महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई
(स्वायत्त) (ISO 9001:2015) (ISO/IEC 27001:2013)

अभियांत्रिकी आणि तंत्रज्ञान पदविका

शिक्षण पुस्तिका
(Learning Material)

JAVA PROGRAMMING
(314317)

K Scheme

जावा प्रोग्रामिंग

संगणक अभियांत्रिकी गट
(के स्कीम)

मराठी–इंग्रजी (द्विभाषिक) माध्यम
(अभियांत्रिकी व तंत्रज्ञानातील चौथे सत्र पदविका)

जावा प्रोग्रामिंग
JAVA PROGRAMMING
(314317)

मार्गदर्शक

प्रा.विजय नामदेवराव कुकरे
विभागप्रमुख, संगणक अभियांत्रिकी

प्रकल्प समन्वयक

प्रा. गोरक्षनाथ भागवतराव काळे
उपप्राचार्य आणि विभागप्रमुख, संगणक तंत्रज्ञान

संकलक

प्रा.एस.एस.बाहेती
प्रा.एस.एस.बोऱ्हाडे
प्रा.व्ही.एम.भाबड
प्रा.डी.एस.मुसळे
प्रा.बी.बी.शिंदे
अधिव्याख्याते, संगणक तंत्रज्ञान

प्रा.एन.डी.नवले
अधिव्याख्याता, माहिती तंत्रज्ञान

शिक्षण पुस्तिका

(Learning Material)

जावा प्रोग्रामिंग

Java Programming

(314317)

संगणक अभियांत्रिकी गट

(के-स्कीम)

K-Scheme

मराठी-इंग्रजी (द्विभाषिक) माध्यम

अभियांत्रिकी व तंत्रज्ञानातील चौथे सत्र

महाराष्ट्र राज्य तंत्रशिक्षण मंडळ, मुंबई

(स्वायत्त) (ISO 9001:2015) (ISO/IEC 27001:2013)

महाराष्ट्र राज्य तंत्र शिक्षण मंडळ

(स्वायत्त) (ISO: ९००१:२०१५) (ISO/IES: २७००१-२०१३)

शासकीय तंत्रनिकेतन इमारत, चौथा मजला, ४९, खेरवाडी, बांद्रा (पूर्व), मुंबई - ४०० ०५१.

दूरध्वनी क्र.: ०२२-६२५४२१७० / १६१

Email : director@msbte.com Web : www.msbte.org.in

प्रास्ताविक

महाराष्ट्र राज्यातील पदविका स्तरावरील तंत्रशिक्षणामध्ये विद्यार्थ्यांचे रोजगार कौशल्य विकसित करून विद्यार्थ्यांचा सर्वांगीण विकास घडवून आणण्याकरिता महाराष्ट्र राज्य तंत्रशिक्षण मंडळ कटिबद्ध आहे. उद्योगधंद्यातील बदलत्या तंत्रज्ञानाशी संबंधित गरजा लक्षात घेऊन महाराष्ट्र राज्य तंत्र शिक्षण मंडळाकडून पदविका अभ्यासक्रम वेळोवेळी अद्यावत करण्यात येतो . अभियांत्रिकी पदविका अभ्यासक्रम शिकत असतांना संकल्पनात्मक ज्ञान, सुसंगत संदर्भ, प्रश्न विचारणे, विश्वसनिय पुरावे, कारणमीमांसा आणि सुस्पष्ट निकष यांचा वापर करून अर्थाची उकल करण्याची, विश्लेषण व मूल्यमापन करण्याची तसेच तर्काने अनुमान काढण्याची क्षमता म्हणजेच चिकित्सक विचार विद्यार्थ्यांमध्ये अधिक दृढ होतील असा मला विश्वास आहे. जेव्हा विद्यार्थी ज्ञान मिळवण्याच्या माध्यमाशी पूर्णपणे परिचित आणि सोयीस्कर असतात, तेव्हा त्यांच्यासाठी वर्गातील चर्चेत भाग घेणे सोपे होते, संकल्पनात्मक व सैद्धांतिक बाबींचे आकलन परिपूर्ण होते, संज्ञानात्मक क्षमता सुधारते आणि त्यांचा आत्मविश्वास देखील वाढतो या सर्व गोष्टींचा विचार करून मंडळाकडून शैक्षणिक सामुग्रीची निर्मिती करण्यात आलेली आहे . भारत देश हा खेड्यापाड्यातून विकसित झालेला देश असून ग्रामीण भागातील विद्यार्थ्यांना तांत्रिक शिक्षण घेतांना भाषेचा अडसर न येता तांत्रिक बाबींचा आशय समजून घेणे शक्य होईल या दृष्टिकोनातून महाराष्ट्र राज्य तंत्र शिक्षण मंडळाने पदविका स्तरावरील तांत्रिक शिक्षणाकरीता विद्यार्थ्यांना मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय शैक्षणिक वर्ष २०२१-२२ पासून उपलब्ध करून दिलेला आहे .

राष्ट्रीय शैक्षणिक धोरण-२०२० प्रादेशिक भाषेतील शिक्षणास प्रोत्साहन देते, ज्यामुळे विद्यार्थ्यांना तांत्रिक अभ्यासक्रमांसाठी प्रादेशिक भाषांतुन शिक्षणाचे माध्यम निवडता येते. सदर धोरणामुळे प्रादेशिक भाषांमध्ये तांत्रिक सामग्री आणि अभ्यास सामग्रीचा विकास आणि भाषांतर निर्माण करण्याची आवश्यकता आहे . त्यास अनुसरून मंडळाने मराठी-इंग्रजी द्विभाषिक माध्यमाचा पर्याय द्वितीय व तृतीय वर्षाकरिताही उपलब्ध करून देण्यात आला आहे. तसेच त्याकरिताची शैक्षणिक सामग्रीही संबंधीत भागधारकरांना उपलब्ध करून देण्यात येत आहे.

पदविका स्तरावरील तंत्रशिक्षण अधिक दर्जेदार करण्यासाठी महाराष्ट्रातील अनुभवी व तज्ञ अध्यापकांनी व्यवहारिक मराठी भाषा व इंग्रजी भाषेतील तांत्रिक शब्दावली यांचा वापर करून मराठी - इंग्रजी भाषेचा सुवर्णमध्य साधण्याचा प्रयत्न केलेला आहे. मंडळाच्या स्तरावर गठीत सुकाणू समितीमार्फत सदर शैक्षणिक सामुग्रीचा दर्जा, तसेच इतर बाबींची तपासणी करण्यात आलेली आहे. त्यामुळे सदर शैक्षणिक सामुग्री अधिक सम्पन्न झालेली असून विद्यार्थी त्यांच्या व्यक्तिमत्त्वाचा सुसंवादी आणि सर्वांगीण विकास साधतील. परिणामतः विश्वस्तरीय मनुष्यबळाच्या गरजा पूर्ण करण्यात महाराष्ट्र राज्य अग्रेसर राहील व पर्यायाने राष्ट्रनिर्मिती करीता निश्चितच हातभार लागेल , असा मला विश्वास आहे.

अभियांत्रिकी पदविका अभ्यासक्रमातील प्रमुख विषयांची मराठी-इंग्रजी द्विभाषिक शैक्षणिक सामुग्री बनविण्यासाठी अध्यापक व सुकाणू समितीचे सदस्य यांनी दर्शविलेले समर्पण व वचनबद्धता कौतुकास पात्र आहे , या सर्वांचे मी मनः पूवक अभिनंदन करतो !

(प्रमोद नाईक)
संचालक
म. रा. तंत्र शिक्षण मंडळ, मुंबई.

अनुक्रमणिका

अ.नु.	युनिटचे नाव	पृष्ठ क्रमांक
1	जावा प्रोग्रामिंग मधील मूलभूत वाक्यरचना **(Basic Syntactical Construct in java)**	1
2	इनहेरिटेन्स इंटरफेस आणि पॅकेजेस **(Inheritance, Interface and Packages)**	49
3	एक्सेप्शन हँडलिंग आणि मल्टीथ्रेडिंग **(Exception Handling and Multithreading)**	68
4	ॲबस्ट्रॅक्ट विंडो टूलकिट (AWT) आणि स्विंग्ज वापरून इव्हेंट हँडलिंग **(Event handling using Abstract Window Toolkit (AWT) & Swings Components)**	84
5	बेसिक ऑफ नेटवर्किंग **(Basics of Network Programming)**	102
6	डेटाबेसशी संवाद **(Interacting with Database)**	118

युनिट - 1
जावा प्रोग्रामिंग मधील मूलभूत वाक्यरचना
(Basic Syntactical Constructs in Java)

विषय निष्पत्ती (Course Outcome):

CO1-क्लास आणि ऑब्जेक्ट्स वापरुन जावा प्रोग्राम विकसित करा.

सिद्धांत शिक्षण परिणाम (Theory Learning Outcomes (TLO)):

1. दिलेल्या समस्येसाठी क्लास आणि ऑब्जेक्ट वापरून प्रोग्राम लिहा.
2. दिलेल्या जावा टोकनची वैशिष्ट्ये स्पष्ट करा.
3. दिलेल्या एक्स्प्रेशन चे मूल्यांकन करण्यासाठी प्रोग्राम लिहा.
4. दिलेल्या समस्येचे निराकरण करण्यासाठी संबंधित कन्ट्रोल स्ट्रक्चर वापरुन प्रोग्राम लिहा.
5. दिलेल्या समस्येसाठी वेक्टर आणि वग्रापर क्लासेसचा वापर करून प्रोग्राम विकसित करा.
6. प्रोग्रामिंग च्या समस्येसाठी कन्स्ट्रक्टर वापरा.

1.1 जावाची वैशिष्ट्ये आणि जावा प्रोग्रामिंग इन्व्हायर्नमेंट
(Java Features and the Java Programming Environment)

1.1.1 जावाची वैशिष्ट्ये (Features of Java)

1.सिम्पल (Simple)

जावा शिकायला खूप सोपे आहे, आणि त्याची वाक्यरचना सोपी आहे आणि समजण्यास सोपी आहे. जावा ची वाक्यरचना C++ वर आधारित आहे म्हणून java सोपी भाषा आहे . Java ने अनेक क्लिष्ट (Complicated) आणि क्वचित वापरलेली वैशिष्ट्ये काढून टाकली आहेत. Java मध्ये ऑटोमॅटिक गार्बेज कलेक्शन असल्यामुळे संदर्भ नसलेल्या ऑब्जेक्ट (object) काढण्याची गरज नाही.

2.ऑब्जेक्ट ओरिएंटेड(Object-Oriented)

जावा ही ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग भाषा (object-oriented programming language)आहे. जावा मधील प्रत्येक गोष्ट एक ऑब्जेक्ट (object) आहे. ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग (OOPs) ही एक पद्धत आहे जी काही नियम प्रदान करून सॉफ्टवेअर विकास आणि देखभाल सुलभ करते. ऑब्जेक्ट-ओरिएंटेड म्हणजे आम्ही आमचे सॉफ्टवेअर विविध प्रकारच्या ऑब्जेक्ट्सचे संयोजन म्हणून व्यवस्थापित करतो ज्यामध्ये डेटा आणि वर्तन दोन्ही समाविष्ट असतात.

3.पोर्टेबल(Portable)

जावा हे पोर्टेबल आहे कारण ते तुम्हाला कोणत्याही प्लॅटफॉर्मवर जावा bytecode वापरण्याची सुविधा देते. त्याला कोणत्याही अंमलबजावणीची आवश्यकता नाही.

4.प्लॅटफॉर्म स्वतंत्र (Platform independent)

जावा हे प्लॅटफॉर्म स्वतंत्र आहे कारण ते C, C++, इत्यादी इतर भाषांपेक्षा वेगळे आहे जे प्लॅटफॉर्म विशिष्ट मशीनमध्ये संकलित केले जाते .जावाही एकदा लिहा, कुठेही चालवा. प्लॅटफॉर्म म्हणजे हार्डवेअर किंवा सॉफ्टवेअर वातावरण ज्यामध्ये प्रोग्राम चालतो. जावा प्लॅटफॉर्म हा सॉफ्टवेअर-आधारित प्लॅटफॉर्म आहे जो इतर हार्डवेअर-आधारित प्लॅटफॉर्मच्या वर चालतो या अर्थाने इतर प्लॅटफॉर्मपेक्षा वेगळा आहे. जावाकोड एकाधिक प्लॅटफॉर्मवर कार्यान्वित केला जाऊ शकतो, उदाहरणार्थ, Windows, Linux, Sun Solaris, Mac/OS, इ जावा कोड कंपाइलरद्वारे संकलित(compile) केला जातो आणि बाइट कोड(bytecode)मध्ये रूपांतरित केला जातो. हा बाइटकोड एक प्लॅटफॉर्म-स्वतंत्र कोड आहे कारण तो एकाधिक प्लॅटफॉर्मवर चालवला जाऊ शकतो, म्हणजे एकदा लिहा आणि कुठेही चालवा (Write Once and Run Anywhere WORA).

5.सुरक्षित(Secured) -जावा त्याच्या सुरक्षिततेसाठी प्रसिद्ध आहे. जावासह, आम्ही व्हायरस-मुक्त प्रणाली विकसित करू शकतो. जावासुरक्षित आहे कारण: कोणतेही स्पष्ट सूचक नाही (No explicit pointer) जावा प्रोग्राम वर्च्युअल मशीन सँडबॉक्समध्ये (virtual machine sandbox) चालतात.

- **क्लासलोडर (Classloader):** Java मधील क्लासलोडर हा Java Runtime Environment (JRE) चा एक भाग आहे जो Java क्लासेस जावा वर्च्युअल मशीनमध्ये डायनामिकली लोड करण्यासाठी वापरला जातो. स्थानिक फाइल सिस्टमच्या वर्गांसाठी(Class) लागणारे पॅकेजला नेटवर्क सौरस कडून आयात केलेल्या वर्गांपासून(Class) वेगळे करून सुरक्षा जोडली जाते.
- **बाईटकोड व्हेरीफायर** (**Bytecode Verifier)** हे बेकायदेशीर कोडसाठी कोडचे तुकडे (Code Fragment)तपासते जे ऑब्जेक्ट्सच्या अधिकारांचे उल्लंघन करू शकतात .
- **सिक्युरिटी मॅनेजर(Security Manager):** लोकल डिस्कवर वाचणे(read) आणि लिहिणे(write) यासारख्या कोणत्या संसाधन क्लास वापर करू शकतो हे ते ठरवते.

6.रोबस्ट(Robust)

Robust चा मराठी अर्थ मजबूत असा आहे. जावामजबूत आहे कारण: हे मजबूत मेमरी व्यवस्थापन वापरते. सुरक्षा समस्या टाळणाऱ्या पॉइंटरचा अभाव आहे. जावास्वयंचलित गार्बेज कलेक्टर (Garbage Collector) प्रदान करते जे जावाअपलीकेशन द्वारे वापरल्या जात नसलेल्या ऑब्जेक्ट (object) पासून मुक्त होण्यासाठी जावाव्हर्च्युअल मशीनवर चालते. जावामध्ये एक्षेपषण हंड्लिंग (exception handling)आणि टाईप चेक्किंग (type checking)यंत्रणा आहे. हे सर्व मुद्दे जावाला मजबूत करतात.

7. आर्किटेक्चर न्यूट्रल (Architecture neutral)

जावाआर्किटेक्चर न्यूट्रल आहे कारण तेथे अंमलबजावणीवर अवलंबून वैशिष्ट्ये नाहीत, उदाहरणार्थ, प्रीमिटीव डेटा टाईपची साईज निश्चित आहे.

8. हाय पेर्फोरमस (High Performance)

जावा इतर पारंपारिक व्याख्या केलेल्या प्रोग्रामिंग भाषांपेक्षा वेगवान आहे कारण जावा bytecode नेटिव्ह कोडच्या "जवळ" आहे. संकलित भाषेपेक्षा (Compiled language) जावा अजून थोडी हळू आहे.

9. मल्टी थ्रेडेड (Multithreaded)

थ्रेड(Thread) हा एका वेगळ्या प्रोग्रामसारखा असतो, जो एकाच वेळी कार्यान्वित होतो. आपण वेग वेगळे जावा प्रोग्राम लिहू शकतो जे एकाच वेळी अनेक थ्रेड्स (Multiple Threads) परिभाषित करून अनेक कार्ये हाताळतात. मल्टी-थ्रेडिंगचा मुख्य फायदा म्हणजे तो प्रत्येक थ्रेडसाठी मेमरी व्यापत नाही. हे एक सामान्य मेमरी(Common Memory) क्षेत्र वापरते . मल्टी-मीडिया, वेब ऍप्लिकेशन्स इत्यादींसाठी थ्रेड महत्त्वाचे आहेत.

10. डीस्ट्रीब्यूटेड (Distributed)

जावाडीस्ट्रीब्यूटेड Distributed आहे कारण ते वापरकर्त्यांना जावामध्ये डीस्ट्रीब्यूटेड अप्पलीकेशन तयार करण्यास सुलभ करते. डीस्ट्रीब्यूटेड ॲप्लिकेशन तयार करण्यासाठी RMI आणि EJB वापरले जातात. जावाचे हे वैशिष्ट्य आम्हाला इंटरनेटवरील कोणत्याही मशीनवरून मेथडस कॉल करून फाईल एक्सेस करण्यास सक्षम करते.

11. डायनॅमिक (Dynamic)

जावाही डायनॅमिक भाषा आहे. हे क्लास च्या डायनॅमिक लोडिंगला समर्थन देते. याचा अर्थ क्लास मागणीनुसार लोड केले जातात.

1.1.2 जावा प्रोग्रामिंग एनवायरमेंट जावा (Programming Environment)

जावाहे जेम्स गोस्लिंग यांनी सन मायक्रोसिस्टम्स (Sun Microsystems) येथे सन 1995 मध्ये विकसित केले होते, नंतर ओरॅकल कॉर्पोरेशनने (Oracle Corporation) विकत घेतले. ही एक सोपी प्रोग्रामिंग (programming) भाषा आहे. जावाप्रोग्रामिंग लेखन, कंपैलिंग (compiling) आणि डीबगिंग(debugging) सुलभ करते. पुन्हा वापरता येण्याजोगे कोड आणि मॉड्यूलर प्रोग्राम तयार करण्यास जावा मदत करते. जावाही वर्ग-आधारित(class-based),ऑब्जेक्ट ओरिएंटेड प्रोग्रामिंग (object-oriented programming) भाषा आहे. जावाकोड सर्व प्लॅटफॉर्मवर रन होऊ शकतो (write once run anywhere) बनविलेली एक सामान्य-उद्देश प्रोग्रामिंग भाषा आहे. कमपाइल (compile) केलेला जावाकोड जावाला सपोर्ट करणाऱ्या सर्व प्लॅटफॉर्मवर चालू शकतो.

जावाॲप्लिकेशन्स बाइट कोडमध्ये (byte code) कमपाइल (compile) केले जातात जे कोणत्याही जावा व्हर्च्युअल मशीनवर (java Virtual Machine) चालू शकतात.

Java चे वाक्यरचना (syntax) C /C++ सारखे आहे. Java शिकण्यापूर्वी, Java च्या खालील सामान्य संज्ञांशी (Terminology) परिचित असणे आवश्यक आहे.

- **जावा व्हर्च्युअल मशीन (Java Virtual Machine (JVM))**

प्रोग्रामच्या अंमलबजावणीचे तीन टप्पे असतात.ते प्रोग्राम लिहीले जातात, कमपाइल (compile) करतात आणि अंमलात(Execute) आणतात. प्रोग्राम लिहिणे Java प्रोग्रामरद्वारे केले जाते. कमपाइल (compile) javac कंपाइलरद्वारे केले जाते जे Java डेवलपमेंट किट (JDK) मध्ये समाविष्ट केलेले प्राथमिक Java कंपाइलर(compiler) आहे. हे Java प्रोग्राम इनपुट(Input) म्हणून घेते आणि आउटपुट (output)म्हणून बाइट कोड(bytecode)तयार करते. प्रोग्रॅमच्या रनिंग फेजमध्ये, कंपाइलरद्वारे उत्पादित बाइट कोड(bytecode) JVM द्वारे कार्यान्वित (Execute)केला जातो. जावा व्हर्च्युअल मशीनचे (JVM) कार्य कंपाइलरद्वारे उत्पादित बाइट कोड(bytecode) कार्यान्वित(Execute) करणे आहे. प्रत्येक ऑपरेटिंग सिस्टीममध्ये भिन्न JVM असते परंतु बाईटेकोडच्या अंमलबजावणीनंतर ते जे आउटपुट तयार करतात ते सर्व ऑपरेटिंग सिस्टममध्ये सारखेच असते. म्हणूनच Java प्लॅटफॉर्म- इनडीपेनडेंट (platform-independent)भाषा म्हणून ओळखली जाते.

- **बाइट कोड इन डेवलपमेंट प्रोसेस (Bytecode in the Development process)**

JDK चा Javac कंपायलर java सोर्स कोड(source code) बाइट कोड(bytecode) मध्ये संकलित करतो जेणेकरून तो JVM द्वारे कार्यान्वित केला जाऊ शकतो. कंपाइलरद्वारे(Compiler) ती .class फाइल म्हणून सेव्ह(save) केली जाते. बाइटकोड(bytecode) पाहण्यासाठी, javap सारखे डिस्सेम्बलर(disassembler) वापरले जाऊ शकते.

- **Java डेवलपमेंट किट (Java Development Kit(JDK)):**

हे संपूर्ण जावा डेव्हलपमेंट किट आहे ज्यामध्ये कंपाइलर(compiler), जावा रनटाइम एन्व्हायर्नमेंट (Java Runtime Environment (JRE)), जावा डीबगर्स(java debuggers), जावा डॉक्स (java docs)इत्यादी सर्व गोष्टींचा समावेश आहे. java मध्ये प्रोग्राम कार्यान्वित(execute) करण्यासाठी, जावा प्रोग्राम तयार करण्यासाठी, संकलित(compile) करण्यासाठी आणि रन करण्यासाठी(run) आम्हाला आमच्या संगणकावर JDK स्थापित (Install) करणे आवश्यक आहे.

- **Java रनटाइम एन्व्हायर्नमेंट (Java Runtime Environment (JRE)**

JDK मध्ये JRE समाविष्ट आहे. संगणकावर JRE इंस्टॉलेशन असेल तर जावा प्रोग्रामला चालवण्यास(Run) अनुमती देते, नाहीतर ते जावा प्रोग्राम संकलित(Compile) करू शकत नाही. JRE मध्ये ब्राउझर(browser), JVM, ऍपलेट (applet)सपोर्ट आणि प्लगइन(plugins) समाविष्ट आहेत. जावा प्रोग्राम चालवण्यासाठी(Run), संगणकाला JRE आवश्यक आहे.

- **गार्बेज कलेक्टर (Garbage Collector):**

Java मध्ये, प्रोग्रामर ऑब्जेक्ट डिलीट करू शकत नाहीत.मेमरी फ्री करण्यासाठी JVM मध्ये प्रोग्राम आहे त्याला गार्बेज कलेक्टर (Garbage Collector)म्हणतात . गार्बेज कलेक्टर (Garbage Collector) संदर्भित नसलेल्या ऑब्जेक्टला recollect करतात. त्यामुळे Java मेमरी व्यवस्थापन हाताळून प्रोग्रामरचे काम सोपे करते. तथापि, प्रोग्रामरने त्यांच्या कोडबद्दल सावधगिरी बाळगली पाहिजे की ते बर्याच काळापासून वापरल्या गेलेल्या ऑब्जेक्ट वापरत आहेत. कारण गार्बेज कलेक्टर संदर्भित ऑब्जेक्टची मेमरी पुनर्प्राप्त करू शकत नाही.

- **क्लासपाथ(classpath):**

क्लासपाथ हा फाईल पाथ आहे जिथे Java रनटाइम आणि जावा कंपाइलर(java Compiler).क्लास (class)फाइल्स लोड करण्यासाठी शोधतात. डीफॉल्टनुसार, JDK अनेक लायब्ररी प्रदान करते. जर तुम्हाला बाह्य लायब्ररी समाविष्ट करायच्या असतील तर त्या क्लासपाथ (ClassPath)मध्ये जोडल्या पाहिजेत.

1.2.1 क्लास बनवणे (Defining a class)

Java मधील क्लास हा केवळ तार्किक घटक(logical entity) आहे. क्लास म्हणजे ऑब्जेक्ट चा समूह ज्यामध्ये सामान्य गुणधर्म असतात.

हे एक टेम्पलेट(Template) किंवा ब्लूप्रिंट (Blueprint) आहे ज्यातून ऑब्जेक्ट तयार केले जातात. Java मधील क्लास मध्ये खालील गोष्टी समाविष्ट असू शकते:

- फील्ड(Fields)
- पद्धती(Methods)
- कन्स्ट्रक्टर(Constructor)
- अवरोध(Blocks)
- नेस्टेड क्लास आणि इंटरफेस(nested class and interfaces)

- **क्लास घोषित करण्यासाठी वाक्यरचना (Syntax to declare a class)**

```
class <class_name>
 {     field;
       method;
  }
```

क्लास (class) मेमरी व्यापत नाही. क्लास (class) हा विविध डेटा प्रकारांच्या (data types) व्हेरीएबलचा (variable) समूह आणि पद्धतींचा (Methods)समूह आहे .

1.2.2 ऑब्जेक्ट तयार करण्यासाठी वाक्यरचना (syntax for creating an object)

ऑब्जेक्ट बनवणे (creating object) ऑब्जेक्ट OOPs भाषेचा मूलभूत बिल्डिंग ब्लॉक आहे. Java मध्ये, आपण ऑब्जेक्ट तयार केल्याशिवाय कोणताही प्रोग्राम कार्यान्वित करू शकत नाही. जावा ऑब्जेक्ट तयार करण्याचे पाच मार्ग प्रदान करते. new कीवर्ड वापरणे हा क्लासचा ऑब्जेक्ट तयार करण्याचा सर्वात लोकप्रिय मार्ग आहे. हे नवीन तयार केलेल्या ऑब्जेक्टसाठी मेमरी वाटप (allocate)करते आणि त्या ऑब्जेक्टचा संदर्भ(reference) त्या मेमरीमध्ये परत करते. new कीवर्ड अॅरे(array) तयार करण्यासाठी देखील वापरला जातो.

```
ClassName object = new ClassName();
```

1.2.3 क्लास मेम्बर वापरणे (Accessing class members)

(.) डॉट ऑपरेटरचा वापर क्लास मधील मेम्बर्सना एक्सेस करण्यासाठी केला जातो.

```
class Bicycle {
  // field of class
  int gear = 5;
  // method of class
  void speed() {
    ...
  } }
Bicycle cycle = new Bicycle();
// access field and method
cycle.gear;
cycle.speed();
```

1.3 Java टोकन आणि डेटा टाईप (Java Tokens and Data types)

1.3.1 Java टोकन- Java कंपाइलर कोडच्या ओळीला मजकूर(text) (शब्द) मध्ये मोडतो त्याला Java टोकन म्हणतात. हे जावा प्रोग्रामचे सर्वात लहान घटक आहेत. Java कंपाइलरने हे शब्द टोकन म्हणून ओळखते. हे टोकन डिलिमिटरद्वारे वेगळे केले जातात. कंपाइलर्सना त्रुटी शोधण्यासाठी ते उपयुक्त आहे. डिलिमिटर Java टोकन्सचा भाग नाहीत.

```
public class Demo
{
public static void main(String args[])
```

```
{
System.out.println("Amrutvahini");
} }
```

वरील कोड मध्ये, public, class, Demo, {, static, void, main, (, String, args, [,],), System, . , out, println, Amrutvahini etc. हे Java टोकन आहे .

- **टोकनचे प्रकार (Types of Tokens)**
  - कीवर्ड Keywords
  - आयडेनटीफायर Identifiers
  - लिटरल Literals
  - ऑपरेटर्स Operators
  - सेपरेटर Separators
  - कमेंट Comments
- **कीवर्ड (Keywords)** : हे कोणत्याही प्रोग्रामिंग भाषेचे पूर्व-परिभाषित आरक्षित शब्द(**pre-defined reserved words)** आहेत. प्रत्येक कीवर्डचा एक विशेष अर्थ असतो. हे नेहमी लहान अक्षरात(Lower case) लिहिले जाते. Java खालील कीवर्ड प्रदान करते:

Table 1.1 कीवर्ड(Keywords)

abstract	Boolean	Byte	break	class
case	Catch	Char	continue	default
do	Double	Else	extends	final
finally	Float	For	if	implements
import	Instanceof	Int	interface	long
native	Super	Package	private	protected
public	Return	Short	static	new
switch	synchronized	This	thro	throws
transient	Try	Void	volatile	while
assert	Const	Enum	goto	strictfp

आयडेनटीफायर (Identifier): व्हेरिएबल, कॉन्स्टंट, फंक्शन, क्लास आणि अॅरे यांना नाव देण्यासाठी आयडेंटिफायर वापरले जातात. हे सहसा युजरद्वारे परिभाषित(Define) केले जाते. आयडेनटीफायर साठी अक्षरे, अंडरस्कोअर(_) किंवा डॉलर चिन्ह($)प्रथम वर्ण म्हणून वापरले जातात .लेबलला एक विशेष प्रकारचा आयडेनटीफायर म्हणून देखील ओळखले जाते जे गोटू(goto) स्टेटमेंटमध्ये वापरले जाते. लक्षात ठेवा की आयडेनटीफायर चे नाव आरक्षित कीवर्डपेक्षा वेगळे असणे आवश्यक आहे.

आयडेनटीफायर घोषित करण्यासाठी काही नियम आहेत:

- आयडेनटीफायर पहिले अक्षर अक्षर, अंडरस्कोर किंवा डॉलरचे चिन्ह असणे आवश्यक आहे. ते अंकांनी सुरू होऊ शकत नाही परंतु त्यात अंक असू शकतात.
- व्हाईटस्पेस आयडेनटीफायर मध्ये समाविष्ट केली जाऊ शकत नाही.
- आयडेंटिफायर केस सेन्सेटिव्ह (case sensitive) असतात.

खालील काही वैध आयडेंटिफायर(valid identifiers) आहेत

```
PhoneNumber    _phonenumber    PRICE    $circumference
radius         a               a1       jagged_array
12radius   //invalid
```

- **लिटरल (Literals)**:

प्रोग्रामिंगमध्ये लिटरल Literals एक नोटेशन आहे जे कोडमध्ये निश्चित मूल्य (Constant) दर्शवते. त्याचे वर्गीकरण इनटीजर लिटरल(integer literal), स्ट्रिंग लिटरल(string literal), बुलियन लिटरल(Boolean literal), इ.
हे प्रोग्रामरद्वारे परिभाषित केले जाते. एकदा व्याख्या केली की बदलता येत नाही. Java खालीलप्रमाणे पाच प्रकारची लिटरल प्रदान करते:
इनटीजर Integer
फ्लोटिंग पोईट Floating Point
क्यारेक्टर Character
स्ट्रिंग String
बुलियन Boolean

Table 1.2 लिटरल (Literals)

Literal	**Type**
25	Int
3.14	Double
false, true	Boolean
'S', '1', '-'	Char
"Amrutvahini"	String
Null	any reference type

- **ऑपरेटर्स (Operators)**

प्रोग्रामिंगमध्ये, ऑपरेटर हे विशेष चिन्ह आहे जे कंपाइलरला विशेष ऑपरेशन करण्यास सांगते. Java विविध प्रकारचे ऑपरेटर प्रदान करते जे ते प्रदान केलेल्या कार्यक्षमतेनुसार वर्गीकृत केले जातात. जावामध्ये आठ प्रकारचे ऑपरेटर आहेत, ते खालीलप्रमाणे आहेत

- अरीथ्मेटिक ऑपरेटर्स Arithmetic Operators
- असायनमेंट ऑपरेटर्स Assignment Operators
- रिलेश्नल ऑपरेटर्स Relational Operators
- युनरी ऑपरेटर्स Unary Operators
- लोजीकॅल ऑपरेटर्स Logical Operators
- टरनरी ऑपरेटर्स Ternary Operators
- बीटवाईज ऑपरेटर्स Bitwise Operators
- शिफ्ट ऑपरेटर्स Shift Operators

Table 1.3 ऑपरेटर्स (Operators)

Operator	Symbols
Arithmetic	+ , - , / , * , %
Unary	++ , - - , !
Assignment	= , += , -= , *= , /= , %= , ^=
Relational	==, != , < , >, <= , >=
Logical	&& , ||
Ternary	(Condition) ? (Statement1) : (Statement2);
Bitwise	& , | , ^ , ~
Shift	<< , >> , >>>

- **सेपरेटर (Separators)**:

जावामधील विभाजकांना(Separator)विरामचिन्हे (punctuators)असेही म्हणतात. जावामध्ये नऊ विभाजक(Separator) आहेत, ते खालीलप्रमाणे आहेत:

- **स्क्रेअर ब्रयकेट Square Brackets []:**हे अॅरे घटक परिभाषित करण्यासाठी वापरले जाते. स्क्रेअर ब्रयकेट ची जोडी एकल-आयामी अॅरे(one dimensional) दर्शवते, चौरस कंसाच्या दोन जोड्या द्विमितीय अॅरेचे(two dimensional) प्रतिनिधित्व करतात.
- **प्यारेनथेसिस Parentheses ():**हे फंक्शन्स कॉल करण्यासाठी आणि पॅरामीटर्स पार्स करण्यासाठी वापरले जाते.
- **करली ब्रेसेस Curly Braces {}: करली ब्रेसेस** कोड ब्लॉकची सुरुवात आणि शेवट दर्शवतात.
- **कॉमा Comma (,):**हे दोन मूल्ये(values), विधाने (statements)आणि मापदंड(parameters) वेगळे करण्यासाठी वापरले जाते.
- **असायनमेंट ऑपरेटर्स Assignment Operator (=):** हे व्हेरिएबल आणि स्थिरांक (Constant) नियुक्त करण्यासाठी वापरले जाते.
- **सेमीकोलन Semicolon (;):**विधानांच्या(statement) शेवटी आढळणारे हे चिन्ह आहे. हे दोन विधान(statement) वेगळे करते.
- **पिरेड Period (.):** हे उप-पॅकेजेस आणि पॅकेज वर्गाचे नाव वेगळे करते. हे वेरिएबल किंवा पद्धत संदर्भ व्हेरिएबलपासून वेगळे करते.
- **कमेंट (Comments):** कमेंट आम्हाला आमच्या Java कोडमध्ये प्रोग्रामबद्दल माहिती निर्दिष्ट करण्याची परवानगी देतात. Java कंपाइलर या कमेंट ना टोकन म्हणून ओळखतो परंतु पुढील प्रक्रियेसाठी ते वगळतो. Java कंपाइलर कमेंट हाताळतो. Java कंपाइलर टिप्पण्यांना व्हाइटस्पेस म्हणून हाताळतो.प्रकारच्या टिप्पण्या प्रदान करते:
  लाइन ओरिएंटेड: हे फॉरवर्डिंग स्लॅशच्या जोडीने सुरू होते (//).
  ब्लॉक-ओरिएंटेड: हे /* ने सुरू होते आणि */ सापडत नाही तोपर्यंत ते चालू राहते.

1.3.2 डेटा टाईप (Data type)

डेटा प्रकार भिन्न आकार(Size) आणि मूल्ये (value)निर्दिष्ट करतात जे व्हेरिएबलमध्ये संग्रहित केले जाऊ शकतात. Java मध्ये दोन प्रकारचे डेटा प्रकार आहेत:

प्रीमिटीव (primitive) डेटा प्रकार: प्रीमिटीव (primitive) डेटा प्रकारांमध्ये बुलियन(boolean) ,कयार (char) बाइट (byte), शॉर्ट (short),इंट(int,),लाँग (long) फ्लोट (float) आणि डबल (double)यांचा समावेश होतो.

नॉन-प्रिमिटिव डेटा प्रकार **(non primitive data type)**: नॉन-प्रिमिटिव डेटा प्रकारांमध्ये क्लास (class), इंटरफेस (interface)आणि अॅरे(Array)यांचा समावेश होतो.

Java **प्रीमिटीव (primitive**) डेटा प्रकार

जावा भाषेत, प्रीमिटीव (primitive) डेटा प्रकार हे डेटा हाताळणीचे बिल्डिंग ब्लॉक्स आहेत. हे Java भाषेत उपलब्ध असलेले सर्वात मूलभूत डेटा प्रकार आहेत.

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type

Table 1.4 : Java प्रीमिटीव (primitive) डेटा प्रकार

डेटा प्रकार **Data Type**	**डीफॉल्ट मूल्य** **Default Value**	**डीफॉल्ट** आकार **Default size**
Boolean	False	1 bit
Char	'\u0000'	2 byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
Long	0L	8 byte
Float	0.0f	4 byte
Double	0.0d	8 byte

- **बुलियन डेटा प्रकार Boolean Data Type**

बुलियन डेटा प्रकार फक्त दोन संभाव्य मूल्ये संग्रहित करण्यासाठी वापरला जातो: सत्य(true) आणि असत्य(false).बुलियन डेटा प्रकार एक बिट माहिती निर्दिष्ट करतो.

उदाहरण Example:

boolean one = false

- **बाइट डेटा प्रकार Byte Data Type**

बाइट डेटा प्रकार हे प्रीमिटीव (primitive)डेटा प्रकाराचे उदाहरण आहे. हे 8-बिट साइनड(signed) टूज कॉम्पलीमेंट आहे. त्याची मूल्य-श्रेणी -128 ते 127 दरम्यान आहे

- **शॉर्ट डेटा प्रकार Short Data Type**

शॉरट डेटा प्रकार हे 16-बिट साइनड(signed) टूज कॉम्पलीमेंट आहे. त्याची मूल्य-श्रेणी -32,768 to 32,767 दरम्यान आहे

उदाहरण Example:

short s = 10000, short r = -5000

- **इंट डेटा प्रकार Int Data Type**

हे 32-बिट साइनड(signed) टूज कॉम्पलीमेंट आहे. त्याची मूल्य-श्रेणी - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) दरम्यान आहे.

उदाहरण Example:

int a = 100000, int b = -200000

- **लाँग डेटा प्रकार Long Data Type**

लाँग डेटा प्रकार हे 64-बिट साइनड (signed) टूज कॉम्पलीमेंट आहे. त्याची मूल्य-श्रेणी 9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1) दरम्यान आहे. जेव्हा तुम्हाला int द्वारे प्रदान केलेल्या मूल्यांपेक्षा अधिक मूल्यांची श्रेणी आवश्यक असते तेव्हा लाँग डेटा प्रकार वापरला जातो.

उदाहरण Example:

long a = 100000L, long b = -200000L

- **फ्लोट डेटा प्रकार Float Data Type**

फ्लोट डेटा प्रकार एकल-परिशुद्धता(single-precision) 32-बिट IEEE 754 फ्लोटिंग पॉइंट आहे.

उदाहरण Example:

float f1 = 234.5f

डबल डेटा प्रकार Double Data Type

- **डबल डेटा प्रकार डबल-परिशुद्धता(double-precision)**

64-बिट IEEE 754 फ्लोटिंग पॉइंट आहे. डबल डेटा प्रकार सामान्यतः फ्लोट प्रमाणे दशांश मूल्यांसाठी (decimal values)वापरला जातो.

उदाहरण Example: double d1 = 12.3

- **क्यार डेटा प्रकार Char Data Type**

क्यार डेटा प्रकार एक सिंगल 16-बिट युनिकोड वर्ण(Unicode character) आहे. त्याची मूल्य-श्रेणी 0 ते 65,535 दरम्यान आहे. वर्ण(characters.) संग्रहित करण्यासाठी char डेटा प्रकार वापरला जातो.

Example:

char letterA = 'A'

1.3.2 कौन्सटन्ट (Constants and Symbolic Constants)

कौन्सटन्ट हा प्रोग्रामिंगमधील एक घटक आहे जो अपरिवर्तनीय(immutable) आहे. दुसऱ्या शब्दांत, मूल्य(value) जे बदलले जाऊ शकत नाही. Java थेट कौन्सटन्टना समर्थन देत नाही. जावामध्ये स्टेटीक(static) आणि अंतिम(final) नॉन-एक्सेस मॉडिफायर्स वापरून कौन्सटन्ट परिभाषित करण्याचा पर्यायी मार्ग आहे. जावा नामकरण नियमानुसार आयडेनटीफायरचे नाव मोठ्या अक्षरात असणे आवश्यक आहे.

Syntax**:**

```
static final datatype identifier_name=value;
```

Example :

```
static final float PI=3.14;
```

वरील विधानात, स्टॅटिक (static)मॉडिफायर व्हेरिएबलला त्याचा परिभाषित क्लासचा (class) इन्सटन्स (Instance)वापरल्या शिवाय उपलब्ध करून देतो आणि फायनल (final) सुधारक व्हेरिएबल निश्चित करतो.

1.3.3 स्कोप ऑफ वेरीएबल (Scope of Variables)

व्हेरिएबल हे कंटेनर आहे जे Java प्रोग्राम कार्यान्वित असताना मूल्य(value) धारण करते. डेटा टाईप(data type) सह व्हेरिएबल नियुक्त केले आहे.व्हेरिएबल्स चे मूल्य बदलले जाऊ शकते.

व्हेरिएबल हे मेमरी स्थानाचे नाव आहे. जावामध्ये तीन प्रकारचे व्हेरिएबल्स आहेत: लोकल (local),इन्सटन्स (instance) and स्टॅटिक (static.)जावामध्ये दोन प्रकारचे डेटा प्रकार आहेत:

प्रीमिटीव (primitive)आणि नॉन प्रीमिटीव(non primitive).

int data=50;//Here data is variable

प्रोग्रामिंगमध्ये, व्हेरिएबलची व्याप्ती परिभाषित करते की विशिष्ट व्हेरिएबल प्रोग्राममध्ये किंवा सर्व क्लास मध्ये कसे प्रवेशयोग्य आहे. प्रोग्रामिंगमध्ये, व्हेरिएबल क्लास, पद्धत किंवा ब्लॉकमध्ये घोषित आणि परिभाषित केले जाऊ शकते. हे व्हेरिएबलची व्याप्ती परिभाषित करते म्हणजे व्हेरिएबलची दृश्यमानता किंवा प्रवेशयोग्यता. ब्लॉक किंवा पद्धतीमध्ये घोषित केलेले व्हेरिएबल बाहेरून दिसत नाहीत. आम्ही असे करण्याचा प्रयत्न केल्यास, आम्हाला एक संकलन त्रुटी मिळेल.

आम्ही प्रोग्राममध्ये कुठेही व्हेरिएबल्स घोषित करू शकतो परंतु त्याला मर्यादित व्याप्ती आहे.

1. व्हेरिएबल हे मेथड किंवा कन्स्ट्रक्टरचे पॅरामीटर असू शकते.
2. व्हेरिएबल मेथड आणि कन्स्ट्रक्टरच्या मुख्य भागामध्ये परिभाषित आणि घोषित केले जाऊ शकते.
3. हे ब्लॉक्स आणि लूपमध्ये देखील परिभाषित केले जाऊ शकते.
4. main() फंक्शनमध्ये घोषित केलेले व्हेरिएबल main() फंक्शनच्या बाहेर ॲक्सेस करता येत नाही

- **Java मधील व्हेरिएबल स्कोप बद्दल काही महत्वाचे मुद्दे:**

सर्वसाधारणपणे, कुरळे कंसाचा संच (opening and closing breces) { } व्याप्ती(scope) परिभाषित करतो.Java मध्ये आपण सामान्यतः व्हेरिएबल ॲक्सेस करू शकतो जोपर्यंत तो आपण लिहित असलेल्या कोडच्या समान कंसाच्या सेटमध्ये किंवा ज्या कर्ली ब्रॅकेटमध्ये व्हेरिएबल परिभाषित केले आहे त्यामध्ये कोणत्याही कर्ली ब्रॅकेटमध्ये परिभाषित केले आहे.

कोणत्याही पद्धतीच्या बाहेर वर्गामध्ये परिभाषित केलेले कोणतेही व्हेरिएबल सर्व सदस्य पद्धतींद्वारे वापरले जाऊ शकते. जेव्हा एखाद्या मेथडमध्ये सदस्यासारखे स्थानिक व्हेरिएबल असते,तेव्हा "this" कीवर्ड वापरला जाऊ शकतो .

- **लोकल व्हेरिएबल (Local Variable)**

पद्धतीच्या(methods) मुख्य भागामध्ये घोषित केलेल्या व्हेरिएबलला लोकल व्हेरिएबल म्हणतात. तुम्ही हे व्हेरिएबल फक्त त्या पद्धतीतच(methods)वापरू शकता आणि क्लासमधील इतर पद्धतींना(methods)व्हेरिएबल अस्तित्त्वात असल्याची माहितीही नसते.

लोकल व्हेरिएबलस ना static कीवर्ड के वापरू शकत नही

- **इन्स्टन्स व्हेरिएबल (Instance Variable)**

क्लासच्या आत पण मेथडच्या मुख्य भागाच्या बाहेर घोषित केलेल्या व्हेरिएबलला इन्स्टन्स व्हेरिएबल म्हणतात. ते स्थिर म्हणून घोषित केलेले नाही.याला इन्सटन्स व्हेरिएबल व्हेरिएबल असे म्हणतात कारण त्याचे मूल्य उदाहरण-विशिष्ट आहे आणि इन्सटन्स व्हेरिएबल मध्ये सामायिक केले जात नाही.

- **स्टॅटिक व्हेरिएबल(Static Variable)**

स्टॅटिक म्हणून घोषित केलेल्या व्हेरिएबलला स्टॅटिक व्हेरिएबल म्हणतात. ते स्थानिक असू शकत नाही. तुम्ही स्टॅटिक व्हेरिएबलची एकच प्रत तयार करू शकता आणि ती क्लास मधील सर्व घटनांमध्ये सामायिक करू शकता. जेव्हा मेमरीमध्ये क्लास लोड केला जातो तेव्हा स्टॅटिक व्हेरिएबल्ससाठी मेमरी वाटप फक्त एकदाच होते

```
public class A
{
   static int m=100;//static variable
   void method()
   {
      int n=90;//local variable
   }
   public static void main(String args[])
   {
      int data=50;//instance variable
   }
}//end of class
```

उदाहरण :

```
public class Demo
{
 //instance variable
String name = "Andrew";
//class and static variable
static double height= 5.9;
public static void main(String args[])
{
//local variable
int marks = 72;
} }
```

उदाहरण:

VariableScopeExample1.java

```
public class VariableScopeExample1
{
public static void main(String args[])
{
```

```
int x=10;
{
//y has limited scope to this block only
int y=20;
System.out.println("Sum of x+y = " + (x+y));
}
//here y is unknown
y=100;
//x is still known
x=50;
}
}
```

Output:

```
/VariableScopeExample1.java:12: error: cannot find symbol
y=100;
^
  symbol:   variable y
  location: class VariableScopeExample1
1 error
```

1.3.4 Java मध्ये टाइप कास्टिंग (Typecasting in Java)

Java मध्ये, टाइप कास्टिंग ही एक पद्धत किंवा प्रक्रिया आहे जी डेटा प्रकार दुसऱ्या डेटा प्रकारात व्यक्तिचलितपणे आणि स्वयंचलितपणे दोन्ही प्रकारे रूपांतरित करते. स्वयंचलित रूपांतरण कंपाइलर आणि मॅन्युअल रूपांतरण प्रोग्रामरद्वारे केले जाते.

कास्टिंगचे दोन प्रकार आहेत:

- **वायडनिंग टाईप कास्टिंग (Widening Type Casting)**

खालच्या डेटा प्रकाराचे उच्चमध्ये रूपांतर करणे याला वाइडनिंग प्रकार कास्टिंग म्हणतात. याला इम्पलीसीट रूपांतरण implicit conversion किंवा कास्टिंग डाउन असेही म्हणतात. ते आपोआप होते. हे सुरक्षित आहे कारण डेटा गमावण्याची संधी नाही.

हे घडते जेव्हा:

- दोन्ही डेटा प्रकार एकमेकांशी सुसंगत असले पाहिजेत.
- लक्ष्य प्रकार स्त्रोत प्रकारापेक्षा मोठा असणे आवश्यक आहे

byte -> **short** -> **char** -> **int** -> **long** -> **float** -> **double**

उदाहरणार्थ, न्यूमेरिक डेटा प्रकारातील क्यार किंवा बुलियनमधील रूपांतरण आपोआप होत नाही. तसेच, क्यार आणि बुलियन डेटा प्रकार एकमेकांशी सुसंगत नाहीत. एक उदाहरण पाहू.

```
public class WideningTypeCastingExample
{
public static void main(String[] args)
{
int x = 10;
//automatically converts the integer type into long type
long y = x;
//automatically converts the long type into float type
float z = y;
System.out.println("Before conversion, int value "+x);
System.out.println("After conversion, long value "+y);
```

```
System.out.println("After conversion, float value "+z);
} }
```

Output

Before conversion, the value is: 10
After conversion, the long value is: 10
After conversion, the float value is: 10.0

वरील उदाहरणात आपण x हे व्हेरिएबल घेतले आहे आणि त्याचे लाँग प्रकारात रूपांतर केले आहे. त्यानंतर, लाँग प्रकार फ्लोट प्रकारात रूपांतरित केला जातो.

• **न्यारोविंग प्रकार कास्टिंग (Narrowing Type Casting)**

उच्च डेटा प्रकाराला खालच्या डेटामध्ये रूपांतरित करणे याला न्यारोविंग प्रकार कास्टिंग म्हणतात. हे स्पष्ट रूपांतरण explicit conversion किंवा कास्टिंग अप म्हणून देखील ओळखले जाते. हे प्रोग्रामरद्वारे व्यक्तिचलितपणे केले जाते. जर आपण कास्टिंग केले नाही तर कंपाइलर कंपाइल-टाइम एररचा अहवाल देतो.

double -> **float** -> **long** -> **int** -> **char** -> **short** -> **byte**

narrowing type casting चे उदाहरण.

खालील उदाहरणात, आम्ही दोन वेळा narrowing प्रकार कास्टिंग केले आहे. प्रथम, आम्ही डबल प्रकार लाँग डेटा प्रकारात रूपांतरित केला आहे नंतर तो लाँग डेटा प्रकार इंट प्रकारात रूपांतरित केला आहे.

```
public class NarrowingTypeCasting
{
public static void main(String args[])
{
double d = 186.66;
//converting double data type into long data type
long l = (long)d;
//converting long data type into int data type
int i = (int)l;
System.out.println("Before conversion: "+d);
//fractional part lost
System.out.println("After conversion into long type: "+l);
//fractional part lost
System.out.println("After conversion into int type: "+i);
} }
```

Output

Before conversion: 186.66
After conversion into long type: 186
After conversion into int type: 186

1.3.5 Java मधील ऑपरेटर

जावा मधील ऑपरेटर हे एक चिन्ह आहे जे ऑपरेशन्स करण्यासाठी वापरले जाते. उदाहरणार्थ: , -, *, / इ. Java मध्ये अनेक प्रकारचे ऑपरेटर आहेत जे खाली दिले आहेत:

- युनरी ऑपरेटर, Unary Operator
- अंकगणित ऑपरेटर, Arithmetic Operator
- शिफ्ट ऑपरेटर, Shift Operator
- रिलेशनल ऑपरेटर, Relational Operator
- बिटवाइज ऑपरेटर, Bitwise Operator

- लॉजिकल ऑपरेटर, Logical Operator
- टर्नरी ऑपरेटर Ternary Operator
- असाइनमेंट ऑपरेटर. Assignment Operator.

जावा ऑपरेटर प्राधान्य (Java Operator Precedence)

Table 1.5 जावा ऑपरेटर प्राधान्य (Java Operator Precedence)

<table>
<tr><th>Operator Type</th><th>Category</th><th>Precedence</th></tr>
<tr><td rowspan="2">Unary</td><td>Postfix</td><td>expr++ expr--</td></tr>
<tr><td>Prefix</td><td>++expr --expr +expr -expr ~ !</td></tr>
<tr><td rowspan="2">Arithmetic</td><td>Multiplicative</td><td>* / %</td></tr>
<tr><td>Additive</td><td>+ -</td></tr>
<tr><td>Shift</td><td>Shift</td><td><< >> >>></td></tr>
<tr><td rowspan="2">Relational</td><td>Comparison</td><td>< > <= >= instanceof</td></tr>
<tr><td>Equality</td><td>== !=</td></tr>
<tr><td rowspan="3">Bitwise</td><td>bitwise AND</td><td>&</td></tr>
<tr><td>bitwise exclusive OR</td><td>^</td></tr>
<tr><td>bitwise inclusive OR</td><td>|</td></tr>
<tr><td rowspan="2">Logical</td><td>logical AND</td><td>&&</td></tr>
<tr><td>logical OR</td><td>||</td></tr>
<tr><td>Ternary</td><td>Ternary</td><td>? :</td></tr>
<tr><td>Assignment</td><td>Assignment</td><td>= += -= *= /= %= &= ^= |= <<= >>= >>>=</td></tr>
</table>

1.जावा युनरी ऑपरेटर (Java unary operator)

Java unary ऑपरेटरना फक्त एक ऑपरेंड आवश्यक आहे. युनरी ऑपरेटर विविध ऑपरेशन्स करण्यासाठी वापरले जातात उदा:

- एक मूल्य वाढवणे/कमी करणे
- निगे टिंग अन एक्षप्रेशन
- बुलियनचे मूल्य उलटे करणे

जावा युनरी ऑपरेटर उदाहरण: ++ आणि –

```
public class OperatorExample{
public static void main(String args[]){
int x=10;
System.out.println(x++); //10 (11)
System.out.println(++x); //12
System.out.println(x--); //12 (11)
System.out.println(--x); //10
}}
```

Output

10
12
12
10

1. जावा युनरी ऑपरेटर उदाहरण: ~ and !

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a); //-11 (minus of total positive value which starts from 0)
System.out.println(~b); //9 (positive of total minus, positive starts from 0)
System.out.println(!c); //false (opposite of boolean value)
System.out.println(!d); //true
}
}
```

Output:

-11
9
False

2. Java अंकगणित ऑपरेटर (Arithmetic Operator)

जावा अंकगणित ऑपरेटर बेरीज, वजाबाकी, गुणाकार आणि भागाकार करण्यासाठी वापरले जातात. ते मूलभूत गणिती क्रिया म्हणून कार्य करतात.

Java अंकगणित ऑपरेटर उदाहरण :

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
}}
```

Output:

15
5
50
2
0

4. जावा लेफ्ट शिफ्ट ऑपरेटर (Java Left Shift Operator)

जावा लेफ्ट शिफ्ट ऑपरेटर << हे मूल्यातील सर्व बिट्स एका विनिर्दिष्ट संख्येच्या डाव्या बाजूला शिफ्ट करण्यासाठी वापरले जाते.

जावा लेफ्ट शिफ्ट ऑपरेटरचे उदाहरण

```
public class OperatorExample{
public static void main(String args[]){
System.out.println(10<<2);//10*2^2=10*4=40
```

```
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}}
```

Output:

```
40
80
80
240
```

5. Java राईट शिफ्ट ऑपरेटर (Java Right Shift Operator)

Java राईट शिफ्ट ऑपरेटर >> उजव्या ऑपरेंडने निर्दिष्ट केलेल्या बिट्सच्या संख्येनुसार डाव्या ऑपरेंडचे मूल्य उजवीकडे हलविण्यासाठी वापरले जाते.

जावा राईट शिफ्ट ऑपरेटरचे उदाहरण

```
public OperatorExample{
public static void main(String args[]){
System.out.println(10>>2);//10/2^2=10/4=2
System.out.println(20>>2);//20/2^2=20/4=5
System.out.println(20>>3);//20/2^3=20/8=2
}}
```

Output:

```
2
5
2
```

6. Java AND Operator Example: Logical && and Bitwise &

पहिली अट असत्य असल्यास लॉजिकल && ऑपरेटर दुसरी अट तपासत नाही. पहिली अट खरी असेल तरच ती दुसरी अट तपासते.पहिली अट खरी किंवा खोटी असली तरी बिटवाईज & ऑपरेटर नेहमी दोन्ही अटी तपासतो .

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a<b&&a<c);//false && true = false
System.out.println(a<b&a<c);//false & true = false
}}
```

Output:

```
false
false
```

- **Java and ऑपरेटर उदाहरण (लोजीकल ऑपरेटर && बिटवाईज &)**

Java AND Operator Example: Logical && vs Bitwise &

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
```

```
System.out.println(a<b&&a++<c);//false && true = false
System.out.println(a);//10 because second condition is not checked
System.out.println(a<b&a++<c);//false && true = false
System.out.println(a);//11 because second condition is checked
}}
```

Output:

false
10
false
11

7. Java ऑर ऑपरेटर उदाहरण Java OR Operator Example: लोजीकल ऑपरेटर || Logical || and बिटवाईज |Bitwise |

लोजीकल ऑपरेटर || पहिली अट सत्य असल्यास दुसरी अट तपासत नाही. पहिली अट खोटी असेल तरच ती दुसरी अट तपासते.पहिली अट खरी किंवा खोटी असली तरी बिटवाईज | ऑपरेटर नेहमी दोन्ही अटी तपासतो .

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=5;
int c=20;
System.out.println(a>b||a<c);//true || true = true
System.out.println(a>b|a<c);//true | true = true
System.out.println(a>b||a++<c);//true || true = true
System.out.println(a);//10 because second condition is not checked
System.out.println(a>b|a++<c);//true | true = true
System.out.println(a);//11 because second condition is checked
}}
```

Output:

true
true
true
10
true
11

8. Java टरनरी ऑपरेटर (Java Ternary Operator)

Java Ternary ऑपरेटरचा वापर if-then-else स्टेटमेंटसाठी एक ओळ बदली म्हणून केला जातो आणि Java प्रोग्रामिंगमध्ये भरपूर वापरला जातो. हा एकमेव सशर्त ऑपरेटर आहे जो तीन ऑपरेंड घेतो.

Java टरनरी ऑपरेटर उदाहरण Java Ternary Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=2;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
}}
```

Output: 2

9. Java असाइनमेंट ऑपरेटर (Java Assignment Operator)

Java असाइनमेंट ऑपरेटर सर्वात सामान्य ऑपरेटरपैकी एक आहे. हे त्याच्या उजवीकडील मूल्य त्याच्या डावीकडील ऑपरेंडला नियुक्त करण्यासाठी वापरले जाते.

Java असाइनमेंट ऑपरेटरचे उदाहरण Java Assignment Operator Example

```
public class OperatorExample{
public static void main(String args[]){
int a=10;
int b=20;
a+=4;//a=a+4 (a=10+4)
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}}
```

Output:

14
16

10. Java असाइनमेंट ऑपरेटर उदाहरण(Java Assignment Operator Example)

```
public class OperatorExample{
public static void main(String[] args){
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
}}
```

Output:

13
9
18
9

1.3.6 डीसीजन मेकिंग आणि लुपिंग (Decision making and looping Statement)

1.3.6.1 डीसीजन मेकिंग विधान

- **Java If-else विधान (Java If-else Statement)**

स्थिती तपासण्यासाठी Java if स्टेटमेंट वापरले जाते. हे बुलियन स्थिती तपासते: सत्य किंवा खोटे. Java मध्ये if स्टेटमेंटचे विविध प्रकार आहेत.

- if statement
- if-else statement
- if-else-if ladder
- nested if statement

1. Java if विधान (Java if Statement)

Java if विधान कंडिशनची चाचणी घेते. कंडिशन सत्य असल्यास ते if ब्लॉक कार्यान्वित करते.

Syntax:

```
if(condition){
//code to be executed
}
```

उदाहरण:

```
public class IfExample {
public static void main(String[] args) {
   //defining an 'age' variable
   int age=20;
   //checking the age
   if(age>18){
      System.out.print("Age is greater than 18");
   }
}  }
```

Output:

Age is greater than 18

2 . Java if-else विधान Java if-else Statement

Java if-else विधान देखील स्थितीची चाचणी करते. कंडिशन सत्य असल्यास ते if ब्लॉक कार्यान्वित करते अन्यथा एल्स (else)ब्लॉक कार्यान्वित केला जातो.

Syntax:

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

उदाहरण 1

```
/*A Java Program to demonstrate the use of if
else statement.  It is a program of odd and even number.  */
public class IfElseExample {
public static void main(String[] args) {
   //defining a variable
   int number=13;
   //Check if the number is divisible by 2 or not
   if(number%2==0){
      System.out.println("even number");
   }else{
      System.out.println("odd number");
   }
}
}
```

Output: odd number

उदाहरण 2:

```
public class LeapYearExample {
public static void main(String[] args) {
   int year=2020;
   if(((year % 4 ==0) && (year % 100 !=0)) || (year % 400==0)){
      System.out.println("LEAP YEAR");
   }
   else{
      System.out.println("COMMON YEAR");
   }
}
}
```

Output: LEAP YEAR

3. टर्नरी ऑपरेटर (Ternary Operator)

if...else स्टेटमेंटचे कार्य करण्यासाठी आपण ternary operator (? :) देखील वापरू शकतो. स्थिती तपासण्याचा हा एक लघुलेखन मार्ग आहे. जर अट खरी असेल तर त्याचा परिणाम ? परत केले जाते. परंतु, जर अट चुकीची असेल, तर : चा परिणाम परत केला जातो.

Syntax **:**

Variable=condition**?**statement to be executed if condition is true**:**
statement to be executed if condition is false

उदाहरण

```
public class IfElseTernaryExample {
public static void main(String[] args) {
   int number=13;
   //Using ternary operator
   String output=(number%2==0)?"even number":"odd number";
   System.out.println(output);
}
}
```

Output:

odd number

4.Java if-else-if ल्याडर विधान (Java if-else-if ladder Statement)

if-else-if ल्याडर विधान एकाधिक विधानांमधून एक अट कार्यान्वित करते.

Syntax:

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
```

//code to be executed if all the conditions are false }

उदाहरण:

```
/*Java Program to demonstrate the use of If else-
if ladder.  It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+.  */
public class IfElseIfExample {
public static void main(String[] args) {
   int marks=65;
   if(marks<50){
      System.out.println("fail");
   }
   else if(marks>=50 && marks<60){
      System.out.println("D grade");
   }
   else if(marks>=60 && marks<70){
      System.out.println("C grade");
   }
   else if(marks>=70 && marks<80){
      System.out.println("B grade");
   }
   else if(marks>=80 && marks<90){
      System.out.println("A grade");
   }else if(marks>=90 && marks<100){
      System.out.println("A+ grade");
   }else{
      System.out.println("Invalid!");
   }  }
}
```

Output: C grade

Program to check POSITIVE, NEGATIVE or ZERO:

```
public class PositiveNegativeExample {
public static void main(String[] args) {
   int number=-13;
   if(number>0){
   System.out.println("POSITIVE");
   }else if(number<0){
   System.out.println("NEGATIVE");
   }else{
   System.out.println("ZERO");
  }
}
}
```

Output:

NEGATIVE

5. नेस्टेड इफ स्टेटमेंट (Java Nested if statement)

नेस्टेड इफ स्टेटमेंट दुसऱ्या इफ ब्लॉकमधील if ब्लॉक दर्शवते. येथे, जर ब्लॉक कंडिशन सत्य असेल तरच बाह्य इफ ब्लॉक कंडिशन कार्यान्वित होते

Syntax:

```
if(condition){
    //code to be executed
        if(condition){
          //code to be executed
    }
}
```

उदाहरण:

```
//Java Program to demonstrate the use of Nested If Statement.
public class JavaNestedIfExample {
public static void main(String[] args) {
   //Creating two variables for age and weight
   int age=20;
   int weight=80;
   //applying condition on age and weight
   if(age>=18){
      if(weight>50){
         System.out.println("You are eligible to donate blood");
      }
   }
}}
```

Output:

You are eligible to donate blood

6 .जावा स्विच स्टेटमेंट (Java Switch Statement)

जावा स्विच स्टेटमेंट एकाधिक परिस्थितींमधून (Multiple cases)एक विधान कार्यान्वित करते. हे if-else-if ल्याडर विधानासारखे आहे. स्विच स्टेटमेंट बाइट, शॉर्ट, इंट, लाँग, एनम प्रकार, स्ट्रिंग आणि बाइट, शॉर्ट, इंट आणि लाँग सारख्या काही रॅपर प्रकारांसह कार्य करते. Java 7 पासून, तुम्ही स्विच स्टेटमेंटमध्ये स्ट्रिंग वापरू शकता.

दुसऱ्या शब्दांत, स्विच स्टेटमेंट एकाधिक मूल्यांच्या विरूद्ध व्हेरिएबलच्या समानतेची चाचणी करते.

- स्विच अभिव्यक्तीसाठी केस मूल्यांची एक किंवा N संख्या असू शकते.
- केस(case) व्हॅल्यू फक्त स्विच एक्सप्रेशन प्रकारातील असणे आवश्यक आहे. केस(case) मूल्य शाब्दिक किंवा स्थिर असणे आवश्यक आहे. हे व्हेरिएबल्सना परवानगी देत नाही.
- केस (case) मूल्ये अद्वितीय असणे आवश्यक आहे. डुप्लिकेट मूल्याच्या बाबतीत, ते कंपाइल-टाइम त्रुटी प्रस्तुत करते.
- Java स्विच एक्सप्रेशन बाइट, शॉर्ट, इंट, लाँग ,एनम् आणि स्ट्रिंगचे असावे.
- प्रत्येक केस स्टेटमेंटमध्ये ब्रेक स्टेटमेंट असू शकते जे ऐच्छिक आहे. जेव्हा नियंत्रण ब्रेक स्टेटमेंटपर्यंत पोहोचते, तेव्हा ते स्विच अभिव्यक्तीनंतर नियंत्रण उडी मारते. जर ब्रेक स्टेटमेंट आढळले नाही, तर ते पुढील केस चालवते.
- केस व्हॅल्यूमध्ये डीफॉल्ट लेबल असू शकते जे ऐच्छिक आहे.

Syntax:

```
switch(expression){
case value1:
```

```
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......
   default:
  code to be executed if all cases are not matched;
}
```

उदाहरण

Program to check Vowel or Consonant:

If the character is A, E, I, O, or U, it is vowel otherwise consonant. It is not case-sensitive.

SwitchVowelExample.java

```
public class SwitchVowelExample {
public static void main(String[] args) {
   char ch='O';
   switch(ch)
   {
      case 'a':
         System.out.println("Vowel");
         break;
      case 'e':
         System.out.println("Vowel");
         break;
      case 'i':
         System.out.println("Vowel");
         break;
      case 'o':
         System.out.println("Vowel");
         break;
      case 'u':
         System.out.println("Vowel");
         break;
      case 'A':
         System.out.println("Vowel");
         break;
      case 'E':
         System.out.println("Vowel");
         break;
      case 'I':
         System.out.println("Vowel");
         break;
      case 'O':
         System.out.println("Vowel");
```

```
        break;
      case 'U':
        System.out.println("Vowel");
        break;
      default:
        System.out.println("Consonant");
    }
}
}
```

Output:

Vowel

1.3.6.2 जावा मध्ये लूप (Loops in Java)

जावा फॉर लूप प्रोग्रामचा एक भाग अनेक वेळा पुनरावृत्ती करण्यासाठी वापरला जातो. पुनरावृत्तीची संख्या निश्चित असल्यास, लूपसाठी वापरण्याची शिफारस केली जाते.

Java मध्ये तीन प्रकारचे for loops आहेत.

- सिम्पल फॉर लूप Simple for Loop
- फॉर –इच किंवा एन्हानस फॉर लूप For-each or Enhanced for Loop
- लेबल फॉर लूप Labeled for Loop

1.सिम्पल फॉर लूप(Simple for Loop)

आपण व्हेरिएबल इनिशियल करू शकतो, कंडिशन आणि इन्क्रिमेंट/डिक्रिमेंट व्हॅल्यू तपासू शकतो. यात चार भाग असतात:

1. इनिशियलायझेशन: ही प्रारंभिक स्थिती आहे जी एकदा लूप सुरू झाल्यावर अंमलात आणली जाते. येथे, आपण व्हेरिएबल इनिशियल करू शकतो किंवा आधीच इनिशियलाइज केलेले व्हेरिएबल वापरू शकतो. ती एक ऐच्छिक अट आहे.
2. अट: ही दुसरी अट आहे जी प्रत्येक वेळी लूपची स्थिती तपासण्यासाठी अंमलात आणली जाते. अट खोटी होईपर्यंत त्याची अंमलबजावणी सुरू राहते. ते खरे किंवा खोटे एकतर बुलियन मूल्य परत करणे आवश्यक आहे. ती एक ऐच्छिक अट आहे.
3. वाढ/घट: हे व्हेरिएबल मूल्य वाढवते किंवा कमी करते. ती एक ऐच्छिक अट आहे.
4. विधान: दुसरी अट असत्य होईपर्यंत प्रत्येक वेळी लूपचे विधान कार्यान्वित केले जाते

```
for(initialization; condition; increment/decrement){
//statement or code to be executed   }
```

उदाहरण:

ForExample.java

```
//Java Program to demonstrate the example of for loop  which prints table of 1
public class ForExample {
public static void main(String[] args) {
   //Code of Java for loop
   for(int i=1;i<=10;i++){
      System.out.println(i);
   }
} }
```

2. जावा नेस्टेड फॉर लूप Java Nested for Loop

जर दुसऱ्या लूपमध्ये फॉर लूप असेल तर त्याला नेस्टेड फॉर लूप असे म्हणतात. जेव्हा बाह्य लूप कार्यान्वित होतो तेव्हा आतील लूप पूर्णपणे कार्यान्वित होते

उदाहरण: Example

NestedForExample.java

```
public class NestedForExample {
public static void main(String[] args) {
//loop of i
for(int i=1;i<=3;i++){
//loop of j
for(int j=1;j<=3;j++){
        System.out.println(i+" "+j);
}//end of i
}//end of j
}
}
```

Output:

1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3

3. फॉर –इच किंवा एन्हानस फॉर लूप (For-each or Enhanced for Loop)

लूप जावामधील ॲरे किंवा संग्रह ट्रॅव्हर्स करण्यासाठी वापरला जातो. साध्या लूपपेक्षा वापरणे सोपे आहे कारण आम्हाला मूल्य वाढवण्याची आणि सबस्क्रिप्ट नोटेशन वापरण्याची आवश्यकता नाही.

हे घटकांच्या आधारावर कार्य करते आणि निर्देशांकावर नाही. हे परिभाषित व्हेरिएबलमध्ये एक-एक करून घटक परत करते.

Syntax:

```
for(data_type variable : array_name){
}
```

उदाहरण:

ForEachExample.java

```
//Java For-each loop example which prints the  elements of the array
public class ForEachExample {
public static void main(String[] args) {
   //Declaring an array
   int arr[]={12,23,44,56,78};
   //Printing array using for-each loop
   for(int i:arr){
      System.out.println(i);
   }
}
}
```

Output:

12

23
44
56
78

4 लेबल फॉर लूप(Labeled for Loop)

प्रत्येक लूपसाठी नाव असू शकते. असे करण्यासाठी, आम्ही for loop च्या आधी लेबल वापरतो. नेस्टेड फॉर लूप वापरताना हे उपयुक्त आहे कारण आपण लूपसाठी विशिष्ट खंडित/सुरू ठेवू शकतो.

Syntax:

```
labelname:
for(initialization; condition; increment/decrement){
//code to be executed
}
```

उदाहरण:

Example:LabeledForExample.java

```
//A Java program to demonstrate the use of labeled for loop
public class LabeledForExample {
public static void main(String[] args) {
   //Using Label for outer and for loop
   aa:
      for(int i=1;i<=3;i++){
         bb:
            for(int j=1;j<=3;j++){
               if(i==2&&j==2){
                  break aa;
               }
               System.out.println(i+" "+j);
            }
      }
}   }
```

Output:

```
1 1
1 2
1 3
2 1
```

5. Java while लूप (Java While Loop)

Java while लूपचा वापर प्रोग्रामचा एक भाग वारंवार पुनरावृत्ती करण्यासाठी निर्दिष्ट बुलियन कंडिशन सत्य होईपर्यंत वापरला जातो. बुलियन कंडिशन खोटी होताच, लूप आपोआप थांबतो.

while लूप हे रिपीट इफ स्टेटमेंट म्हणून मानले जाते. पुनरावृत्तीची संख्या निश्चित नसल्यास, वळण लूप वापरण्याची शिफारस केली जाते.

Syntax:

```
while (condition){
//code to be executed
 increment / decrement statement
}
```

उदाहरण:

WhileExample.java

```
public class WhileExample {
public static void main(String[] args) {
   int i=1;
   while(i<=4){
      System.out.println(i);
   i++;
   }
}
}
```

Output:

1
2
3
4

6. डू व्हाईल लूप (Java do-while Loop)

डू व्हाईल हे इटरेटिव्ह स्टेटमेंट आहे, जो ठराविक अटी पूर्ण होईपर्यंत कोड ब्लॉकला पुन्हा पुन्हा चालवतो. पुनरावृत्तीची संख्या निश्चित नसल्यास आणि तुम्हाला किमान एकदा लूप कार्यान्वित करणे आवश्यक असल्यास, डू-व्हाइल लूप वापरण्याची शिफारस केली जाते. Java do-while लूपला एक्झिट कंट्रोल लूप म्हणतात. म्हणून, while loop आणि for loop च्या विपरीत, do-while लूप बॉडीच्या शेवटी स्थिती तपासा. Java do-while लूप किमान एकदा कार्यान्वित केला जातो कारण लूप बॉडीनंतर स्थिती तपासली जाते.

Syntax:

```
do{
//code to be executed / loop body
 //update statement
}while (condition);
```

उदाहरण:

DoWhileExample.java

```
public class DoWhileExample {
public static void main(String[] args) {
   int i=1;
   do{
      System.out.println(i);
   i++;
   }while(i<=4);
}
}
```

Output:

1
2
3
4

7. जावा ब्रेक स्टेटमेंट java break statement

- जेव्हा लूपमध्ये ब्रेक स्टेटमेंट आढळते, तेव्हा लूप ताबडतोब बंद केला जातो आणि लूपनंतर पुढील स्टेटमेंटवर प्रोग्राम कंट्रोल पुन्हा सुरू होतो.
- Java ब्रेक स्टेटमेंटचा वापर लूप किंवा स्विच स्टेटमेंट ब्रेक करण्यासाठी केला जातो. हे निर्दिष्ट स्थितीत प्रोग्रामचा वर्तमान प्रवाह खंडित करते. आतील लूपच्या बाबतीत, ते फक्त अंतर्गत लूप तोडते.
- आपण जावा ब्रेक स्टेटमेंट सर्व प्रकारच्या लूपमध्ये वापरू शकतो जसे की for loop, while loop आणि do-while loop.

Syntax:

```
jump-statement;
break;
```

उदाहरण:

BreakExample.java

```
//Program to demonstrate the use of break statement inside the for loop.
public class BreakExample {
public static void main(String[] args) {
   //using for loop
   for(int i=1;i<=10;i++){
      if(i==5){
         //breaking the loop
         break;
      }
      System.out.println(i);
   }
}  }
```

Output:

```
1
2
3
4
```

8. कंटिन्यू स्टेटमेंट (Java Continue Statement)

- कंटिन्यू स्टेटमेंट लूप कंट्रोल स्ट्रक्चरमध्ये वापरले जाते जेव्हा तुम्हाला लूपच्या पुढील पुनरावृत्तीवर लगेच जावे लागते. हे for loop किंवा while loop सह वापरले जाऊ शकते.
- लूप चालू ठेवण्यासाठी Java continue स्टेटमेंट वापरले जाते. तो प्रोग्रामचा वर्तमान प्रवाह चालू ठेवतो आणि निर्दिष्ट स्थितीत उर्वरित कोड वगळतो. आतील लूपच्या बाबतीत, ते फक्त आतील लूप चालू ठेवते.
- आपण जावा कंटिन्यू स्टेटमेंट सर्व प्रकारच्या लूपमध्ये वापरू शकतो जसे की for loop, while loop

Syntax:

```
jump-statement;
continue;
```

उदाहरण:

```
//Program to demonstrate the use of continue statement  inside the for loop.
public class ContinueExample {
public static void main(String[] args) {
   //for loop
```

```
   for(int i=1;i<=6;i++){
      if(i==5){
         //using continue statement
         continue;//it will skip the rest statement
      }
      System.out.println(i);
   }
}
}
```

Output:

1
2
3
4
6

1.4.1 ॲरे (Java Arrays)

ॲरे हा सिमिलर प्रकारच्या घटकांचा संग्रह आहे ज्यात मेमरी स्थान सलग असते. आम्ही जावा ॲरेमध्ये घटकांचा(elements) फक्त निश्चित संच(fixed set) ठेवू शकतो. Java मधील ॲरे अनुक्रमणिका-आधारित आहे, ॲरेचा पहिला घटक 0व्या अनुक्रमणिकेवर संग्रहित केला जातो, दुसरा घटक 1ल्या निर्देशांकावर संग्रहित केला जातो आणि असेच.

ॲरेचे दोन प्रकार आहेत.

1. सिंगल डायमेंशनल ॲरे Single Dimensional Array
2. बहुआयामी ॲरे Multidimensional Array

सिंगल डायमेंशनल ॲरे Single Dimensional Array in Java

Java मध्ये ॲरे घोषित करण्यासाठी सिंटॅक्स Syntax to Declare an Array in Java

```
dataType[] arr;
 (or)
dataType []arr;
(or)
dataType arr[];
```

Java मध्ये ॲरेची स्थापना Instantiation of an Array in Java

```
arrayRefVar=new datatype[size];
```

Java array चे साधे उदाहरण, जिथे आपण ॲरे घोषित(declare), इन्स्टंट(instantiate,), इनिशिएट(, initialize) आणि ट्रॅव्हर्स (traverse)करणार आहोत.

```
class Testarray{
public static void main(String args[]){
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=30;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
```

}}

Output

10

20

30

40

50

```
class Testarray1{
public static void main(String args[]){
int a[]={10,20,30,40,50};//declaration, instantiation and initialization  //printing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}}
```

Output

10

20

30

40

50

- **बहुआयामी ॲरे Multidimensional Array in Java**

डेटा पंक्ती(row) आणि स्तंभ(column) आधारित अनुक्रमणिकेमध्ये संग्रहित केला जातो (ज्याला मॅट्रिक्स फॉर्म देखील म्हणतात).

Java मध्ये ॲरे घोषित करण्यासाठी सिंटॅक्स(Syntax to Declare Multidimensional Array in Java)

```
dataType[][] arrayRefVar;
 (or)
dataType [][]arrayRefVar;
(or)
dataType arrayRefVar[][];
 (or)
dataType []arrayRefVar[];
```

Java मध्ये ॲरेची स्थापना (**Example to instantiate Multidimensional Array in Java)**

int[][] arr=**new int**[3][3];//3 row and 3 column

Java मध्ये बहुआयामी ॲरे इनिशिएट करण्याचे उदाहरण

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

बहुआयामी ॲरेचे उदाहरण (Example of Multidimensional Java Array)

```
class Testarray3{
public static void main(String args[]){
//declaring and initializing 2D array
int arr[][]={
{1,2,3},
{2,4,5},
{4,4,5}
};
//printing 2D array
for(int i=0;i<3;i++){
 for(int j=0;j<3;j++){
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}  }}
```

Output

```
1 2 3
2 4 5
4 4 5
```

1.4.2 Java स्ट्रींग (Java String)

Java मध्ये, स्ट्रिंग ही मूलत: एक ऑब्जेक्ट आहे जी क्यार (char) मूल्यांचा क्रम दर्शवते. वर्णांची ॲरे (character array). Java स्ट्रिंग प्रमाणेच कार्य करते.

Example

```
char[] ch={'A','m','r','u','t','v','a','h','i','n','i'}
String s=new String(ch);
or
String s="Amrutvahini";
```

जावा स्ट्रिंग क्लास compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() यांसारख्या स्ट्रिंगवर ऑपरेशन्स करण्यासाठी अनेक पद्धती पुरवतो. इ.

जावा स्ट्रिंग अपरिवर्तनीय(immutable) आहे म्हणजे ती बदलली जाऊ शकत नाही. जेव्हा आपण कोणतीही स्ट्रिंग बदलतो तेव्हा एक नवीन उदाहरण (instance)तयार केले जाते.

साधारणपणे, स्ट्रिंग हा वर्णांचा क्रम(Character sequence) असतो. परंतु Java मध्ये, स्ट्रिंग ही एक ओब्जेक्ट object आहे जी वर्णांच्या क्रमाचे प्रतिनिधित्व करते. java.lang.String क्लास स्ट्रिंग ऑब्जेक्ट तयार करण्यासाठी वापरला जातो.

स्ट्रिंग ऑब्जेक्ट तयार करण्याचे दोन मार्ग आहेत:

1. स्ट्रिंग शाब्दिक द्वारे(String literal)
2. न्यू कीवर्डद्वारे(new keyword)

जावा स्ट्रिंग शाब्दिक(String literal) दुहेरी अवतरण(double quotes) वापरून तयार केले आहे. **उदाहरणार्थ:**

```
 String s="Welcome";
```

न्यू कीवर्डद्वारे(new keyword) By new keyword

String s=**new** String("Welcome");

उदाहरण : StringExample.java

public class StringExample{

```
public static void main(String args[]){
String s1="Amrutvahini";//creating string by Java string literal
char ch[]={'P','o','l','y','t','e','c','h','n','i','c'};
String s2=new String(ch);//converting char array to string
String s3=new String("Sangamner");//creating Java string by new keyword
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```

Output:

Amrutvahini
Polytechnic
Sangamner

स्ट्रिंग क्लास String class

स्ट्रिंग क्लास अनेक पद्धती परिभाषित करतो ज्यामुळे आम्हाला स्ट्रिंग मॅनिपुलेशनची विविध कामे पूर्ण करता येतात. Table 1.6 मध्ये काही सामान्यतः वापरल्या जाणाऱ्या स्ट्रिंग पद्धती आणि त्यांची कार्ये सूचीबद्ध आहेत.

Table 1.6 जावा स्ट्रिंग क्लास पद्धती (Java String class methods)

Sr.No.	Method	Description
1.	char charAt(int index)	It returns char value for the particular index हे विशिष्ट निर्देशांकासाठी वर्ण मूल्य परत करते
2.	int length()	It returns string length हे स्ट्रिंगची लांबी मिळवते
3.	String substring(int beginIndex)	It returns substring for given begin index. हे दिलेल्या आरंभ निर्देशांकासाठी सबस्ट्रिंग परत करते.
4.	String substring(int beginIndex, int endIndex)	It returns substring for given begin index and end index. हे दिलेल्या आरंभ इंडेक्स आणि एंड इंडेक्ससाठी सबस्ट्रिंग परत करते.
5.	boolean contains(CharSequence s)	It returns true or false after matching the sequence of char value. वर्ण मूल्याच्या क्रमाशी जुळल्यानंतर ते खरे True किंवा असत्य false परत करते.
6.	static String join(CharSequence delimiter, CharSequence... elements)	It returns a joined string. ते जोडलेली स्ट्रिंग मिळवते.
7.	static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)	It returns a joined string. ते जोडलेली स्ट्रिंग मिळवते.
8.	boolean equals(Object another)	It checks the equality of string with the given object. हे दिलेल्या ऑब्जेक्टसह स्ट्रिंगची समानता तपासते.

9.	boolean isEmpty()	It checks if string is empty. स्ट्रिंग रिक्त असेल तर true मूल्य होते
10.	String concat(String str)	It concatenates the specified string. हे निर्दिष्ट स्ट्रिंगला जोडते.
11.	String replace(char old, char new)	It replaces all occurrences of the specified char value. हे निर्दिष्ट वर्ण मूल्याच्या सर्व घटनांना पुनर्स्थित करते.
12.	String replace(CharSequence old, CharSequence new)	It replaces all occurrences of the specified CharSequence. हे निर्दिष्ट वर्ण क्रमाच्या सर्व घटनांना पुनर्स्थित करते.
13.	static String equalsIgnoreCase(String another)	It compares another string. It doesn't check case. ते दुसऱ्या स्ट्रिंगची तुलना करते. हे case तपासत नाही.
14.	String[] split(String regex)	It returns a split string matching regex. हे स्प्लिट स्ट्रिंग जुळणारे regex परत करते.
15.	String[] split(String regex, int limit)	It returns a split string matching regex and limit. हे regex आणि मर्यादा जुळणारी स्प्लिट स्ट्रिंग मिळवते.
16.	int indexOf(int ch)	It returns the specified char value index. हे निर्दिष्ट वर्ण मूल्य निर्देशांक परत करते.
17.	int indexOf(int ch, int fromIndex)	It returns the specified char value index starting with given index. हे दिलेल्या निर्देशांकापासून सुरू होणारी निर्दिष्ट वर्ण मूल्य निर्देशांक परत करते.
18.	int indexOf(String substring)	It returns the specified substring index. हे निर्दिष्ट सबस्ट्रिंग निर्देशांक परत करते.
19.	int indexOf(String substring, int fromIndex)	It returns the specified substring index starting with given index. हे दिलेल्या निर्देशांकापासून सुरू होणारी निर्दिष्ट सबस्ट्रिंग अनुक्रमणिका परत करते.
20.	String toLowerCase()	It returns a string in lowercase. ते लोअरकेसमध्ये स्ट्रिंग परत करते.
21.	String toLowerCase(Locale l)	It returns a string in lowercase using specified locale. हे निर्दिष्ट लोकेल वापरून लोअरकेसमध्ये एक स्ट्रिंग परत करते.
22.	String toUpperCase()	It returns a string in uppercase. ते अपरकेसमध्ये एक स्ट्रिंग परत करते.
23.	String toUpperCase(Locale l)	It returns a string in uppercase using specified locale. हे निर्दिष्ट लोकेल वापरून अपरकेसमध्ये एक स्ट्रिंग परत करते.

24.	String trim()	It removes beginning and ending spaces of this string. ते या स्ट्रिंगची सुरुवात आणि शेवटची जागा काढून टाकते.
25.	static String valueOf(int value)	It converts given type into string. It is an overloaded method.हे दिलेल्या प्रकाराला स्ट्रिंगमध्ये रूपांतरित करते. ही एक ओव्हरलोड पद्धत आहे.

उदाहरण :

```
// Alphabetical ordering of strings
class StringOrdering
{
  static String name[] = ("Madras", "Delhi", "Ahmedabad", "Calcutta", "Bombay");
  public static void main (String args[ ])
  {
        int size = name.length;
        String temp = null;
        for (int i = 0; i < size; i++)
        {
          for (jot j = i+l; j < size; j++)
          {
            if (name [j .compareTo (name [l] ) < 0)
            {
                // swap the strings
                temp= name[i];
                name[i] = name[j];
                name[j] = temp;
          }
        for (int i = 0; i < size; i++)
        {
          System.out.println(name[il);
      }
  } }
```

(Program produces the following sorted list)Output:

Ahmedabad
Bombay
Calcutta
Delhi
Madras

1.4.3 स्ट्रिंग बफर क्लास (String Buffer class):

स्ट्रिंगबफर हा स्ट्रिंगचा समवयस्क वर्ग आहे. स्ट्रिंग निश्चित लांबीचे स्ट्रिंग तयार करते, तर स्ट्रिंगबफर लवचिक लांबीचे स्ट्रिंग जे लांबी आणि सामग्री दोन्हीच्या दृष्टीने सुधारित केले जाऊ शकतात. आपण स्ट्रिंगच्या मध्यभागी सबस्ट्रिंग्स घालू शकतो किंवा शेवटी दुसरी स्ट्रिंग जोडू शकतो.

स्ट्रिंगबफर पद्धती:

Table 1.7 जावा स्ट्रिंगबफर वर्ग पद्धती

Method	Task
sl.setChartAt(n, 'x')	Modifies the nth character to x n व्या वर्णाला x मध्ये बदलते
sl.append(s2)	Appends the string s2 to s1 at the end शेवटी s1 ला स्ट्रिंग s2 जोडते
sl.insert(n, s2)	Inserts the string s2 at the position n of the string s1 स्ट्रिंग s1 च्या n स्थानावर स्ट्रिंग s2 घालते
sl.setLength(n)	Sets the length of the string s1 to n. If n < s1.length ( ) s1 is truncated, If n>s1.length() zeros are added to s1 स्ट्रिंग s1 ची लांबी n वर सेट करते. जर n < s1.length ( ) s1 कापला जातो, जर n>s1.length() असेल तर s1 मध्ये शून्य जोडला जातो

स्ट्रिंग हाताळण्यासाठी चे उदाहरण :

```
class StringManipulation
{
   public static void main(String args[ ])
    {
        StringBuffer str = new StringBuffer("Object language");
       System.out.println("Original String :" + str);
       // obtaining string length
      System.out.println("Length of string :" + str.length());
     // Accessing characters in a string
     for (int i = 0; i < str.length(); i++)
     {
        int p = i +1;
       System.out.println("Character at position "+ p + "is" +
       str.charAt(i));
   }
         // Inserting a string in the middle
         String aString = new String(str.toString());
         int pos = aString index0f(" language"); str.insert(process," Oriented ");
         System. out. println("Modified string " + str);
         / / Modifying characters
         str.setCharAt(6, '-');
         System. out. println("String now " + str);
         / / Appending a string at the end
         str.append(" improves security.");
         System.out.println ("Appended string " + str);
   }
}
```

Output of Program would be:

Original String : Object

Length of string : 6

Character at position : 1 is 0
Character at position : 2 is b
Character at position : 3 is j
Character at position : 4 is e
Character at position : 5 is c
Character at position : 6 is t
Modified string : Object Oriented language
String now : Object Oriented language
Appended string : Object Oriented language improves security.

1.4.4 व्हेक्टर (VECTORS):

java.util पॅकेजमध्ये व्हेक्टर क्लास समाविष्ट आहे. या वर्गाचा वापर व्हेक्टर म्हणून ओळखला जाणारा सामान्य डायनॅमिक ॲरे तयार करण्यासाठी केला जाऊ शकतो जो कोणत्याही प्रकारच्या आणि कोणत्याही संख्येच्या वस्तू ठेवू शकतो. वस्तू एकसंध असणे आवश्यक नाही. ॲरे सहजपणे व्हेक्टर म्हणून लागू केले जाऊ शकतात. व्हेक्टर खालीलप्रमाणे ॲरेप्रमाणे तयार केले जातात.

Vector intVect = new Vector( ); // आकाराशिवाय घोषित करणे

Vector list = new Vector (3); // आकारासह घोषित करणे

Vector list = new Vector (5,2); // आकार आणि वाढीसह घोषित करणे

व्हेक्टर कन्स्ट्रक्टर:

1.Vector (): प्रारंभिक क्षमता 10 चे डीफॉल्ट व्हेक्टर तयार करते.

उदाहरण

Vector intVect = new Vector( );

2. Vector(int size):

एक व्हेक्टर तयार करतो ज्याची प्रारंभिक क्षमता आकारानुसार निर्दिष्ट केली जाते.

उदाहरण Vector list = new Vector (3);

3. Vector(int size, int incr):

एक व्हेक्टर तयार करतो ज्याची प्रारंभिक क्षमता आकाराद्वारे निर्दिष्ट केली जाते आणि वाढ incr द्वारे निर्दिष्ट केली जाते. प्रत्येक वेळी व्हेक्टरचा आकार बदलून वरच्या दिशेने वाटप करण्यासाठी घटकांची संख्या निर्दिष्ट करते.

उदाहरण : Vector list = new Vector (5,2);

4. Vector(Collection c):

एक व्हेक्टर तयार करतो ज्यामध्ये संकलन c चे घटक असतात.

लक्षात घ्या की कोणताही आकार स्पष्टपणे न सांगता व्हेक्टर घोषित केला जाऊ शकतो. आकार नसलेला सदिश अज्ञात वस्तूंची संख्या सामावून घेऊ शकतो. अगदी, जेव्हा आकार निर्दिष्ट केला जातो, तेव्हा याकडे दुर्लक्ष केले जाऊ शकते आणि व्हेक्टरमध्ये भिन्न संख्या ठेवली जाऊ शकते. लक्षात ठेवा, याउलट, ॲरेचा आकार नेहमी निर्दिष्ट केलेला असणे आवश्यक आहे.

व्हेक्टरचे ॲरेवर अनेक फायदे आहेत.

1. वस्तू साठवण्यासाठी व्हेक्टर वापरणे सोयीचे आहे.
2. आकारात भिन्न असू शकतील अशा वस्तूंची सूची संग्रहित करण्यासाठी व्हेक्टरचा वापर केला जाऊ शकतो.
3. आवश्यकतेनुसार आम्ही सूचीमधून वस्तू जोडू आणि हटवू शकतो.

Table 1.8 व्हेक्टर पद्धती (Vector Methods):

Method	Task
v.addElement(item)	सूचीमध्ये शेवटी नमूद केलेला आयटम जोडतो

	Adds the item specified to the list at the end
v.elementAt(10)	10व्या वस्तूचे नाव देतो Gives the name of the 10th object
v.size( )	उपस्थित वस्तूंची संख्या देते Gives the number of objects present
v.removeElement(item)	सूचीमधून निर्दिष्ट आयटम काढून टाकते Removes the specified item from the list
v.removeElementAt(n)	सूचीच्या nव्या स्थानावर संग्रहित आयटम काढून टाकते Removes the item stored in the nth position of the list
v.removeAllElements( )	सूचीतील सर्व घटक काढून टाकते Removes all the elements in the list
v.copyInto(array)	सूचीमधून ॲरेमध्ये सर्व आयटम कॉपी करते Copies all items from list to array
v.insertElementAt(item, n)	n व्या स्थानावर आयटम घालतो Inserts the item at nth position
v.capacity()	या व्हेक्टरची वर्तमान क्षमता मिळविण्यासाठी याचा वापर केला जातो. It is used to get the current capacity of this vector.
v.contains(item)	व्हेक्टरमध्ये निर्दिष्ट घटक असल्यास ते सत्य मिळवते. It returns true if the vector contains the specified element.
v.equals(item)	हे समानतेसाठी निर्दिष्ट ऑब्जेक्टची व्हेक्टरसह तुलना करण्यासाठी वापरले जाते. It is used to compare the specified object with the vector for equality.
v.containsAll()	जर व्हेक्टरमध्ये निर्दिष्ट संग्रहातील सर्व घटक असतील तर ते सत्य परत येईल. It returns true if the vector contains all of the elements in the specified collection.
firstElement()	हे व्हेक्टरचा पहिला घटक मिळविण्यासाठी वापरला जातो. It is used to get the first component of the vector.
v.lastElement()	हे व्हेक्टरचा शेवटचा घटक मिळविण्यासाठी वापरला जातो. It is used to get the last component of the vector.

व्हेक्टर उदाहरण :

```
import java.util.*;
public class
{
    public static void main(String args[])
     {
       //Create an empty vector with initial capacity 4
       Vector<String> vec = new Vector<String>(4);
       //Adding elements to a vector
       vec.add("Tiger");
       vec.add("Lion");
       vec.add("Dog");
```

```
        vec.add("Elephant");
        //Check size and capacity
        System.out.println("Size is: "+vec.size());
        System.out.println("Default capacity is: "+vec.capacity());
        //Display Vector elements
        System.out.println("Vector element is: "+vec);
        vec.addElement("Rat");
        vec.addElement("Cat");
        vec.addElement("Deer");
        //Again check size and capacity after two insertions
        System.out.println("Size after addition: "+vec.size());
        System.out.println("Capacity after addition is: "+vec.capacity());
        //Display Vector elements again
        System.out.println("Elements are: "+vec);
        //Checking if Tiger is present or not in this vector
         if(vec.contains("Tiger"))
         {
           System.out.println("Tiger is present at the index " +vec.indexOf("Tiger"));
         }
         else
         {
           System.out.println("Tiger is not present in the list.");
         }
         //Get the first element
        System.out.println("The first animal of the vector is = "+vec.firstElement());
        //Get the last element
        System.out.println ("The last animal of the vector is = "+vec.lastElement());
    }
}
```

Output:

Size is: 4
Default capacity is: 4
Vector element is: [Tiger, Lion, Dog, Elephant]
Size after addition: 7
Capacity after addition is: 8
Elements are: [Tiger, Lion, Dog, Elephant, Rat, Cat, Deer]
Tiger is present at the index 0
The first animal of the vector is = Tiger
The last animal of the vector is = Deer

Table 1.9 ॲरे आणि व्हेक्टर मधील फरक (Difference between Array and Vector)

ॲरे (Array)	व्हेक्टर (Vector)
ॲरे हा वर्ग नाही. Array is not a class.	व्हेक्टर हा एक वर्ग आहे. Vector is a class.

ॲरे म्हणजे स्टॅटिक मेमरी ॲलोकेशन. An array is the static memory allocation.	व्हेक्टर म्हणजे डायनॅमिक मेमरी ॲलोकेशन. Vector is the dynamic memory allocation.
ॲरेमध्ये रॅपर वर्ग वापरले जात नाहीत Wrapper classes are not use in array.	व्हेक्टरमध्ये रॅपर क्लासेसचा वापर केला जातो. Wrapper classes are use in vector.
Syntax :int arr [ ] = new int [4];	**Syntax :**: Vector list = new Vector (8)
ॲरेद्वारे संचयित केलेला प्रिमिटिव प्रकार डेटा घटक. Primitive type data element stored by array.	व्हेक्टरद्वारे संग्रहित नॉन-प्रिमिटिव प्रकार डेटा घटक. Non-primitive type data elements stored by vector.
ॲरेमध्ये, घटक जोडण्यासाठी आणि काढण्यासाठी कोणत्याही पद्धती प्रदान केल्या जात नाहीत In array, no methods are provided for adding and removing elements.	व्हेक्टरमध्ये, घटक जोडण्यासाठी आणि काढण्यासाठी पद्धती प्रदान केल्या जातात. In vector, methods are provided for adding and removing elements.
निर्मितीनंतर, ॲरे ही निश्चित-लांबीची रचना असते. After creation, an array is a fixed-length structure.	व्हेक्टर तयार झाल्यानंतर आयटम जोडणे आणि काढून टाकणे यासाठी व्हेक्टरचा आकार आवश्यकतेनुसार वाढू शकतो किंवा लहान होऊ शकतो. The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the vector has been created.
Array is unsynchronized. ॲरे असंक्रमित आहे.	व्हेक्टर सिंक्रोनाइझ आहे. Vector is synchronized.
ॲरेद्वारे निश्चित आकाराची मेमरी वाटप केली जाते.Fixed-size memory allocates by the array.	व्हेक्टरमध्ये, मेमरी डायनॅमिकद्वारे वाटप करते. In vector, memory allocates by the dynamic.
ॲरेमध्ये एकाच प्रकारचे एलिमेंट आहेत. An array is a homogeneous.	व्हेक्टर मध्ये विविध प्रकारचे एलिमेंट आहेत. Vector are heterogeneous.

1.4.5 रॅपर क्लासेस (Wrapper classes):

व्हेक्टर इनटीजर , फ्लोट, लाँग, क्यारेक्टर आणि डबल सारखे आदिम डेटा प्रकार हाताळू शकत नाहीत. जावामध्ये असलेल्या रॅपर क्लासेसचा वापर करून आदिम डेटा प्रकार ऑब्जेक्ट प्रकारात रूपांतरित केले जाऊ शकतात. lang पॅकेज. सारणी साधे डेटा प्रकार आणि त्यांचे संबंधित रॅपर वर्ग प्रकार दर्शवते. java.lang पॅकेजचे आठ वर्ग जावामध्ये रॅपर क्लासेस म्हणून ओळखले जातात. आठ रॅपर वर्गांची यादी खाली दिली आहे:

Table 1.10 रॅपर क्लासेस

Primitive Type	**Wrapper class**
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

ऑटोबॉक्सिंग Autoboxing :

प्रिमिटिव्ह डेटा प्रकाराचे त्याच्या संबंधित रॅपर क्लासमध्ये स्वयंचलित रूपांतरण ऑटोबॉक्सिंग म्हणून ओळखले जाते, उदाहरणार्थ, byte to Byte, char to Character, int to Integer, long to Long, float to Float, boolean to Boolean,

double to Double, and short to Short.

Wrapper class Example: Primitive to Wrapper

```
//program to convert primitive into objects Autoboxing example of int to Integer
public class WrapperExample1
{
  public static void main(String args[])
   {
        //Converting int into Integer
        int a=20;
        Integer i=Integer.valueOf(a);//converting int into Integer explicitly
        Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
          System.out.println(a+" "+i+" "+j);
   }
}
```

Output:

20 0 20

अनबॉक्सिंग Unboxing :

रॅपर प्रकाराचे त्याच्या संबंधित आदिम प्रकारात स्वयंचलित रूपांतर अनबॉक्सिंग म्हणून ओळखले जाते. ही ऑटोबॉक्सिंगची उलट प्रक्रिया आहे

Wrapper class Example: Wrapper to Primitive

Program to convert object into primitives Unboxing example of Integer to int

```
public class WrapperExample2
{
  public static void main(String args[])
  {
        //Converting Integer to int
        Integer a=new Integer(3);
        int i=a.intValue();   //converting Integer to int explicitly
        int j=a;                    //unboxing, now compiler will write a.intValue() internally
        System.out.println(a+" "+i+" "+j);
 }}
```

1.5.1 **कन्स्ट्रक्टर आणि मेथड (Constructors and methods)**

1.5.1.1 कन्स्ट्रक्टर

जावा कन्स्ट्रक्टर Java Constructors नावाच्या एका विशिष्ट प्रकारच्या पद्धतीला(method) सपोर्ट करते, जे ऑब्जेक्ट तयार केल्यावर स्वतःला आरंभ करण्यास सक्षम असतात.

कन्स्ट्रक्टरची वैशिष्ट्ये खालीलप्रमाणे आहेत:

- कन्स्ट्रक्टरचे (Constructor) नाव क्लाससारखेच(class) असते
- ते रिटर्न टाईप निर्दिष्ट करत नाहीत अगदी शून्य ***सुद्धा*** कारण ते वर्गाचेच उदाहरण परत करतात

1.5.1.2 Java मध्ये रिटर्न स्टेटमेंट Return Statement in Java

Java मध्ये रिटर्न स्टेटमेंट म्हणजे काय?

Java प्रोग्रामिंगमध्ये, रिटर्न स्टेटमेंटचा वापर ब्लॉकची अंमलबजावणी पूर्ण झाल्यावर मूल्य परत करण्यासाठी केला जातो. लूपमधील रिटर्न स्टेटमेंटमुळे लूप खंडित होईल आणि कंपाइलरद्वारे पुढील विधानांकडे दुर्लक्ष केले जाईल.

पद्धतीवरून मूल्य परत करणे Returning a Value from a Method

जावामध्ये, प्रत्येक पद्धत इंट, फ्लोट, डबल, स्ट्रिंग इत्यादी रिटर्न प्रकारासह घोषित केली जाते.
या रिटर्न प्रकारांना पद्धतीच्या शेवटी रिटर्न स्टेटमेंट आवश्यक आहे. परिणामी मूल्य परत करण्यासाठी रिटर्न कीवर्ड वापरला जातो.
व्हॉइड रिटर्न प्रकाराला कोणत्याही रिटर्न स्टेटमेंटची आवश्यकता नसते. व्हॉइड पद्धतीने व्हॅल्यू रिटर्न करण्याचा प्रयत्न केल्यास कंपाइलर एरर दाखवतो.

Syntax:

return returnvalue;

returnvalue रिटर्न करायच्या मूल्य.

उदाहरण:

```
public class SampleReturn1
{
   /* Method with an integer return type and no arguments */
  public int CompareNum()
 {
  int x = 3;
 int y = 8;
 System.out.println("x = " + x + "\ny = " + y);
 if(x>y)
   return x;
  else
    return y;    }
    public static void main(String ar[])
  {
        SampleReturn1 obj = new SampleReturn1();
         int result = obj.CompareNum();
        System.out.println("The greater number among x and y is: " + result);
    }  }
```

Output:

x = 3

y = 8

The greater number among x and y is: 8

1.5.2 कन्स्ट्रक्टरचे प्रकार (Types of Constructors)

दोन प्रकारचे कन्स्ट्रक्टर आहेत.

1डीफॉल्ट कन्स्ट्रक्टर (Default constructor):

कोणताही पॅरामीटर नसलेला कन्स्ट्रक्टर (No argument constructor) डीफॉल्ट कन्स्ट्रक्टर म्हणून ओळखला जातो. C ++प्रमाणे, वापरकर्त्याने लिहिलेले कोणतेही डीफॉल्ट किंवा पॅरामीटराइज्ड कन्स्ट्रक्टर नसल्यास Java आपोआप डीफॉल्ट कन्स्ट्रक्टर तयार करतो आणि (C++ प्रमाणे) डीफॉल्ट कन्स्ट्रक्टर स्वयंचलितपणे पॅरेंट डीफॉल्ट कन्स्ट्रक्टरला parent default constructor कॉल करतो. परंतु C++ च्या विपरीत, जावा मधील डिफॉल्ट कन्स्ट्रक्टर सदस्य डेटा व्हेरिएबलला डीफॉल्ट मूल्यांवर प्रारंभ करतो (संख्यात्मक मूल्ये 0 म्हणून आरंभ केली जातात, बूलियन्स मूल्ये खोटे(false) म्हणून आरंभ केली जातात आणि निरर्थक (null) म्हणून आरंभ केले जातात).

प्रोग्राम: डीफॉल्ट कन्स्ट्रक्टर वापरून आयताचे क्षेत्र शोधणे Program to calculate area of rectangle using default constructor

```
class Rectangle
```

```
{
    int l,b,a;
    Rectangle()
    {
        l=135;
        b=12;
    }
    void GetData()
    {
    a=l*b;
    System.out.println ("Area of Rectangle is : "+a);
    }
}
class RectangleDefaultConstructor
 {
    public static void main(String args[])
    {
         Rectangle Rect = new Rectangle();
        Rect.GetData();
    }
}
```

2 .पॅरामीटराइज्ड कन्स्ट्रक्टर (Parameterized Constructor):

पॅरामीटर्स असलेल्या कन्स्ट्रक्टरला पॅरामीटराइज्ड कंस्ट्रक्टर म्हणून ओळखले जाते. जर आपल्याला स्वतःच्या व्हॅल्यूसह क्लासचे फील्ड इनिशिलाइज करायचे असतील तर पॅरामीटराइज्ड कन्स्ट्रक्टर वापरा

प्रोग्राम: पॅरामीटराइज्ड कन्स्ट्रक्टर वापरून आयताचे क्षेत्रफळ शोधणे Program to calculate area of rectangle using parameterised constructor

```
import java.io.*;
class Rectangle
{
    int l,b,a;
    Rectangle(int x, int y)
    {
        l = x;
        b = y;
    }
    void GetArea()
    {
        a=l*b;
        System.out.println ("Area of Rectangle is :"+a);
    }
}
class RectangleParameterisedConstructor
{
    public static void main (String args[])
```

```
    {
        Rectangle r1=new Rectangle (10,25);
        r1.GetArea();}   }
```

1.5.3 कन्स्ट्रक्टर ओव्हरलोडिंग:

जावा मध्ये, आपण मेथड्स सारखे कन्स्ट्रक्टर ओव्हरलोड करू शकतो. कन्स्ट्रक्टर ओव्हरलोडिंगची व्याख्या वेगवेगळ्या पॅरामीटर्ससह एकापेक्षा जास्त कन्स्ट्रक्टर असण्याची संकल्पना म्हणून केली जाऊ शकते जेणेकरून प्रत्येक कन्स्ट्रक्टर वेगळे कार्य करू शकेल.

उदाहरण:

खालील जावा प्रोग्राममध्ये आपण क्लासमध्ये वेगवेगळे कन्स्ट्रक्टर वापरले आहेत.

```
public class Student
{
//instance variables of the class
int id;
String name;
  Student()
{
    System.out.println ("this a default constructor");
}
 Student(int i, String n)
{
    Id = i;
    Name = n;
}
  public static void main (String[] args)
    {
    //object creation
    Student s = new Student ();
    System.out.println("\nDefault Constructor values: \n");
    System.out.println ("Student Id: "+s.id + "\nStudent Name: "+s.name);
    System.out.println("\nParameterized Constructor values: \n");
    Student student = new Student (10, "Avanish");
    System.out.println("Student Id: "+student.id + "\student Name: "+student.name);
    }
}
```

Output:

```
this a default constructor
Default Constructor values:
Student Id : 0
Student Name : null
Parameterized Constructor values:
Student Id : 10
Student Name : Avanish
```

1.5.4 नेस्टिंग ऑफ मेथड (Nesting of methods)

डॉट ऑपरेटर (dot operator) वापरून क्लासची (class) पद्धत फक्त त्या वर्गाच्या ऑब्जेक्टद्वारे (class object) (किंवा स्वतः

वर्ग) कॉल केली जाऊ शकते. तथापि याला अपवाद आहे. एखाद्या पद्धतीचे (method) नाव वापरून त्याच (class)वर्गाच्या दुसऱ्या पद्धतीद्वारे(method) कॉल केले जाऊ शकते. याला नेस्टिंग ऑफ मेथड म्हणतात.

प्रोग्राम: नेस्टिंग ऑफ मेथड वापर करून दोन संख्यापैकी सर्वात मोठी संख्या शोधणे

```
import java.io.*;
class nesting
{
     int m,n;
    nesting(int x,int y)
   {
        m=x;
        n=y;
}
int largest()
{
    if(m>=n)
         return (m);
else
        return (n);
}
void display()
{
    int large=largest();
    System.out.println("largest value="+large);
}
}
class nestingTest
{
     public static void main(String args[])
    {
        nesting nest=new nesting (50,20);
       nest.display();
}
}
```

Output:

largest value =50

1.5.5 कमांड लाइन आर्ग्युमेंट (Command line argument)

कमांड लाइन आर्ग्युमेंट हा प्रोग्राम चालवताना त्या वेळी पास केलेला आर्ग्युमेंट आहे. जावा प्रोग्राममध्ये कमांड लाइन आर्ग्युमेंट देणे खूप सोपे आहे कारण ते मुख्य पद्धतीच्या आर्ग्युमेंट पॅरामीटरला पास केलेल्या स्ट्रिंग अ‍ॅरेमध्ये स्ट्रिंग म्हणून संग्रहित केले जातात.

उदाहरण : **क्लास डेमो चालवताना, तुम्ही कमांड लाइन आर्ग्युमेंट्स म्हणून निर्दिष्ट करू शकता**

```
java Demo arg1 arg2 arg3
```

1. तुमचा ऍप्लिकेशन लॉन्च करताना कॉन्फिगरेशन माहिती निर्दिष्ट करण्यासाठी कमांड लाइन आर्ग्युमेंट्स वापरले जाऊ शकतात.
2. जावा कमांड लाइन आर्ग्युमेंट्सच्या संख्येवर कोणतेही बंधन नाही .

3. माहिती स्ट्रिंग्स म्हणून पास केली जाते.
4. ते तुमच्या मुख्य पद्धतीच्या String args मध्ये कॅप्चर केले जातात.

जावा कमांड लाइन आर्ग्युमेंट्स शिकण्यासाठी प्रोग्राम

```
class Demo
{
public static void main(String b[])
	{
		System.out.println ("Argument one = "+b[0]);
		System.out.println ("Argument two = "+b[1]); } }
compile by > javac Demo.java
run by > java Demo Java PHP
```

Output

```
 Argument one = Java
 Argument two = PHP
```

कमांड लाइन पॅरामीटर वापरून विषम किंवा सम तपासण्यासाठी प्रोग्राम

```
class OddOrEven
{
  public static void main(String args[])
	{
	int n=Integer.parseInt(args[0]);
	if (n % 2 == 0)
		System.out.println ("The number is even.");
	else
		System.out.println("The number is odd.");
	}
}
```

1.5.6 गारबेज कलेक्शन (Garbage Collection):

Java मधील गारबेज कलेक्शन ही प्रक्रिया आहे ज्याद्वारे Java प्रोग्राम स्वयंचलित मेमरी व्यवस्थापन करतात. जावा प्रोग्राम्स बाइट कोडवर संकलित करतात जे Java व्हर्च्युअल मशीनवर किंवा थोडक्यात JVM वर चालवले जाऊ शकतात. जेव्हा Java प्रोग्राम्स JVM वर चालतात, तेव्हा ऑब्जेक्ट्स हीपवर तयार होतात, जो प्रोग्रामला समर्पित मेमरीचा एक भाग असतो. अखेरीस, काही वस्तूंची यापुढे आवश्यकता राहणार नाही. गारबेज कलेक्शन करणाऱ्याला या न वापरलेल्या ऑब्जेक्ट सापडतात आणि मेमरी मोकळी करण्यासाठी त्या रिमूव्ह करतात.

• **फायनलाईझ मेथड (finalize ( ) method):**

जेव्हा एखादी वस्तू घोषित केली जाते तेव्हा कन्स्ट्रक्टर पद्धत वापरली जाते. ही प्रक्रिया आरंभ म्हणून ओळखली जाते. त्याचप्रमाणे, जावा फायनलायझेशन, जे इनिशिएलायझेशनच्या अगदी विरुद्ध आहे. जावा रन-टाइम ही एक स्वयंचलित गारबेज कलेक्शन करणारी यंत्रणा आहे. हे ऑब्जेक्ट्सद्वारे वापरलेली मेमरी संसाधने स्वयंचलितपणे मुक्त करते. परंतु ऑब्जेक्ट्समध्ये इतर नॉन-ऑब्जेक्ट संसाधने असू शकतात जसे की फाइल वर्णनकर्ता किंवा विंडो सिस्टम फॉन्ट. गारबेज कलेक्शन हे स्त्रोत मुक्त करू शकत नाहीत. ही संसाधने मुक्त करण्यासाठी आपण फायनलायझर पद्धत वापरणे आवश्यक आहे. हे C++ मधील विध्वंसकांसारखेच आहे.

Finalize() ही ऑब्जेक्ट क्लासची पद्धत आहे. finalize() पद्धत प्रणाली संसाधनांची विल्हेवाट लावण्यासाठी, क्लीन-अप करण्यासाठी आणि मेमरी फ्री करण्यासाठी ओव्हरराइड करतात.

Syntax:

```
protected void finalize() throws Throwable
```

Throw

Throwable -

अपवाद या पद्धतीद्वारे उठविला जातो (the Exception is raised by this method)

उदाहरण :

```
public class JavafinalizeExample1
{
public static void main(String[] args)
{
  JavafinalizeExample1 obj = new JavafinalizeExample1();
  System.out.println(obj.hashCode());
  obj = null;
  // calling garbage collector
  System.gc();
  System.out.println("end of garbage collection");
}
@Override
protected void finalize()
{
  System.out.println("finalize method called");
  }
 }
```

1.5.7 व्हिसबलिटी कंट्रोल (Visibility Control):

आपण हे देखील पाहिले आहे की क्लासचे व्हेरिएबल्स आणि पद्धती प्रोग्राममध्ये सर्वत्र दिसतात. तथापि, काही परिस्थितींमध्ये वर्गाच्या बाहेरील विशिष्ट व्हेरिएबल्स आणि पद्धतींवर प्रवेश प्रतिबंधित करणे आवश्यक असू शकते. उदाहरणार्थ, आम्हाला क्लासचे ऑब्जेक्ट्स थेट व्हेरिएबल ऍक्सेस पद्धतीचे मूल्य बदलू शकत नाहीत. इंस्टेन्स व्हेरिएबल्स पद्धतींमध्ये दृश्यमानता सुधारक लागू करून आम्ही हे जावा मध्ये साध्य करू शकतो. दृश्यमानता सुधारकांना प्रवेश सुधारक म्हणून देखील ओळखले जाते. जावा तीन प्रकारचे ऍक्सेस स्पेसिफायर प्रदान करते: सार्वजनिक or पब्लिक, खाजगी or प्राइवेट Private आणि संरक्षित or प्रोटेक्टेड Protected. खाली वर्णन केल्याप्रमाणे ते संरक्षणाचे विविध स्तर प्रदान करतात.

1.पब्लिक (Public):

कोणतीही व्हेरिएबल किंवा पद्धत ज्या वर्गामध्ये ती परिभाषित केली आहे त्या संपूर्ण वर्गासाठी दृश्यमान आहे. या वर्गाबाहेरील सर्व वर्गांना ते दृश्यमान करायचे असेल तर,व्हेरिएबल किंवा पद्धत सार्वजनिक **पब्लिक** म्हणून घोषित करून हे शक्य आहे.

उदाहरण :

```
public int number;
public void sum( ) {………….}
```

उदाहरण :

```
//save by A.java
  package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;
```

```
 class B{
 public static void main(String args[]){
  A obj = new A();
  obj.msg();
 }
}
```

Output: Hello

या उदाहरणात, पब्लिक ऍक्सेस मॉडिफायर सर्वत्र प्रवेश करण्यायोग्य आहे.

2.फ्रेंडली (Friendly):

जेव्हा कोणताही ऍक्सेस मॉडिफायर निर्दिष्ट केलेला नसतो, तेव्हा सदस्य "फ्रेंडली" ऍक्सेस स्तर म्हणून ओळखल्या जाणाऱ्या सार्वजनिक प्रवेशयोग्यतेच्या मर्यादित आवृत्तीवर डीफॉल्ट करतो. "पब्लिक" ऍक्सेस आणि "फ्रेंडली" ऍक्सेसमधला फरक असा आहे की पब्लिक मॉडिफायर सर्व क्लासेसमध्ये फील्ड्स दृश्यमान बनवतो, त्यांच्या पॅकेजची पर्वा न करता, तर फ्रेंडली ऍक्सेस फील्ड्स फक्त त्याच पॅकेजमध्ये दृश्यमान करतो, परंतु इतर पॅकेजमध्ये नाही.

3.प्रोटेक्टेड (Protected):

"प्रोटेक्टेड " फील्डची दृश्यमानता पातळी सार्वजनिक पब्लिक प्रवेश आणि मैत्रीपूर्ण फ्रेंडली प्रवेश दरम्यान असते. म्हणजेच, प्रोटेक्टेड सुधारक फील्ड केवळ एकाच पॅकेजमधील सर्व वर्ग आणि उपवर्गांनाच नाही तर इतर पॅकेजेसमधील उपवर्गांना देखील दृश्यमान बनवतो.

उदाहरण :

```
//save by A.java
package pack;
public class A
{
protected void msg(){System.out.println("Hello");}
}
//save by B.java
package mypack;
import pack.*;

class B extends A
{
 public static void main(String args[])
{
  B obj = new B();
  obj.msg();
 }
}
```

या उदाहरणात, आम्ही दोन पॅकेजेस पॅक आणि मायपॅक तयार केले आहेत. पॅक पॅकेजचा एक वर्ग सार्वजनिक आहे, त्यामुळे पॅकेजच्या बाहेरून प्रवेश केला जाऊ शकतो. परंतु या पॅकेजची msg पद्धत संरक्षित म्हणून घोषित केली आहे, म्हणून ती केवळ वारसाहक्काद्वारे वर्गाबाहेरून प्रवेश करू शकते.

4. प्रायव्हेट (Private) :

प्राइवेट क्षेत्रांना सर्वोच्च संरक्षण असते. ते फक्त त्यांच्या स्वतःच्या वर्गासह प्रवेशयोग्य आहेत. ते उपवर्गांद्वारे वारशाने मिळू शकत नाहीत आणि म्हणून उपवर्गांमध्ये प्रवेशयोग्य नाहीत. ओव्हरराइड करण्याच्या बाबतीत सार्वजनिक पब्लिक पद्धती खाजगी प्रायव्हेट प्रकार म्हणून पुन्हा परिभाषित केल्या जाऊ शकत नाहीत.

उदाहरण :

```
class A
{
private int data=40;
private void msg()
{
System.out.println("Hello java");}
}
public class Simple
{
 public static void main(String args[])
{
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }  }
```

या उदाहरणात आपण A आणि साधे असे दोन वर्ग तयार केले आहेत. वर्गात खाजगी डेटा सदस्य आणि खाजगी पद्धत असते. आम्ही या खाजगी सदस्यांना वर्गाच्या बाहेरून प्रवेश देत आहोत, त्यामुळे संकलित वेळेची त्रुटी आहे.

5.प्रायव्हेट प्रोटेक्टेड (Private Protected):

फील्ड खाजगी प्राइवेट आणि संरक्षित प्रोटेक्टेड अशा दोन कीवर्डसह घोषित केले जाऊ शकते. हे "संरक्षित" प्रोटेक्टेड प्रवेश आणि "खाजगी" प्राइवेट प्रवेश दरम्यान दृश्यमानता पातळी देते. हे सुधारक फील्ड्स कोणत्या पॅकेजमध्ये आहेत याची पर्वा न करता सर्व उपवर्गांमध्ये दृश्यमान करते. लक्षात ठेवा, ही फील्ड समान पॅकेजमधील इतर वर्गांद्वारे प्रवेशयोग्य नाहीत.

नियम

योग्य ॲक्सेस मॉडिफायर लागू करण्यासाठी खाली काही सोपे नियम दिले आहेत.

1. फील्ड सर्वत्र दिसायचे असल्यास सार्वजनिक Public पब्लिक वापरा.
2. जर फील्ड सध्याच्या पॅकेजमध्ये सर्वत्र दृश्यमान असेल आणि इतर पॅकेजेसमध्ये देखील उपवर्ग असेल तर संरक्षित Protected प्रोटेक्टेड वापरा.
3. फील्ड फक्त वर्तमान पॅकेजमध्ये सर्वत्र दृश्यमान असेल तर "डिफॉल्ट" वापरा.
4. फील्ड केवळ उपवर्गांमध्ये दिसायचे असल्यास, पॅकेजेसची पर्वा न करता खाजगी संरक्षित प्राइवेट प्रोटेक्टेड वापरा.
5. फील्ड स्वतःच्या वर्गाशिवाय कोठेही दृश्यमान नसल्यास खाजगी Private प्राइवेट वापरा.

खालील सारणी विविध ॲक्सेस मॉडिफायर्सद्वारे प्रदान केलेल्या दृश्यमानतेचा सारांश देते.

Table 1.11 व्हिजिबिलिटी कंट्रोल

Access modifier or Visibility Control ->	**Public पब्लिक**	**Protected प्रोटेक्टेड**	**Friendly फ्रेंडली**	**private protected प्राइवेट प्रोटेक्टेड**	**Private प्राइवेट**
Own class	Y	Y	Y	Y	Y
Sub class in same package	Y	Y	Y	Y	N
Other classes in same package	Y	Y	Y	N	N
Sub class in other package	Y	Y	N	Y	N
Other classes In other package	Y	N	N	N	N

संदर्भ (Web References):

1. https://www.w3schools.com/java/java_constructors.asp
2. https://www.geeksforgeeks.org/operators-in-java/
3. https://www.javatpoint.com/java-string
4. https://www.geeksforgeeks.org/wrapper-classes-java.

युनिट - 2
इनहेरिटेन्स इंटरफेस आणि पॅकेजेस
(Inheritance, Interface and Packages)

विषय निष्पत्ती: Course Outcome (CO)

CO2 - कोड रेयूझ्याबिलिटी संकल्पना अंमलात आणण्यासाठी जावा प्रोग्राम विकसित करा.

सिद्धांत शिक्षण परिणाम (Theory Learning Outcomes (TLO)):

1. दिलेल्या प्रोग्रामिंग समस्येसाठी ओळखल्या जाणाऱ्या प्रकाराचा इनहेरिटेन्स लागू करा.
2. उदाहरणांच्या मदतीने ओव्हरलोडिंग आणि ओव्हरराइडिंग दरम्यान फरक करा.
3. इंटरफेसचा वापर करून प्रोग्राम विकसित करा.
4. दिलेल्या प्रोग्रामसाठी युजर डिफाइन पॅकेज तयार करा.

2.1 इनहेरिटेन्सची संकल्पना (Concept of Inheritance)

- जावा मध्ये, इनहेरिटन्स हा एक ऑब्जेक्ट-ओरिएंटेड प्रोग्रॅमिंग संकल्पना आहे, जो एका क्लास ला दुसऱ्या क्लास चे गुणधर्म (फिल्ड्स) आणि वर्तन (मेथड्स) प्राप्त करण्याची परवानगी देतो.
- ही अशी प्रक्रिया आहे जिथे एक ऑब्जेक्ट पालक ऑब्जेक्ट चे सर्व गुणधर्म आणि वर्तन मिळवतो.
- जावा मधील इनहेरिटन्स च्या संकल्पनेमागील उद्दिष्ट असे आहे की आपण विद्यमान क्लासेस वर आधारित नवीन क्लासेस तयार करू शकतो.
- जेव्हा आपण विद्यमान क्लास कडून मेथड्स इनहेरिट करतो, तेव्हा आपण पालक क्लास चे मेथड्स आणि फिल्ड्स पुन्हा वापरू शकतो. तसेच आपण आपल्या सध्याच्या क्लास मध्ये नवीन मेथड्स आणि फिल्ड्स देखील जोडू शकतो.
- ज्या क्लास चे वैशिष्ट्ये इनहेरिट केली जातात, त्याला सुपरक्लास (बेस क्लास किंवा पॅरेंट क्लास) म्हणतात.
- जो क्लास दुसऱ्या क्लास ला इनहेरिट करते, तिला सबक्लास (डिराइव्ड क्लास, एक्सटेंडेड क्लास किंवा चाइल्ड क्लास) म्हणतात. सबक्लास सुपरक्लास च्या फिल्ड्स आणि मेथड्स व्यतिरिक्त स्वतःचे फिल्ड्स आणि मेथड्स जोडू शकते.

➢ **जावामध्ये इनहेरिटन्स आवश्यक आहे कारण**

1. कोड रीउसएबिलिटी (Code Reusability)

सुपरक्लास मध्ये लिहिलेला कोड सबक्लास मध्ये थेट वापरू शकतो. यामुळे कोडची पुनरावृत्ती टाळली जाते आणि DRY (डोंट रिपीट युअरसेल्फ) तत्त्व पाळले जाते.

2. मेथड ओव्हररायडिंग (Method Overriding)

सबक्लास हा सुपरक्लासमधील पद्धती (मेथड) अधिलिखित करू शकतो आणि स्वतःची अंमलबजावणी (Implementation) करू शकतो. यामुळे चालू अवस्थेतील बहुरूपता (रनटाइम पॉलिमॉर्फिझम) साध्य होते, ज्यामुळे चालू अवस्थेत (रनटाइम) योग्य पद्धत कॉल केली जाते.

3. ॲब्स्ट्रक्शन (Abstraction):

जेव्हा आपल्याला सर्व तपशील प्रदान करण्याची गरज नसते, तेव्हा ही संकल्पना इनहेरिटन्स (Inheritance) द्वारे साध्य केली जाते. ॲब्स्ट्रक्शन वापरकर्त्याला केवळ कार्यक्षमता दर्शवतो.

· **जावा इनहेरिटन्सचा सिंटॅक्स:**

```
class Subclass-name extends Superclass-name
{
   // मेथड्स (methods) आणि फील्ड्स (fields)
}
```

extends ही कीवर्ड दर्शवते की आपण आधीपासून अस्तित्वात असलेल्या वर्गापासून (class) नवीन वर्ग तयार

करत आहोत. extends म्हणजे कार्यक्षमता वाढवणे.

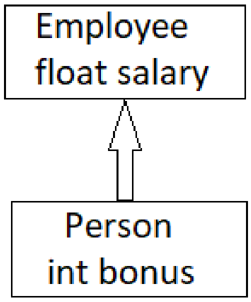


Fig 2.1: इनहेरिटन्सचे उदाहरण

वरील आकृती 2.1 मध्ये दर्शविल्याप्रमाणे, Person हा सबक्लास (Subclass) आहे आणि Employee हा सुपरक्लास (Superclass) आहे. या दोन वर्गांमधील संबंध "Programmer IS-A Employee" असा आहे. याचा अर्थ Person हा Employee चा एक प्रकार आहे.

- **जावा इनहेरिटन्स उदाहरण**

File Name: Programmer.java

```
 class Employee
{
  float salary=40000;
 }
 Class Person extends Employee
{
 int bonus=10000;
public static void main(String args[])
{
   Person p=new Programmer ();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
 }  }
```

Output

```
Programmer salary is:40000.0
Bonus of programmer is:10000
```

वरील उदाहरणात**, Programmer** ऑब्जेक्ट स्वतःच्या वर्गाचे फील्ड तसेच **Employee** वर्गाचे फील्ड ॲक्सेस करू शकतो, म्हणजेच **Code Reusability** साध्य होते.

2.2 जावा मधील इनहेरिटन्सचे प्रकार (Types of Inheritance in Java)

खाली **जावा** मध्ये समर्थित विविध प्रकारचे इनहेरिटन्स दिले आहेत:

1. सिंगल इनहेरिटन्स (Single Inheritance)
2. मल्टीलेव्हल इनहेरिटन्स (Multilevel Inheritance)
3. हायरार्किकल इनहेरिटन्स (Hierarchical Inheritance)
4. मल्टिपल इनहेरिटन्स (Multiple Inheritance) – इंटरफेसद्वारे
5. हायब्रिड इनहेरिटन्स (Hybrid Inheritance) – इंटरफेसद्वारे

1.सिंगल इनहेरिटन्स (Single Inheritance)

सिंगल इनहेरिटन्समध्ये, सब-क्लास फक्त एका सुपर-क्लासपासून प्राप्त (derive) होतो. हा आपल्या पालक वर्गाच्या (Parent Class) सर्व गुणधर्म (Properties) आणि वर्तन (Behavior) वारसा म्हणून घेतो. कधीकधी याला सिंपल

इनहेरिटन्स (Simple Inheritance) असेही म्हणतात.खालील आकृतीत, '**A**' हा पॅरेंट क्लास आहे आणि '**B**' हा चाइल्ड क्लास आहे. '**B**' हा वर्ग '**A**' वर्गाचे सर्व गुणधर्म वारसा म्हणून प्राप्त करतो.

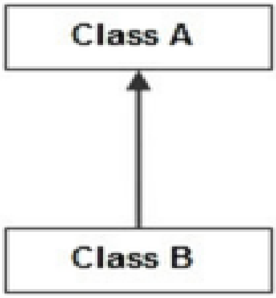


Fig 2.2: सिंगल इनहेरिटन्स

खाली सिंगल क्लास इनहेरिटन्सचे उदाहरण दिले आहे, ज्यामध्ये Calculation हा सुपर क्लास आहे आणि my_Calculation हा सबक्लास (एक्सटेंडेड क्लास) आहे. येथे सबक्लास सुपर क्लासच्या सर्व मेथड्स इनहेरिट करू शकतो.

```
class Calculation
 {
       int z;
       public void addition(int x, int y) {
       z = x + y;
       System.out.println("The sum of the given numbers:"+z); }
        public void Subtraction(int x, int y) {
         z = x - y;
         System.out.println("The difference between the given numbers:"+z);
     }   }
      public class My_Calculation extends Calculation {
      public void multiplication(int x, int y) {
          z = x * y;
          System.out.println("The product of the given numbers:"+z);  }
     public static void main(String args[]) {
     int a = 20, b = 10;
     My_Calculation demo = new My_Calculation();
     demo.addition(a, b);
     demo.Subtraction(a, b);
     demo.multiplication(a, b);
   }  }
```

Output

The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200

दिलेल्या प्रोग्राममध्ये, जेव्हा My_Calculation क्लासचा ऑब्जेक्ट तयार केला जातो, तेव्हा त्यामध्ये सुपरक्लासची प्रत (copy) तयार केली जाते. हे आकृती 2.3 मध्ये दर्शविले आहे. यामुळे**,** सबक्लासच्या ऑब्जेक्टचा वापर करून सुपरक्लासचे मेंबर्स ॲक्सेस करता येतात**.**सुपरक्लासचा रेफरन्स व्हेरिएबल सबक्लासचा ऑब्जेक्ट धारण करू शकतो. परंतु, त्या व्हेरिएबलद्वारे फक्त सुपरक्लासचे मेंबर्स ॲक्सेस करता येतात. त्यामुळे, दोन्ही क्लासचे मेंबर्स ॲक्सेस करण्यासाठी सबक्लासचा रेफरन्स

व्हेरिएबल तयार करणे योग्य ठरते.जर तुम्ही वरील प्रोग्राम विचारात घेतला, तर खालीलप्रमाणे क्लास इन्स्टंटिएट करू शकता. पण, जर सुपरक्लासचा रेफरन्स व्हेरिएबल (उदा. cal) वापरला, तर My_Calculation या सबक्लासमधील multiplication() ही मेथड कॉल करता येणार नाही.

```
Calculation demo = new My_Calculation();
demo.addition(a, b);
demo.Subtraction(a, b);
```

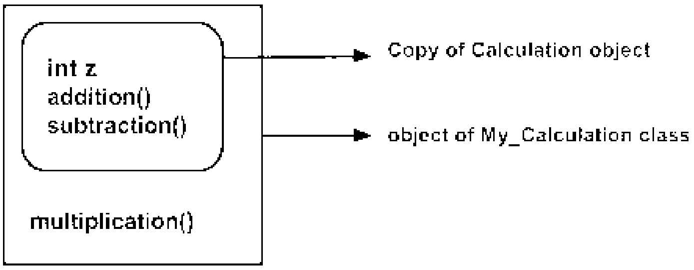


Fig 2.3: मेमरीतील ऑब्जेक्ट संरचना

2.मल्टीलेव्हल इनहेरिटन्स (Multilevel Inheritance)

मल्टीलेव्हल इनहेरिटन्समध्ये, एक डिरॅव्ह क्लास (Derived Class) हा बेस क्लासला (Base Class) इनहेरिट करत असतो, तसेच तो डिरॅव्ह क्लास पुढील इतर क्लाससाठी बेस क्लास म्हणून कार्य करतो.खालील प्रतिमेत, क्लास A हा एक डिरॅव्ह क्लास B साठी बेस क्लास आहे, आणि B हा पुढे डिरॅव्ह क्लास C साठी बेस क्लास म्हणून कार्य करतो.जावा मध्ये, एका क्लासला थेट त्याच्या ग्रँडपॅरेंट क्लासचे मेंबर्स ॲक्सेस करता येत नाहीत.

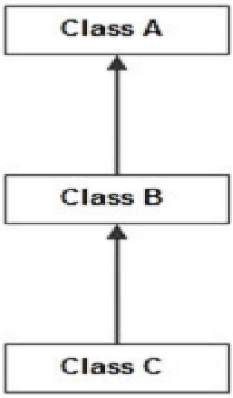


Fig 2.4: मल्टीलेव्हल इनहेरिटन्स

- **मल्टीलेव्हल इनहेरिटन्सचे उदाहरण**

```
// Importing required libraries
import java.io.*;
import java.lang.*;
import java.util.*;
// Parent class One
class One {
   // Method to print "Hello"
   public void print_hello() {
      System.out.println("Hello");
   }
}
// Child class Two inherits from class One
```

```
class Two extends One {
   // Method to print "To_java"
   public void print_Tojava() {
      System.out.println("To Java");
   }
}
// Child class Three inherits from class Two
class Three extends Two {
   // Method to print "World"
   public void print_world() {
      System.out.println("World");
   }
}
// Driver class
public class Multi_level {
   public static void main(String[] args) {
      // Creating an object of class Three
      Three g = new Three();
      // Calling method from class One
      g. print_hello();
      // Calling method from class Two
      g. print_Tojava();
      // Calling method from class Three
      g.print_ print_world();
   }
}
```

Output

Hello

To Java

World

3.हायरार्किकल इनहेरिटन्स (Hierarchical Inheritance)

हायरार्किकल इनहेरिटन्समध्ये, एकाच क्लासला (सुपरक्लास/बेस क्लास) एकापेक्षा जास्त सबक्लास इन्हेरीत करतात खालील प्रतिमेत, क्लास A हा डिरॅव्ह क्लास B, C आणि D साठी बेस क्लास म्हणून कार्य करतो.

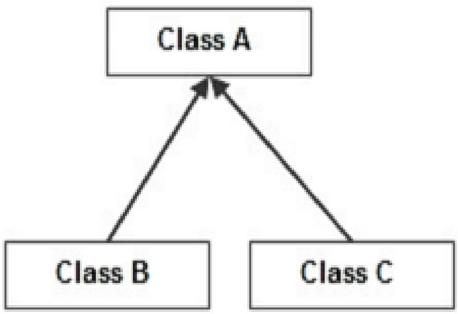


Fig 2.5: हायरार्किकल इनहेरिटन्स

- **हायरार्किकल इनहेरिटन्सचे उदाहरण**

```
// Java program to illustrate the
// concept of Hierarchical  inheritance
class A {
   public void print_A() { System.out.println("Class A"); }
```

```
}
class B extends A {
   public void print_B() { System.out.println("Class B"); }
}
class C extends A {
   public void print_C() { System.out.println("Class C"); }
}
class D extends A {
   public void print_D() { System.out.println("Class D"); }
}
// Driver Class
public class Test {
   public static void main(String[] args)
   {
      B obj_B = new B();
      obj_B.print_A();
      obj_B.print_B();
      C obj_C = new C();
      obj_C.print_A();
      obj_C.print_C();
      D obj_D = new D();
      obj_D.print_A();
      obj_D.print_D();}  }
```

Output

```
Class A
Class B
Class A
Class C
Class A
Class D
```

2.3 मेथड ओव्हररायडिंग (Method Overriding)

जेव्हा सबक्लास सुपरक्लासमध्ये आधीच परिभाषित (Define) केलेल्या मेथडची विशिष्ट अंमलबजावणी (Implementation) प्रदान करतो, तेव्हा त्याला मेथड ओव्हररायडिंग म्हणतात. ओव्हररायडिंग मेथड (Overriding Method) ही सबक्लासमध्ये असते आणि ती सुपरक्लासमधील मेथडसारखीच नाव (Name), परतावा प्रकार (Return Type), आणि पॅरामीटर्स (Parameters) असणे आवश्यक असते. जेव्हा सुपरक्लास आणि सबक्लास दोन्हीमध्ये एकसारखी मेथड परिभाषित (Define) केली जाते, तेव्हा सबक्लासमधील मेथड ही सुपरक्लासमधील मेथडला ओव्हरराइड करते. यालाच मेथड ओव्हररायडिंग (Method Overriding) असे म्हणतात.

मेथड ओव्हररायडिंगचे उदाहरण

```
class Animal {
  public void displayInfo() {
    System.out.println("I am an animal.");
  }
}
class Dog extends Animal {
```

```
  @Override
  public void displayInfo() {
    System.out.println("I am a dog.");
  }
}
class Main {
  public static void main(String[] args) {
    Dog d1 = new Dog();
    d1.displayInfo();
  }
 }
```

Output:

I am a dog.

वरील प्रोग्राममध्ये, displayInfo() ही मेथड Animal सुपरक्लास आणि Dog सबक्लास दोन्हीमध्ये उपलब्ध आहे.जेव्हा d1 ऑब्जेक्ट (जो सबक्लास Dog चा आहे) वापरून displayInfo() मेथड कॉल केली जाते, तेव्हा सबक्लास Dog मधील मेथड कार्यान्वित होते.यामुळे, सबक्लासमधील displayInfo() मेथड ही सुपरक्लासमधील त्याच मेथडला ओव्हरराइड करते.

वरील उदाहरणामध्ये @Override ॲनोटेशनचा वापर दर्शविला आहे. जावा मध्ये, annotations म्हणजे मेटाडेटा (metadata) जे कंपाइलरला अतिरिक्त माहिती प्रदान करण्यासाठी वापरले जाते.@Override वापरणे अनिवार्य (mandatory) नाही, परंतु हे कोड अधिक वाचनीय बनवते आणि चुकीचा ओव्हररायडिंग टाळण्यास मदत करते.

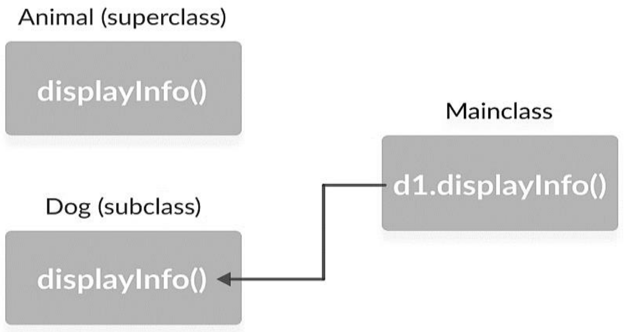


Fig 2.6: मेथड ओव्हररायडिंग

वरील आकृती 2.6 मध्ये मेथड ओव्हररायडिंग संकल्पना दर्शविली आहे.

- **जावा ओव्हररायडिंग नियम (Java Overriding Rules)**
  1. सुपरक्लास आणि सबक्लास दोन्हीमध्ये एकसारखी मेथड असावी, म्हणजेच समान नाव (Method Name), समान परतावा प्रकार (Return Type), आणि समान पॅरामीटर्स (Parameter List) असणे आवश्यक आहे.
  2. final आणि static म्हणून घोषित केलेल्या मेथड्स ओव्हरराइड करता येत नाहीत.
  3. सुपरक्लासमधील abstract मेथड्स नेहमीच ओव्हरराइड करायला हव्यात.

2.4 final व्हेरिएबल्स आणि final मेथड्स

जावा मध्ये, final कीवर्डचा वापर व्हेरिएबल्स, मेथड्स आणि क्लासेसवर निर्बंध लागू करण्यासाठी केला जातो.

2.4.1 final व्हेरिएबल्स

final व्हेरिएबल हा एक कॉन्स्टन्ट (constant) आहे, ज्याची मूल्य एकदा ठरवल्यानंतर बदलता येत नाही.

उदाहरण: final व्हेरिएबल

```
class Test {
  final int MAX_VALUE = 100;  // Final variable (constant)
    void display() {
      System.out.println("MAX_VALUE: " + MAX_VALUE);
```

```
  }
   public static void main(String[] args) {
    Test obj = new Test();
    obj.display();
    // obj.MAX_VALUE = 200; // ❌ Compilation error (Cannot assign a value to final variable)
   } }
```

- **final व्हेरिएबल्ससाठी नियम**
  1. final व्हेरिएबलची व्हॅल्यू डिकलयीर करताना किंवा कन्स्ट्रक्टरमध्ये इनिशियलाइज करणे आवश्यक आहे.
  2. एकदा व्हॅल्यू ठरवल्यानंतर ते बदलता येत नाही.
  3. final static व्हेरिएबल्सना कॉन्स्टन्ट (constants) मानले जाते (बहुतेक वेळा uppercase मध्ये लिहिले जातात).

उदाहरण

```
class Constants {
     static final double PI = 3.14159;  // Final static variable (constant)
    public static void main(String[] args) {
    System.out.println("Value of PI: " + PI);
  }
}
```

Output

Value of PI: 3.14159

2.4.2 final मेथड्स

final मेथडला सबक्लासद्वारे ओव्हरराइड करता येत नाही.

उदाहरण: final मेथड

```
class Parent {
  final void show() {  // Final method
    System.out.println("This is a final method in Parent class.");
  }
}
class Child extends Parent {
  // void show() {  // ❌ Compilation error: Cannot override final method
  //     System.out.println("Trying to override final method.");
  // }
}
public class Main {
  public static void main(String[] args) {
    Child obj = new Child();
    obj.show();  // Calls the final method from Parent class
 }
}
```

- **final मेथड्ससाठी नियम**
  1. final मेथडला सबक्लासमध्ये ओव्हरराइड करता येत नाही.
  2. final मेथडला इनहेरिट (inherit) करून चाइल्ड क्लासमध्ये वापरता येते.
- **final कधी वापरावे?**

1. final व्हेरिएबल्स: जेव्हा तुम्हाला स्थिरांक (constants) तयार करायचे असतात.
2. final मेथड्स: जेव्हा तुम्हाला ओव्हररायडिंग प्रतिबंधित करायचे असते आणि निश्चित वर्तन (fixed behavior) सुनिश्चित करायचे असते.
3. final क्लासेस: जेव्हा तुम्हाला इनहेरिटन्स रोखायचे असते (उदाहरणार्थ, Java मधील String क्लास हा final आहे).

2.5 super कीवर्डचा वापर

1. जर सुपरक्लास आणि सबक्लासमधील मेंबर्सचे नाव एकसारखे असतील, तर त्यांना वेगळे करण्यासाठी super कीवर्डचा वापर केला जातो.
2. सबक्लासमधून सुपरक्लासचा कन्स्ट्रक्टर कॉल करण्यासाठी super कीवर्डचा वापर केला जातो.

- **मेंबर्स वेगळे करण्यासाठी (Differentiating the Members)**

जर एखादा क्लास दुसऱ्या क्लासला इनहेरीट करत असेल आणि सुपरक्लास व सबक्लासमधील मेंबर्सची नावे सारखी असतील, तर त्यांना वेगळे करण्यासाठी super कीवर्डचा वापर केला जातो, जसे खाली दाखवले आहे.

```
super.variable
super.method();
```

इनहेरिटन्समध्ये super कीवर्डच्या वापराचे उदाहरण

```
class Super_class
 {
  int num = 20;
  // display method of superclass
  public void display() {
    System.out.println("This is the display method of superclass");
  }
}
public class Sub_class extends Super_class
{
  int num = 10;
  // display method of sub class
  public void display() {
    System.out.println("This is the display method of subclass");
  }
  public void my_method()
 {
    // Instantiating subclass
    Sub_class sub = new Sub_class();
    // Invoking the display() method of sub class
    sub.display();
    // Invoking the display() method of superclass
    super.display();
    // printing the value of variable num of subclass
    System.out.println("value of the variable named num in sub class:"+ sub.num);
    // printing the value of variable num of superclass
    System.out.println("value of the variable named num in super class:"+ super.num);
  }
  public static void main(String args[])
```

```
 {
    Sub_class obj = new Sub_class();
    obj.my_method();
  }
 }
```

Output

This is the display method of subclass
This is the display method of superclass
value of the variable named num in sub class:10
value of the variable named num in super class:20

2.5.2 सुपरक्लास कन्स्ट्रक्टर कॉल करणे(Invoking Superclass Constructor)

जर एखादा क्लास दुसऱ्या क्लासच्या गुणधर्म इनहेरीट करत असेल, तर सबक्लास आपोआप सुपरक्लासचा डिफॉल्ट कन्स्ट्रक्टर कॉल होतोपरंतु, सुपरक्लासचा पॅरामीटराइज्ड कन्स्ट्रक्टर कॉल करायचा असल्यास, super कीवर्डचा वापर करावा लागतो, जसे खाली दर्शविले आहे.

Super(value);

खाली दिलेला प्रोग्राम super कीवर्डचा वापर करून सुपरक्लासचा पॅरामीटराइज्ड कन्स्ट्रक्टर कसा कॉल करावा हे दर्शवतो.या प्रोग्राममध्ये सुपरक्लास आणि सबक्लास असतो, जिथे सुपरक्लासमध्ये एक पॅरामीटराइज्ड कन्स्ट्रक्टर आहे, जो एक integer व्हॅल्यू स्वीकारतो.सबक्लासमध्ये super कीवर्डचा वापर करून हा पॅरामीटराइज्ड कन्स्ट्रक्टर कॉल केला जातो

उदाहरण

```
class Superclass {
  int age;
  Superclass(int age) {
    this.age = age;
  }
  public void getAge() {
    System.out.println("The value of the variable named age in super class is: " +age);
  }
}
public class Subclass extends Superclass {
int a;
 Subclass(int age) {
    super(age);
 }
  public static void main(String args[]) {
    Subclass s = new Subclass(24);
    s.getAge(); }}
```

Output

The value of the variable named age in super class is: 24

2.6 अब्स्ट्रॅक्ट मेथड्स आणि क्लासेस

2.6.1 अब्स्ट्रॅक्ट क्लासेस

जावा मधील abstract क्लास इन्स्टँशिएट करता येत नाही (म्हणजेच, आपण abstract क्लासचे ऑब्जेक्ट तयार करू शकत नाही). abstract कीवर्डचा वापर करून अब्स्ट्रॅक्ट क्लास घोषित केला जातो.

अब्स्ट्रॅक्ट क्लासमध्ये काय असू शकते?

1. अब्स्ट्रॅक्ट मेथड्स (ज्यांना अंमलबजावणी [implementation] नसते)
2. काँक्रीट मेथड्स (अंमलबजावणी असलेल्या साध्या मेथड्स)
3. इन्स्टन्स व्हेरिएबल्स, कन्स्ट्रक्टर्स आणि static मेथड्स

अब्स्ट्रॅक्ट क्लास कधी वापरावा?

- जेव्हा तुमच्याकडे सामायिक कोड असलेला एक बेस क्लास असतो.
- जेव्हा तुम्हाला सबक्लासमध्ये काही ठराविक मेथड्सची अंमलबजावणी सक्तीने लागू करायची असते.
- जेव्हा तुम्हाला एकाच प्रकारच्या संबंधित क्लासेससाठी टेम्प्लेट तयार करायचे असते.

उदाहरण: अब्स्ट्रॅक्ट क्लास

```
abstract class Animal {
   // Abstract method (does not have a body)
   abstract void makeSound();
   // Concrete method (has a body)
   void sleep() {
      System.out.println("Sleeping...");
   }
}
```

एका अब्स्ट्रॅक्ट क्लासमध्ये नियमित (Concrete) मेथड्स आणि abstract मेथड्स दोन्ही असू शकतात.

2.6.2 अब्स्ट्रॅक्ट मेथड्स

ज्या मेथडला बॉडी (अंमलबजावणी) नसते, तिला अब्स्ट्रॅक्ट मेथड म्हणतात.abstract कीवर्डचा वापर करून अब्स्ट्रॅक्ट मेथड तयार केली जाते.

अब्स्ट्रॅक्ट मेथडचे नियम:

- जर एखाद्या क्लासमध्ये अब्स्ट्रॅक्ट मेथड असेल, तर तो क्लास अब्स्ट्रॅक्ट म्हणून घोषित करावा लागतो.
- जर अब्स्ट्रॅक्ट क्लासमध्ये अब्स्ट्रॅक्ट मेथड असेल आणि ती सबक्लासमध्ये ओव्हरराइड केली नाही, तर सबक्लासलाही अब्स्ट्रॅक्ट घोषित करावे लागते.
- अब्स्ट्रॅक्ट मेथडची अंमलबजावणी सबक्लासमध्ये द्यावी लागते, अन्यथा रनटाईम एरर येऊ शकतो.

उदाहरण:

```
abstract class Animal {
  abstract void makeSound();
  public void eat() {
    System.out.println("I can eat.");
   }
 }
class Dog extends Animal {
  // provide implementation of abstract method
  public void makeSound() {
    System.out.println("Bark bark");
  }
}
class Main {
  public static void main(String[] args) {
    // create an object of Dog class
    Dog d1 = new Dog();

    d1.makeSound();
    d1.eat();
  }
```

```
}
```

Output

Bark bark

I can eat.

- **पेमेंट सिस्टममध्ये अब्स्ट्रॅक्ट क्लासचे वास्तविक उदाहरण**

```
abstract class Payment {
   // Abstract method (to be implemented by subclasses)
   abstract void processPayment(double amount);
    // Concrete method
   void printReceipt() {
      System.out.println("Transaction complete. Receipt generated.");
    }
 }
// Subclass for Credit Card payment
class CreditCardPayment extends Payment {
   @Override
   void processPayment(double amount) {
      System.out.println("Processing credit card payment of $" + amount);
    }
 }
// Subclass for PayPal payment
class PayPalPayment extends Payment {
   @Override
   void processPayment(double amount) {
      System.out.println("Processing PayPal payment of $" + amount);
    }
 }
public class PaymentSystem {
   public static void main(String[] args) {
      Payment payment1 = new CreditCardPayment();
      payment1.processPayment(100.0);
      payment1.printReceipt();
      Payment payment2 = new PayPalPayment();
      payment2.processPayment(200.0);
      payment2.printReceipt();
   }}
```

Output:

Processing credit card payment of $100.0

Transaction complete. Receipt generated.

Processing PayPal payment of $200.0

Transaction complete. Receipt generated

2.7 इंटरफेसची व्याख्या (Define interface)

इंटरफेस हा पूर्णतः अब्स्ट्रॅक्ट क्लास असतो. यात अब्स्ट्रॅक्ट मेथड्स (ज्या मेथड्सना बॉडी नसते) असतात. जावामध्ये इंटरफेस तयार करण्यासाठी interface कीवर्डचा वापर केला जातो. जावामधील interface हा एक ब्लूप्रिंट आहे, जो फक्त abstract

मेथड्स (Java 8 पूर्वी) आणि default/static मेथड्स (Java 8 नंतर) समाविष्ट करतो. याचा उपयोग abstraction आणि मल्टीपल इनहेरिटन्स साध्य करण्यासाठी केला जातो.

इंटरफेसची मुख्य वैशिष्ट्ये:

1. interface कीवर्ड वापरून घोषित केला जातो.
2. instance variables समाविष्ट करू शकत नाही (public static final कॉन्स्टंट्स असतात).
3. constructors नसतात (कारण इंटरफेसचे instantiation करता येत नाही).
4. सर्व मेथड्स public आणि abstract असतात (डिफॉल्टने).
5. multiple inheritance ला सपोर्ट करतो (क्लासेसच्या विपरीत).
6. Java 8 पासून इंटरफेसमध्ये default आणि static मेथड्स अंमलबजावणीसह असू शकतात.
7. Java 9 पासून इंटरफेसमध्ये private मेथड्स असू शकतात.

- **इंटरफेस डिक्लरेशन (Declaring an Interface)**

  इंटरफेस interface कीवर्ड वापरून डिक्लर केला जातो. हे पूर्णतः abstraction प्रदान करते; म्हणजेच, इंटरफेसमधील सर्व मेथड्स empty body सह परिभाषित केल्या जातात आणि सर्व फील्ड्स public, static आणि final असतात (डिफॉल्टने).जो क्लास एखाद्या इंटरफेसला implement करतो, त्याला त्या इंटरफेसमधील सर्व abstract मेथड्स लागू कराव्या लागतात.

सिंटॅक्स

```
interface <interface name>
  {
        // declare constant fields
        // declare methods that abstract
        // by default.
    }
```

उदाहरण

```
interface Animal
        {
           void makeSound(); // Abstract method (by default public and abstract)
        }
```

2.8 इंटरफेसची अंमलबजावणी (Implementing an Interface)

एखादा क्लास implements कीवर्डचा वापर करून इंटरफेस लागू करतो

उदाहरण

या उदाहरणात, Animal इंटरफेसमध्ये **फक्त एकच मेथड** आहे, आणि त्याची अंमलबजावणी Dog क्लासमध्ये केली आहे.

```
interface Animal {
    void makeSound();  // Abstract method
}
// Dog class implements Animal interface
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}
public class InterfaceExample {
```

```
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.makeSound();
    }
}
```

Output:

Dog barks

या उदाहरणात, Printable इंटरफेसमध्ये **फक्त एकच मेथड** आहे, आणि त्याची अंमलबजावणी InterfaceExample क्लासमध्ये प्रदान केली आहे.

```
interface printable
{
void print();
}
class InterfaceExample implements printable
{
public void print( ){System.out.println("Hello");
}
public static void main(String args[])
{
InterfaceExample obj = new InterfaceExample();
obj.print( );
 }   }
```

Output:

Hello

- **जावा इंटरफेस उदाहरण: बँक**

```
interface Bank{
float rateOfInterest();
}
class SBI implements Bank{
public float rateOfInterest(){return 9.15f;}
}
class PNB implements Bank{
public float rateOfInterest(){return 9.7f;}
    }
    class TestInterface2{
    public static void main(String [] args){
    Bank b=new SBI();
    Bank b1= new PNB();
    System.out.println("ROI: "+b.rateOfInterest());
    System.out.println("ROI: "+b1.rateOfInterest());
    } }
```

Output:

ROI: 9.15

ROI: 9.7

2.9 इंटरफेसमधील व्हेरिएबल्स आणि मेथड्स ॲक्सेस करणे (Accessing Variables and Methods in Interface)

जावामध्ये, इंटरफेस व्हेरिएबल हे आपोआप public, static, आणि final असते. याचा अर्थ एकदा मूल्य असाइन केल्यावर ते बदलता येत नाही. तसेच, इंटरफेस व्हेरिएबल्स सर्व इम्प्लिमेंटिंग क्लासेससाठी ॲक्सेसिबल असतात, ज्यामुळे कोड पुनर्वापर (code reusability) आणि प्रमाणीकरण (standardization) सोपे होते.

खालील उदाहरणातून इंटरफेस व्हेरिएबल्स अधिक स्पष्ट होईल

```
public interface Shape
 {
   int DEFAULT_SIZE = 10;
   void draw();
  }
```

या उदाहरणात, Shape इंटरफेसमध्ये DEFAULT_SIZE नावाचा इंटरफेस व्हेरिएबल परिभाषित आहे, ज्याला मूल्य 10 असाइन केले आहे. इम्प्लिमेंटिंग क्लासेस हे व्हेरिएबल वापरून वेगवेगळ्या आकारांसाठी डिफॉल्ट साइज प्रदान करू शकतात.

खालील उदाहरण विचारात घ्या.

```
public interface Constants {
   String DATABASE_URL = "jdbc:mysql://localhost:3306/mydatabase";
   String USERNAME = "root";
   String PASSWORD = "password123";
}
public class DatabaseConnection {
   // Code for establishing a database connection using the constants
}
```

या उदाहरणात, कॉन्स्टंट्स इंटरफेस डेटाबेस URL, वापरकर्तानाव (username), आणि पासवर्डसाठी (password) व्हेरिएबल्स परिभाषित करते. हे कॉन्स्टंट्स वापरून, कोणताही क्लास जो डेटाबेस कनेक्शन स्थापन करू इच्छितो, तो त्यांचा संदर्भ घेऊ शकतो, ज्यामुळे सुसंगतता (consistency) आणि सुलभ देखभाल (easy maintenance) सुनिश्चित होते.

- इंटरफेसद्वारे जावामधील मल्टीपल इनहेरिटन्स (Multiple Inheritance in Java by Interface)

```
interface Printable
{
void print();
}
interface Showable{
void show();
}
class A7 implements Printable,Showable
{
public void print(){System.out.println("Hello");
}
public void show()
{       System.out.println("Welcome");
}
public static void main(String args[]){
A7 obj = new A7();
```

```
  obj.print();
  obj.show();  }  }
```

Output

Hello
Welcome

2.10 Extending Interface

इंटरफेसमध्ये व्हेरिएबल्स आणि मेथड्स असतात, जसे की क्लासमध्ये असतात, परंतु क्लासच्या विपरीत, इंटरफेसमधील सर्व मेथड्स डीफॉल्टने ॲब्स्ट्रॅक्ट (abstract) असतात.इंटरफेस इनहेरिटन्स (Interface Inheritance) मध्ये, इंटरफेस दुसऱ्या इंटरफेसला "extends" करतो, जसे की क्लास इंटरफेसला "implements" करतो

```
interface A {
  void funcA();
}
interface B extends A {
  void funcB();
}
class C implements B {
  public void funcA() {
    System.out.println("This is funcA");
  }
  public void funcB() {
    System.out.println("This is funcB");
  }
}
public class Demo {
  public static void main(String args[]) {

    C obj = new C();
    obj.funcA();
    obj.funcB();
  }
}
```

Output

This is funcA
This is funcB

2.11 पॅकेजची व्याख्या (Define Package)

जावा पॅकेज (Java Package) म्हणजे संबंधित क्लासेस आणि इंटरफेसचा समूह (Namespace) जो त्यांना एकत्र ठेवतो.हे नेमिंग कॉन्फ्लिक्ट्स (Name Conflicts) टाळण्यास मदत करते आणि बेहतर ॲक्सेस कंट्रोल (Access Control) प्रदान करते.जावा पॅकेज म्हणजे समान प्रकारच्या क्लासेस (Classes), इंटरफेस (Interfaces) आणि सब-पॅकेजेस (Sub-Packages) चा समूह आहे.

- **जावा पॅकेजचे फायदे (Advantage of Java Package)**
  1. जावा पॅकेज क्लासेस आणि इंटरफेस वर्गीकृत (Categorize) करण्यासाठी वापरले जाते, ज्यामुळे त्यांचे व्यवस्थापन सोपे होते.
  2. जावा पॅकेज ॲक्सेस संरक्षण (Access Protection) प्रदान करते.
  3. जावा पॅकेज नेमिंग कोलिजन (Naming Collision) टाळते.

2.12 पॅकेजचे प्रकार (Types of Packages)

जावामधील पॅकेज दोन प्रकारात विभागले जातात:

- बिल्ट-इन पॅकेज (Built-in Package) - जावामध्ये पूर्वनिर्धारित (Predefined) पॅकेज उपलब्ध आहेत, जसे की java.util, java.io, java.lang इत्यादी.
- युजर-डिफाइन्ड पॅकेज (User-Defined Package) – वापरकर्त्याने स्वतःच्या आवश्यकतेनुसार तयार केलेले पॅकेज

1. **बिल्ट-इन पॅकेज (Built-in Packages) (Java API Packages)**

जावा एपीआय (Java API) हे मूलभूत प्रोग्रामिंग कार्ये जसे की लूपिंग, GUI फॉर्म प्रदर्शित करणे इत्यादी करण्यासाठी मोठ्या प्रमाणात लायब्ररी रूटीनचा संग्रह आहे. जास्त वेळा, आपण Java API सह उपलब्ध असलेल्या पॅकेजेस वापरतो. Java API मध्ये, क्लासेस आणि इंटरफेसेस पॅकेजमध्ये गटबद्ध केलेले असतात. हे सर्व क्लासेस जावा प्रोग्रामिंग भाषेत लिहिलेले असून JVM वर चालतात.

जावा अनेक पूर्वनिर्धारित (Predefined) पॅकेज प्रदान करते, जसे की:

- **Java.lang**
- **Java.util**
- **Java.awt**
- **Java.io**
- **Java.net**

खालील आकृती वारंवार वापरल्या जाणाऱ्या API पॅकेजेस दर्शवते.

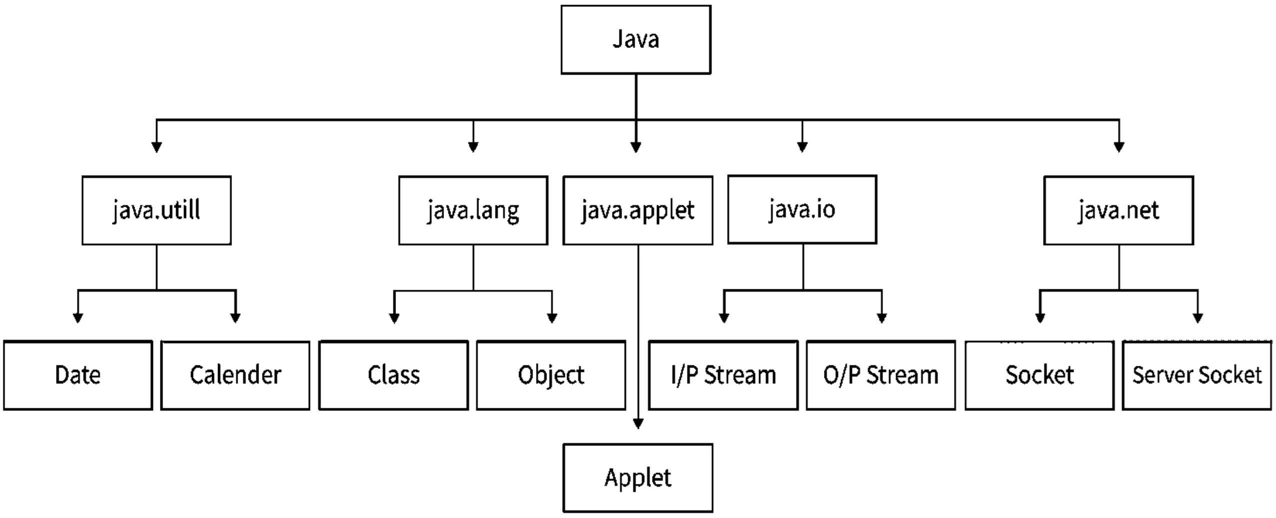


Fig 2.7: जावा पॅकेज श्रेणीक्रम (Java Package Hierarchy)

2. युझर डिफाइन पॅकेजेस (User-Defined Packages): ही डेव्हलपर्सद्वारे त्यांच्या क्लासेसचे व्यवस्थापन करण्यासाठी तयार केलेली पॅकेजेस आहेत.

2.13 पॅकेज नामकरण आणि निर्मिती (Naming and Creating Package)

- **पॅकेजसाठी नामकरण (Naming a Package)**
  1. नावाच्या गोंधळापासून बचाव करण्यासाठी लोअरकेस (Lowercase) अक्षरे वापरा.
  2. अद्वितीयता(uniqueness) राखण्यासाठी रिव्हर्स डोमेन नेम नोटेशन वापरा.
     उदाहरण: com.companyname.projectname.module

पॅकेज तयार करणे (Creating Package)

जावामध्ये पॅकेज तयार करण्यासाठी package कीवर्डचा वापर केला जातो.

सिंटॅक्स

```
package packagename; // पॅकेज डिक्लेअर करणे
```

उदाहरण

```
package mypackage; // Declaring package
public class MyClass {
    public void display() {
        System.out.println("Hello from MyClass in mypackage!");
    }
}
```

Compiling a Package

```
javac -d . MyClass.java
```

The -d . flag creates the mypackage directory and places MyClass.class inside it.

2.14 पॅकेज ॲक्सेस करणे (Accessing Package)

एखाद्या पॅकेजमधील क्लास वापरण्यासाठी आपण खालील दोन पद्धती वापरू शकतो:

- फुल्ली क्वालिफाइड नेम (Fully Qualified Name) वापरणे.
- इम्पोर्ट (import) स्टेटमेंट वापरणे.

मेथड 1: पूर्णपणे क्वालिफाइड (Fully Qualified) नाव वापरणे

```
mypackage.MyClass obj = new mypackage.MyClass();
obj.display();
```

ही मेथड वापरताना **import** स्टेटमेंटची आवश्यकता नसते.

मेथड 2: इम्पोर्ट (Import) स्टेटमेंट

पूर्णपणे क्वालिफाइड नाव वापरण्याऐवजी, **import** कीवर्डचा वापर करून कोड साधा पॅकेज ॲक्सेस करता येतो.

सिंटॅक्स

import package_name.ClassName; // विशिष्ट क्लास **इम्पोर्ट** करणे.

import package_name.*; // संपूर्ण पॅकेजमधील सर्व क्लास **इम्पोर्ट** करणे.

उदाहरण

```
import mypackage.MyClass;
public class Test {
    public static void main(String[] args) {
        MyClass obj = new MyClass();  // No need for fully qualified name
        obj.display();
    }
}
```

- **Compiling and Running**

```
javac -d. Test.java
java Test
```

- **स्टॅटिक इम्पोर्ट (static import)**

स्टॅटिक इम्पोर्ट वैशिष्ट्यामुळे, क्लासचे नाव न वापरता थेट स्टॅटिक मेथड्स आणि फील्ड्स ॲक्सेस करता येतात.

सिंटॅक्स

```
import static package_name.ClassName.staticMember;
```

उदाहरण

```
import static java.lang.Math.*;  // Importing all static members of Math class
public class TestStaticImport
{
    public static void main(String[] args) {
        System.out.println(sqrt(16));  // Using sqrt() directly
        System.out.println(PI);        // Accessing PI constant directly
```

```
  }
}
```

2.15 पॅकेजमध्ये क्लास आणि इंटरफेस जोडणे (Adding Class and Interfaces to a Package)

आधीच्या अस्तित्वात असलेल्या पॅकेजमध्ये अधिक क्लासेस किंवा इंटरफेस जोडण्यासाठी, नवीन फाईल्समध्ये त्याच पॅकेज नाव डिक्लेरेशन मध्ये समाविष्ट करावे.

उदाहरण

- **पॅकेज तयार करणे (Creating a Package)**

```
package mypackage;
public class MyClass
{
public void display ()
 { System.out.println("This is Myclass Method."); }}
```

- **त्याच पॅकेजमध्ये आणखी एक क्लास जोडणे**

```
package mypackage;
public class AnotherClass
 {
public void showMessage()
 {
System.out.println("This is another class Method.");
} }
```

- **पॅकेजमध्ये इंटरफेस जोडणे**

```
package mypackage;
public interface MyInterface
 {
 void myMethod();
}
```

- **इम्प्लिमेंटेशन करणारा क्लास**

```
package mypackage;
 public class InterfaceImpl implements MyInterface
{
public void myMethod()
{
System.out.println("This is implementing class .");}}
```

संदर्भ(Web References):

1. https://www.javatpoint.com/java-tutorial
2. https://www.w3schools.com/java/
3. https://www.tutorialspoint.com/java/java_inheritance.htm
4. https://www.programiz.com/java-programming/inheritance.

युनिट -3
एक्सेप्शन हँडलिंग आणि मल्टीथ्रेडिंग
(Exception Handling and Multithreading)

विषय निष्पत्ती: Course Outcome (CO)

CO 3 – त्रुटी हाताळणी आणि मल्टीथ्रेडिंग साठी प्रोग्राम बनविणे .

सिद्धांत शिक्षण परिणाम: थेअरी लर्निंग आऊटकम्स: (Theory Learning Outcomes)

1. त्रुटी व त्रुटी हाताळणी यांच्यातील फरक उदाहरणा सह दाखवा
2. सांगितलेल्या त्रुटी उदाहरण सह हाताळा.
3. थ्रेड चा वापर करून मल्टीपल प्रोसेस रन करा .
4. वेगवेगळ्या थ्रेड लाइफ सायकलच्या मेथड वापरून प्रोग्राम तयार करा .

3.1 एरर आणि एक्सेप्शन (Errors & Exception)

एरर आणि एक्सेप्शन हे दोनीही थ्रोवेबल क्लास चे सबक्लास आहेत. एरर प्रोग्रॅम मध्ये असणाऱ्या चुका असतात ज्या मुख्यतः सिस्टम संसाधनांच्या कमतरतेमुळे उद्भवते आणि बनवलेल्या प्रोग्राम या प्रकारच्या समस्या पकड मध्ये येत नाही . एक्सेप्शन हे रनटाइम आणि कंपाइल टाइममध्ये उद्भवू शकणाऱ्या समस्या आहेत. हे प्रामुख्याने डेव्हलपर लिहिलेल्या कोडमध्ये आढळते. जावा व्हर्च्युअल मशीन (JVM) मेमरी संपणे, मेमरी लीक होणे, स्टॅक ओव्हरफ्लो एरर, लायब्ररी विसंगतता, अनंत पुनरावृत्ती इ. यासारख्या अपरिवर्तनीय परिस्थितीचे एरर प्रतिनिधित्व करतात. त्रुटी सहसा प्रोग्रामरच्या नियंत्रणाबाहेर असतात.

3.1.2 एरर चे प्रकार (Types Of Errors)

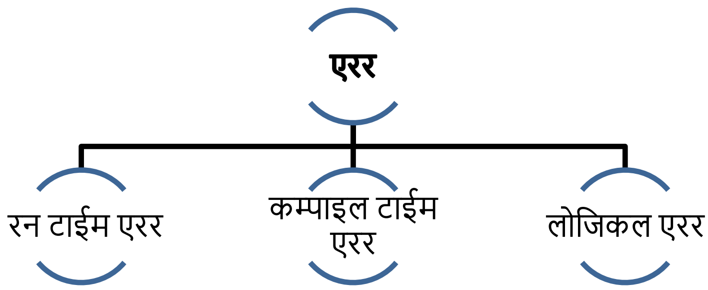


Fig 3.1 : एरर चे प्रकार

- **एरर तीन श्रेणींमध्ये विभागले गेले आहेत.**

1. रन टाईम एरर(Runtime Error)

प्रोग्रामच्या अंमलबजावणी दरम्यान आढळून येतात ज्या त्रुटी उद्भवतात त्यांना रन टाइम त्रुटी म्हणतात. जेव्हा वापरकर्ता अवैध डेटा किंवा डेटा प्रविष्ट करतो जो संबंधित नाही हे शोधले जाते. जेव्हा प्रोग्राममध्ये कोणत्याही सिंटॅक्स त्रुटी नसतात तेव्हा रनटाइम त्रुटी उद्भवतात परंतु संगणकाला असे काहीतरी करण्यास सांगते जे संगणक विश्वसनीयपणे करू शकत नाही.

उदाहरण: संगणक पूर्णांकाची अपेक्षा करत असताना वापरकर्त्याने स्ट्रिंग स्वरूपाचा डेटा इनपुट केल्यास, रनटाइम त्रुटी असेल.

2. कम्पाइल टाईम एरर(Compile time Error)

कंपाइल टाईम एरर्स या त्रुटी आहेत ज्या चुकीच्या वाक्यरचनामुळे कोडला चालण्यापासून रोखतात जसे की विधानाच्या शेवटी अर्धविराम नसणे किंवा कंस, वर्ग सापडला नाही इ. या त्रुटी जावा कंपाइलरद्वारे शोधल्या जातात आणि संकलित करताना स्क्रीनवर एक त्रुटी संदेश प्रदर्शित होतो.कंपाइल टाइम एरर्सना काहीवेळा सिंटॅक्स एरर असेही संबोधले जाते. या प्रकारच्या त्रुटी शोधणे आणि सुधारणे सोपे आहे कारण जावा कंपाइलर आपल्यासाठी त्या शोधतो.

उदाहरण : चुकीचे स्पेलिंग व्हेरिएबल नाव किंवा पद्धतीची नावे.

- **लोजिकल एरर(Logical Error)**

लॉजिक एरर म्हणजे जेव्हा तुमचा प्रोग्राम संकलित करतो आणि कार्यान्वित करतो, परंतु चुकीची गोष्ट करतो किंवा चुकीचा परिणाम देतो किंवा आउटपुट परत करत असताना आउटपुट मिळत नाही. या त्रुटी कंपाइलर किंवा JVM द्वारे शोधल्या जात नाहीत. Java सिस्टमला तुमचा प्रोग्राम काय करायचा आहे याची कल्पना नाही, त्यामुळे तुम्हाला त्रुटी शोधण्यात मदत करण्यासाठी ती कोणतीही अतिरिक्त माहिती पुरवत नाही.तार्किक त्रुटींना सिमेंटिक एरर्स देखील म्हणतात. कोडिंग करताना प्रोग्रामरने वापरलेल्या चुकीच्या कल्पना किंवा संकल्पनेमुळे या त्रुटी उद्भवतात.

उदाहरण: ऑपरेशन करण्यासाठी चुकून व्हेरिएबल्सवर चुकीचा ऑपरेटर वापरणे ('%' वापरण्याऐवजी मॉड्यूलस मिळविण्यासाठी '/' ऑपरेटर वापरणे).

3.2 एक्सेप्शन (Exceptions)

जावा मधील एक्सेप्शन हँडलिंग हे रनटाइम त्रुटी हाताळण्याचे एक प्रभावी माध्यम आहे जेणेकरून अनुप्रयोगाचा नियमित प्रवाह जतन केला जाऊ शकतो. जावा एक्सेप्शन हँडलिंग ही रनटाइम एरर हाताळण्याची एक यंत्रणा आहे जसे की ClassNotFoundException, IOException, SQLException, RemoteException इ.

ही एक अनपेक्षित घटना आहे, जी प्रोग्रामच्या अंमलबजावणीदरम्यान घडते, म्हणजे रन टाइममध्ये, जी प्रोग्रामच्या सूचनांच्या सामान्य प्रवाहात व्यत्यय आणते. कार्यक्रमाद्वारे एक्सेप्शन पकडले आणि हाताळले जाऊ शकतात. जेव्हा एखाद्या पद्धतीमध्ये एक्सेप्शन येतो, तेव्हा ते ऑब्जेक्ट तयार करते. या ऑब्जेक्टला एक्सेप्शन ऑब्जेक्ट असे म्हणतात. यात एक्सेप्शन बद्दल माहिती असते, जसे की एस्केपशनचे नाव आणि वर्णन आणि एक्सेप्शन आला तेव्हा प्रोग्रामची स्थिती.

3.2.1 एक्सेप्शन का होतो याची प्रमुख कारणे

- अवैध वापरकर्ता इनपुट
- डिव्हाइस शोधण्यास अयशस्वी
- नेटवर्क कनेक्शन गमावले
- डिस्क मेमरी मर्यादा
- कोड एरर
- अनुपलब्ध फाइल उघडत आहे.

3.2.2 एक्सेप्शन दोन श्रेणींमध्ये विभागले गेले आहेत

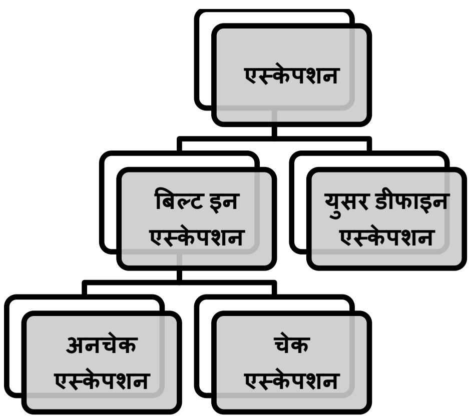


Fig 3.2 : एक्सेप्शन श्रेणी

- **बिल्ट इन एक्सेप्शन (Built-in Exceptions):** बिल्ट इन एक्सेप्शन हे असे एक्सेप्शन आहेत जे जावा लायब्ररीमध्ये उपलब्ध आहेत. हे एक्सेप्शन विशिष्ट त्रुटी परिस्थिती स्पष्ट करण्यासाठी योग्य आहेत.

- **चेक एक्सेप्शन (Checked Exception)**

तपासलेल्या एक्सेप्शन कंपाइल-टाइम एक्सेप्शन म्हणतात कारण हे एक्सेप्शन कंपाइलरद्वारे कंपाइल-टाइमवर तपासले जातात.

- **अनचेक एक्सेप्शन (Unchecked Exception)**

अनचेक केलेले एक्सेप्शन चेक केलेल्या एस्केपशनच्या अगदी विरुद्ध आहेत. कंपाइलर संकलित वेळी हे एक्सेप्शन तपासणार नाही. सोप्या शब्दात सांगायचे तर, जर एखादा प्रोग्राम अनचेक केलेला एक्सेप्शन टाकतो आणि जरी ते हाताळले नाही किंवा घोषित केले नाही, तर प्रोग्राम संकलन त्रुटी देणार नाही.

- **युसर डीफाइन एक्सेप्शन (User-Defined Exceptions)**

काहीवेळा, Java मधील अंगभूत अपवाद विशिष्ट परिस्थितीचे वर्णन करण्यास सक्षम नसतात. अशा परिस्थितीत, वापरकर्ते एक्सेप्शन देखील तयार करू शकतात, ज्यांना 'वापरकर्ता-परिभाषित एक्सेप्शन म्हणतात.

3.2.4 जावा मधील एक्सेप्शन हाताळणीचे फायदे खालीलप्रमाणे आहेत:

- कार्यक्रमाची अंमलबजावणी पूर्ण करण्यासाठी तरतूद
- प्रोग्राम कोड आणि एरर-हँडलिंग कोडची सहज ओळख
- एरर प्रसार
- अर्थपूर्ण एरर अहवाल
- एरर प्रकार ओळखणे

3.3.5 ट्राय अंड कॅच स्टेटमेन्ट (Try & Catch Statement)

ट्राय स्टेटमेंट तुम्हाला कोडच्या ब्लॉकची अंमलबजावणी करताना त्रुटींसाठी चाचणी करण्यासाठी परिभाषित करण्याची परवानगी देते.ट्राय ब्लॉकचा वापर कोड चेक करण्यासाठी केला जातो जो एक्सेप्शन देऊ शकतो. असा कोड ट्राय ब्लॉक मध्ये वापरला जाणे आवश्यक आहे. ट्राय ब्लॉकमधील विशिष्ट विधानात एक्सेप्शन आढळल्यास, उर्वरित ब्लॉक कोड कार्यान्वित होणार नाही. म्हणून इतर, कोड ट्राय ब्लॉकमध्ये न ठेवण्याची शिफारस केली जाते ज्यामुळे एक्सेप्शन होणार नाही. ट्राय ब्लॉकमध्ये त्रुटी आढळल्यास, कॅच स्टेटमेंट तुम्हाला अंमलात आणण्यासाठी कोडचा ब्लॉक परिभाषित करण्यास अनुमती देतो कॅच ब्लॉकचा वापर पॅरामीटरमध्ये एस्केपशनचा प्रकार घोषित करून एक्सेप्शन हाताळण्यासाठी केला जातो. घोषित एक्सेप्शन हा पॅरेंट क्लास एक्सेप्शन असला पाहिजे किंवा जनरेट केलेला एक्सेप्शन असणे आवश्यक आहे. जनरेट केलेला एक्सेप्शन घोषित करणे हा चांगला दृष्टिकोन आहे.
उदा: आपण ट्राय-कॅच ब्लॉक चा वापर न केल्यास आपणास येणाऱ्या समस्या समजून घेण्याचा प्रयत्न करूया.

```
public class TryCatchExample1
{
public static void main(String[] args)
{
int data=50/0; //may throw exception
System.out.println("rest of the code");
}
}
```

Output:

Exception in thread "main" java.lang.ArithmeticException: / by zero

एक्सेप्शन हंड्लिंग :

```
public class TryCatchExample2 {
    public static void main(String[] args) {
      try
      {
      int data=50/0; //may throw exception
      }
         //handling the exception
      catch(ArithmeticException e)
```

```
        {
          System.out.println(e);
        }
        System.out.println("rest of the code");
    }
}
```

Output:

java.lang.ArithmeticException: / by zero
rest of the code

3.3 नेस्टेड ट्राय स्टेटमेंट (Nested Try Statement)

दुसऱ्या ट्राय ब्लॉकमध्ये ट्राय ब्लॉक वापरण्याची परवानगी आहे. त्याला नेस्टेड ट्राय ब्लॉक असे म्हणतात. प्रत्येक स्टेटमेंट जे आपण ट्राय ब्लॉकमध्ये स्टेटमेंट टाकतो, त्या एक्सेप्शन संदर्भ स्टॅकवर ढकलला जातो.

- **Throw(थ्रो)-** हे प्रामुख्याने युसर डिफाईन एक्सेप्शन थ्रो करण्या करिता वापर केला जातो.

उदा: throw new myException(“Invalid number”);
गृहीत धरून myException जस कि हे युसर डिफाईन एस्केपशन

- **Throws(थ्रोस)-**

हे पद्धतीच्या घोषणेसह वापरले जाऊ शकते ज्यामध्ये काही रनटाइम त्रुटी असू शकतात.
उदा : public static void main(String args[]) throws IOException

- **Finally Statement (फायनली स्टेटमेंट):**

हे एक्सेप्शन हाताळण्यासाठी वापरले जाऊ शकते जे मागील कोणत्याही कॅच स्टेटमेंटद्वारे पकडले जात नाही. ट्राय ब्लॉकद्वारे व्युत्पन्न केलेले कोणतेही विधान हाताळण्यासाठी फायनली ब्लॉक वापरले जाऊ शकते. हे प्रयत्न केल्यानंतर किंवा शेवटच्या कॅच ब्लॉकनंतर लगेच जोडले जाऊ शकते.

मांडणी:

```
finally
{
// block of code to be executed before try block ends
}
```

उदा.

```
finally{
System.out.println("finally block is always executed");
}
```

- **बिल्ट इन एस्केपशन (Build-In Exceptions)**

बिल्ट इन एक्सेप्शन हे एक्सेप्शन आहेत जे Java लायब्ररीमध्ये उपलब्ध आहेत. हे एक्सेप्शन विशिष्ट त्रुटी परिस्थिती स्पष्ट करण्यासाठी योग्य आहेत.

3.4 काही महत्त्वाच्या बिल्ट इन एक्सेप्शन ची यादी खाली दिली आहे.

अंकगणित एक्सेप्शन (Arithmetic exception) : जेव्हा अंकगणित ऑपरेशनमध्ये अपवादात्मक स्थिती उद्भवते तेव्हा ते थ्रो जाते.

- **ArrayIndexOutOfBounds Exception:** बेकायदेशीर निर्देशांकासह अॅरेमध्ये प्रवेश केला गेला आहे हे सूचित करण्यासाठी ते थ्रो केले जाते. निर्देशांक एकतर ऋणात्मक आहे किंवा अॅरेच्या आकारापेक्षा जास्त आहे.
- **ClassNotFoundException :** ज्या वर्गाची व्याख्या सापडत नाही अशा वर्गात प्रवेश करण्याचा प्रयत्न करताना हा एक्सेप्शन वापरला जातो.
- **FileNotFoundException :** जेव्हा एखादी फाइल प्रवेशयोग्य नसते किंवा उघडत नसते तेव्हा हे एक्सेप्शन वापरले जातो.

उदा .

// Java program to demonstrate
// ArithmeticException

```
class ArithmeticException_Demo {
public static void main(String args[])
{
try {
int a = 30, b = 0;
int c = a / b; // cannot divide by zero
System.out.println("Result = " + c);
}
catch (ArithmeticException e) {
System.out.println("Can't divide a number by 0");
}
}
}
```

Chained Exceptions(चेन एस्केपशन):

शृंखलाबद्ध (साखळी) एक्सेप्शन एका एस्केपशनलादुसऱ्या एस्केपशनशी जोडण्याची परवानगी देतात, म्हणजे एक एक्सेप्शन दुसऱ्या एस्केपशनच्या कारणाचे वर्णन करतो.

उदाहरणार्थ, शून्याने विभाजित करण्याच्या प्रयत्नामुळे एक पद्धत अंकगणित एक्सेप्शन थ्रो करते अशा परिस्थितीचा विचार करा परंतु एस्केपशनचे वास्तविक कारण I/O त्रुटी होती ज्यामुळे विभाजक शून्य झाला.

// Java program to demonstrate working of chained exceptions

```
public class ExceptionHandling
{
public static void main(String[] args)
{
try{
// Creating an exception
NumberFormatException ex =new NumberFormatException("Exception");
// Setting a cause of the exception
ex.initCause(new NullPointerException("This is actual cause of the exception"));// Throwing an exception
with cause.
throw ex;
}
catch(NumberFormatException ex)
{
// displaying the exception
System.out.println(ex);
// Getting the actual cause of the exception
System.out.println(ex.getCause());
}}}
```

Output:

java.lang.NumberFormatException: Exception
java.lang.NullPointerException: This is actual cause of the exception

स्वतःचे एक्सेप्शन करणे (Throw Clause)(Creating Own Exception):
जावा मधील थ्रो कीवर्ड कोडच्या कोणत्याही ब्लॉकमधून एक्सेप्शन टाकण्यासाठी वापरला जातो. एकतर चेक केलेला किंवा अनचेक केलेला एक्सेप्शन टाकू शकतो. थ्रो कीवर्ड प्रामुख्याने सानुकूल एक्सेप्शन टाकण्यासाठी वापरला जातो.

- **थ्रोस (Throws):**

थ्रो हा जावा मधील एक कीवर्ड आहे जो मेथडच्या स्वाक्षरीमध्ये वापरला जातो हे दर्शविण्यासाठी की ही पद्धत सूचीबद्ध प्रकारच्या अपवादांपैकी एक टाकू शकते. या पद्धतींचा कॉल करणाऱ्याला ट्राय-कॅच ब्लॉक वापरून अपवाद हाताळावा लागतो.
type method_name(parameters) throws exception_list

दोन प्रकारे हाताळू शकतो

१. ट्राय कॅच वापरून(Using try catch)
२. थ्रो कीवर्ड वापरुन(using throw keyword)

मल्टीथ्रेडेड प्रोग्रामिंग थ्रेड तयार करणे (Multithreaded Programming Creating A Thread):
(**मल्टीथ्रेडेड)Multithreaded:** हे एकाच वेळी अनेक कामे हाताळू शकते. मल्टीथ्रेडिंगच्या वैशिष्ट्यासह जावा हे शक्य करते. याचा अर्थ असा आहे की दुसरे काम सुरू करण्यापूर्वी एक कार्य पूर्ण करण्यासाठी पूर्ण होण्याची प्रतीक्षा करण्याची गरज नाही.
उदा:
throw Instance
Example:
throw new ArithmeticException("/ by zero");

प्रोग्राम शीर्षक :- मल्टीथ्रेडिंग आणि सिंक्रोनाइझेशनवर आधारित प्रोग्राम्सची अंमलबजावणी.

```
import java.util.*;
import java.lang.*;
class prime extends Thread
{
static int count;
public void run()
{
try
{
System.out.println("Prime no. from 1 to 20 are:");
for(int i=1;i<=i;j++)
{
if(i%j==0) count++;
}
if(count==1||count==2)
{
Thread.sleep(1000);
System.out.println(i);
}
}
} catch(InterruptedException e) { }
}
}
class prime1 extends Thread
{
```

```
static int count;
public void run()
{
try
{
for(int i=11;i<=i;j++)
{ if(i%j==0) count++;
 }
if(count==1||count==2)
{ Thread.sleep(1000);
System.out.println(i);
}
}
} catch(InterruptedException e) { }
}
}
class Pract13
{
public static void main(String args[])
{
new prime().start();
new prime1().start();
}
}
```

Output:- C:\Program Files\Java\jdk1.7.0\bin>java Pract13 Prime no. from 1 to 20 are:

1
11
2
13
3
17
5
19
 7

2) import java.util.*;

```
import java.lang.*;
class SyncObj
{ void disp(String msg)
{
System.out.println("HELLO"+msg);
try
{
Thread.sleep(500);
} catch(InterruptedException e) {}
```

```
System.out.println("Exit from SyncObj");
}
}
class SyncTest implements Runnable
{
String z =" ";
Thread t;
SyncObj s;
SyncTest(SyncObj a, String b)
{
s=a; z=b;
t=new Thread(this);
t.start();
}
public void run()
{
synchronized(s)
{
s.disp(z);
 }
 }
 }
class Pract13_2
{
public static void main(String args[])
{
SyncObj s = new SyncObj();
SyncTest s1=new SyncTest(s, "Nitin");
SyncTest s2=new SyncTest(s, "Ajinkya");
}
}
```

Output:- C:\Program Files\Java\jdk1.7.0\bin>java Pract13_2

HELLONitin

Exit from SyncObj

HELLOAjinkya

Exit from SyncObj

जावा थ्रेड्सचा परिचय(Introduction to Java Threads) :

1. थ्रेड एक मूलभूत प्रक्रिया युनिट आहे ज्यासाठी OS प्रोसेसर वेळ वाटप करू शकते.
2. थ्रेड ही एक लाईट वेट प्रोसेस आहे .
3. प्रोग्रामला एक पेक्षा जास्त थ्रेड असु शकतात .
4. थ्रेड हा काही रिसोर्सेस वापरतो म्हणून गरजेपेक्षा जास्त थ्रेड वापरू नये.
5. Java थ्रेड्स सामान्य मेमरी क्षेत्र सामायिक करतात म्हणून प्रत्येक थ्रेडसाठी मेमरी वाटप आवश्यक नसते. त्यामुळे कन्टेकस स्विचिंगला मल्टी-प्रोसेसिंग वापरण्यापेक्षा कमी वेळ लागतो.
6. प्रत्येक जावा प्रोग्राममध्ये किमान एक थ्रेड असतो म्हणजेच जावा प्रोग्राम कार्यान्वित करणारा थ्रेड.

7. थ्रेड्स वेगवान आणि अधिक कार्यक्षम प्रोग्राम आहेत जे अंमलबजावणीची गती वाढवतात

8. एका वेळी फक्त एक थ्रेड कार्यान्वित केला जातो.

थ्रेड्स तयार करण्याचे दोन मार्ग (Two ways to create a thread)

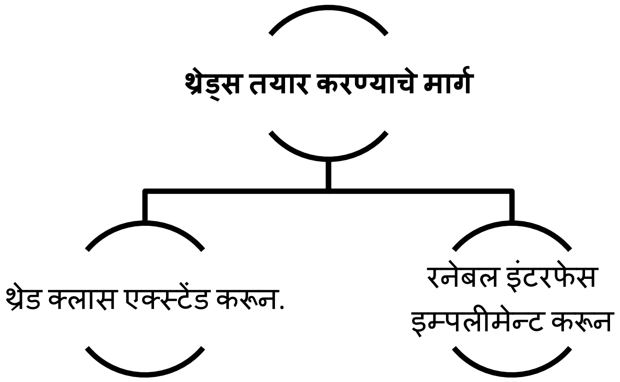


Fig 3.3 : थ्रेड्स तयार करण्याचे दोन मार्ग

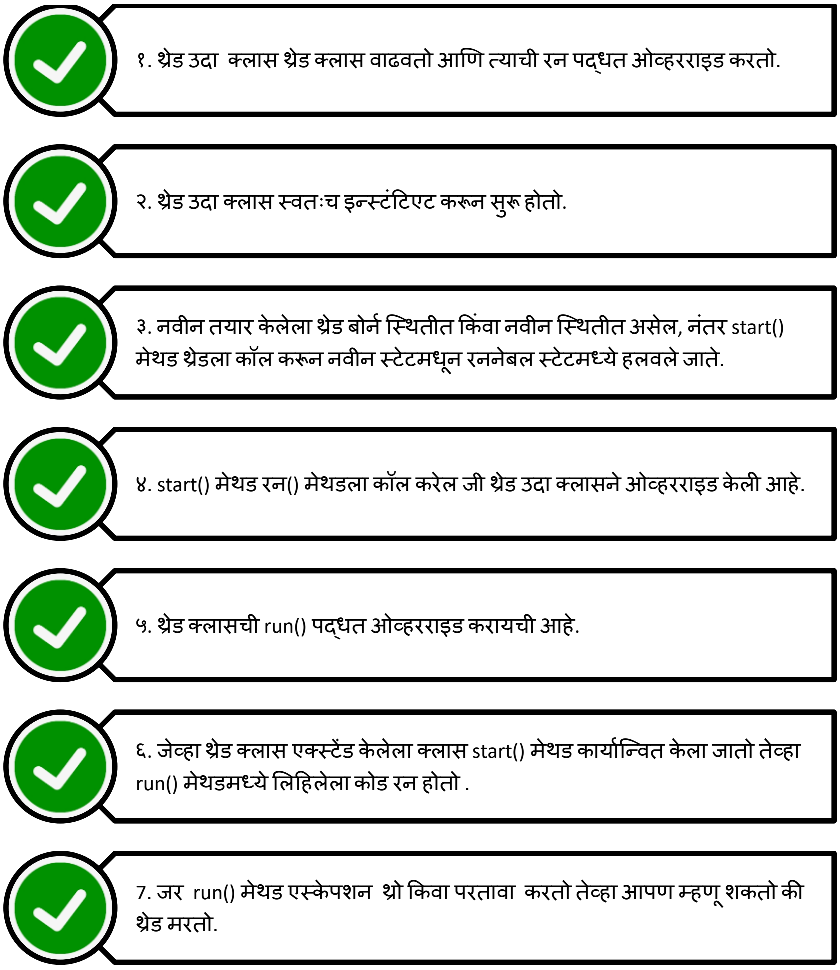


Fig 3.4: स्टेप्स ऑफ रनिंग थ्रेड

काही निष्कर्ष: आता आपण वर लिहिलेल्या प्रोग्राममधून काही निष्कर्ष काढू शकतो

रनेबल इंटरफेस इम्पलीमेन्ट करणे (Implementing Runnable Interface)

खालील उदाहरणाचा विचार करा

प्रथम खालील उदाहरणाचा विचार करा –

```
public class ThreadExample extends Thread
{
```

```
public void run()
{
for (int i = 1; i <= 10; i++)
{
System.out.println(i); }}
 public static void main(String[] args)
{
ThreadExample t1 = new ThreadExample();
t1.start(); }}
```

output : Thread is running !!

उदाहरण

```
public class RunnableThread implements Runnable
{
public void run()
{
for (int i = 1; i <= 10; i++)
{ System.out.println(i);
}
}
public static void main(String[] args)
{
RunnableThread t1 = new RunnableThread();
Thread thread = new Thread(t1); thread.start();
}
}
```

output :

```
1
2
3
4
5
6
7
8
9
10
```

स्पष्टीकरण : वरील प्रोग्राममध्ये, प्रथम वर्गाला रन करण्यायोग्य इंटरफेस लागू करणे आवश्यक आहे

```
public class RunnableThread implements Runnable {}
```

Runnable interface have run() method which we need to implement.

```
public void run()
{
for (int i = 1; i <= 10; i++)
{
System.out.println(i);
```

```
 }
 }
```

आता आपल्याला क्लासचा एक ऑब्जेक्ट तयार करावा लागेल जो रननेबल इंटरफेस लागू करेल.

RunnableThread t1 = new RunnableThread();

जर आपण Runnable चा वापर केला तर आपल्याला थ्रेड क्लास इन्स्टंशिएट करावा लागेल आणि त्यावर Runnable पास करावा लागेल.

Thread thread = new Thread(t1);

थ्रेड ऑब्जेक्ट तयार केल्यानंतर रन() मेथड कार्यान्वित करण्यासाठी आपण start() मेथडचा वापर करू शकतो.sleep()पद्धतीचा वापर निर्दिष्ट वेळेसाठी थ्रेड स्लीप करण्यासाठी वापरला जातो.

InterruptedException एक्सेप्शन थ्रो केले जाते , त्याची प्यारामीटर मिलीसेकंदात असते .

थ्रेड क्लास दोन पद्धती प्रदान करतो –

```
public static void sleep(long miliseconds)
public static void sleep(long miliseconds, int nanos)
class SleepExample extends Thread {
   public void run() {
      for (int i = 1; i <= 5; i++) {
         System.out.println("थ्रेड चालू आहे: " + i);
         try {
            Thread.sleep(1000); // 1 सेकंद थांबा
         } catch (InterruptedException e) {
            System.out.println("थ्रेड इंटरप्ट झाला: " + e);
         }}}}
public class SleepDemo {
   public static void main(String[] args) {
      SleepExample t1 = new SleepExample();
      t1.start();}}
```

हे थ्रेड प्रत्येक इटरेशननंतर (प्रत्येक फेरीत) 1 सेकंदासाठी थांबेल.

- **थ्रेड ची लाइफ सायकल(Life Cycle Of Thread):**

  जावा मध्ये थ्रेड लाइफसायकल हा एक महत्त्वाचा घटक आहे. प्रत्येक थ्रेड वेगवेगळ्या स्थितींमधून (States) जातो. जावा मध्ये Thread State Enum (java.lang.Thread.State) वापरून थ्रेडच्या स्थितीबद्दल माहिती मिळवू शकतो.

थ्रेड (Thread) च्या वेगवेगळ्या अवस्थांना (States)

1. नवीन (New)
2. चालू होण्यास सज्ज (Runnable)
3. चालू (Running)
4. ब्लॉक केलेले (Blocked)
5. प्रतीक्षा (Waiting) किंवा वेळेची प्रतीक्षा (Timed Waiting)
6. समाप्त (Terminated/Dead)

- जावा मध्ये थ्रेड हे वेगवेगळया स्टेट्समधून जातात.
- थ्रेड स्टेट्स getState() वापरून तपासता येतात.
- सिंक्रोनायझेशन आणि wait(), sleep(), join() यांचा वापर थ्रेड स्टेट्स बदलण्यासाठी होतो.

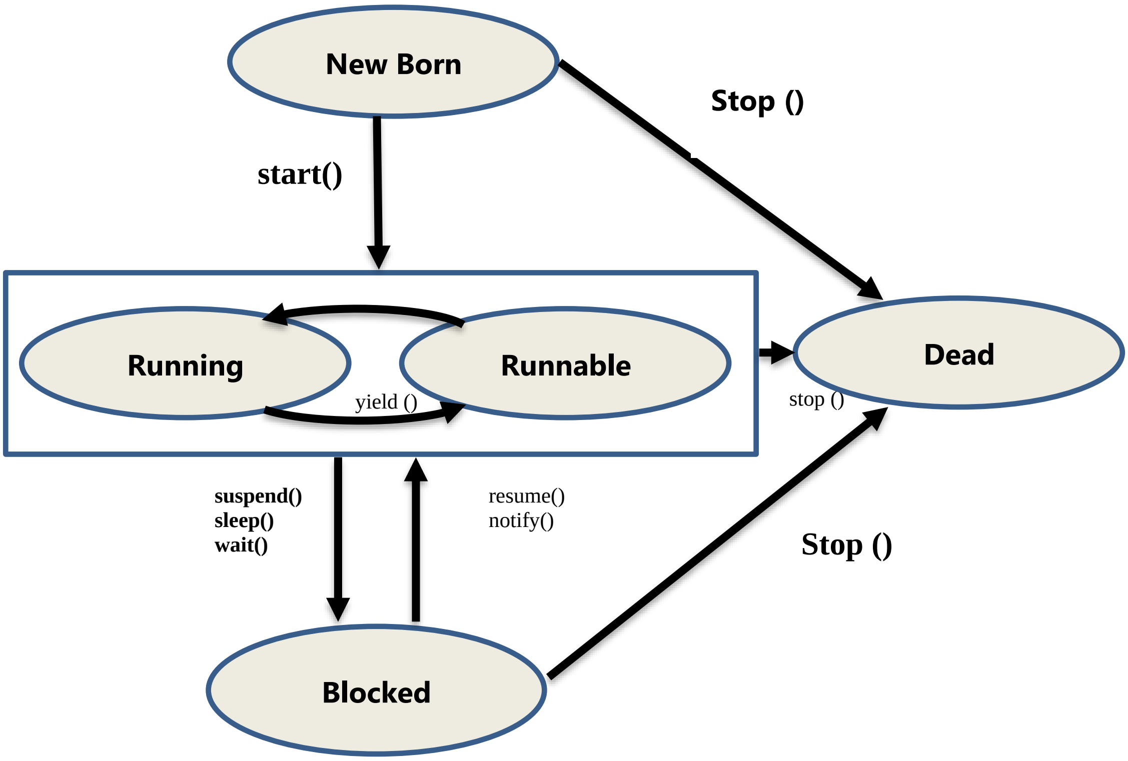


Fig 3.5: थ्रेड ची लाइफ सायकल

थ्रेड च्य मेथड्स (Thread Methods)

1. **वेट (Wait):** wait() मेथड कॉल केल्यावर, कॉलिंग थ्रेड इतर थ्रेडद्वारे notify() किंवा notifyAll() पद्धत सुरू करेपर्यंत त्याची अंमलबजावणी थांबवते.
2. **नोटीफाई Notify ():** एखाद्या वस्तूची वाट पाहत असलेला एकच थ्रेड जागृत करण्यासाठी त्याचा वापर केला जातो आणि तो थ्रेड नंतर कार्यान्वित करण्यास सुरुवात करतो.
3. **रीसियुम Resume (): syntax** : public void resume() ही पद्धत suspend() पद्धत वापरून निलंबित केलेला थ्रेड पुन्हा सुरू करते.
4. **सस्पेन्ड Suspend():** ही पद्धत थ्रेडला निलंबित स्थितीत ठेवते आणि resume() पद्धत वापरून पुन्हा सुरू करता येते.

   syntax : public void suspend()
5. **स्टोप Stop():**थ्रेड अचानक थांबवते.
6. **run ( )** – थ्रेडमध्ये चालवायचा कोड. run() ला थेट कॉल केल्यास नवीन थ्रेड तयार होत नाही, त्यामुळे start() वापरणे आवश्यक आहे.

run ( ) मेथड चे उदाहरण

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread is Running!");
   }
}
```

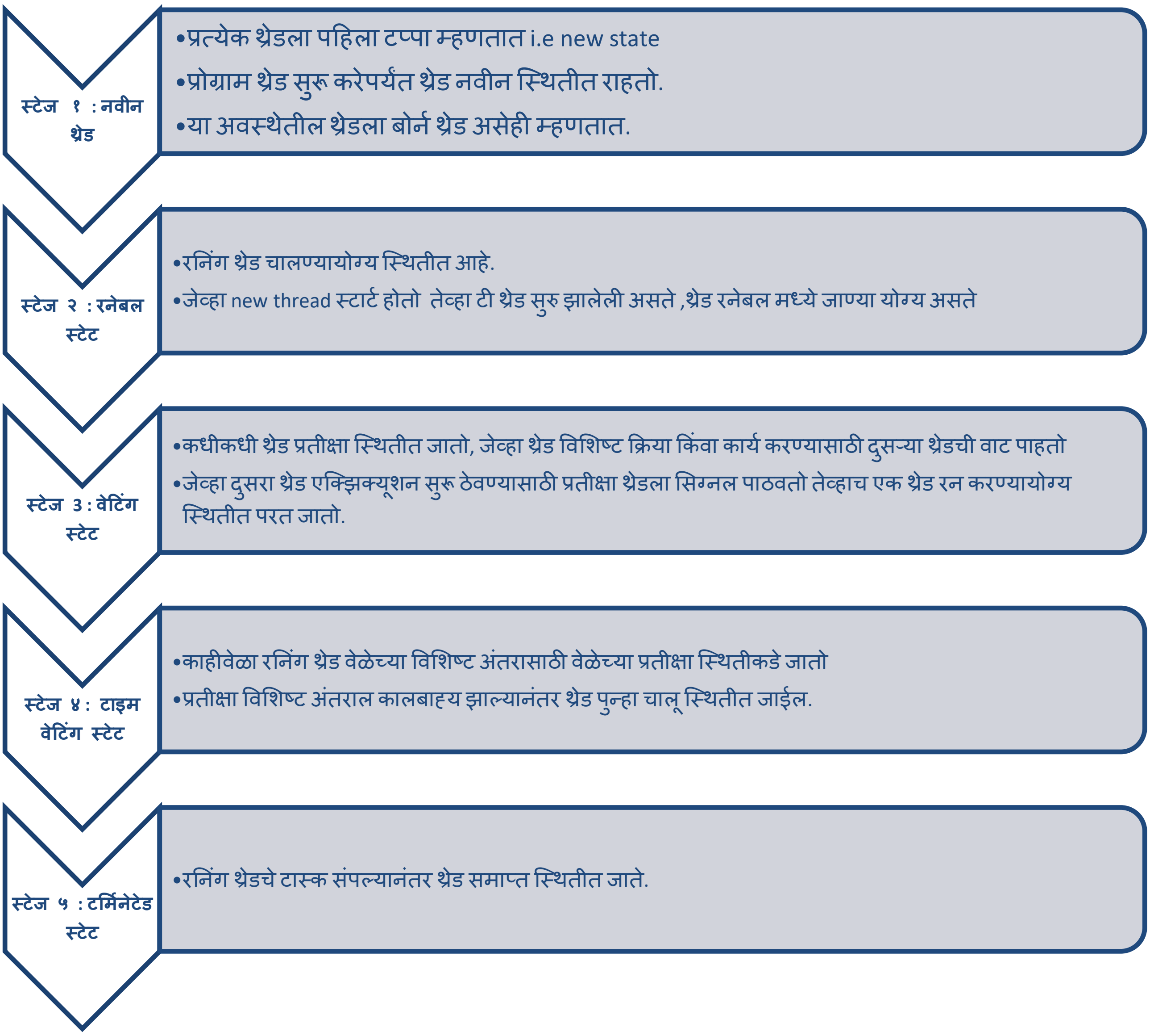


Fig 3.6: थ्रेड स्टेज

थ्रेड प्रायोरिटी (Thread Priority):

1. तार्किकदृष्ट्या आपण असे म्हणू शकतो की थ्रेड एकाच वेळी चालतात परंतु व्यावहारिकदृष्ट्या ते खरे नाही, एका वेळी फक्त एकच थ्रेड अशा प्रकारे चालू शकतो की वापरकर्त्याला ते समवर्ती वातावरण वाटेल.
2. फिक्स्ड प्रायॉरिटी शेड्युलिंग अल्गोरिदमचा वापर प्राधान्याने एक थ्रेड निवडण्यासाठी केला जातो.
3. आपण setPriority() पद्धत वापरून थ्रेड प्रायोरिटी बदलू शकतो.
4. थ्रेडला 1 ते 10 दरम्यान पूर्णांक प्राधान्य असू शकते
5. Java थ्रेड क्लास खालील कॉन्स्ट/टस चा वापर करतो –
6. एका वेळी अनेक थ्रेड अंमलबजावणीसाठी तयार असू शकतात परंतु अंमलबजावणीसाठी सर्वोच्च प्राधान्य असलेला धागा निवडला जातो
7. थ्रेडला 5 च्या बरोबरीचे डीफॉल्ट प्राधान्य असते.

थ्रेड प्रायॉरिटी (Thread Priority)	**कॉन्स्टन्ट (Constant)**
किमान प्रायॉरिटी/ MIN_PRIORITY	1
जास्तीत जास्त प्रायॉरिटी /MAX_PRIORITY	10
सामान्य प्रायॉरिटी /NORM_PRIORITY	5

इन्हेरीटन्स (Inheritance)-इन्हेरीटन्स ही एक प्रक्रिया आहे जिथे एक ऑब्जेक्ट दुसर्या ऑब्जेक्टचे गुणधर्म प्राप्त करतो.
सबक्लास(Subclass)-ज्या क्लासला दुसऱ्या वस्तूचे गुणधर्म मिळतात त्याला सबक्लास म्हणतात.
सुपरक्लास(Superclass)-ज्या क्लासचे गुणधर्म उपवर्गाद्वारे वारशाने मिळतात त्यांना सुपरक्लास म्हणतात.
किवर्ड (Keywords) एक्स्टेंडस (extends) and इम्प्लीमेंटस (implements)

IS-A Relationship उदाहरण :

IS-A म्हणण्याची एक पद्धत आहे: ही वस्तू त्या वस्तूचा एक प्रकार आहे.

```
public class Vehicle{ }
public class FourWheeler extends Vehicle{ }
public class TwoWheeler extends Vehicle{ }
public class WagonR extends FourWheeler{ }
```

1. वाहन हा दुचाकी वर्गाचा सुपरक्लास आहे Vehicle is the superclass of TwoWheeler class.
2. वाहन हा फोरव्हीलर वर्गाचा सुपरक्लास आहे Vehicle is the superclass of FourWheeler class.
3. दुचाकी आणि चारचाकी हे वाहन वर्गाचे उपवर्ग आहेत TwoWheeler and FourWheeler are sub classes of Vehicle class.
4. वॅगनआर हा चारचाकी आणि वाहन या दोन्ही वर्गांचा उपवर्ग आहे WagonR is the subclass of both FourWheeler and Vehicle classes.

IS-A relationship वरील उदाहरण आहे –

TwoWheeler IS-A Vehicle
FourWheeler IS-A Vehicle
WagonR IS-A FourWheeler
Hence WagonR IS-A Vehicle

सिंक्रोनाइझेशन (Synchronization):

जेव्हा दोन किंवा अधिक थ्रेड्सना सामायिक संसाधनामध्ये प्रवेश आवश्यक असतो, एका वेळी फक्त एकाच थ्रेडद्वारे संसाधन वापरले जाईल याची खात्री करण्यासाठी त्यांना काही मार्ग आवश्यक आहेत. ज्या प्रक्रियेद्वारे हे साध्य केले जाते त्याला सिंक्रोनाइझेशन म्हणतात.

सिंक्रोनाइझेशनचे फायदे:

1. **डेटा करप्ट होण्या पासून वाचवतो (prevent data corruption):** जेव्हा एकाधिक थ्रेड्स समान डेटावर कार्य करतात तेव्हा सिंक्रोनाइझेशन सुनिश्चित करते की डेटा सुसंगत राहतो आणि एकाचवेळी सुधारणांद्वारे दूषित होत नाही.
2. थ्रेड हस्तक्षेप रोखणे: सिंक्रोनाइझेशन हे सुनिश्चित करते की सामायिक संसाधनांमध्ये प्रवेश करणारे थ्रेड एकमेकांना हस्तक्षेप करणार नाहीत.
3. सुसंगतता राखून ठेवतात : जेव्हा एकापेक्षा अधिक थ्रेड्समध्ये समान ऑब्जेक्टमध्ये प्रवेश करण्याची आवश्यकता असते, तेव्हा सिंक्रोनाइझेशन ऑब्जेक्टची स्थिती सुसंगत राहण्याची खात्री देते .

सिंक्रोनाइझेशनवर आधारित प्रोग्राम :

```
class Callme
{
void call(String msg)
{
System.out.print("[" +msg);
try
{
Thread.sleep(1000);
 }
```

```
catch(InterruptedException e)
{
System.out.println("Interrupted ");
} System.out.print("]");
}
}
class Caller implements Runnable
{ String msg; Callme target;
Thread t;
public Caller(Callmetarg,String s)
{
target=targ;
msg=s;
t=new Thread(this);
t.start();
 }
public void run()
{
synchronized(target)
{
target.call(msg);
}
}
class Synch
{
public static void main(String args[])
{
Callme target=new Callme();
Caller ob1=new Caller(target,"Hello");
Caller ob2=new Caller(target,"Synchronized");
try
{
ob1.t.join();
ob2.t.join();
}
catch(InterruptedException e) { System.out.println("Interrupted ");
 } } }
```

- **इंटर थ्रेड कम्युनिकेशन (Inter-Thread Communication):**

जावा मधील इंटर-थ्रेड कम्युनिकेशन ही एक यंत्रणा आहे ज्यामध्ये थ्रेडला त्याच्या गंभीर विभागात विराम दिला जातो आणि त्याच क्रिटिकल विभागात दुसऱ्या थ्रेडला प्रवेश (किंवा लॉक) करण्याची परवानगी दिली जाते.इंटर-थ्रेड कम्युनिकेशनला जावामधील सहकार्य म्हणूनही ओळखले जाते. एखादी स्थिती खरी होईपर्यंत वारंवार तपासण्याची प्रक्रिया मतदान म्हणून ओळखली जाते. एखादी विशिष्ट स्थिती खरी आहे की नाही हे तपासण्यासाठी लूपच्या मदतीने मतदानाची अंमलबजावणी केली जाते. जर ते खरे असेल तर निश्चित कारवाई केली जाते. यामुळे अनेक CPU सायकल वाया जातात आणि अंमलबजावणी अकार्यक्षम बनते.
उदाहरणार्थ, क्लासिक रांगेतील समस्येमध्ये जिथे एक थ्रेड डेटा तयार करत आहे आणि दुसरा त्याचा वापर करत आहे.

संदर्भ(Web References):

1. https://www.javatpoint.com/java-tutorial
2. https://www.w3schools.com/java/
3. https://www.tutorialspoint.com/java/index.html

युनिट- 4
ॲबस्ट्रॅक्ट विंडो टूलकिट (AWT) आणि स्विंग्ज वापरून इव्हेंट हँडलिंग
(Event handling using Abstract Window Toolkit (AWT) & Swings Components)

विषय निष्पत्ती: Course Outcome (CO)

CO4 - विंडो-आधारित ॲप्लिकेशन घटकांचा वापर करून इव्हेंट हँडलिंग अंमलात आणण्यासाठी जावा प्रोग्राम विकसित करा.

सिद्धांत शिक्षण परिणाम: (Theory Learning Outcomes(TLO):

1. दिलेल्या समस्येसाठी फ्रेमसह AWT घटकांचा वापर करून ग्राफिकल यूजर इंटरफेस (GUI) विकसित करण्यासाठी पायऱ्या लिहा.
2. दिलेल्या समस्येसाठी मेनू आणि डायलॉग बॉक्स वापरून प्रोग्राम विकसित करा.
3. दिलेल्या समस्येसाठी प्रगत स्विंग घटकांचा वापर करून ग्राफिकल यूजर इंटरफेस (GUI) विकसित करण्यासाठी पायऱ्या लिहा.
4. दिलेल्या समस्येसाठी इव्हेंट चालित प्रोग्राम विकसित करण्यासाठी डेलिगेशन इव्हेंट मॉडेल वापरा.
5. दिलेल्या इव्हेंट हाताळण्यासाठी संबंधित AWT/स्विंगकम्पोनंट वापरा.

4.1 कम्पोनंट, कंटेनर, विंडो, फ्रेम, पॅनेल, AWT कंट्रोल्स (Component, Container, Window, Frame, Panel, AWT Controls)

- **कम्पोनंट (Component)**-हे GUI अनुप्रयोगातील मूलभूतकम्पोनंट आहेत. ते बटणे,
- टेक्स्टफील्ड लेबल्स इत्यादी असू शकतात. AWT मध्ये, सर्वकाही कम्पोनंट क्लास पासून विस्तारित(inherits) होते.
- **कंटेनर(Container)-**हा एक विशेष प्रकारचा कम्पोनंट**(**Component) आहे जो इतर कम्पोनंट ना(**Components)** सामावून घेऊ शकतो.
- **विंडो-** हा एक उच्च-स्तरीय कंटेनर आहे, सजावट नसलेली एक रिकामी विंडो. ती सामान्यतः थेट वापरली जात नाही.
- **फ्रेम-** ही टायटल बार आणि बॉर्डर असलेली एक विशेष विंडो आहे आणि ती मुख्य विंडोसाठी सामान्यतः वापरली जाते.
- **पॅनेल-** हा एक कंटेनर आहे जो फ्रेममध्ये कम्पोनंट आयोजित करण्यासाठी वापरला जातो.

➢ **AWT कंट्रोल्स :**

- **लेबल:** टेक्स्ट किंवा प्रतिमा प्रदर्शित करते.
- **बटण:** क्लिक केल्यावर कृती(Action) सुरू करते.
- **चेकबॉक्स:** वापरकर्त्याला(Users) पर्याय निवडण्याची किंवा निवड रद्द करण्याची परवानगी देते.
- **चेकबॉक्स ग्रुप:** एका वेळी फक्त एकच निवडता येईल म्हणून अनेक चेकबॉक्स गटबद्ध करण्यासाठी वापरले जाते.
- **टेक्स्टफील्ड:** मजकूरासाठी एकल-लाइन(singale line) इनपुट फील्ड.
- **टेक्स्टएरिया:** मोठ्या टेक्स्ट इनपुटसाठी एक बहु-लाइन(multiline) टेक्स्ट फील्ड.

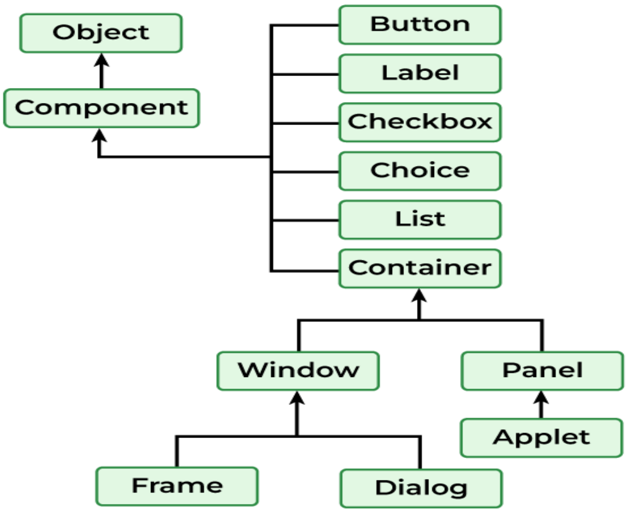


Fig 4.1 एडब्ल्यूटी पदानुक्रम (AWT Hierarchy)

उदाहरण (Example):

```
import java.awt.*;
public class AWTComponentsExample {
    public static void main(String[] args) {
        Frame frame = new Frame("AWT Components Example");
        Label label = new Label("Enter your name:");
        TextField textField = new TextField(20);
        Button button = new Button("Submit");
        // Adding components to the frame
        frame.add(label);
        frame.add(textField);
        frame.add(button);
        // Setting layout manager (FlowLayout arranges components in a line)
        frame.setLayout(new FlowLayout());
        // Frame settings
        frame.setSize(400, 200);
        frame.setVisible(true);
    }
}
```

आउटपुट (Output):

4.2 लेआउट मॅनेजर्सचा वापर (Use of Layout Managers)

लेआउट मॅनेजर्स कंटेनरमधील घटकांची स्थिती आणि आकारमान नियंत्रित करतात. जावामध्ये लेआउट मॅनेजर्सचे अनेक प्रकार आहेत:

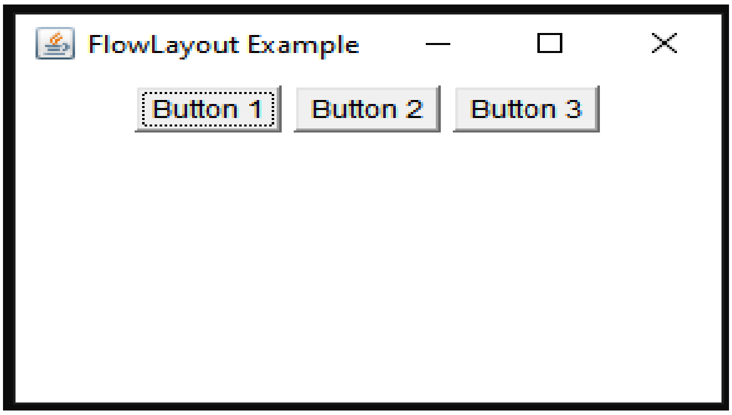
- **फ्लोलेआउट (FlowLayout):** डावीकडून उजवीकडे कम्पोनंट ची व्यवस्था करते.
- **बॉर्डरलेआउट (BorderLayout):** कंटेनरला पाच क्षेत्रांमध्ये (regions) (उत्तर, दक्षिण, पूर्व, पश्चिम, केंद्र) (North, South, East, West, Center). विभागते.
- **ग्रिड लेआउट (GridLayout):** निश्चित संख्येच्या पंक्ती आणि स्तंभां (Rows and columns) सह कम्पोनंट ना ग्रिडमध्ये व्यवस्थित (Orginize) करते.
- **ग्रिडबॅग लेआउट (GirdBagLayout):** कम्पोनंट चा आकार (resized)कसा बदलायचा आणि त्यांची स्थिती कशी ठेवायची(positioned) हे निर्दिष्ट करून अधिक जटिल व्यवस्थ (more complex arrangements)करण्यास अनुमती देते.
- **कार्ड लेआउट (CardLayout):** एकाधिक "कार्ड" (भिन्न पॅनेल) व्यवस्थापित करते आणि त्यांच्यामध्ये स्विच करण्यास अनुमती देते.

लेआउट मॅनेजर्सची उदाहरणे:

➢ **फ्लोलेआउट(FlowLayout):**

```
import java.awt.*;
public class FlowLayoutExample {
   public static void main(String[] args) {
      Frame frame = new Frame("FlowLayout Example");
       // Creating buttons
      Button btn1 = new Button("Button 1");
      Button btn2 = new Button("Button 2");
      Button btn3 = new Button("Button 3");
       // Adding buttons to the frame
      frame.setLayout(new FlowLayout());
      frame.add(btn1);
      frame.add(btn2);
      frame.add(btn3);
       // Frame settings
      frame.setSize(300, 200);
      frame.setVisible(true);}}
```

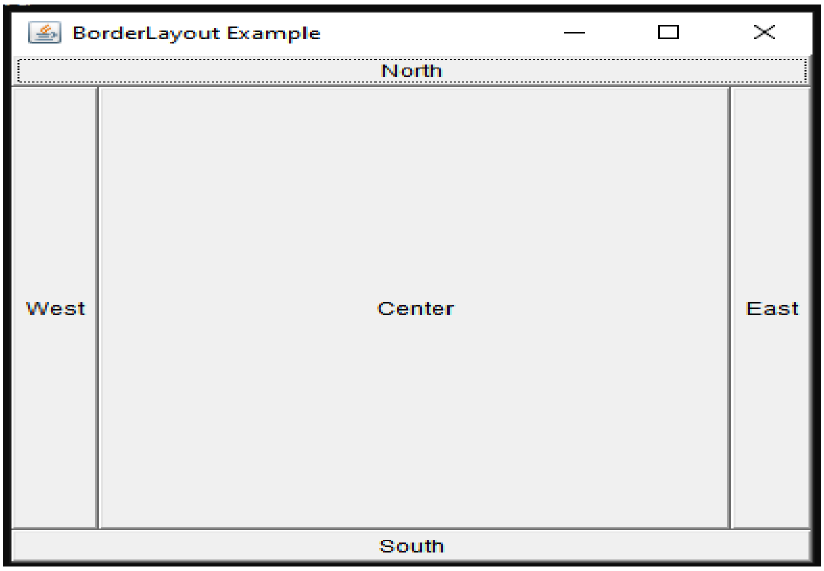
आउटपुट (Output):



➢ बॉर्डरलेआउट(BorderLayout):

```
import java.awt.*;
public class BorderLayoutExample {
    public static void main(String[] args) {
        Frame frame = new Frame("BorderLayout Example");
         // Creating buttons
        Button btn1 = new Button("North");
        Button btn2 = new Button("South");
        Button btn3 = new Button("East");
        Button btn4 = new Button("West");
        Button btn5 = new Button("Center");
        // Setting BorderLayout and adding components
        frame.setLayout(new BorderLayout());
        frame.add(btn1, BorderLayout.NORTH);
        frame.add(btn2, BorderLayout.SOUTH);
        frame.add(btn3, BorderLayout.EAST);
        frame.add(btn4, BorderLayout.WEST);
        frame.add(btn5, BorderLayout.CENTER);
        // Frame settings
        frame.setSize(400, 400);
        frame.setVisible(true);
    }
}
```

आउटपुट (Output):

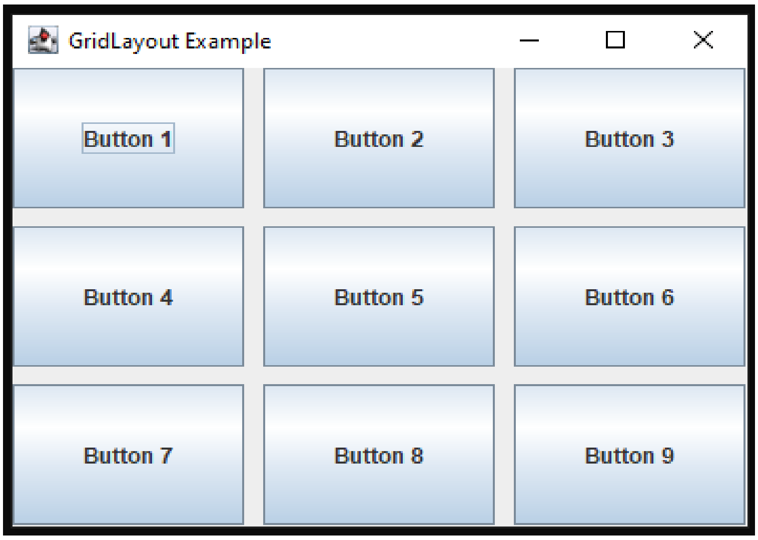


➢ ग्रिडलेआउट(GridLayout):

```
import java.awt.*;
import javax.swing.*;
public class GridLayoutExample {
    public static void main(String[] args) {
        // Create a frame
        JFrame frame = new JFrame("GridLayout Example");
```

```
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        // Set GridLayout (rows, columns, horizontal gap, vertical gap)
        frame.setLayout(new GridLayout(3, 3, 10, 10));
        // Add buttons to the grid
        for (int i = 1; i <= 9; i++) {
            frame.add(new JButton("Button " + i));
        }
        // Make the frame visible
        frame.setVisible(true);
    }
}
```

आउटपुट (Output):

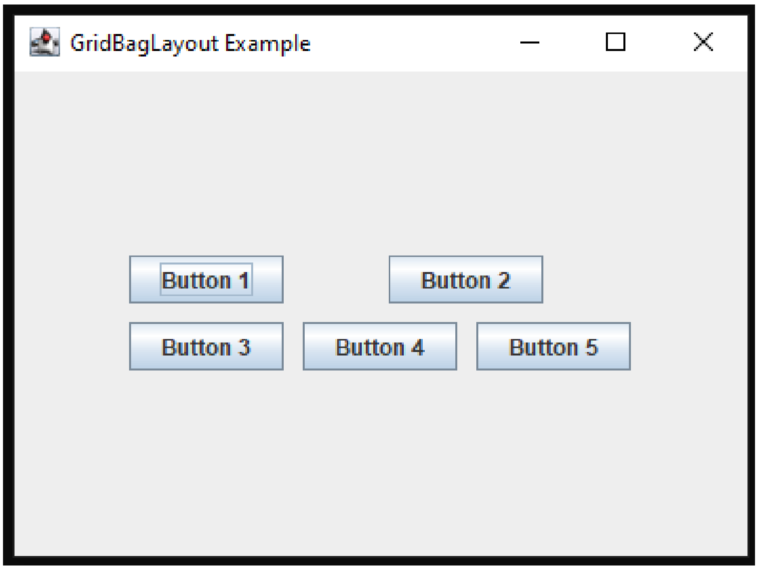


- **ग्रिडबॅगलेआउट(GridBagLayout):**

```
import java.awt.*;
import javax.swing.*;
public class GridBagLayoutExample {
    public static void main(String[] args) {
        // Create a frame
        JFrame frame = new JFrame("GridBagLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
            // Set GridBagLayout
        frame.setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        // Configure constraints and add components
        gbc.insets = new Insets(5, 5, 5, 5); // Padding between components
        // First button (Row 0, Column 0)
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 1; // Takes 1 column
        gbc.gridheight = 1; // Takes 1 row
```

```
        frame.add(new JButton("Button 1"), gbc);
        // Second button (Row 0, Column 1, spanning 2 columns)
        gbc.gridx = 1;
        gbc.gridy = 0;
        gbc.gridwidth = 2; // Spanning 2 columns
        frame.add(new JButton("Button 2"), gbc);
        // Third button (Row 1, Column 0)
        gbc.gridx = 0;
        gbc.gridy = 1;
        gbc.gridwidth = 1;
        frame.add(new JButton("Button 3"), gbc);
        // Fourth button (Row 1, Column 1)
        gbc.gridx = 1;
        gbc.gridy = 1;
        gbc.gridwidth = 1;
        frame.add(new JButton("Button 4"), gbc);
        // Fifth button (Row 1, Column 2)
        gbc.gridx = 2;
        gbc.gridy = 1;
        frame.add(new JButton("Button 5"), gbc);
        // Set frame visibility
        frame.setVisible(true);
    }
}
```

आउटपुट (Output):

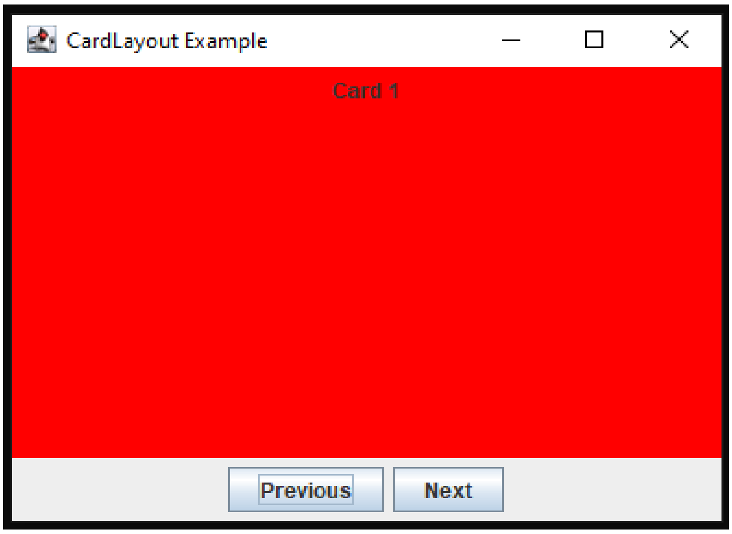


➢ **कार्डलेआउट(Card Layout):**

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample {
    public static void main(String[] args) {
```

```
        // Create the main frame
        JFrame frame = new JFrame("CardLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        // Create CardLayout and main panel
        CardLayout cardLayout = new CardLayout();
        JPanel cardPanel = new JPanel(cardLayout);
        // Create different panels (cards)
        JPanel panel1 = new JPanel();
        panel1.setBackground(Color.RED);
        panel1.add(new JLabel("Card 1"));
        JPanel panel2 = new JPanel();
        panel2.setBackground(Color.GREEN);
        panel2.add(new JLabel("Card 2"));
        JPanel panel3 = new JPanel();
        panel3.setBackground(Color.BLUE);
        panel3.add(new JLabel("Card 3"));
        // Add panels (cards) to the card panel
        cardPanel.add(panel1, "Card 1");
        cardPanel.add(panel2, "Card 2");
        cardPanel.add(panel3, "Card 3");
        // Create buttons to switch between cards
        JButton nextButton = new JButton("Next");
        JButton prevButton = new JButton("Previous");
        // Action listeners for buttons
        nextButton.addActionListener(e -> cardLayout.next(cardPanel));
        prevButton.addActionListener(e -> cardLayout.previous(cardPanel));
        // Create a control panel for buttons
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(prevButton);
        buttonPanel.add(nextButton);
        // Add components to the frame
        frame.setLayout(new BorderLayout());
        frame.add(cardPanel, BorderLayout.CENTER);
        frame.add(buttonPanel, BorderLayout.SOUTH);
        // Set frame visibility
        frame.setVisible(true);
    }
}
```

आउटपुट (Output):



4.2 स्विंगचा परिचय(Introduction to Swing)

स्विंग ही एक GUI टूलकिट आहे जी AWT च्या तुलनेत अधिक फ्लेक्सिबल आणि रिच User इंटरफेस कम्पोनंट प्रदान (provide) करते.

स्विंगकम्पोनंट लाइटवेट असतात (ते मूळ OS कम्पोनंट वर अवलंबून नसतात).

➢ **AWT आणि स्विंगमधील प्रमुख फरक:**

- **AWT:** हेवीवेटकम्पोनंट (OS वर अवलंबून).
- **स्विंग:** लाइटवेट कम्पोनंट (जावा वापरून काढलेले).

स्विंग चांगले कस्टमायझेशन करण्यास अनुमती देते, प्लग करण्यायोग्य लुक-अँड-फीलला समर्थन देते आणि अधिक वैशिष्ट्यांसह ट्री, TABLE आणि टेक्स्ट एरिया यासारखे कम्पोनंट प्रदान (Provide) करते.

➢ **AWT आणि स्विंगमधील फरक**

खालील तक्ता AWT आणि स्विंगमधील प्रमुख फरकांचे वर्णन करतो

Table 4.1 AWT आणि स्विंगमधील फरक

अ. क्र.	जावा AWT	जावा स्विंग
1.	AWT कम्पोनंट प्लॅटफॉर्मवर अवलंबून आहेत (AWT components are platform-dependent.)	जावा स्विंगकम्पोनंट टफॉर्मवर अवलंबून नाही आहेत.(Java swing components are platform-independent.)
2.	AWTकम्पोनंट हेवीवेट आहेत. (AWT components are heavyweight.)	स्विंगकम्पोनंट लाइटवेट आहेत. (Swing components are lightweight.)
3.	AWT प्लगेबल लूक आणि फीलला सपोर्ट करत नाही.(AWT doesn't support pluggable look and feel.)	स्विंग प्लगेबल लूक आणि फीलला सपोर्ट करते. (Swing supports pluggable look and feel.)

4.	AWT स्विंगपेक्षा कमीकम्पोनंट प्रदान करते. (AWT provides less components than Swing.)	स्विंग TABLE, लिस्ट, स्क्रोलपेन, कलरचूजर, टॅब्डपेन इत्यादीसारखे अधिक शक्तिशालीकम्पोनंट प्रदान करते. (Swing provides more powerful components such as tables, lists, scrollpanes,colorchooser, tabbedpane etc.)
5.	AWT MVC (मॉडेल ,व्ह्यू,कंट्रोलर) चेअनुसरण करत नाही (AWT doesn't follows MVC(Model View Controller)	स्विंग MVC चे अनुसरण करते. (Swing follows MVC.)

➢ **जावा स्विंगची प्रमुख वैशिष्ट्ये**

- **प्लॅटफॉर्म स्वातंत्र्य(Platform Independence)**: स्विंगचा एक प्रमुख फायदा म्हणजे त्याचे प्लॅटफॉर्म स्वातंत्र्य. स्विंग वापरून विकसित केलेले अनुप्रयोग (programs) जावाला समर्थन देणाऱ्या कोणत्याही प्लॅटफॉर्मवर बदल न करता चालू शकतात.
- **कम्पोनंट चा समृद्ध संच (Rich Set of Components)**: स्विंगमध्ये विविध कम्पोनंट चा समावेश आहे ज्यांचा वापर जटिल(complex) GUI तयार करण्यासाठी केला जाऊ शकतो. डेव्हलपर बटणे आणि लेबल्स सारख्या मूलभूत कम्पोनंट पासून ते TABLE, ट्री आणि स्क्रोल पेन सारख्या प्रगत कम्पोनंट पर्यंत निवडू शकतात.
- **सानुकूलितता (Customizability):** स्विंगकम्पोनंट अत्यंत सानुकूलित आहेत, ज्यामुळे डेव्हलपर त्यांच्या देखावा आणि वर्तनाच्या विविध पैलूंवर नियंत्रण ठेवू शकतात. आकार, रंग, फॉन्ट आणि लेआउट सारखे गुणधर्म विशिष्ट डिझाइन आवश्यकता पूर्ण करण्यासाठी सहजपणे समायोजित केले जाऊ शकतात.
- **इव्हेंट हँडलिंग(Event Handling):** स्विंग एक मजबूत इव्हेंट हँडलिंग यंत्रणा प्रदान करते जी डेव्हलपरना बटण क्लिक आणि माऊस मुहमनेट सारख्या वापरकर्त्यांच्या परस्परसंवादांना (users interaction) प्रतिसाद देण्यास अनुमती देते. हे परस्परसंवादी आणि प्रतिसादात्मक अनुप्रयोगांची निर्मिती सक्षम करते.
- **लेआउट व्यवस्थापक(Layout Managers)**: स्विंगमध्ये लेआउट व्यवस्थापकांचा एक संच समाविष्ट आहे जो कंटेनरमधील कम्पोनंट ची व्यवस्था सुलभ करतो. लेआउट मॅनेजर कंटेनरच्या आकारानुसार कम्पोनंट ची स्थिती आणि आकार स्वयंचलितपणे समायोजित करतात, ज्यामुळे वेगवेगळ्या स्क्रीन रिझोल्यूशन आणि डिव्हाइसेसमध्ये सुसंगत वर्तन सुनिश्चित होते.

➢ **जावा स्विंग क्लासेसची पदानुक्रम(Hierarchy of Java Swing classes)**

जावा स्विंग एपीआयची पदानुक्रम(hierarchy) खाली दिली आहे.

- JComponent हा सर्व Swing घटकांसाठी बेस क्लास आहे. त्याचे काही महत्त्वाचे सबक्लास खाली दिले आहेत:
- JLabel → मजकूर किंवा प्रतिमा दाखवतो.
- JButton → क्लिक करण्यासाठी बटण.
- JTextField → एकाच ओळीत टेक्स्ट इनपुट करण्यासाठी.
- JTextArea → बहु-ओळी टेक्स्ट इनपुटसाठी.
- JCheckBox → सिलेक्शन बॉक्स (होय/नाही).
- JRadioButton → रेडिओ बटण (एकच सिलेक्ट करता येते).
- JList → लिस्ट दाखवण्यासाठी.
- JTable → TABLE डेटा दाखवण्यासाठी.
- JTree → झाडासारखा (हायरेरकी) डेटा दाखवण्यासाठी.
- JScrollPane → मोठा कंटेंट स्क्रोल करण्यासाठी.

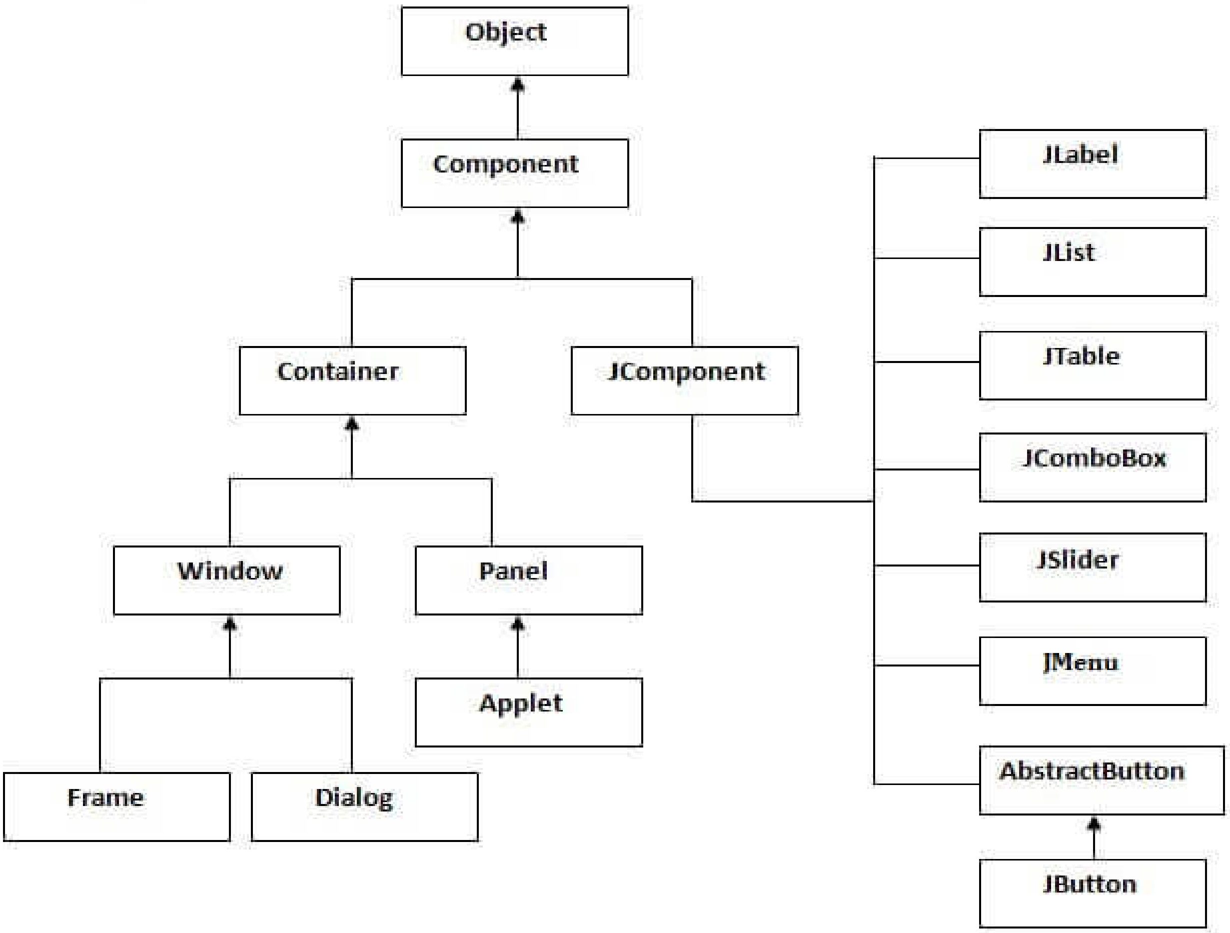


Fig 4.2 जावा स्विंग पदानुक्रम (java swing API Hierarchy)

- कम्पोनंट क्लासच्या पद्धती(Methods of Component Class)

जावा स्विंगमध्ये कम्पोनंट **क्लासच्या** पद्धतीमोठ्या प्रमाणात वापरल्या जातात ज्या खाली दिल्या आहेत.

पद्धती (Method)	**वर्णन (Description)**
public void add(Component c)	ते एका कम्पोनंट ला दुसऱ्या कम्पोनंट वर जोडते (It add a component on another component.)
public void setSize(int width,int height)	हे कम्पोनंट चा आकार सेट करते.(It sets size of the component.)
public void setLayout(LayoutManager m)	हे कम्पोनंट साठी लेआउट मॅनेजर सेट करते.(It sets the layout manager for the component.)
public void setVisible(boolean b)	हे कम्पोनंट ची दृश्यमानता सेट करते. ते बाय डिफॉल्ट फॉल्स असते.(It sets the visibility of the component. It is by default false.)

➢ जावा स्विंग उदाहरणे

फ्रेम तयार करण्याचे दोन मार्ग आहेत:

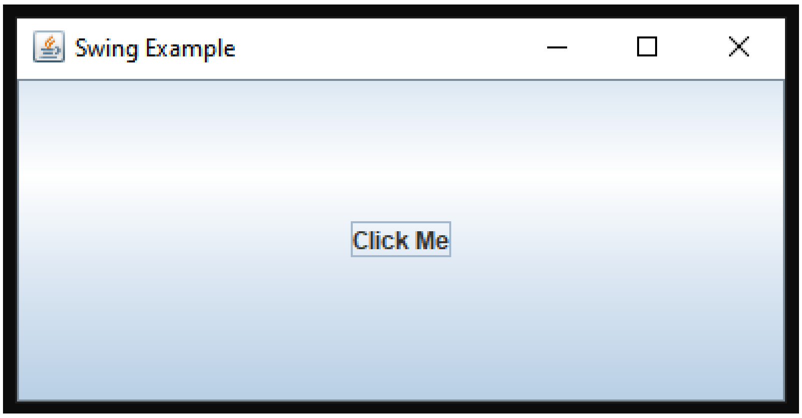
- फ्रेम क्लास (असोसिएशन) चे ऑब्जेक्ट तयार करून
- फ्रेम क्लास (इनहेरिटन्स) एक्सटेन्ड करून (By extending) आपण स्विंगचा कोड मेन(), कन्स्ट्रक्टर किंवा इतर कोणत्याही पद्धतीमध्ये लिहू शकतो.

जावा स्विंग उदाहरण(Swing Example):

फाइलचे नाव: FirstSwingExample.java

```
import javax.swing.*;
public class SwingExample {
   public static void main(String[] args) {
      // Creating a JFrame (top-level container for Swing components)
      JFrame frame = new JFrame("Swing Example");
      // Creating a JButton (Swing button component)
      JButton button = new JButton("Click Me");
      // Adding button to the frame
      frame.add(button);
      // Setting frame properties
      frame.setSize(400, 200);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

आउटपुट (Output):



4.4 स्विंग कम्पोनंट

स्विंग कम्पोनंट फ्लेक्सिबल आणि वैशिष्ट्यपूर्ण असतात. सर्वात जास्त वापरल्या जाणाऱ्या काही कम्पोनंट मध्ये हे समाविष्ट आहे:

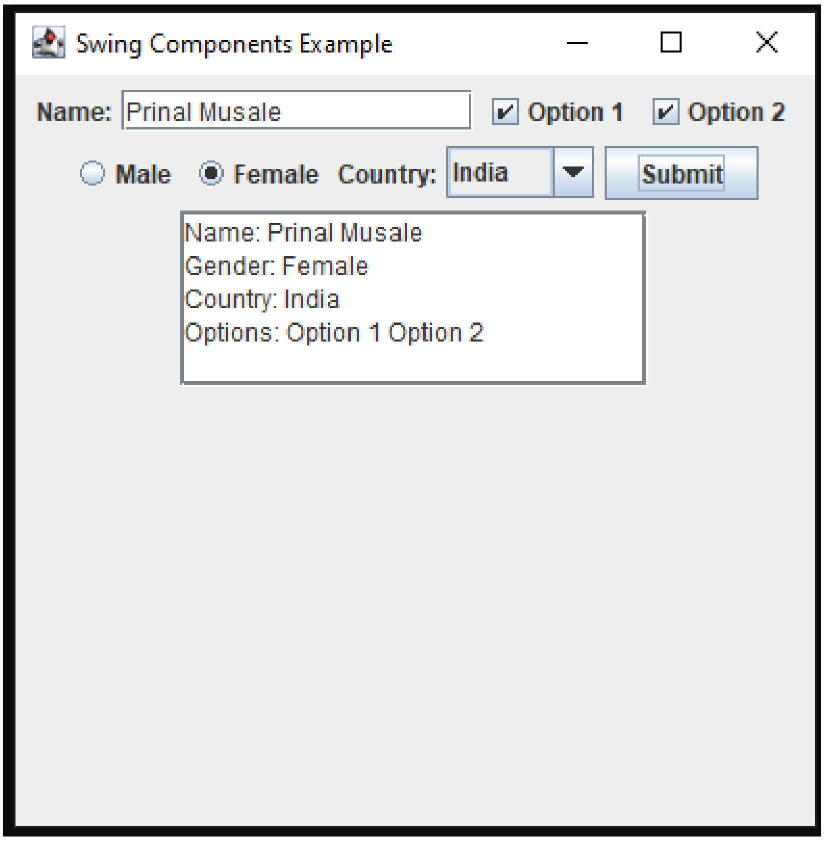
- **JLabel:** टेक्स्ट किंवा प्रतिमा प्रदर्शित करते.
- **JTextField:** एकल-लाइन टेक्स्ट फील्ड.
- **JComboBox:** ड्रॉप-डाउन सूची.
- **JButton:** क्रिया सुरू करू शकणारे बटण.
- **JCheckBox:** वापरकर्ता निवडू किंवा निवड रद्द करू शकणारा बॉक्स.
- **JRadioButton:** एका गटातील एक बटण जिथे एका वेळी फक्त एकच निवडता येतो.

अनेक स्विंग कम्पोनंटचा वापर करून उदाहरण(Example of using several Swing components):

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SwingComponentsExample {
  public static void main(String[] args) {
    // Create the main frame
    JFrame frame = new JFrame("Swing Components Example");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(400, 400);
    frame.setLayout(new FlowLayout());
    // Create labels and text fields
    JLabel nameLabel = new JLabel("Name:");
    JTextField nameField = new JTextField(15);
    // Create checkboxes
    JCheckBox checkBox1 = new JCheckBox("Option 1");
    JCheckBox checkBox2 = new JCheckBox("Option 2");
    // Create radio buttons and group them
    JRadioButton maleRadio = new JRadioButton("Male");
    JRadioButton femaleRadio = new JRadioButton("Female");
    ButtonGroup genderGroup = new ButtonGroup();
    genderGroup.add(maleRadio);
    genderGroup.add(femaleRadio);
    // Create a combo box (drop-down)
    String[] countries = { "USA", "Canada", "UK", "India" };
    JComboBox<String> countryBox = new JComboBox<>(countries);
    // Create a text area
    JTextArea textArea = new JTextArea(5, 20);
    textArea.setBorder(BorderFactory.createLineBorder(Color.GRAY));
    // Create a button
    JButton submitButton = new JButton("Submit");
    // Add action listener to button
    submitButton.addActionListener(e -> {
      String name = nameField.getText();
      String gender = maleRadio.isSelected() ? "Male" : femaleRadio.isSelected() ? "Female" :
   "Not selected";
      String country = (String) countryBox.getSelectedItem();
      String options = (checkBox1.isSelected() ? "Option 1 " : "") + (checkBox2.isSelected() ?
   "Option 2" : "");
      textArea.setText("Name: " + name + "\nGender: " + gender + "\nCountry: " + country +
   "\nOptions: " + options);
    });
    // Add components to the frame
    frame.add(nameLabel);
```

```
        frame.add(nameField);
        frame.add(checkBox1);
        frame.add(checkBox2);
        frame.add(maleRadio);
        frame.add(femaleRadio);
        frame.add(new JLabel("Country:"));
        frame.add(countryBox);
        frame.add(submitButton);
        frame.add(new JScrollPane(textArea)); // Add scroll pane to text area
        // Set frame visibility
        frame.setVisible(true);
    }
```

आउटपुट (Output):



4.5 प्रगत स्विंग कम्पोनंट

प्रगत स्विंगकम्पोनंट तुम्हाला अधिक जटिल अनुप्रयोग(Complex Program) तयार करण्याची परवानगी देतात. या कम्पोनंट मध्ये हे समाविष्ट आहे:

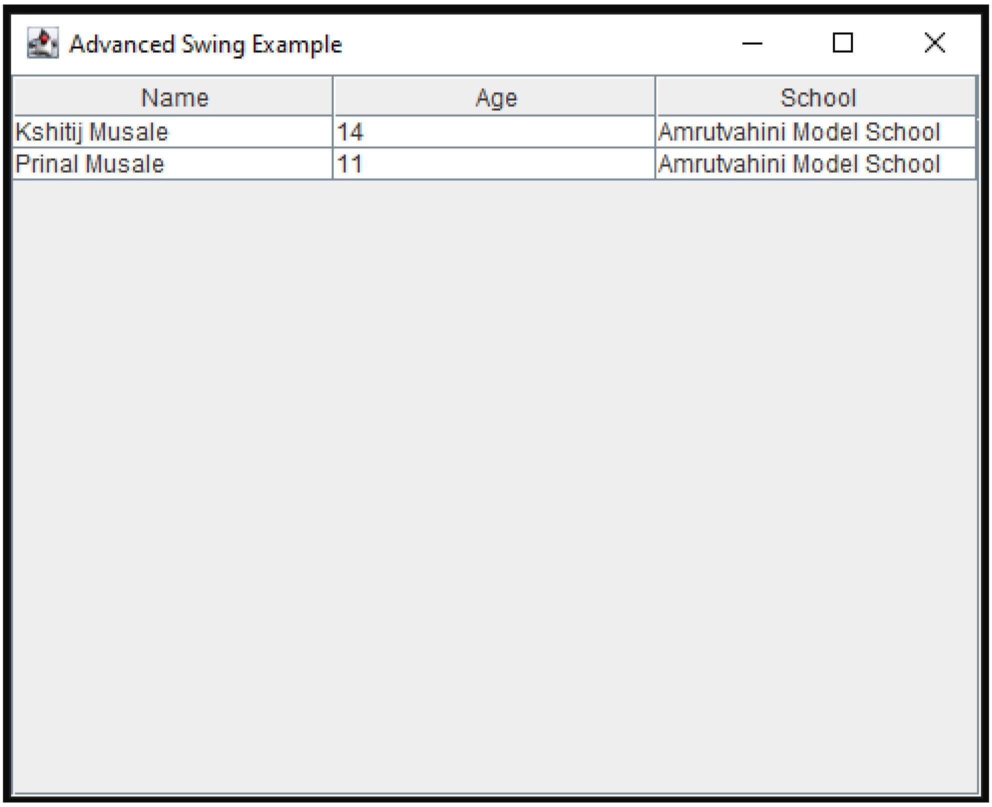
- **JTabbedPane:** टॅब्ड नेव्हिगेशनला अनुमती देते.
- **JScrollPane:** स्क्रोल कार्यक्षमता जोडते.
- **JTable:** TABLE स्वरूपात डेटा प्रदर्शित करते.
- **JProgressBar:** प्रोग्रेस टास्क प्रदर्शित करते.
- **JToolTip:** कम्पोनंट वर मुहमेन्ट करताना माहिती दर्शवते.

JTable आणि JScrollPane वापरण्याचे उदाहरण(Example of using a JTable and scrollPane):

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
```

```
public class AdvancedSwingExample {
   public static void main(String[] args) {
      // Create a frame
      JFrame frame = new JFrame("Advanced Swing Example");
      // Creating a table
      String[] columns = {"Name", "Age"};
      String[][] data = {{"Kshitij Musale", "14","Amrutvahini Model School"}, {"Prinal Musale",
    "11" ,"Amrutvahini Model School"}};
      JTable table = new JTable(data, columns);
      // Adding table to a scroll pane
      JScrollPane scrollPane = new JScrollPane(table);
      frame.add(scrollPane);
      // Frame settings
      frame.setSize(500, 400);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

आउटपुट (Output):



4.6 इव्हेंट हँडलिंगचा परिचय (Introduction to Event Handling)

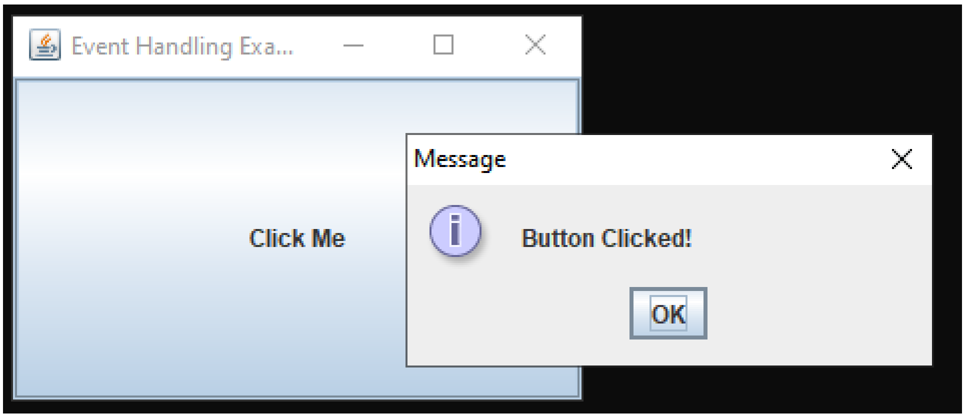
इव्हेंट-चालित प्रोग्रामिंगमध्ये, तुम्ही बटण क्लिक, माऊस मुहमेन्ट इत्यादी वापरकर्त्यांच्या कृतींना(User Action) प्रतिसाद देता. जावा डेलिगेशन इव्हेंट मॉडेल **(Delegation Event Model)** वापरते, जे कम्पोनंट ना इव्हेंट **हँडलिंग** साठी जबाबदार असलेल्या इव्हेंट लिसनर ना इव्हेंट पाठविण्याची परवानगी देते.

- **इव्हेंट स्रोत (Event sources)**: इव्हेंट निर्माण करणारे ऑब्जेक्ट (उदा., बटणे,टेक्स्ट फील्ड).
- **इव्हेंट लिसनर:** इव्हेंट हाताळणारे ऑब्जेक्ट (उदा.,ॲक्शन लिसनर, आयटमलिसनर).

स्विंग (बटण क्लिक) मध्ये इव्हेंट हँडलिंगचे उदाहरण:

```
import javax.swing.*;
import java.awt.event.*;
public class EventHandlingExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Event Handling Example");
      // Creating a button
      JButton button = new JButton("Click Me");
      // Adding an ActionListener to the button
      button.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            JOptionPane.showMessageDialog(frame, "Button Clicked!");
         }
      });
      // Adding the button to the frame
      frame.add(button);
         // Frame settings
      frame.setSize(300, 200);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

आउटपुट (Output):



4.7 इव्हेंट क्लासेस(Event Classes)

इव्हेंट क्लासेस वेगवेगळ्या प्रकारच्या वापरकर्त्यांच्या कृती (Users Action) दर्शवतात.

उदाहरणार्थ:

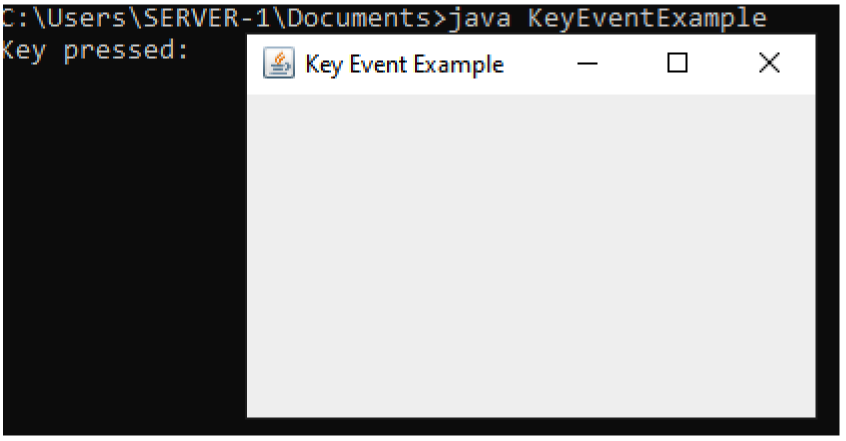
- **ActionEvent:** बटण, मेनू आयटम किंवा तत्सम कम्पोनंट वर क्लिक केल्यावर ट्रिगर होतो.
- **ItemEvent:** आयटम (जसे की चेकबॉक्स किंवा रेडिओ बटण) निवडल्यावर ट्रिगर होतो.
- **KeyEvent:** की दाबली किंवा सोडली की ट्रिगर होतो.
- **MouseEvent:** क्लिक, मुहमेन्ट किंवा कम्पोनंट मध्ये प्रवेश करणे/बाहेर पडणे यासारख्या माऊसच्या कृतींमुळे(Action) ट्रिगर होतो.

KeyEvent वापरण्याचे उदाहरण:

```
import java.awt.event.*;
```

```
import javax.swing.*;
public class KeyEventExample {
   public static void main(String[] args) {
      JFrame frame = new JFrame("Key Event Example");
      // Adding a KeyListener
      frame.addKeyListener(new KeyAdapter() {
         public void keyPressed(KeyEvent e) {
            System.out.println("Key pressed: " + e.getKeyChar());
         }
      });
      // Frame settings
      frame.setSize(300, 200);
      frame.setVisible(true);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```

आउटपुट (Output):



4.8 इव्हेंट लिसनर इंटरफेस

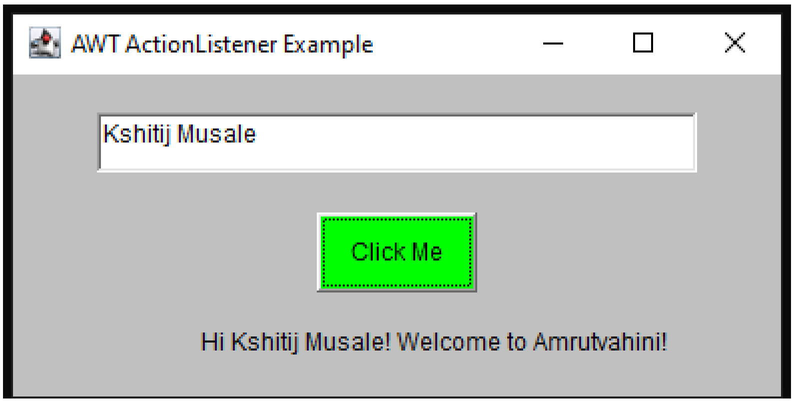
- **ॲक्शनलिसनर**: ॲक्शन इव्हेंट्स ऐकतो, उदा. बटण क्लिक्स.
- **आयटमलिसनर**: आयटम इव्हेंट्स ऐकतो, उदा. चेकबॉक्स किंवा कॉम्बो बॉक्स बदल.
- **कीलिसनर:** कीबोर्ड इव्हेंट्स ऐकतो, उदा. की दाबणे.
- **माउसलिसनर**: क्लिक्स आणि होव्हर सारख्या माउस इव्हेंट्स ऐकतो.
- **माउसमोशनलिसनर:** माउस मुहमेन्ट इव्हेंट्स ऐकतो.

ॲक्शनलिसनर आणि आयटमलिसनर वापरण्याचे उदाहरण:

```
// Java Program to implement
// AWT ActionListener
import java.awt.*;
import java.awt.event.*;
// Driver Class
public class Main {
    // main function
   public static void main(String[] args){
      // Create a frame
      Frame f = new Frame("AWT ActionListener Example");
```

```
        // Set the size
        f.setSize(400, 200);
        // Set the layout
        f.setLayout(null);
        // Make the frame visible
        f.setVisible(true);
        // Set the background color of the frame
        f.setBackground(Color.LIGHT_GRAY);
        // Create a button
        Button b = new Button("Click Me");
        // Set the positions
        b.setBounds(160, 100, 80, 40);
        // Add button to the frame
        f.add(b);
        // Set the background color of the button
        b.setBackground(Color.GREEN);
        // Create a text field
        TextField tf = new TextField();
        // Set the positions
        tf.setBounds(50, 50, 300, 30);
        // Add text field to the frame
        f.add(tf);
        // Create a label
        Label lb = new Label();
        // Set the positions
        lb.setBounds(100, 150, 300, 30);
        // Add label to the frame
        f.add(lb);
        // Add an action listener to the button
        b.addActionListener(new ActionListener() {
            // Override the actionPerformed() method
            public void actionPerformed(ActionEvent e){
                // Update the text of the label
                lb.setText("Hi " + tf.getText() + "! "
                        + "Welcome to Amrutvahini!");
            }
        });
}}
```

आउटपुट (Output):



संदर्भ(References):

1. https://www.javatpoint.com/awt-and-swing-in-java
2. https://docs.oracle.com/javase/8/docs/technotes/guides/awt/
3. https://dotnettutorials.net/lesson/abstract-windows-toolkit-awt-in-java/
4. https://docs.oracle.com/javase/1.5.0/docs/guide/awt/
5. https://www.geeksforgeeks.org/java-actionlistener-in-awt/

युनिट - 5
बेसिक ऑफ नेटवर्किंग
(Basic of Network Programming)

विषय निष्पत्ती: Course Outcome (CO)

CO5 - जावामध्ये नेटवर्क प्रोग्रामिंगची अंमलबजावणी करा.

सिद्धांत शिक्षण परिणाम: (Theory Learning Outcomes(TLO)):

1. जावामध्ये सॉकेट्सच्या संकल्पनांचे वर्णन करा.
2. नेटवर्किंग क्लासेस वापरून होस्ट तपशील मिळवा.
3. दिलेल्या समस्येसाठी TCP/IP सर्वर सॉकेट्सद्वारे क्लायंट/सर्वर संवादासाठी प्रोग्राम विकसित करा.

- **परिचय (Introducation)**

सन्स मायक्रोसिस्टम्स, ज्यांनी जावा विकसित केला, त्यांचे ग्रीटिंग वाक्य आहे, 'नेटवर्क म्हणजे संगणक'. त्यामुळे त्यांनी जावा प्रोग्रामिंग भाषा नेटवर्किंगसाठी अधिक योग्य बनवली, जसे की, C++ किंवा FORTRAN पेक्षा. जावाला नेटवर्किंगसाठी उत्तम भाषा बनवणारे कारण म्हणजे java.net पॅकेजमध्ये परिभाषित केलेल्या क्लासेस. हे नेटवर्किंग क्लासेस सॉकेट्सच्या संकल्पनेस समाविष्ट करतात, जे बर्कली सॉफ्टवेअर डिस्ट्रिब्यूशन (BSD) मध्ये, युनिव्हर्सिटी ऑफ कॅलिफोर्निया, बर्कली येथे सुरू करण्यात आले होते. इंटरनेट नेटवर्किंग लायब्ररींच्या कोणत्याही चर्चेत UNIX आणि BSD सॉकेट्सच्या इतिहासाचा थोडक्यात उल्लेख केल्याशिवाय पूर्ण होणार नाही.

5.1 सॉकेट ओव्हरव्हीयु (Socket Overview)

नेटवर्क सॉकेट हे इलेक्ट्रिकल सॉकेटसारखे असते. नेटवर्कच्या सभोवतालच्या विविध प्लगमध्ये त्यांचे पेलोड वितरित करण्याचा एक मार्ग आहे. जे कोणतेही उपकरण या प्रमाणित प्रोटोकॉलला समजते, ते त्या सॉकेटमध्ये जोडले जाऊ शकते आणि संवाद साधू शकते.जोपर्यंत ते 50Hz, 115-व्होल्ट विजेची अपेक्षा करत आहेत, तोपर्यंत उपकरणे कार्य करतील. इलेक्ट्रिक बिल कसे तयार होते याचा विचार करा. घर आणि उर्वरित नेटवर्कमध्ये कुठेतरी एक मीटर आहे. त्या मीटरमधून जाणाऱ्या प्रत्येक किलोवॅट पॉवरसाठी बिल दिले जाते. त्यामुळे पॉवर ग्रिडच्या आजूबाजूला वीज मुक्तपणे वाहत असली तरी घरातील सर्व सॉकेटचा एक विशिष्ट पत्ता असतो. हीच संकल्पना नेटवर्क सॉकेटलाही लागू होते, परंतु येथे आपण इलेक्ट्रॉन्स आणि स्ट्रीट अड्रेसऐवजी TCP/IP पॅकेट्स आणि IP ॲड्रेस वापरतो.

इंटरनेट प्रोटोकॉल (IP) हा एक लो-लेव्हल रूटिंग प्रोटोकॉल आहे जो डेटा लहान पॅकेट्समध्ये विभागतो आणि नेटवर्कवर विशिष्ट पत्त्यावर पाठवतो, पण हे पॅकेट्स योग्य ठिकाणी पोहोचतीलच याची हमी देत नाही.

ट्रान्समिशन कंट्रोल प्रोटोकॉल (TCP) हा एक हाय-लेव्हल प्रोटोकॉल आहे जो या पॅकेट्सना योग्य क्रमाने ठेवतो, हरवलेले पॅकेट्स पुन्हा पाठवतो आणि डेटा अचूकपणे पोहोचेल याची खात्री करतो.

तिसरा प्रोटोकॉल, यूझर डेटाग्राम प्रोटोकॉल (UDP), TCPच्या शेजारी कार्य करतो आणि वेगवान, कनेक्शन-विरहित, आणि अनिश्चित डेटा ट्रान्समिशनसाठी वापरला जातो.

5.1.1 क्लायंट/सर्व्हर

सर्व्हर म्हणजे कोणतेही असे साधन जे काही रिसोर्स (resource) शेअर करू शकते. वेगवेगळ्या प्रकारचे सर्व्हर असतात—

- **कम्प्युट सर्व्हर**-जे संगणकीय क्षमता प्रदान करतात.
- **प्रिंट सर्व्हर**-जे प्रिंटरच्या संचालने व्यवस्थापित करतात.
- **डिस्क सर्व्हर**-जे नेटवर्कवर स्टोरेज स्पेस प्रदान करतात.
- **वेब सर्व्हर**-जे वेब पृष्ठे संग्रहित करतात.

क्लायंट म्हणजे कोणताही घटक ज्याला एखाद्या विशिष्ट सर्व्हरशी कनेक्ट व्हायचे असते.क्लायंट आणि सर्व्हर यांच्यातील परस्परसंवाद विद्युत सॉकेट आणि दिव्याच्या (लॅम्पच्या) परस्परसंवादासारखा आहे. घरातील वीज ग्रिड म्हणजे सर्व्हर, आणि दिवा हा क्लायंट आहे. सर्व्हर हा नेहमी उपलब्ध असलेला स्रोत आहे, तर क्लायंट त्याच्या गरजेनुसार कनेक्ट आणि डिस्कनेक्ट होऊ शकतो.बर्कले सॉकेट्समध्ये, एका संगणकाला अनेक वेगवेगळ्या क्लायंट्सना एकाच वेळी सेवा पुरवता यावी म्हणून "सॉकेट" ही संकल्पना अस्तित्वात आहे. तसेच, वेगवेगळ्या प्रकारची माहिती सर्व्ह करण्यासाठी सॉकेट्स वापरले जातात.पोर्ट

ही संकल्पना यासाठीच निर्माण करण्यात आली आहे. पोर्ट म्हणजे एका विशिष्ट संगणकावरील संख्या असलेला सॉकेट.

- सर्व्हर प्रोसेस हा एका पोर्टवर "लिसन" करत राहतो, जोपर्यंत एखादा क्लायंट त्याला कनेक्ट करत नाही.
- एकाच पोर्ट नंबरवर अनेक क्लायंट्स कनेक्ट होऊ शकतात, परंतु प्रत्येक सेशन (सत्र) हे वेगळे आणि स्वतंत्र असते.
- अनेक क्लायंट कनेक्शन्स हाताळण्यासाठी, सर्व्हर प्रोसेस मल्टीथ्रेडेड (अनेक थ्रेड्स असलेली) असावी किंवा एकाच वेळी अनेक I/O हाताळण्यासाठी मल्टिप्लेक्सिंग तंत्र वापरले जाते.

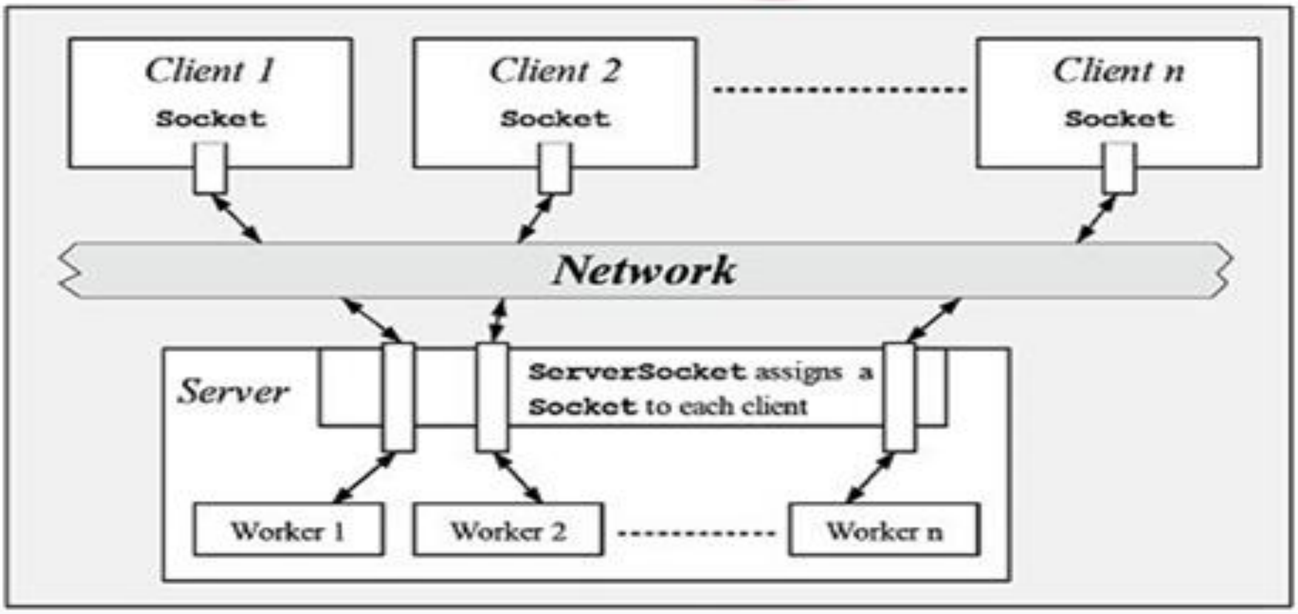


Fig 5.1: क्लायंट-सर्व्हर कम्युनिकेशन

5.1.2 रिझर्वड सॉकेट्स (Reserved Sockets)

TCP/IP मध्ये 1,024 पोर्ट्स विशिष्ट प्रोटोकॉलसाठी राखीव असतात.TCP/IP प्रणालीमध्ये खालील 1,024 पोर्ट्स वेगवेगळ्या प्रोटोकॉलसाठी राखीव ठेवले जातात. जर आपण इंटरनेट ब्राउझिंग केले असेल, तर हे पोर्ट आपल्याला परिचित असतील. उदाहरणार्थ:

- **पोर्ट 21** - FTP (File Transfer Protocol)
- **पोर्ट 23 -** Telnet
- **पोर्ट 25 -** ई-मेल (SMTP - Simple Mail Transfer Protocol)
- **पोर्ट 79 -** Finger
- **पोर्ट 80 -** HTTP (Hypertext Transfer Protocol)
- **पोर्ट 119 -** Net-News (NNTP - Network News Transfer Protocol)

प्रत्येक प्रोटोकॉल ठरवतो की क्लायंट त्या पोर्टशी कसा संवाद साधेल.उदाहरणार्थ, HTTP (Hypertext Transfer Protocol) हा वेब ब्राउझर्स आणि वेब सर्व्हर्स वेबपृष्ठे आणि प्रतिमा हस्तांतरित करण्यासाठी वापरतात. हा तुलनेने सोपा प्रोटोकॉल आहे, जो पृष्ठ ब्राउझिंगसाठी वापरला जातो.

5.1.3 प्रॉक्सी सर्व्हरप्रॉक्सी सर्व्हर हा क्लायंटच्या वतीने दुसऱ्या सर्व्हरशी संवाद साधतो**.** हा पर्याय तेव्हा उपयुक्त ठरतो जेव्हा क्लायंटवर काही विशिष्ट सर्व्हरशी थेट कनेक्ट होण्यावर निर्बंध असतात**.**

प्रॉक्सी सर्व्हर कसा काम करतो?

1. क्लायंट प्रथम प्रॉक्सी सर्व्हरशी कनेक्ट होतो.
2. प्रॉक्सी सर्व्हर, ज्यावर अशा प्रकारचे निर्बंध नसतात, क्लायंटच्या वतीने इच्छित सर्व्हरशी संपर्क साधतो.
3. प्रॉक्सी सर्व्हर नुसते डेटा फॉरवर्ड करत नाही तर विनंत्या (requests) फिल्टर करू शकतो आणि काही प्रतिसाद (responses) कॅश करून भविष्यातील वापरासाठी साठवू शकतो.

- **प्रॉक्सी सर्व्हरचे फायदे**
  1. **नेटवर्कवरील बँडविड्थ कमी होते** – लोकप्रिय वेबपृष्ठे अनेक वापरकर्त्यांकडून वारंवार पाहिली जात असल्यास, प्रॉक्सी सर्व्हर ती एकदाच डाउनलोड करून साठवतो**.** त्यामुळे प्रत्येक वेळी इंटरनेटवरून डेटा पुन्हा पुन्हा डाउनलोड करण्याची गरज राहत नाही.
  2. **स्पीड वाढतो** – कॅशमधून डेटा मिळाल्यास क्लायंटला वेगवान प्रतिसाद मिळतो.
  3. **सुरक्षितता आणि गोपनीयता सुधारते** – काही प्रॉक्सी सर्व्हर डेटा एन्क्रिप्ट करून सुरक्षितता वाढवतात आणि वापरकर्त्यांची ओळख लपवू शकतात.

उदाहरण:

समजा, एका संस्थेतील शंभर कर्मचारी रोज त्याच वेबपृष्ठावर जातात. जर प्रत्येक व्यक्तीने थेट इंटरनेटवरून हे पृष्ठ उघडले तर मोठ्या प्रमाणात बँडविड्थ वापरली जाईल. प्रॉक्सी सर्व्हर एकदाच हे पृष्ठ डाउनलोड करून स्थानिकरित्या साठवू शकतो, त्यामुळे पुढील सर्व कर्मचारी प्रॉक्सी सर्व्हरमधूनच वेगाने पृष्ठ उघडू शकतात.प्रॉक्सी सर्व्हर सुरक्षेच्या दृष्टिकोनातून आणि कार्यक्षमतेच्या दृष्टीने खूप महत्त्वाचा ठरतो.

5.1.4 इंटरनेट ॲड्रेसिंग (Internet Addressing)

प्रत्येक संगणकाला एक इंटरनेट ऍ़ड्रेस असतो.इंटरनेटवर असलेल्या प्रत्येक संगणकाचा एक अद्वितीय (unique) पत्ता असतो**.** हा इंटरनेट ऍ़ड्रेस म्हणजे एक संख्या असते जी प्रत्येक संगणकाला ओळखते**.**

IPv4 आणि IPv6 चे स्वरूप

पूर्वी, सर्व इंटरनेट ऍ़ड्रेस 32-बिट मूल्यांचे होते**,** जे IPv4 (Internet Protocol version 4) म्हणून ओळखले जात होते. परंतु**,** नवीन IPv6 (Internet Protocol version 6) प्रणाली वापरली जात आहे, जी 128-बिट मूल्यांचा वापर करते**.**IPv6 मध्ये अधिक मोठे ऍ़ड्रेस स्पेस आहे**,** ज्यामुळे अधिक संगणक आणि उपकरणे इंटरनेटला जोडता येतात.IPv4 शी सुसंगत (backward compatible) आहे**,** त्यामुळे IPv4 प्रणाली असलेल्या उपकरणांसोबत IPv6 कार्य करू शकतो.सध्या IPv4 अधिक प्रमाणात वापरले जाते**,** पण भविष्यात IPv6 चा वापर वाढण्याची शक्यता आहे.

नेटवर्क क्लास (class) आणि त्यांचे प्रकार

IPv4 ऍ़ड्रेस अ, ब, क, ड आणि ई (A, B, C, D, E) अशा 5 वर्गांमध्ये विभागले जातात.

- Class A – मोठ्या नेटवर्कसाठी वापरला जातो.
- Class B – मध्यम आकाराच्या नेटवर्कसाठी.
- Class C – सामान्य इंटरनेट वापरकर्त्यांसाठी सर्वाधिक वापरला जातो.
  - Class C ऍ़ड्रेसचे पहिले बाइट 192 ते 224 दरम्यान असते.
  - एका Class C नेटवर्कमध्ये 256 संगणक (डिव्हाइसेस) कनेक्ट होऊ शकतात.
  - जगभरात अर्धा अब्ज उपकरणे Class C नेटवर्कमध्ये जोडली जाऊ शकतात.

5.2 Java आणि नेट (Java and the Net)

Java मध्ये TCP/IP नेटवर्किंगसाठी उत्कृष्ट समर्थन आहे. हे समर्थन स्ट्रीम I/O इंटरफेसला (stream I/O interface) विस्तार देऊन आणि नेटवर्कवरील I/O ऑब्जेक्ट्स तयार करण्यासाठी आवश्यक वैशिष्ट्ये जोडून मिळवले जाते.

5.3 InetAddress क्लास (InetAddress Class)

आपण फोन कॉल करतो, ई-मेल पाठवतो किंवा इंटरनेटवर कनेक्शन स्थापन करतो, तेव्हा पत्ते (addresses) महत्त्वाचे असतात**.** Java मध्ये `InetAddress` क्लास वापरून संख्यात्मक (numerical) IP पत्ता आणि डोमेन नेम साठवता येतो.

InetAddress क्लास म्हणजे काय?

- हा IP पत्ता आणि डोमेन नेम एकत्र साठवतो (encapsulate करतो)**.**
- आपण डोमेन नेम **(**`www.google.com`**)** वापरून IP पत्ता मिळवू शकतो**,** जो संख्यात्मक IP पत्त्याच्या तुलनेत समजण्यास सोपा असतो.
- `InetAddress` IPv4 आणि IPv6 दोन्ही हाताळू शकतो (Java 2, वर्जन 1.4 पासून).
- हा क्लास IP पत्त्याचा तपशील अंतर्गत व्यवस्थापित करतो, त्यामुळे वापरकर्त्याला फक्त डोमेन नेमवरून IP मिळवता येतो.

5.3.1 फॅक्टरी पद्धती (Factory Methods)

`InetAddress` क्लासचे कोणतेही कन्स्ट्रक्टर (constructor) दिसत नाहीत.यामुळे, `InetAddress` ऑब्जेक्ट तयार करण्यासाठी फॅक्टरी मेथड्स (Factory Methods) वापराव्या लागतात.फॅक्टरी मेथड ही एक प्रथा (convention) आहे, जिथे स्टॅटिक (`static`) मेथड्स एखाद्या क्लासचे ऑब्जेक्ट परत करतात.यामुळे अनेक प्रकारचे कन्स्ट्रक्टर ओव्हरलोड करण्याऐवजी वेगवेगळ्या नावांची मेथड्स वापरता येतात.

Table 5.1 InetAddress चे ३ प्रमुख फॅक्टरी मेथड्स

मेथड	वर्णन
static InetAddress getLocalHost() throws -UnknownHostException-	स्थानिक संगणकाचा (localhost) InetAddress ऑब्जेक्ट परत करते.
static InetAddress getByName(String hostName) -throws UnknownHostException-	दिलेल्या होस्ट नेमचा (hostName) IP पत्ता शोधते आणि InetAddress ऑब्जेक्ट परत करते.
static InetAddress[] getAllByName(String hostName) throws UnknownHostException-	एका होस्टच्या सर्व उपलब्ध IP पत्त्यांची यादी (`Array`) परत करते. काही डोमेनसाठी एकापेक्षा जास्त IP असतात.

उदाहरण (Example)

```
Import java.net.*;
class InetAddressTest
{
        public static void main(String args[]) throws UnknownHostException
        {
                InetAddress Address = InetAddress.getLocalHost ();
                system.out.println(Address);
                Address = InetAddress.getByName("google.com");
                System.out.println(Address);
                InetAddress SW[] =  InetAddress.getAllByName("www.yahoo.com");
                for (int i=0; i<SW.length; i++)
                system.out.println(SW[i]);
        }
}
```

5.3.2 InetAddress क्लासच्या इंस्टन्स (Instance) मेथड्स

InetAddress क्लासमध्ये काही इंस्टन्स मेथड्स उपलब्ध आहेत, ज्या InetAddress ऑब्जेक्ट्सवर (Objects) लागू करता येतात.खालील काही सर्वाधिक वापरल्या जाणाऱ्या (commonly used) मेथड्स दिल्या आहेत.

1. boolean equals(Object other)

हे दोन InetAddress ऑब्जेक्ट्स समान आहेत का ते तपासते. जर दोन्ही ऑब्जेक्ट्सचे IP पत्ते सारखे असतील, तर true परत करते, अन्यथा false परत करते.

उदाहरण

```
InetAddress addr1 = InetAddress.getByName ("www.google.com");
InetAddress addr2 = InetAddress.getByName ("www.google.com");
System.out.println (addr1.equals (addr2)); // true (कारण दोन्ही IP सारखे आहेत)
```

2. Byte [] getAddress ()

IP पत्ता byte अ‍ॅरे (array) स्वरूपात परत करते. हा पत्ता नेटवर्क बाइट ऑर्डरमध्ये (network byte order) असतो.

उदाहरण

```
InetAddress addr = InetAddress.getByName ("www.google.com");
byte [] ip = addr.getAddress();
```

3. String getHostAddress ()

IP पत्ता String स्वरूपात (dotted decimal format) परत करते.

उदाहरण

```
InetAddress addr = InetAddress.getByName ("www.google.com");
System.out.println ("Google चा IP: " + addr.getHostAddress ());
```

4. String getHostName ()

होस्टचे नाव (Host Name) String स्वरूपात परत करते.

जर डोमेन नाव नसेल, तर IP पत्ता परत केला जातो.

उदाहरण

```
InetAddress addr = InetAddress.getByName ("142.250.183.36");
System.out.println ("Host Name: " + addr.getHostName ());
```

5. boolean isMulticastAddress()

IP पत्ता Multicast आहे का नाही, हे तपासते.

जर Multicast Address असेल, तर true, अन्यथा false परत करते.

उदाहरण:

```
InetAddress multicastAddr = InetAddress.getByName ("224.0.0.1");
System.out.println ("Multicast आहे का? " + multicastAddr.isMulticastAddress ());
```

6. String toString()

IP पत्ता आणि होस्ट नेम String स्वरूपात परत करते.

उदाहरण:

```
InetAddress addr = InetAddress.getByName("www.google.com");
System.out.println (addr.toString ());
```

5.4 TCP/IP सॉकेट्स (TCP/IP Sockets)

TCP/IP सॉकेट्स हे इंटरनेटवरील विश्वसनीय (Reliable), द्विदिशात्मक (Bidirectional), कायमस्वरूपी (Persistent), आणि Point-to-Point कनेक्शन प्रदान करतात. सॉकेटचा उपयोग Java च्या I/O प्रणालीला (Input/Output System) इतर प्रोग्रॅम्सशी (Programs) जोडण्यासाठी केला जातो.हे प्रोग्रॅम्स स्थानिक संगणकावर (Local Machine) किंवा कोणत्याही इंटरनेटवरील संगणकावर (Remote Machine) असू शकतात.

TCP/IP सॉकेट कनेक्शनचे वैशिष्ट्ये:

- विश्वसनीय (Reliable) **–** डेटाचा सुरक्षित आणि शुद्ध (error-free) प्रवाह सुनिश्चित करतो**.**
- द्विदिशात्मक (Bidirectional) – डेटा प्रेषित (Send) आणि ग्रहण (Receive) दोन्ही करू शकतो.
- कायमस्वरूपी (Persistent) – एकदा कनेक्शन स्थापन झाल्यावर ते टिकून राहते.
- Point-to-Point संप्रेषण – दोन संगणकांदरम्यान थेट डेटा ट्रान्सफर करतो.
- Stream-Based – डेटा Byte Stream स्वरूपात पाठवला जातो.
- फ्लो कंट्रोल (Flow Control)-डेटा पाठवणाऱ्या आणि स्वीकारणाऱ्या सिस्टिम्समधील स्पीड संतुलित ठेवतो.
- पोर्ट-आधारित कम्युनिकेशन (Port-Based Communication)-TCP/IP सॉकेट प्रत्येक सेवेकरिता पोर्ट नंबर वापरतो (उदा. HTTP साठी पोर्ट 80, FTP साठी पोर्ट 21).

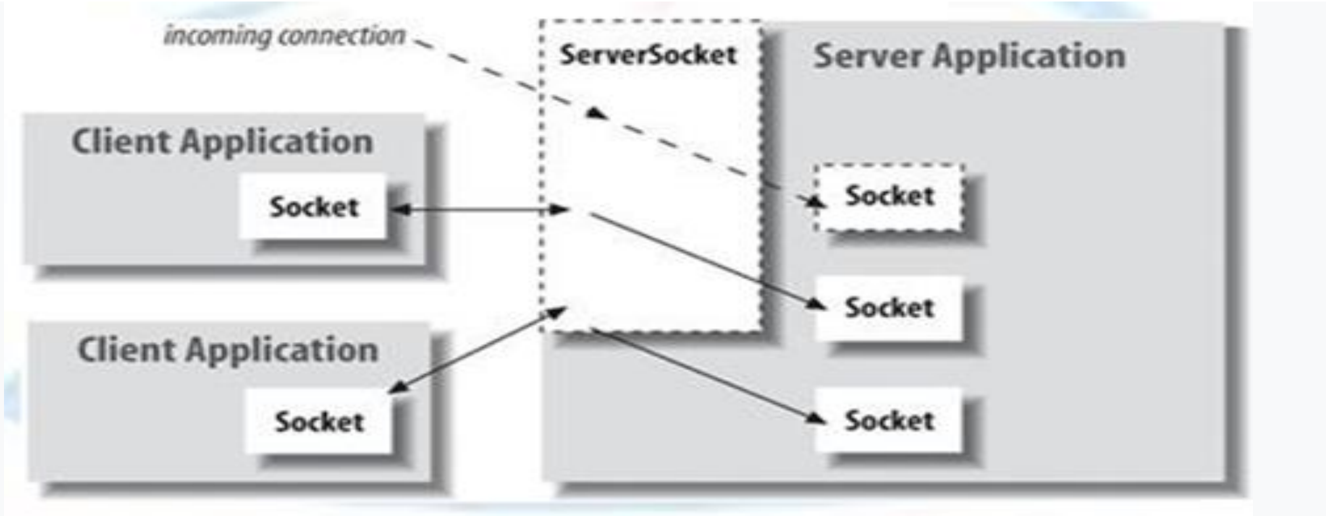


Fig 5.: **TCP/IP सॉकेट्स (TCP/IP Sockets)**

5.4.1 TCP/IP क्लायंट सॉकेट्स

Socket ऑब्जेक्ट तयार केल्याने स्वयंचलितपणे (implicitly) क्लायंट आणि सर्व्हर दरम्यान कनेक्शन स्थापित होते.यासाठी कोणत्याही पद्धती (`methods`) किंवा कन्स्ट्रक्टर्स (constructors) द्वारे स्पष्ट (explicitly) कनेक्शन स्थापन करण्याचा तपशील दिला जात नाही.

`Socket` क्लासचे कन्स्ट्रक्टर्स

1. Socket(String hostName, int port)
   - दिलेल्या होस्टनेम (hostName) आणि पोर्ट (port) वर सॉकेट कनेक्ट करते.
   - UnknownHostException किंवा IOException थ्रो करू शकते.
2. Socket(InetAddress ipAddress, int port)
   - InetAddress आणि port चा वापर करून सॉकेट तयार करते.
   - IOException थ्रो करू शकते

- **`Socket` क्लासचे महत्त्वाचे मेथड्स**

1. InetAddress getInetAddress()
   - संबंधित InetAddress परत करते.
2. int getPort()
   - संबंधित रिमोट पोर्ट (remote port) परत करते, जिथे Socket कनेक्ट आहे.
3. int getLocalPort()
   - संबंधित स्थानिक पोर्ट (local port) परत करते, जिथे Socket कनेक्ट आहे.

- **`InputStream` आणि `OutputStream` मिळवण्यासाठी मेथड्स**

एकदा `Socket` ऑब्जेक्ट तयार झाल्यावर डेटा पाठवण्यासाठी (`send`) आणि प्राप्त (`receive`) करण्यासाठी खालील मेथड्स वापरता येतात:

1. InputStream getInputStream()–संबंधित InputStream परत करते.
2. OutputStream getOutputStream()-संबंधित OutputStream परत करते.

उदाहरण:

```
Import java.net.*;
Import java.io.*;
Public class LowPortScanner
{
 Public static void main (String [] args)  {
        String host = "localhost";
```

```
        for (int i = 1; i < 1024; i++)
        {
        try {
        Socket s = new Socket(host, i);
system.out.println("There is a server on port " + i  + "        of " + host);
    }
    catch (UnknownHostException ex)   {
     System.err.println (ex);        break;       }
    catch (IOException ex)   {
      // must not be a server on this port
    }
   } // end for
  }  // end main
}  // end PortScanner
```

5.4.2 TCP/IP सर्व्हर सॉकेट्स

Java मध्ये सर्व्हर ॲप्लिकेशन्स (server applications) तयार करण्यासाठी वेगळा सॉकेट क्लास वापरावा लागतो. ServerSocket क्लासचा वापर स्थानीय (local) किंवा दूरस्थ (remote) क्लायंट प्रोग्रॅम्ससाठी सर्व्हर तयार करण्यासाठी केला जातो, जे प्रकाशित (published) पोर्ट्सवर कनेक्शनसाठी वाट पाहतात.

Server Scoket आणि Socket मधील महत्त्वाचा फरक

- ServerSocket तयार केल्यावर, सिस्टमला क्लायंट कनेक्शनमध्ये रस आहे असे नोंदवते (register).
- ServerSocket च्या कन्स्ट्रक्टर्समध्ये, कोणत्या पोर्टवर कनेक्शन स्वीकारायचे आहे आणि किती मोठी queue ठेवायची आहे हे निर्धारित करता येते.
- डिफॉल्ट queue लांबी (length) 50 आहे. याचा अर्थ, 50 क्लायंट्सपर्यंतच्या रिक्वेस्ट्स pending राहू शकतात, त्यानंतर नवीन कनेक्शन्स नाकारले जातील.

ServerSocket क्लासचे कन्स्ट्रक्टर्स

1. ServerSocket(int port) throws BindException, IOException
- दिलेल्या port वर सर्व्हर सॉकेट तयार करते.
- डिफॉल्ट queue लांबी 50 असते.
- BindException किंवा IOException थ्रो करू शकते.
2. ServerSocket(int port, int maxQueue) throws BindException, IOException
- दिलेल्या port वर सर्व्हर सॉकेट तयार करते आणि
- दिलेल्या maxQueue (queue लांबी) मध्ये क्लायंट कनेक्शन pending ठेवू शकते.
3. ServerSocket(int port, int maxQueue, InetAddress localAddress) throws IOException
- दिलेल्या `port` वर सर्व्हर सॉकेट तयार करते.
- दिलेल्या `maxQueue` मध्ये `pending` क्लायंट कनेक्शन्स ठेवते.
- localAddress वापरून सॉकेट विशिष्ट IP address वर bind केले जाते (जर होस्टला एकापेक्षा जास्त `IP` addresses असतील, तर कोणत्या IP वर सर्व्हर सुरू करायचा ते ठरवू शकतो).

accept() मेथड

- accept() ही ब्लॉकिंग कॉल आहे
- ती क्लायंटशी कनेक्शन स्थापित होईपर्यंत (`waiting for connection`) थांबते.
- कनेक्शन मिळाल्यावर एक `Socket` ऑब्जेक्ट परत करते, जो सर्व्हर-क्लायंट संवादासाठी वापरला जातो.

उदाहरण

```
import java.net.*;
import java.io.*;
public class LocalPortScanner
{
  public static void main(String[] args)
  {
    for (int port = 1; port <= 65535; port++)
    {       try
      {
        // the next line will fail and drop into the catch block if
        // there is already a server running on the port
        ServerSocket server = new ServerSocket(port);
      }
      catch (IOException ex)
      {
        System.out.println("There is a server on port " + port  + ".");
      } // end catch
    } // end for
  }
}
```

उदाहरण:

```
/*
क्लायंट आणि सर्व्हर यांच्यात संवाद तयार करा. क्लायंट संवादासाठी पोर्ट क्रमांक 3642 वर विनंती पाठवेल. त्यानंतर सर्व्हर पासवर्ड विचारेल. जोपर्यंत पासवर्ड बरोबर नाही तोपर्यंत संप्रेषण स्थापित केले जाणार नाही. योग्य पासवर्ड टाकल्यानंतर, सर्व्हर पाठवेल: "स्वागत आहे" आणि संप्रेषण बंद करा.
*/
import java.io.*;
import java.net.*;
class PwdClient
{
  public static void main(String args[])
  {
    try
         {
          Socket so =new Socket("localhost",3642);
          DataInputStream d =new DataInputStream(System.in);
          System.out.println("Enter a Password");
          String passwd =d.readLine();
          PrintStream p = new PrintStream(so.getOutputStream());
          p.println(passwd);
          DataInputStream d1 =new DataInputStream(so.getInputStream());
           String r =d1.readLine();
          System.out.println(r);
```

```
            so.close();
           }
           catch( Exception e)
           {
            System.out.println("Msg from client: "+e);
           }
  }
}
\\ TCP/IP Server program
import java.net.*;
import java.io.*;
public class PwdServer
{
  public static void main(String [] args)
  {
    try
           {
            PWServer p =new PWServer();
          System.out.println("Server Started ...");
            p.activate();
           }
           catch(Exception e)
           {
            System.out.println("Error: InSocket");
           }
  }
}
class PWServer
{
  public void activate()throws Exception
  {
    ServerSocket ss=new ServerSocket(3642);
           while(true)
           {
            Socket so= ss.accept();
            DataInputStream d =new DataInputStream (so.getInputStream());
            String passwd=d.readLine();
            String response;
          if(passwd. Equals("mahesh"))
            {
           response="welcome.....";
              System.out.print(" LOGIN SUCCESSFUL.....\n");
                     }
                     else
```

```
                {
                  response ="Invalid Password";
                }
                PrintStream p =new PrintStream(so.getOutputStream());
                p.println(response);
                so.close();
            }
          }
  }
```

5.5 URL क्लास

वेब (Web) म्हणजे विविध उच्च-स्तरीय प्रोटोकॉल्स (protocols) आणि फाईल स्वरूपांचे (file formats) एक संकलन आहे, जे वेब ब्राउझर (web browser) मध्ये केले जाते.वेबच्या सर्वात महत्त्वाच्या पैलूंपैकी एक म्हणजे टिम बर्नर्स-ली (Tim Berners-Lee) यांनी नेटवरील सर्व संसाधन (resources) लोकेट करण्यासाठी एक स्केलेबल पद्धत तयार केली.

याच समस्येचे समाधान म्हणजे URL (Uniform Resource Locator)

URL ही नेटवरील माहिती ओळखण्यासाठी आणि तिला युनिक (unique) पद्धतीने ॲड्रेस (address) करण्यासाठी वापरली जाणारी संरचना आहे.आजच्या घडीला, वेब हा इंटरनेटवर कार्यरत असलेला संसाधनांचा (resources) संच आहे, जो URL आणि HTML च्या माध्यमातून ॲक्सेस केला जातो.

- URL स्वरूप (URL Format)

URL च्या उदाहरणे:

1. http://www.rediff.com/
2. http://www.rediff.com:80/index.htm/

URL हे चार मुख्य घटकांवर आधारित असते:

घटक (Component)	**वर्णन** (Description)
1.प्रोटोकॉल (Protocol)	- http, https, ftp, file इत्यादी वापरले जातात. - उदाहरणः http:
2.होस्ट नाव (Host Name) किंवा IP पत्ता	- डावीकडे // आणि उजवीकडे / किंवा : द्वारे वेगळे केले जाते. - उदाहरणः //www.rediff.com
3.पोर्ट क्रमांक (Port Number) (पर्यायी)	- होस्ट नावानंतर : ने वेगळा केला जातो. - HTTP साठी डिफॉल्ट port 80 असतो. - उदाहरणः :80
4.फाईल पाथ (File Path)	- मागे / ने वेगळे केले जाते. - जर एखाद्या डिरेक्टरीला URL दिली असेल, तर index.html किंवा index.htm हे डिफॉल्ट पथ (default path) ठरते. - उदाहरणः /index.htm

उदाहरणे

http://www.rediff.com/ आणि http://www.rediff.com/index.htm हे समान आहेत.

URL **क्लासच्या कन्स्ट्रक्टर्स** (Constructors of URL Class)

Java मध्ये URL क्लास वापरण्यासाठी काही ठराविक कन्स्ट्रक्टर्स उपलब्ध आहेत.URL क्लासच्या कोणत्याही कन्स्ट्रक्टरमध्ये MalformedURLException थ्रो होऊ शकतो.

1. URL(String urlSpecifier)

- एक पूर्ण URL (String) स्वीकारतो आणि त्यापासून URL ऑब्जेक्ट तयार करतो.
- उदाहरण: URL url = new URL("http://www.rediff.com/");

2.URL(String protName, String hostName, int port, String path)

- स्वतंत्र प्रोटोकॉल (protocol), होस्ट (host), पोर्ट (port) आणि फाईल पथ (path) स्वीकारतो.
- उदाहरण: URL url = new URL("http", "www.rediff.com", 80, "/index.htm")

3.URL(String protName, String hostName, String path)

- पोर्ट नंबर (port) वगळून फक्त प्रोटोकॉल (protocol), होस्ट (host) आणि पथ (path)स्वीकारतो.
- उदाहरण: URL url = new URL("http", "www.rediff.com", "/index.htm");

4.URL(URL urlObj, String urlSpecifier)

- एका URL ऑब्जेक्टचा (URL object) संदर्भ (reference) घेऊन नवीन URL तयार करण्यासाठी वापरला जातो.
- हे विशेषतः संबंधित (relative) URLs तयार करताना उपयुक्त आहे.
- उदाहरण: URL base = new URL("http://www.rediff.com/")

उदाहरण

```
import java.net.*;
class URLDemo
{
 public static void main(String args[]) throws MalformedURLException
   {
     URL hp = new URL("http://contentind.cricinfo.com/ci/content/current/story/news.html");
      System.out.println("Protocol: " + hp.getProtocol());
      System.out.println("Port: " + hp.getPort());
      System.out.println("Host: " + hp.getHost());
      System.out.println("File: " + hp.getFile());
      System.out.println("Ext:" + hp.toExternalForm());
   }
}
```

Output

```
Protocol: http
Port: -1
Host: content-ind.cricinfo.com
File: /ci/content/current/story/news.html
Ext:http://content-ind.cricinfo.com/ci/content/current/story/news.html
```

5.6 URLConnection (यूआरएल कनेक्शन)

URLConnection ही abstract क्लास आहे, जी एखाद्या URL द्वारे निर्दिष्ट (specified) केलेल्या संसाधनाशी (resource) सक्रिय (active) कनेक्शन दर्शवते.URL क्लासच्या तुलनेत, URLConnection सर्व्हर (server) शी संवाद (interaction) साधण्यावर अधिक नियंत्रण (control) देते.हे विशेषतः HTTP सर्व्हरसाठी (HTTP Server) अधिक फायदेशीर आहे.

URLConnection चे कार्य (Functions of URLConnection)

- सर्व्हरकडून पाठवलेल्या हेडर (Header) ची तपासणी (Inspect) करणे आणि योग्य प्रतिसाद (Respond) देणे.
- क्लायंटच्या विनंती (Request) मध्ये Header फील्ड सेट (Set) करणे.
- Binary फाईल्स डाउनलोड (Download) करणे.
- वेब सर्व्हरला POST, PUT किंवा अन्य HTTP विनंती पद्धती (Request Methods) द्वारे डेटा (Data) पाठवणे.

➢ URLConnection क्लासचा कन्स्ट्रक्टर (**Constructor of URLConnection**)

1. protected URLConnection(URL url);

हा कन्स्ट्रक्टर protected असल्यामुळे, थेट URLConnection ऑब्जेक्ट तयार करता येत नाही.याचा अर्थ, आपण openConnection() पद्धती (Method) चा वापर करूनच URLConnection मिळवू शकतो.

उदाहरण (Example) - URLConnection वापरून डेटा वाचणे

```
import java.net.*;
import java.io.*;
import java.util.*;
public class HeaderViewer
{
public static void main(String args[])
{
try
{
URL u = new URL("http://www.rediffmail.com/index.html");
URLConnection uc = u.openConnection( );
System.out.println("Content-type: " +uc.getContentType( ));
System.out.println("Content-encoding: "+ uc.getContentEncoding( ));
System.out.println("Date: " + new Date(uc.getDate( )));
System.out.println("Last modified: "+ new Date(uc.getLastModified( )));
System.out.println("Expiration date: "+ new Date(uc.getExpiration( )));
System.out.println("Content-length: " +uc.getContentLength( ));
} // end try
catch (MalformedURLException ex)
{
System.out.println("I can't understand this URL...");
}
catch (IOException ex)
{
System.err.println(ex);
}
System.out.println( );
} // end main
}
```

URL आणि URLConnection मधील फरक या उदाहरणाप्रमाणे फक्त साध्या इनपुट प्रवाहाने स्पष्ट होत नाहीत. दोन वर्गांमधील सर्वात मोठे फरक आहेत

1. URLConnection HTTP हेडरमध्ये प्रवेश प्रदान करते.
2. URLConnection सर्व्हरला पाठवलेली विनंती पॅरामीटर्स कॉन्फिगर करू शकते.

URLConnection सर्व्हरवर डेटा लिहू शकतो तसेच सर्व्हरवरील डेटा वाचू शकतो.

5.7 युजर डेटाग्राम प्रोटोकॉल (UDP) - User Datagram Protocol

UDP (User Datagram Protocol) हा IP वर डेटा पाठवण्यासाठी एक पर्यायी (Alternative) प्रोटोकॉल आहे, जो अतिशय वेगवान (Fast) असतो पण विश्वसनीय (Reliable) नसतो.

1. UDP द्वारे पाठवलेला डेटा पोहोचला की नाही, याची खात्री नसते.
2. पाठवलेले डेटा तुकडे (Packets) क्रमाने (Order) पोहोचतीलच असेही निश्चित नाही.
3. **3.** मात्र, जो डेटा पोहोचतो तो तुलनेने जास्त वेगाने (Faster) पोहोचतो.

TCP आणि UDP मधील तुलना (**TCP vs UDP**)

◆ **TCP (Transmission Control Protocol)**	◆ **UDP (User Datagram Protocol)**
फोन सिस्टमसारखा (Phone System) आहे.	पोस्ट ऑफिससारखा (**Post Office**) आहे.
Connection-Oriented प्रोटोकॉल आहे.	Connection-Less प्रोटोकॉल आहे.
विश्वसनीय (Reliable) आहे.	विश्वसनीय नाही (Unreliable).
डेटा क्रमाने (Ordered) पोहोचतो.	डेटा क्रमशः पोहोचेल याची खात्री नाही.
डेटा गहाळ (Lost) झाल्यास तो पुन्हा पाठवला जातो.	गहाळ झालेला डेटा पुन्हा पाठवला जात नाही.
Flow Control आणि Error Control पुरवतो.	Flow Control आणि Error Control नाही.
Multicasting आणि Broadcasting समर्थन करत नाही.	Multicasting आणि Broadcasting समर्थन करते.
TCP चे Packet म्हणजे Segment.	UDP चे Packet म्हणजे User Datagram.
पूर्ण Duplex (दोन दिशेने संवाद) समर्थन करतो.	पूर्ण Duplex समर्थन करत नाही.

5.7.2 DatagramPacket (डेटाग्राम पॅकेट)

DatagramPacket वेगवेगळ्या Constructors वापरतो, हे पॅकेट डेटा पाठवण्यासाठी **(**Send**)** किंवा प्राप्त (Receive) करण्यासाठी वापरले जाणार आहे की नाही यावर अवलंबून असते.हे थोडं असामान्य आहे, कारण सहसा कन्स्ट्रक्टर्स **(**Constructors**)** ओव्हरलोड (Overload**)** करणे म्हणजे विविध प्रकारची माहिती प्रदान करून समान प्रकारचे ऑब्जेक्ट **(**Object**)** तयार करणे होय. पण इथे, समान वर्ग (Class) वापरून वेगवेगळ्या संदर्भात (Context) ऑब्जेक्ट तयार करण्यासाठी ओव्हरलोडिंग (Overloading) केली जाते.या सर्व Constructors मध्ये एक byte ऍरे **(**Array**)** असतो, जो डेटाग्राम **(**Datagram**)** चा डेटा **(**Data**)** ठेवतो**.** तसेच, आपण या byte ऍरे मधून किती बाइट (Bytes) डेटा वापरणार आहोत हे देखील ठरवू शकतो.

डेटाग्राम प्राप्त (Receiving) करण्यासाठी Constructors

हे दोन कन्स्ट्रक्टर्स **(**Constructors**)** नवीन DatagramPacket ऑब्जेक्ट तयार करतात, जे नेटवर्कवरून डेटा प्राप्त (Receive) करण्यासाठी वापरले जाते:

1. public DatagramPacket(byte[] buffer, int length)
2. public DatagramPacket(byte[] buffer, int offset, int length)

डेटा प्राप्त (Receive**) करताना:**

- जेव्हा सॉकेट (Socket) कोणतंही डेटाग्राम (Datagram) प्राप्त (Receive) करते, तेव्हा डेटा buffer मध्ये संग्रहित (Store) केला जातो.
- हा डेटा buffer[0] पासून साठवला जातो आणि संपूर्ण पॅकेट साठवलं जाईपर्यंत किंवा length बाइट्स (Bytes) भरले जाईपर्यंत चालू राहतो.
- दुसरा कन्स्ट्रक्टर (Constructor) वापरल्यास, डेटा buffer[offset] पासून संग्रहित केला जातो.
- length हा buffer.length - offset च्या बरोबरी किंवा कमी असावा.
- जर length चा आकार buffer पेक्षा जास्त असेल, तर IllegalArgumentException (Runtime Exception) फेकला जातो.

उदाहरण

खालील कोड 8192 बाइट्स (Bytes) पर्यंत डेटा प्राप्त (Receive) करण्यासाठी DatagramPacket तयार करतो:

```
byte[] buffer = new byte[8192];
DatagramPacket dp = new DatagramPacket(buffer, buffer.length)
```

डेटाग्राम पाठवण्यासाठी (Sending) Constructors

हे दोन कन्स्ट्रक्टर्स **(**Constructors**)** नवीन DatagramPacket तयार करतात, जे डेटा पाठवण्यासाठी (Send) वापरले जाते:

1. public DatagramPacket(byte[] data, int length, InetAddress destination, int port)
2. public DatagramPacket(byte[] data, int offset, int length, InetAddress destination, int port)

डेटा पाठवताना (Send) काय होते?

- data ऍरे मधून length बाइट्स (Bytes) पॅकेटमध्ये टाकले जातात.
- offset नसल्यास, डेटा शून्य (0) पासून सुरू होतो.
- जर `length` हा `data.length` पेक्षा जास्त असेल, तर `IllegalArgumentException` फेकला जातो.
- InetAddress destination म्हणजे ज्या होस्ट (Host) ला पॅकेट पाठवायचं आहे त्याचा पत्ता.
- port म्हणजे त्या होस्टवरील पोर्ट नंबर.

```
//Construct a DatagramPacket to receive data
import java.net.*;
public class DatagramExample1
{
 public static void main(String[] args)
 {
  String s = "This is a test.";
  byte[] data = s.getBytes( );     try
  {
   InetAddress ia = InetAddress.getByName("www.ibiblio.org");
   int port = 7;
   DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
   System.out.println("This packet is addressed to " + dp.getAddress( ) + " on port " +
   dp.getPort());
   System.out.println("There are " + dp.getLength( ) + " bytes of data in the packet");
System.out.println( new String(dp.getData( ), dp.getOffset( ), dp.getLength( )));
  }
  catch (UnknownHostException e)
  {
   System.err.println(e);
  }
 }
}
```

Output:

This packet is addressed to www.ibiblio.org/152.2.254.81 on port 7

There are 15 bytes of data in the packet This is a test.

5.7.3 DatagramSocket (डेटाग्राम सॉकेट)

UDP DatagramPacket पाठवण्यासाठी (Send) किंवा प्राप्त (Receive) करण्यासाठी DatagramSocket उघडणे आवश्यक असते.जावामध्ये DatagramSocket हे DatagramSocket वर्ग (Class) द्वारे तयार (Create) आणि प्रवेश (Access) केले जाते.

DatagramSocket चे कन्स्ट्रक्टर्स (Constructors)

हे तीन कन्स्ट्रक्टर्स DatagramSocket तयार करतात:

1. public DatagramSocket() throws SocketException
   - अज्ञात (Anonymous) पोर्टवर सॉकेट तयार करते.
2. public DatagramSocket(int port) throws SocketException
   - दिलेल्या पोर्टवर (Port) UDP डेटाग्राम ऐकणारे (Listening) सॉकेट तयार करते.

3. public DatagramSocket(int port, InetAddress interface) throws SocketException
   - विशिष्ट नेटवर्क इंटरफेस (Network Interface) आणि पोर्टवर (Port) सॉकेट तयार करते.
   - Multihomed Hosts (ज्यांच्याकडे एकाहून अधिक IP Addresses आहेत) साठी उपयुक्त.

DatagramSocket वापरून DatagramPacket पाठवणे आणि प्राप्त करणे (Send & Receive Datagrams)UDP द्वारे डेटा पाठवण्यासाठी **(**Send**)** आणि प्राप्त **(**Receive**)** करण्यासाठी DatagramSocket वापरले जाते**.**एकाच DatagramSocket द्वारे एकाधिक (Multiple) होस्ट (Hosts) सोबत संवाद साधता येतो.

DatagramPacket Sending UDP Packets

public void send(DatagramPacket dp) throws IOException

DatagramPacket प्राप्त (Receive) करण्यासाठी

public void receive(DatagramPacket dp) throws IOException

उदाहरण

UDP Client

```
import java.net.*;
import java.io.*;
public class UDPDiscardClient
{
public static void main(String[] args)
{
String hostname = "localhost";
int port = 9;
try
{
InetAddress server = InetAddress.getByName(hostname);
BufferedReader userInput= new BufferedReader(new InputStreamReader(System.in));
DatagramSocket theSocket = new DatagramSocket( );
while (true)
{
String theLine = userInput.readLine( );
if (theLine.equals(".")) break;
byte[] data = theLine.getBytes( );
DatagramPacket theOutput
= new DatagramPacket(data, data.length, server, port);
theSocket.send(theOutput);
} // end while
} // end try
catch (UnknownHostException uhex) {
System.err.println(uhex);
}
catch (SocketException se) {
System.err.println(se);
}
catch (IOException ioex) {
System.err.println(ioex);
}
```

```
} // end main
}
```

UDP Server

```
import java.net.*;
import java.io.*;
public class UDPDiscardServer
{
public static void main(String[] args)
{
int port = 9;
byte[] buffer = new byte[65507];
try
{
DatagramSocket server = new DatagramSocket(port);
DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
while (true)
{
try
{
server.receive(packet);
String s = new String(packet.getData( ),
0, packet.getLength( ));
System.out.println(packet.getAddress( ) + " at port "+ packet.getPort( ) + " says " + s);
// reset the length for the next packet
packet.setLength(buffer.length);
}
catch (IOException ex) {
System.err.println(ex);
}
} // end while
} // end try
catch (SocketException ex)
{
System.err.println(ex);
} // end catch
} // end main
}
```

संदर्भ (Web References)

1. https://www.geeksforgeeks.org/
2. https://www.w3schools.blog/
3. https://www.javatpoint.com/example
4. https://www.tutorialspoint.com/java/java_networking.htm

युनिट- 6
डेटाबेसशी संवाद
(Interacting with Database)

विषय निष्पत्ती (Course Outcome):

CO 6- डेटाबेस व्यवस्थापित करण्यासाठी जावा प्रोग्राम विकसित करा.

घटक निष्पत्ती (Theory Learning Outcomes):

1. डेटाबेस व्यवस्थापित करण्यासाठी जावा प्रोग्राम विकसित करा.
2. जेडीबीसी (JDBC) च्या दोन स्तरीय आणि तीन स्तरीय आर्किटेक्चरचे वर्णन करा.
3. जावा प्रोग्रामिंग संबंधित जेडीबीसी ड्रायव्हर निवडा.
4. डेटाबेसशी कनेक्टिव्हिटी स्थापन करण्यासाठी लागणाऱ्या पायऱ्या (steps) उदाहरणासहित विस्तृतपणे स्पष्ट करा.

6.1 जेडीबीसीचा परिचय (Introduction to JDBC, ODBC)

जावा डेटाबेस कनेक्टिव्हीटी (Java Database Connectivity) ही थोडक्यात JDBC म्हणून ओळखली जाणारी एक Java API (Application Programming Interface) आहे जी जावा प्रोग्रॅमला ला SQL स्टेटमेंट्स कार्यान्वित (execute) करण्यास सक्षम करते. ही एक अँप्लिकेशन प्रोग्रामिंग इंटरफेस (Application Programming Interface) आहे जी ठरवते की जावा डेव्हलपर जावा कोड चा वापर करून डेटाबेस ला सारणीबद्ध (tabular format)मध्ये कसा ऍक्सेस (access) करू शकतो. यासाठी जावा प्रोग्रामिंग मध्ये लिहिलेल्या स्टँडर्ड इंटरफेसेस (standard interfaces)आणि classes चा वापर केला जातो. JDBC रिलेशनल डेटाबेस मॅनेजमेंट सिस्टम (RDBMS) जसे की SQL Server, Oracle, MySQL, MS Access इत्यादींमध्ये डेटा क्वेरी (query) आणि अपडेट करण्यासाठी आवश्यक पध्दती **(**methods) प्रदान करते.JDBC विविध प्लॅटफॉर्मवर जावा सोबत कार्य करते, जसे की Windows, Mac OS आणि विविध UNIX आवृत्त्या.

JDBC लाइब्रेरी खालीलप्रमाणे डेटाबेस वापराशी संबंधित कार्यांसाठी आवश्यक APIs समाविष्ट करते.

थोडक्यात, JDBC प्रोग्रामर्सना जावा अँप्लिकेशन (java applicaltion) लिहिण्यास मदत करते, ज्या खालील तीन प्रोग्रामिंग ॲक्टिव्हिटी व्यवस्थापित करतात:

1. डेटाबेस कनेक्शन स्थापित करणे
2. एसक्यूएल क्वेरीज (SQL Queries) पाठवणे आणि त्यावर प्रक्रिया करणे
3. डेटाबेस मधील डेटा अपडेट करणे आणि व्यवस्थापित करणे

6.1.1 जेडीबीसी आर्किटेक्चर (JDBC Architecture)

जेडीबीसी आर्किटेक्चर दोन स्तरांमध्ये विभागलेले आहे:

1. JDBC API: हे जावा अँप्लिकेशन आणि डेटाबेस मॅनेजर यांच्यातील कनेक्शन प्रदान करते.
2. JDBC Driver API – हे JDBC Manager आणि Driver यांच्यातील कनेक्शनला सपोर्ट देते.

JDBC API Driver Manager आणि डेटाबेस-विशिष्ट Drivers चा वापर करून विविध डेटाबेसशी संपर्क (connectivity) साधते. JDBC Driver Manager हे JDBC आर्किटेक्चरचे मुख्य घटक (backbone) आहे, जे योग्य driver निवडून डेटासोर्स ला योग्यरित्या ऍक्सेस करण्यास मदत करते. Driver Manager एकाच वेळी विविध डेटाबेससह कार्य करणाऱ्या एकापेक्षा अधिक ड्रायव्हर्सना सपोर्ट करू शकतो.

JDBC (Java Database Connectivity) आर्किटेक्चर हे Java Application आणि Database यांच्यातील कनेक्शनसाठी वापरले जाते.

- **JDBC आर्किटेक्चरचे घटक:**
  1. Java Application (जावा अनुप्रयोग)
  2. JDBC API (डेटाबेसशी जोडणारा इंटरफेस)
  3. JDBC Driver (डेटाबेसशी जोडणारा ड्रायव्हर)
  4. Database (डेटाबेस जिथे डेटा संग्रहित केला जातो)

खालील आर्किटेक्चरल डायग्राम Java application, JDBC drivers, आणि Driver Manager यांचे स्थान स्पष्ट करते.

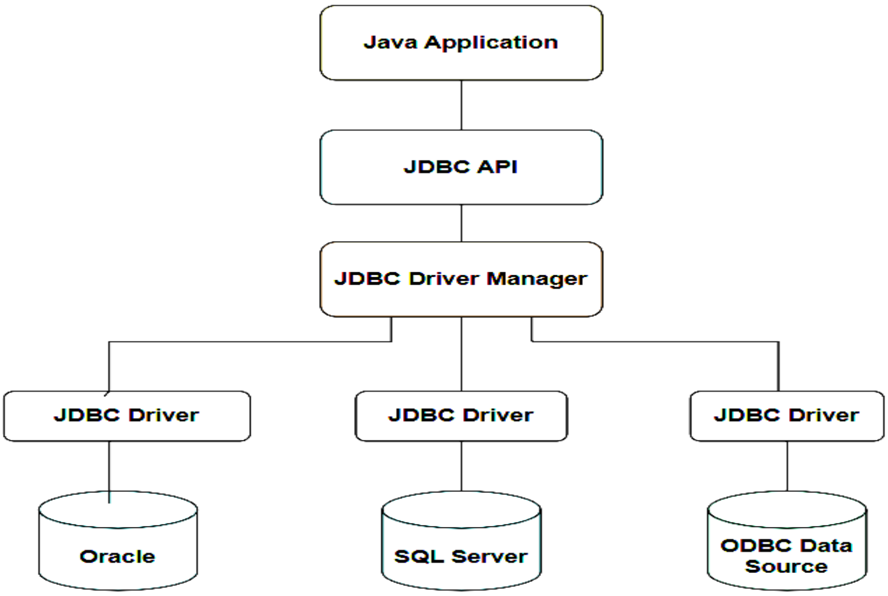


Fig 6.1: **जेडीबीसी आर्किटेक्चर (JDBC Architecture)**

6.1.2 जेडीबीसी मॉडेल्स (JDBC Models)

JDBC API दोन प्रकारच्या प्रोसेसिंग मॉडेल्स (processing models) ला समर्थन देते.

6.1.2.1 दोन-स्तरीय मॉडेल (Two-Tier Model) (Client-Server Architecture)

या मॉडेल मध्ये जावा ॲप्लिकेशन थेट डेटाबेस सर्वर (database server) शी जोडते.

JDBC driver वापरून SQL queries डेटाबेसवर पाठवल्या जातात. डेटाबेस सर्वर त्या queries वर प्रक्रिया करून परिणाम (results) परत पाठवतो.

महत्वाचे मुद्दे:

डेटाबेसशी थेट संवाद साधण्यासाठी योग्य JDBC driver आवश्यक असतो. युजर कमांड (User commands) डेटाबेसला पाठवले जातात आणि तिथून results परत येतात.डेटासोर्स (Database source) वेगळ्या मशीनवर असू शकतो, आणि नेटवर्कद्वारे जोडला जातो. ही प्रणाली Client-Server Architecture म्हणून ओळखली जाते.

Client: वापरकर्ता संगणक ला क्लाईन्ट (Client) मशीन असे संबोधले जाते.

Server: ज्या संगणकावर डेटाबेस प्रस्थापित केलेला आहे अश्या संगणकाला सर्व्हर म्हणून संबोधले जाते

क्लायंट आणि सर्वर संगणक नेटवर्क द्वारे जोडलेले असतात

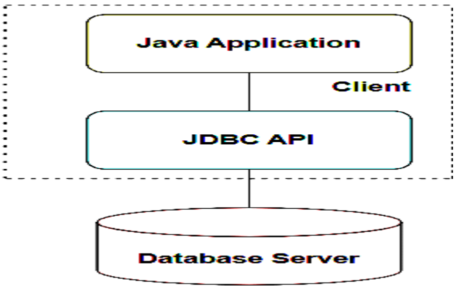


Fig 6.1.2.1: दोन-स्तरीय मॉडेल (Two-Tier Model)

फायदे (Advantages):

1. सोपे आणि सहज वापरण्याजोगे: क्लाईन्ट (Client) थेट डेटाबेसशी संवाद साधतो त्यामुळे दोन-स्तरीय मॉडेल वापरण्यास खूप सोपे असते.
2. वेगवान कार्यक्षमता (Fast Performance)

मधील स्तर (Middle layer) नाही, त्यामुळे विनंती प्रक्रिया (request processing) वेगवान होते.

3. कमी नेटवर्क लोड (Less Network Overhead): क्लाईन्ट आणि डेटाबेस मध्ये थेट संवाद (direct communication) असल्यामुळे नेटवर्क रहदारी कमी होते.
4. लहान अँप्लिकेशन साठी सर्वोत्तम (Best for Small Applications) जर छोट्या application ला कमी user's असतील, तर दोन-स्तरीय मॉडेल अधिक योग्य ठरते.

तोटे (Disadvantages):

मर्यादित स्केलेबिलिटी: प्रत्येक क्लायंट थेट डेटाबेसशी कनेक्ट होत असल्याने मोठ्या प्रमाणावर वापरकर्त्यांसाठी कार्यक्षमता कमी होते.नेटवर्क ओव्हरलोड: प्रत्येक क्लायंटला स्वतंत्र डेटाबेस कनेक्शन आवश्यक असल्याने नेटवर्क ट्रॅफिक वाढतो. सिक्युरिटी समस्या: क्लायंटला थेट डेटाबेसशी संवाद साधावा लागतो, त्यामुळे डेटाबेसची सुरक्षा धोक्यात येऊ शकते. मेंटेनन्स कठीण: कोणताही बदल थेट क्लायंट ॲप्लिकेशनमध्ये करावा लागतो, त्यामुळे अपडेट्स आणि मेंटेनन्स कठीण होते.

6.1.2.2 तीन-स्तरीय मॉडेल (Three-Tier Model)

तीन-स्तरीय मॉडेल मध्ये क्लायंट आणि सर्व्हर दरम्यान दुसरा स्तर असतो. या आर्किटेक्चरमध्ये, क्लायंट थेट सर्व्हरशी संवाद साधू शकत नाही.क्लायंट-एंडवरील ॲप्लिकेशन, ॲप्लिकेशन सर्व्हरशी संवाद साधते जे पुढे डेटाबेस सिस्टमशी संवाद साधते.ॲप्लिकेशन सर्व्हरच्या पलीकडे डेटाबेसच्या अस्तित्वाबद्दल अंतिम वापरकर्त्याला कल्पना नसते. डेटाबेसला देखील ॲप्लिकेशनच्या पलीकडे इतर कोणत्याही वापरकर्त्याबद्दल कल्पना नाही.मोठ्या वेब ऍप्लिकेशनच्या बाबतीत 3- टायर आर्किटेक्चर वापरले जाते

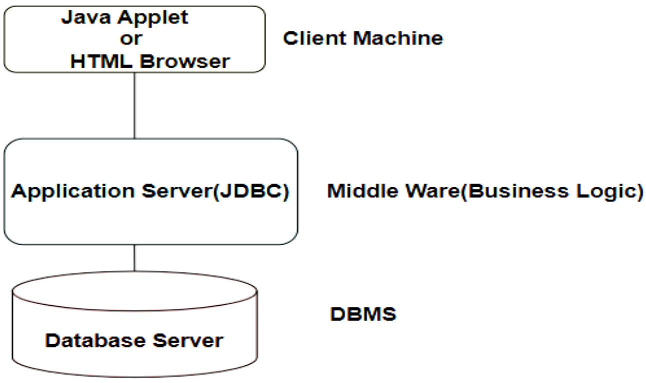


Fig 6.2: तीन-स्तरीय मॉडेल (Three-Tier Model)

फायदे (Advantages):

1. **सुरक्षा (Security)**
   क्लायंट् अँप्लिकेशन थेट डेटाबेस सर्व्हर शी जोडलेले नसते, त्यामुळे डेटा अधिक सुरक्षित राहतो.
2. **स्केलेबिलिटी (Scalability):**
   एकाच वेळी **अनेक क्लायंट्स (Users)** सहजपणे कनेक्ट होऊ शकतात. आवश्यकता असल्यास अँप्लिकेशन सर्व्हर आणि डेटाबेस सर्व्हर स्वतंत्रपणे वाढवता येतात.
3. **मेंटेनन्स आणि अपग्रेड (Maintainability & Upgradability)**
   बिसनेस लॉजिक (Business logic), युजर इंटरफेस (User Interface) आणि डेटाबेस स्वतंत्र असल्यामुळे दुरुस्ती, देखभाल सोपे होते. कोणत्याही एकाच स्तरात बदल केल्यास संपूर्ण सिस्टम प्रभावित होत नाही.
4. **पोर्टेबिलिटी (Portability):**
   वेगवेगळ्या मुख्य वापरकर्ता इंटरफेस (frontend) आणि बॅकएंड तंत्रज्ञान (backend technologies) सहजपणे वापरता येतात. वेब, मोबाइल किंवा डेस्कटॉप ॲप्लिकेशन्ससाठी वापरता येते.

तोटे (Disadvantages):

- अधिक कॉम्प्लेक्स (Complexity):
  दोन-स्तरीय मॉडेल च्या तुलनेत तीन-स्तरीय मॉडेल अधिक जटिल असते. सेटअप आणि कॉन्फिगरेशनसाठी अधिक वेळ आणि संसाधने (resources) लागतात.

- परफॉर्मन्स समस्या (Performance Issues):तीन स्तरांमधील सततच्या डेटा ट्रान्सफर मुळे काही वेळा Latency (विलंब) वाढतो. योग्य हार्डवेअर आणि नेटवर्क सुविधा नसल्यास सिस्टम मंद होऊ शकते.
- अधिक खर्च (Higher Cost):
  ॲप्लिकेशन सर्व्हर साठी अतिरिक्त सर्व्हर्स आणि सॉफ्टवेअर आवश्यक असते, ज्यामुळे खर्च वाढतो.हार्डवेअर आणि मेंटेनन्ससाठी जास्त गुंतवणूक करावी लागते.

6.1.3. ओपन डेटाबेस कनेक्टिव्हिटी (Open Database Connectivity (ODBC))

ODBC (ओपन डेटाबेस कनेक्टिव्हिटी) ही एक प्रमाणित ॲप्लिकेशन प्रोग्रामिंग इंटरफेस आहे, जी ॲप्लिकेशन ला विविध डेटाबेस व्यवस्थापन प्रणाली (DBMS) मधील डेटा प्रवेश करण्याची परवानगी देते.ODBC ॲप्लिकेशन आणि डेटाबेस यांच्यात एक ब्रिज (Bridge) म्हणून कार्य करते. ODBC स्टेटमेंट्स वापरून, एखादा प्रोग्राम SQL Server, MySQL, Oracle आणि फ्लॅट फाइल डेटाबेससारख्या विविध डेटाबेसना कनेक्ट होऊ शकतो, ज्यामुळे विकसक (developer) **डेटाबेस-निरपेक्ष** (database-agnostic) कोड लिहू शकतात.ODBC Windows, Linux, macOS आणि इतर अनेक ऑपरेटिंग सिस्टमवर समर्थित आहे**.** या क्रॉस-प्लॅटफॉर्म सपोर्टमुळे, विविध वातावरणांमध्ये (enviournment) ॲप्लिकेशन विकसित आणि तैनात (deploy) करता येतात, तसेच डेटाबेस प्रवेशामध्ये सातत्य (consistency) राखता येते. उदाहरणार्थ, Linux वर चालणारा एखादा अनुप्रयोग ODBC चा वापर करून Windows वर चालणाऱ्या डेटाबेसला जोडू शकतो. ODBC चे प्रमुख घटक पुढीलप्रमाणे आहेत:

ODBC मुख्य घटक आणि त्यांची कार्ये:

1. **अनुप्रयोग** (Application) अनुप्रयोग वापरकर्त्यासोबत संवाद साधतो आणि ODBC API फंक्शन्सला कॉल करतो.SQL स्टेटमेंट्स प्रक्रिया करण्यासाठी फाइल सिस्टम किंवा डेटाबेस व्यवस्थापन प्रणाली (DBMS) कडे पाठवतो.
2. **ड्रायव्हर व्यवस्थापक** (Driver Manager) अनुप्रयोगाकडून (through application) ODBC API फंक्शन कॉल स्वीकारतो आणि त्यांना ODBC ड्रायव्हरकडे प्रक्रिया करण्यासाठी पाठवतो. ODBC ड्रायव्हरकडून प्राप्त झालेल्या परिणामांना अनुप्रयोगाकडे पुनः पाठवतो.
3. **ODBC ड्रायव्हर (ODBC Driver)** ड्रायव्हर व्यवस्थापकाकडून (driver Manager) प्राप्त ODBC API फंक्शन कॉल्स प्रक्रिया करतो. आवश्यक डेटा मिळवण्यासाठी फाइल सिस्टम किंवा डेटाबेस व्यवस्थापन प्रणालीशी संवाद साधतो.प्राप्त डेटा ड्रायव्हर व्यवस्थापकाकडे परत पाठवतो.
4. **डेटा स्रोत (Data Source)**
   ODBC ड्रायव्हरद्वारे प्रवेश करता येणाऱ्या डेटाचे संच (sets) समाविष्ट असतात.यात डेटाशी संबंधित सर्व वातावरण **(environments)** समाविष्ट असतात, जसे की फाइल प्रवेश सॉफ्टवेअर, डेटाबेस प्रवेश सॉफ्टवेअर, ऑपरेटिंग सिस्टम, आणि नेटवर्किंग प्लॅटफॉर्म**.**

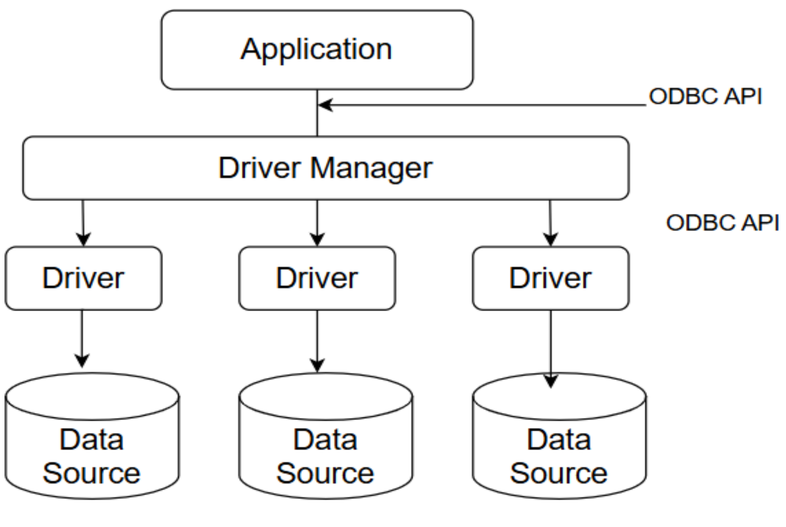


Fig 6.3: ओपन डेटाबेस कनेक्टिव्हिटी (Open Database Connectivity)

6.1.4. ODBC आणि JDBC मधील फरक (Differentiate between ODBC & JDBC)

6.2 JDBC ड्रायव्हर्स (JDBC Drivers)

JDBC ड्रायव्हर म्हणजे जावाच्या काही क्लासेसचा संच आहे, जो विशिष्ट डेटाबेसशी जोडणी करण्यासाठी आणि संवाद साधण्यासाठी आवश्यक असलेल्या JDBC इंटरफेसची अंमलबजावणी करतो. हे ड्रायव्हर जावा कॉल्सना डेटाबेस-स्पेसिफिक कॉल्समध्ये रूपांतरित करतात. ड्रायव्हरची जबाबदारी असते की जावा प्रोग्राम आणि डेटाबेस यांच्यात योग्य प्रकारे संवाद साधला जावा

JDBC ड्रायव्हरची मुख्य जबाबदारी खालीलप्रमाणे

1. जावा प्रोग्रॅम आणि डेटाबेस यांच्यातील कनेक्शन तयार करणे
2. SQL क्वेरी ट्रान्सलेट (अनुवाद) करून डेटाबेसला समजेल अशा स्वरूपात पाठवणे, डेटाबेसकडून उत्तर मिळवून ते जावा प्रोग्रॅमपर्यंत पोहोचवणे.
3. जावा मध्ये थेट (direct) डेटाबेसशी संवाद साधण्याची क्षमता नसते, म्हणून JDBC ड्रायव्हर वापरून जावा कोड आणि डेटाबेसमधील पूल (bridge) तयार केला जातो.

JDBC ड्रायव्हर्स चार प्रकारांमध्ये विभागले जातात, जे त्यांच्या डेटाबेसशी संवाद साधण्याच्या पद्धती आणि आर्किटेक्चरवर

घटक	ODBC (Open Database Connectivity)	JDBC (Java Database Connectivity)
प्लॅटफॉर्म अवलंबित्व	प्लॅटफॉर्म-निरपेक्ष (विविध OS वर चालते)	फक्त जावा आधारित (जावा वापरत असल्यासच वापरता येतो)
भाषा अवलंबित्व	कोणत्याही प्रोग्रॅमिंग भाषेसोबत वापरता येतो	फक्त जावा प्रोग्रॅम्स मध्ये वापरता येतो
API प्रकार	C भाषा (Language) आधारित अँप्लिकेशन प्रोग्रामिंग इंटरफेस	जावा आधारित अँप्लिकेशन प्रोग्रामिंग इंटरफेस
ड्रायव्हर प्रकार	ODBC ड्रायव्हर्स वापरते	JDBC ड्रायव्हर्स वापरते
कार्यक्षमता (Performance)	तुलनेने वेगवान (कारण हे Native Code वापरते)	तुलनेने कमी वेगवान (जावा मध्ये इंटरप्रिटेशन मुळे)
सुरक्षितता	कमी सुरक्षित (नेटिव्ह कोड मुळे)	अधिक सुरक्षित (जावा मुळे)
उदाहरणार्थ वापर	Windows, Linux, macOS वर विविध DBMS साठी	जावा ॲप्लिकेशन्समध्ये SQL डेटाबेसशी जोडण्यासाठी

आधारित असतात:

1. टाइप-1 ड्रायव्हर /JDBC-ODBC ब्रिज ड्रायव्हर (JDBC-ODBC Bridge Driver)
2. टाइप-2 ड्रायव्हर /नेटिव्ह-API ड्रायव्हर (Native API Driver)
3. टाइप-3 ड्रायव्हर /नेटवर्क प्रोटोकॉल ड्रायव्हर (Network Protocol Driver)
4. टाइप-4 ड्रायव्हर /थिन ड्रायव्हर (Thin Driver)

1. टाइप-1 ड्रायव्हर /JDBC-ODBC ब्रिज ड्रायव्हर (JDBC-ODBC Bridge Driver)

वर्णन (Description): हा सर्वात जुना प्रकारचा JDBC ड्रायव्हर आहे, ज्याला JDBC-ODBC ब्रिज असेही म्हणतात. तो ODBC (Open Database Connectivity) चा वापर करून जावा अँप्लिकेशन ला डेटाबेसशी जोडतो. हा ड्रायव्हर JDBC कॉल्सना ODBC कॉल्समध्ये रूपांतरित करतो.

फायदे: साधा आणि वापरण्यास सोपा.

तोटे:

हा **ODBC** ड्रायव्हरवर अवलंबून असतो, जो सर्व प्रणालींवर उपलब्ध नसेल.अनुवादित **(Traslate)** करण्याच्या स्तरामुळे कार्यक्षमतेत घट होऊ शकते.

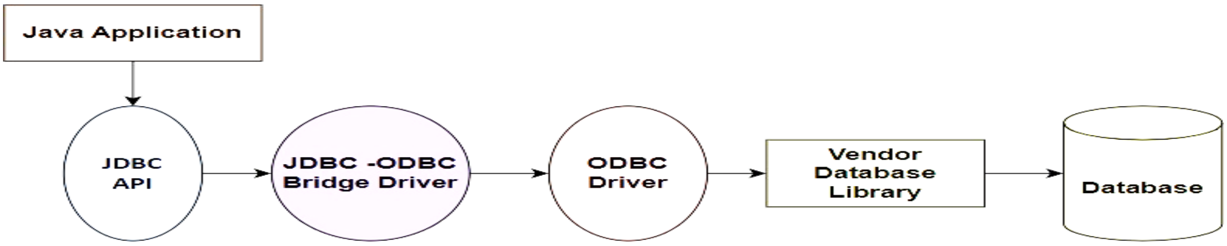


Fig 6..4: टाइप-1 ड्रायव्हर (Type-1 Driver)

2. टाइप-2 ड्रायव्हर /नेटिव्ह-API ड्रायव्हर (Native API Driver)

वर्णन (Description):

हा ड्रायव्हर JDBC कॉल्सना डेटाबेस-स्पेसिफिक कॉल्समध्ये रूपांतरित करतो (नेटिव्ह API चा वापर करून).हा थेट डेटाबेसशी नेटिव्ह लायब्ररीद्वारे संवाद साधतो (उदाहरणार्थ, Oracle साठी Oracle Call Interface).

फायदे (Advantages):

टाइप-1 ड्रायव्हरपेक्षा वेगवान, कारण तो ODBC स्तर टाळतो.

तोटे (Disadvantages):

डेटाबेस-स्पेसिफिक लायब्ररी इंस्टॉल करावी लागते, ज्यामुळे तो कमी पोर्टेबल होतो.

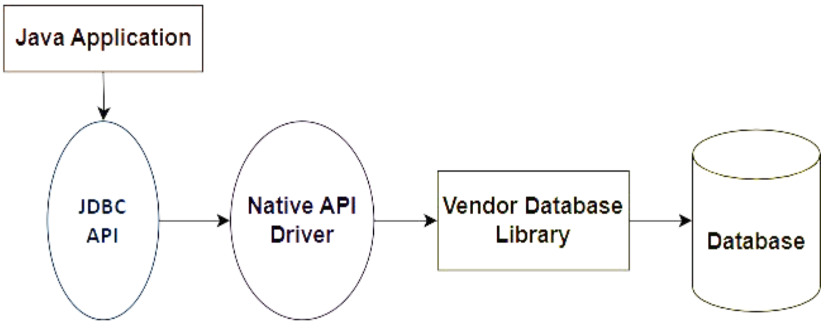


Fig 6.5: नेटिव्ह-API ड्रायव्हर(Type-2 Driver)

3. टाइप-3 ड्रायव्हर /नेटवर्क प्रोटोकॉल ड्रायव्हर (Network Protocol Driver)

हा ड्रायव्हर मिडलवेअर सर्व्हरद्वारे डेटाबेसशी संवाद साधतो. ड्रायव्हर JDBC कॉल्स एका डेटाबेस-स्वतंत्र प्रोटोकॉलमध्ये रूपांतरित करतो, जो नंतर मिडलवेअरद्वारे डेटाबेसच्या विशिष्ट प्रोटोकॉलमध्ये भाषांतरित केला जातो

फायदे(Advantages): यासाठी डेटाबेस-स्पेसिफिक क्लायंट लायब्ररीची आवश्यकता नाही आणि हे नेटवर्क कम्युनिकेशनसाठी अधिक लवचिक आहे.

तोटे(Disadvantages):

यासाठी मध्यस्थ सर्व्हर (Middle Server) आवश्यक असतो, जो अतिरिक्त गुंतागुंती (complexity) आणि ओव्हरहेड (overhead) निर्माण करू शकतो.

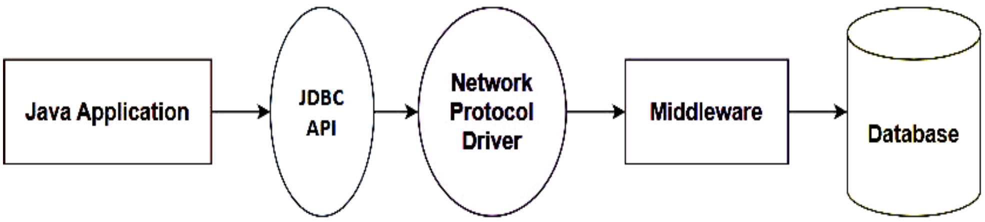


Fig 6.6: Type-3 Driver (नेटवर्क प्रोटोकॉल ड्रायव्हर)

4. टाइप-4 ड्रायव्हर /थिन ड्रायव्हर (Thin Driver)

हा शुद्ध जावा ड्रायव्हर आहे, जो थेट डेटाबेसशी त्याच्या नेटिव्ह प्रोटोकॉलद्वारे संवाद साधतो.हा JDBC कॉल्सना डेटाबेस-स्पेसिफिक कॉल्समध्ये रूपांतरित करतो, आणि हे सर्व फक्त जावा कोडच्या माध्यमातून केले जाते.

फायदे(Advantages):

हलका (Lightweight) म्हणजे कमी संसाधने (CPU, मेमरी, स्टोरेज) वापरणारे आणि पूर्णतः पोर्टेबल, कारण तो पूर्णपणे जावामध्ये लिहिलेला आहे म्हणजे कोणत्याही डेटाबेस-स्पेसिफिक क्लायंट लायब्ररी किंवा मिडलवेअरची आवश्यकता नाही.

तोटे (Disavantgaes)

प्लॅटफॉर्म-निरपेक्ष नाही – हा ड्रायव्हर प्रत्येक डेटाबेससाठी स्वतंत्र असतो, त्यामुळे वेगवेगळ्या डेटाबेससाठी वेगवेगळे ड्रायव्हर वापरावे लागतात.

बऱ्याच वेळा अपडेट्स लागतात – कारण प्रत्येक ड्रायव्हर हा डेटाबेसच्या विशिष्ट आवृत्तीवर (version) अवलंबून असतो, त्यामुळे नवीन अपडेट्स आल्यानंतर तो बदलावा लागू शकतो.

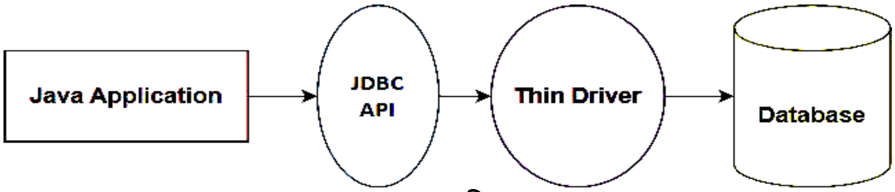


Fig 6..7: Type-3 Driver (थिन ड्रायव्हर)

6.3 जावा इंटरफेसेस व DriverManager क्लास (Java Interfaces and Driver Manager Class)

जावा डेटाबेसशी जोडणी करण्यासाठी आणि SQL स्टेटमेंट्स कार्यान्वित करण्यासाठी अनेक इंटरफेस प्रदान करते. JDBC API चा वापर करून आपण सामान्य तसेच डायनॅमिक SQL स्टेटमेंट्स कार्यान्वित **(execute)** करू शकतो.खालील तक्त्यात JDBC API मधील महत्त्वाचे इंटरफेस दर्शविले आहेत:

Table 6.3: जावा इंटरफेसेस व DriverManager क्लास

इंटरफेसचे नाव (Interface Name)	**वर्णन (Description)**
Driver	JDBC च्या माध्यमातून डेटाबेसशी संवाद साधण्यासाठी आवश्यक असलेल्या पद्धती (methods) परिभाषित करणारा इंटरफेस.हे डेटाबेसशी जोडणी (connection) स्थापन करण्यासाठी आणि विविध ऑपरेशन्स करण्यासाठी वापरले जाते
Connection	डेटाबेसशी संपर्क साधण्यासाठी आणि ट्रान्झॅक्शन व्यवस्थापित करण्यासाठी वापरले जाते. महत्त्वाच्या पद्धती (Methods) createStatement () – साधे (Simple) SQL स्टेटमेंट तयार करण्यासाठी वापरले जाते. prepareStatement () – पूर्व-क्रमांकित SQL स्टेटमेंट (Dynamic SQL Statement) तयार करण्यासाठी वापरले जाते. commit () – ट्रान्झॅक्शन पूर्ण (commit) करण्यासाठी वापरले जाते. close() – कनेक्शन बंद करण्यासाठी वापरले जाते
Statement	स्थिर (static) SQL क्वेरी कार्यान्वित करण्यासाठी वापरले जाते.
PreparedStatement	गतीशील SQL स्टेटमेंट्स (Dynamic SQL Statements) कार्यान्वित करण्यासाठी वापरले जाते.
CallableStatement	Stored Procedures कार्यान्वित करण्यासाठी वापरले जाते.
ResultSet	क्वेरीच्या निकाल संचावर(Resultset) ऑपरेशन्स करण्यासाठी वापरले जाते. डेटाबेसमधून डेटा मिळवण्यासाठी विविध पद्धती प्रदान करते. महत्त्वाच्या पद्धती (Methods): **next()** – पुढील रेकॉर्डकडे जाण्यासाठी. **getString()** – स्ट्रिंग स्वरूपात डेटा मिळवण्यासाठी. **getInt()** – पूर्णांक (Integer) स्वरूपात डेटा मिळवण्यासाठी.

DriverManager	हा इंटरफेस JDBC च्या माध्यमातून डेटाबेसशी संवाद साधण्यासाठी आवश्यक पद्धती (methods) परिभाषित करतो, जसे की कनेक्शन स्थापन करणे आणि डेटाबेस ऑपरेशन्स पार पाडणे.
ResultSetMetaData	हे **ResultSet** संबंधित मेटाडेटा **(Metadata)** प्रदान करते, जसे की स्तंभांची **(Columns)** संख्या आणि त्यांची नावे.

6.3 ड्रायव्हर इंटरफेस (Driver Interface):

JDBC ड्रायव्हर जावा ॲप्लिकेशन ला डेटाबेसशी संवाद साधण्यास सक्षम करतो. प्रत्येक डेटाबेससाठी JDBC ला वेगळ्या ड्रायव्हर्सची आवश्यकता असते.JDBC ड्रायव्हर डेटाबेसशी जोडणी प्रदान करतो आणि SQL क्वेरी आणि निकालांचे (Results) आदानप्रदान करण्यासाठी आवश्यक प्रोटोकॉल अंमलात आणतो.

कारण प्रत्येक डेटाबेसचा प्रोटोकॉल वेगळा असतो, त्यामुळे JDBC क्लायंटला प्रत्येक वेगवेगळ्या डेटाबेससाठी स्वतंत्र ड्रायव्हर्स (बहुधा व्हेंडर-प्रदान केलेले) वापरावे लागतात.

प्रमुख JDBC ड्रायव्हर्स:

1. Oracle डेटाबेससाठी**:** oracle.jdbc.driver.OracleDriver
2. MySQL डेटाबेससाठी: com.mysql.jdbc.Driver
3. MS-Access डेटाबेससाठी: sun.jdbc.odbc.JdbcOdbcDriver

ड्रायव्हर इंटरफेस हा JDBC मधील सर्वांत महत्त्वाचा इंटरफेस आहे**,** आणि प्रत्येक ड्रायव्हर क्लासने त्याची अंमलबजावणी (implement) करणे आवश्यक असते**.**

आपण आवश्यकतेनुसार अनेक डेटाबेस ड्रायव्हर्स लोड करू शकतो, पण सर्व ड्रायव्हर्सनी Driver इंटरफेस लागू करणे आवश्यक आहे.

6.3.1 ड्रायव्हर लोड आणि नोंदणी (Driver Loading and Registration):

ड्रायव्हर लोड करण्यासाठी खालील कोड वापरला जातो:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");    // Oracle डेटाबेस कनेक्शनसाठी
Class.forName("com.mysql.jdbc.Driver");            // MySQL डेटाबेस कनेक्शनसाठी
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");    // MS-Access डेटाबेस कनेक्शनसाठी
```

जसे ड्रायव्हर क्लास लोड होतो, तो आपली स्वतःची एक इंस्टन्स तयार करतो आणि DriverManager मध्ये नोंदणी करतो.JDK 1.6 किंवा त्याहून नवीन आवृत्त्यांमध्ये, Class.forName() ही पद्धत स्पष्टपणे कॉल करण्याची आवश्यकता नाही, कारण पहिल्यांदा कनेक्शन मागितले असता JDBC ड्रायव्हर आपोआप लोड होतो.

6.4 ड्रायव्हर मॅनेजर क्लास (DriverManager class)

- DriverManager हा JDBC API चा घटक (component) आहे आणि java.sql पॅकेजचा सदस्य आहे.
- DriverManager हा वापरकर्ते (user) आणि ड्रायव्हर्स यांच्यातील इंटरफेस म्हणून कार्य करतो.
- तो उपलब्ध असलेल्या JDBC ड्रायव्हर्सचा मागोवा ठेवतो आणि योग्य ड्रायव्हरद्वारे डेटाबेसशी जोडणी स्थापन करतो.
- डेटाबेस ड्रायव्हर क्लास (Driver class) नोंदणी (register) आणि नोंदणी रद्द (deregister) करण्यासाठी आवश्यक सर्व पद्धती (methods) प्रदान करतो. ड्रायव्हर मॅनेजर क्लास जावा ॲप्लिकेशन आणि डेटाबेस यांच्यातील कनेक्शन तयार करण्यासाठी वापरला जातो.
- DriverManager मध्ये नोंदणी (Registration) प्रक्रिया:
  1. DriverManager नोंदणीकृत (Registered) ड्रायव्हर क्लासेसची यादी व्यवस्थापित करतो.
     कोणताही JDBC ड्रायव्हर खालील पद्धती वापरून स्वतःची नोंदणी करू शकतो:
  2. DriverManager.registerDriver (new com.mysql.jdbc.Driver ());
  3. या प्रक्रियेमुळे DriverManager योग्य ड्रायव्हर ओळखतो आणि डेटाबेसशी जोडणी स्थापन करतो.

Table 6.4: Methods ड्रायव्हर मॅनेजर क्लास

Sr.No	पद्धत (Methods)	वर्णन (Description)
1	public static void registerDriver(Driver driver)	दिलेल्या ड्रायव्हरला DriverManager मध्ये नोंदणी करण्यासाठी ही पद्धत वापरली जाते.
2	public static void deregisterDriver(Driver driver)	DriverManager मधून ड्रायव्हर काढून टाकण्यासाठी (deregister) वापरले जाते.
3	public static Connection getConnection(String url) throws SQLException	दिलेल्या URL वरून डेटाबेसशी जोडणी (Connection) प्रस्थापित करण्यासाठी वापरले जाते. जर ड्रायव्हर नोंदणीकृत नसेल तर SQLException फेकले (throw) जाते.
4	public static Connection getConnection(String url, String userName, String password) throws SQLException	दिलेल्या URL, वापरकर्तानाव (username) आणि पासवर्डद्वारे डेटाबेसशी जोडणी करण्यासाठी वापरले जाते. जर ड्रायव्हर नोंदणीकृत नसेल तर SQLException फेकले जाते.

उदाहरण: JDBC चा MySQL शी कनेक्शन (Example: Connecting JDBC with MySQL)

खालील कोड JDBC चा वापर करून MySQL डेटाबेसशी जोडणी (Connection) प्रस्थापित करण्याचे उदाहरण दर्शवतो.प्रारंभ करण्यापूर्वी, localhost सर्व्हर ऍक्सेस करण्यासाठी आवश्यक असलेली ॲप्लिकेशन तुमच्याकडे असल्याची खात्री करा, जसे की XAMPP.जर तुमच्याकडे XAMPP नसेल, तर ते Apache Friends च्या अधिकृत वेबसाइटवरून डाउनलोड आणि इंस्टॉल करा: https://www.apachefriends.org/download.html

इंस्टॉलेशननंतर: XAMPP ॲप्लिकेशन उघडा.Apache आणि MySQL सुरू (Start) करा.

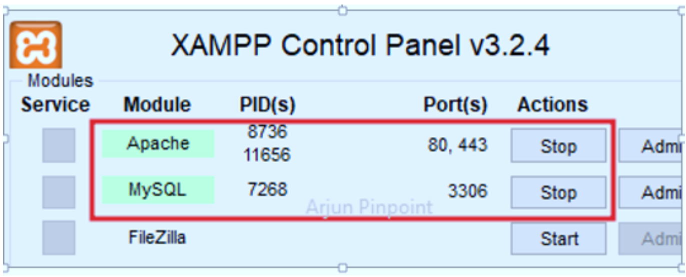


Fig 6.4: Starting Apache and MYSQL

खालील स्टेप्स (steps) पूर्ण करा:

1. डेटाबेस आणि TABLE तयार करा, त्यामध्ये काही मूल्ये (values) समाविष्ट करा आणि जतन (save) करा.
2. MySQL Connector JAR (mysql-connector.jar) डाउनलोड करा.
3. mysql-connector.jar फाईल जावा फाईलच्या स्थानावर चिकटवा(paste).
   उदाहरणार्थ: C:\Program Files\Java\mysql-connector.jar
4. एन्व्हायर्नमेंट व्हेरिएबलमध्ये पाथ सेट करा (set path in enviourment variable):
   "This PC" वर उजवी (right) क्लिक करा. (Properties) वर क्लिक करा. "Advanced System Settings" शोधा आणि त्यावर क्लिक करा. "Environment Variables" वर क्लिक करा (Fig 6.4.1 पहा).
5. खालील पाथ (path) कॉपी करा.
   C:\Program Files\Java\mysql-connector.jar;.;
   "classpath" नावाने नवीन व्हेरिएबल तयार करा आणि वर दिलेला पाथ मूल्य (value) म्हणून पेस्ट करा.
   OK क्लिक करून सेटिंग्स सेव्ह करा.

आता तुम्ही **JDBC** चा वापर करून **MySQL** शी यशस्वीपणे कनेक्ट होऊ शकता.

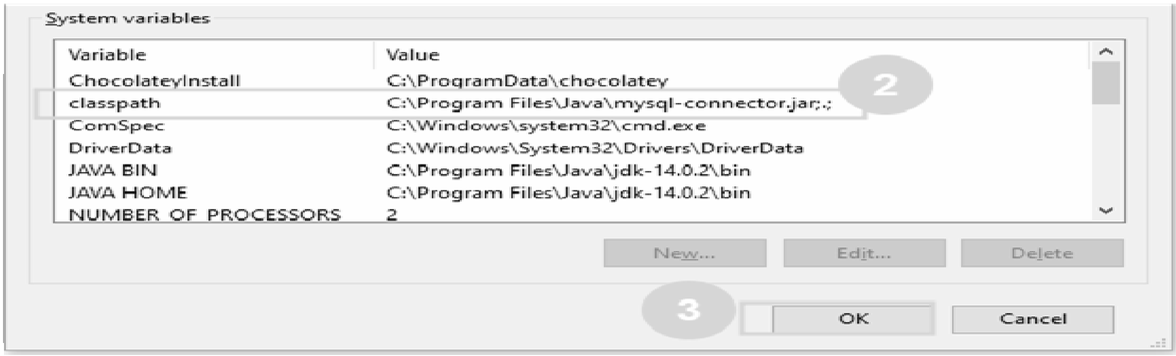


Fig 6.4.1: setting classpath of .jar file

उदाहरण: MYSQL डेटाबेस शी डेटाबेस कनेक्शन यशवीरित्या झाले कि नाही हे तपासणे

निम्नलिखित कोड तपासेल की टेस्ट डेटाबेससह कनेक्शन यशस्वीरीत्या स्थापन झाले आहे की नाही.

(Following code will check the connection to test database established successfully or not)

```
import java.sql.*;
class TestConn
{
  public static void main(String args[]) {
    try
     {
      Class.forName("com.mysql.cj.jdbc.Driver");
      String db_name = "test";
     //test is database name
      String username ="root";
      String password = "";
     String url = "jdbc:mysql://localhost:3306/"+db_name;
    Connection con=
    DriverManager.getConnection(url,username,password);
    System.out.println("Connection to test database is Successfully
    Created"); con.close();
    }
   catch(Exception e) {
    System.out.println(e);
   }}}
```

```
//Output
Connection to test database is Successfully Created
```

6.5. कनेक्शन इंटरफेस (Connection interface)

जावा मधील Connection Interface हा java.sql.Connection पॅकेज अंतर्गत येतो आणि डेटाबेसशी कनेक्शन स्थापित करण्यासाठी आणि SQL स्टेटमेंट execute करण्यासाठी वापरले जाते.Connection इंटरफेस प्रामुख्याने खालील गोष्टींसाठी वापरले जाते: सर्व SQL स्टेटमेंट्स Connection च्या संदर्भातच execute केली जातात. Connection Interface चा उपयोग करून Statement, PreparedStatement, आणि CallableStatement ऑब्जेक्ट तयार करता येतात. डेटाबेसची माहिती **(**metadata) मिळवण्यासाठी देखील कनेक्शन इंटरफेस वापरता येतो. याशिवाय, डेटाबेसचे मेटाडेटा मिळवण्यासाठी देखील याचा वापर करता येतो, जसे की: डेटाबेस प्रॉडक्टचे नाव JDBC ड्रायव्हरचे नाव.

सिंटॅक्स (Syntax**)**

Connection con =   DriverManager.getConnection (url, user, password);

6.5.1. कनेक्शन इंटरफेसद्वारे उपलब्ध काही महत्त्वाच्या मेथड्स (Methods of Connection Interface)

कनेक्शन इंटरफेस अनेक उपयोगी मेथड्स प्रदान करते, ज्या डेटाबेसशी कनेक्शन व्यवस्थापित करण्यासाठी वापरल्या जातात.

Table 6.5.1: कनेक्शन इंटरफेस मेथड्स

क्र.	मेथड (Methods)	वर्णन(Description)
1	Statement createStatement() throws SQLException	SQL क्वेरी कार्यान्वित करण्यासाठी java.sql.Statement ऑब्जेक्ट तयार करते.
2	Statement createStatement(int resultSetType, int resultSetConcurrency) throws SQLException	दिलेल्या प्रकार आणि Concurrency सह ResultSet ऑब्जेक्ट निर्माण करणारे Statement ऑब्जेक्ट तयार करते.
3	void setAutoCommit(boolean status) throws SQLException	Auto-commit मोड सेट करण्यासाठी वापरले जाते. जर true असेल, तर SQL स्टेटमेंट्स स्वतंत्र ट्रान्झॅक्शन्स म्हणून कार्यान्वित व committed होतील. जर false असेल, तर सर्व SQL स्टेटमेंट्स एका ट्रान्झॅक्शनमध्ये गटबद्ध केली जातील. Auto-commit मोड डीफॉल्टने true असतो.
4	void commit() throws SQLException	Auto-commit मोड false असताना या मेथडचा वापर करून ट्रान्झॅक्शन commit केले जाते.
5	void rollback() throws SQLException	सध्याच्या ट्रान्झॅक्शनमधील सर्व बदल पूर्ववत (rollback) करण्यासाठी वापरले जाते. हे फक्त Auto-commit मोड false असताना वापरले पाहिजे.
6	PreparedStatement prepareStatement(String sql) throws SQLException	Parameterized SQL स्टेटमेंट पाठवण्यासाठी java.sql.PreparedStatement ऑब्जेक्ट तयार करते.

कनेक्शन आणि स्टेटमेंट ऑब्जेक्ट तयार करण्यासाठी प्रोग्रॅम

(Program for Creating Connection and Statement object)

```
import java.sql.*;
 class TestConnOb
 {
   public static void main(String args[])
   {
     try {
         class.forName("com.mysql.cj.jdbc.Driver");
         String db_name = "test";
         String username = "root";
         String password = "";
         String url = "jdbc:mysql://localhost:3306/"+db_name;
         Connection
         con=DriverManager.getConnection(url,username,password);
```

```
        System.out.println("Connection to test database is Successfully
        Created"); Statement st=con.createStatement();
        System.out.println("Statement
        object created"); con.close();
      }
      catch(Exception e)
      {
        System.out.println(e);
      }
    }}
```

Output

Connection to test database is Successfully Created
Statement object created

6.6 स्टेटमेंट इंटरफेस (Statement interface)

स्टेटमेंट (Statement) इंटरफेस हा java.sql पॅकेज (package)चा भाग आहे आणि तो डेटाबेसवर SQL क्वेरी कार्यान्वित (execute) करण्यासाठी वापरला जातो. JDBC (Java Database Connectivity) मध्ये डेटाबेसशी संवाद साधण्यासाठी स्टेटमेंट (Statement) हा प्राथमिक इंटरफेस आहे.स्टेटमेंट इंटरफेस चा वापर SQL क्वेरी कार्यान्वित करण्यासाठी, डेटाबेस अपडेट करण्यासाठी, डेटाबेसमधून परिणाम (ResultSet) मिळवण्यासाठी केला जातो. Connection इंटरफेस च्या createStatement() मेथड चा वापर स्टेटमेंट (Statement) ऑब्जेक्ट तयार करण्यासाठी केला जातो.Connection ऑब्जेक्ट तयार केल्यानंतर म्हणजेच डेटाबेसशी कनेक्शन स्थापित केल्यानंतर Statement ऑब्जेक्ट तयार करता येते.

Connection con = DriverManager.getConnection (url, user, password);

आपण createStatement() मेथड कॉल करून स्टेटमेंट (Statement) ऑब्जेक्ट तयार करू शकतो, ते खालीलप्रमाणे: Statement st=con.createStatement (); येथे st हा Statement इंटरफेसचा ऑब्जेक्ट आहे, जो SQL क्वेरी कार्यान्वित करण्यासाठी वापरला जातो. स्टेटमेंट (Statement) इंटरफेसमध्ये अनेक महत्त्वाच्या मेथड्स (methods) उपलब्ध आहेत, ज्या खालील तक्त्यात दर्शविल्या आहेत

Table 6.6: स्टेटमेंट इंटरफेस मेथड्स

क्रमांक	मेथड (Method)	वर्णन (Description)
1	ResultSet executeQuery(String sql)	SELECT क्वेरी कार्यान्वित करण्यासाठी वापरले जाते. हे ResultSet ऑब्जेक्ट परत करते.
2	public int executeUpdate(String sql)	CREATE, DROP, INSERT, UPDATE, DELETE इत्यादी क्वेरी कार्यान्वित करण्यासाठी वापरले जाते. हे int मूल्य परत करते, जे प्रभावित झालेल्या पंक्तींची संख्या दर्शवते. जर DDL स्टेटमेंट (जसे की CREATE किंवा DROP) वापरले असेल, तर 0 परत करते.
3	boolean execute(String sql)	SELECT आणि NON-SELECT दोन्ही प्रकारच्या क्वेरी कार्यान्वित करण्यासाठी वापरले जाते. जर ResultSet ऑब्जेक्ट प्राप्त झाले तर true परत करते, अन्यथा false परत करते.
4	int[ ] executeBatch()	एकाच वेळी बॅचमध्ये एकापेक्षा अधिक SQL स्टेटमेंट्स कार्यान्वित करण्यासाठी वापरले जाते. जर सर्व स्टेटमेंट्स यशस्वीरित्या कार्यान्वित झाली, तर अपडेट केलेल्या पंक्तींची संख्या असलेली int अॅरे परत करते.
5	void close()	Statement ऑब्जेक्ट बंद करण्यासाठी वापरले जाते.

जावा प्रोग्राम: 1

खालील प्रोग्राम test डेटाबेसमध्ये Student नावाचे TABLE तयार करतो. या TABLEमध्ये id, name, city आणि sscper असे चार कॉलम आहेत.

(Program to create Student table in test databasewith columns id, name, city and sscper)

```
import java.sql.*;
 class CreateTable
 {
    public static void main(String args[])
    {
       try
       {
          Class.forName("com.mysql.cj.jdbc.Driver");
          String db_name = "test"; uname =
          "root";
          String pass = "";
          String jdbc_with_sql = "jdbc:mysql://localhost:3306/"+db_name;
           Connection
           con=DriverManager.getConnection(jdbc_with_sql,name,pass);
          Statement stmt=con.createStatement();
          String createquery="create table student(id integer,name varchar(12),city
          varchar(12),sscper decimal(4,2))";
          boolean r=stmt.execute(createquery); //returns false if database table
          is created if(r==false)
          {
             System.out.println("Student table Created Successfully");
          }
          else
          {
             System.out.println("can't create Table Student already exists");
          }
          con.close();
         }
       catch(Exception e)
       {
          System.out.println(e);
       }
    }
 }
 //Output
 Student table Created Successfully
```

जावा प्रोग्राम:2

खालील JDBC प्रोग्राम test डेटाबेसमधील Student TABLEमध्ये एक रेकॉर्ड (विद्यार्थ्यांची माहिती) INSERT करतो.

(Program for Insertion of one record in student table)

```
import java.sql.*;
class InsertData
{
  public static void main(String args[])
  {
     try
     {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String db_name = "test"; //test is
        database name String username =
        "root";
        String password = "";
        String jdbc_with_sql = "jdbc:mysql://localhost:3306/"+db_name;
        Connection
        con=DriverManager.getConnection(jdbc_with_sql,username,passw
        ord); Statement stmt=con.createStatement();
        String insertquery=("insert into student values(1,'Shraddha','Nasik',97.22)");
         int r=stmt.executeUpdate(insertquery);
          //returns false if database table is created if(r==1)
        {
           System.out.println("1 Record Inserted Successfully");
        }
        con.close();
     }
     catch(Exception e)
     {
        System.out.println(e);
     }
  }
}
//Output
1 Record Inserted Successfully
```

6.7 प्रीपेर्ड स्टेटमेंट इंटरफेस (PreparedStatement Interface)

प्रीपेर्ड स्टेटमेंट इंटरफेस हे Statement इंटरफेसचे उपइंटरफेस (subinterface) आहे.हे पॅरामिटराइज्ड क्वेरी (Parameterized Query) execute करण्यासाठी वापरले जाते.

पॅरामिटराइज्ड क्युरी च्या उदाहरणाकडे पाहू:

```
String sql = "INSERT INTO emp VALUES(?,?,?)";
```

जसे तुम्ही पाहू शकता, "?" हे पॅरामिटर (Parameter) म्हणून पास केले आहे**.**

या पॅरामिटरचे मूल्य PreparedStatement च्या सेट्टर (setter) मेथड्स कॉल करून सेट केले जाईल**.**प्रीपेर्ड स्टेटमेंट (PreparedStatement) चा वापर का करावा?

कामगिरी सुधारते (Increases speed of execution of query):

जर तुम्ही PreparedStatement इंटरफेस वापरला, तर जावा अॅप्लिकेशन ची कार्यक्षमता अधिक चांगली होते**,** कारण SQL क्वेरी फक्त एकदाच संकलित (compiled) केली जाते आणि पुन्हा पुन्हा वापरली जाऊ शकते.

6.7.1 प्रीपेर्ड स्टेटमेंट चा सिंटॅक्स (PreparedStatement Syntax)

PreparedStatement pstmt = connection.prepareStatement ("SQL_QUERY");
Connection इंटरफेसमधील prepareStatement () मेथड PreparedStatement चे ऑब्जेक्ट परत करण्यासाठी वापरली जाते.

public PreparedStatement prepareStatement(String query) throws SQLException
prepareStatement() मेथड SQL क्वेरी स्वीकारते आणि PreparedStatement ऑब्जेक्ट परत करते.

उदाहरण

PreparedStatement prepare = prepareStatement(sql)

प्रीपेर्ड स्टेटमेंट इंटरफेसच्या मेथड्स (Methods of PreparedStatement interface)

Table 6.7: प्रीपेर्ड स्टेटमेंट इंटरफेस मेथड्स

अ. क्र	मेथड(Method)	वर्णन(Descrption)
1.	public void setInt(int paramIndex, int value)	निर्दिष्ट पॅरामिटर इंडेक्सवर integer value सेट करतो.
2.	public void setString(int paramIndex, String value)	निर्दिष्ट पॅरामिटर इंडेक्सवर String value सेट करतो.
3.	public void setFloat(int paramIndex, float value)	निर्दिष्ट पॅरामिटर इंडेक्सवर float value सेट करतो.
4.	public void setDouble(int paramIndex, double value)	निर्दिष्ट पॅरामिटर इंडेक्सवर double value सेट करतो.
5.	public int executeUpdate()	INSERT, UPDATE, AND DELETE सारख्या SQL स्टेटमेंट्स चालवतो.
6.	public ResultSet executeQuery()	SELECT क्वेरी चालवतो आणि ResultSet ऑब्जेक्ट परत करतो.

Student TABLE मध्ये डेटा समाविष्ट करणे, अपडेट करणे, दर्शवणे आणि हटवण्यासाठी प्रोग्राम
(**Program for insertion, updating, displaying and deletion of record of Student table)**

```
import java.sql.*;
class PrepareTest
{
public static void main(String args[])
{
try
{
Class.forName("com.mysql.cj.jdbc.Driver");
String db_name = "test"; //test is database name
String username = "root";
String password = "";
String jdbc_with_sql = "jdbc:mysql://localhost:3306/"+db_name;
Connection con=DriverManager.getConnection(jdbc_with_sql,username,password);
String q1="insert into student values(?,?,?,?)";
PreparedStatement p1=con.prepareStatement(q1); p1.setInt(1,10);
p1.setString(2,"Poonam"); p1.setString(3,"Pune"); p1.setDouble(4,78.90);
int r= p1.executeUpdate();
System.out.println( r + " Record/(s) Inserted Successfully");
String q2="Update student set city =? where city=?";
```

```
PreparedStatement p2=con.prepareStatement(q2); p2.setString(1,"NasikCity");
p2.setString(2,"Nasik");
r=p2.executeUpdate();
System.out.println( r + " Record/(s) Updated Successfully");
 String q3="delete from student where city=?";
PreparedStatement p3=con.prepareStatement(q3); p3.setString(1,"Pune");
r=p3.executeUpdate();
System.out.println(r +"Record/(s) Deleted Successfully");
String q4="Select * from student where city=?";
PreparedStatement p4=con.prepareStatement(q4); p4.setString(1,"NasikCity");
ResultSet rs=p4.executeQuery();
 while(rs.next())
{
System.out.print("\n StudentInfo *****" +rs.getInt("id") + " " + rs.getString("Name") +" "
+rs.getString("city")+ " "+rs.getDouble("sscper"));
}
con.close();
}
catch(Exception e)
{
System.out.println(e);
}
}
}
//Output
1 Record/(s) Inserted Successfully
1  Record/(s) Updated Successfully
1   Record/(s) Deleted Successfully
 Shraddha NasikCity  97.22
```

6.8. रिझल्ट सेट इंटरफेस (ResultSet Interface)

ResultSet हे मूलत: डेटाचा एक TABLE असतो, जिथे प्रत्येक पंक्ती (row) म्हणजे एक रेकॉर्ड असतो आणि प्रत्येक स्तंभ (column) म्हणजे डेटाबेसमधील एक फील्ड असतो. ResultSet मध्ये कर्सर (cursor) असतो, जो सद्यस्थितीत असलेल्या पंक्तीवर निर्देश करतो. आपण next(), previous(), first(), आणि last() या मेथड्सद्वारे ResultSet मध्ये हालचाल करू शकतो.

रिज्ल्टसेट (ResultSet) डेटा मिळवण्यासाठी वापरण्यात येणाऱ्या काही मेथड्स:

getString() – स्ट्रिंग स्वरूपात डेटा मिळवण्यासाठी

getInt() – पूर्णांक (Integer) स्वरूपातील डेटा मिळवण्यासाठी

getDouble() – दशांश (Decimal) स्वरूपातील डेटा मिळवण्यासाठी

रिज्ल्टसेट ऑब्जेक्ट डेटाबेस क्वेरीच्या निकालांचा संग्रह साठवण्यासाठी वापरले जाते, विशेषतः SQL SELECT स्टेटमेंट साठी.हे JDBC API चा भाग आहे, जो रिलेशनल डेटाबेसशी संवाद साधण्यासाठी वापरण्यात येतो.सुरुवातीला, कर्सर पहिल्या पंक्तीच्या आधी स्थित असतो. rs.next () ही मेथड कर्सरला पुढील पंक्तीकडे हलवते, rs.next () मेथड बूलियन मूल्य परत करते. जर शेवटचा रेकॉर्ड आल्यास, false परत करते व लूप थांबतो.

6.8.1रिज्ल्टसेट मधील नेव्हिगेशन मेथड्स (Navigating Methods a ResultSet)

अ. क्र	पद्धत (Methods)	वर्णन (Description)
1	next()	ResultSet मधील पुढील रेकॉर्ड (record/row) वर जाण्यासाठी वापरले जाते.
2	previous()	ResultSet मधील मागील रेकॉर्ड (record/row) वर जाण्यासाठी वापरले जाते.
3	first()	ResultSet मधील पहिल्या रेकॉर्ड (record/row) वर जाण्यासाठी वापरले जाते.
4	last()	ResultSet मधील शेवटच्या रेकॉर्ड (record/row) वर जाण्यासाठी वापरले जाते.
5	absolute(int row)	निर्दिष्ट केलेल्या रेकॉर्ड (record/row) वर नेण्यासाठी वापरले जाते.
6	relative(int rows)	दिलेल्या संख्येनुसार पुढे किंवा मागे सरकण्यासाठी वापरले जाते.

6.8.2 ResultSet मधून डेटा मिळवणे (Retrieving Data from a ResultSet):

या पद्धती ResultSet मधील सद्यच्या रेकॉर्डमधून डेटा प्राप्त करण्यासाठी वापरल्या जातात. तसेच, तुम्ही कॉलम क्रमांक किंवा कॉलम नावाद्वारे डेटा मिळवू शकता.

अ. क्र	पद्धत (Methods)	वर्णन (Description)
1.	getInt(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून पूर्णांक (Integer) प्राप्त करण्यासाठी वापरले जाते.
2.	getString(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून स्ट्रिंग (String) प्राप्त करण्यासाठी वापरले जाते.
3.	getDouble(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून डबल (Double) मूल्य प्राप्त करण्यासाठी वापरले जाते.
4.	getBoolean(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून true किंवा false प्राप्त करण्यासाठी वापरले जाते.
5.	getDate(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून java.sql.Date प्राप्त करण्यासाठी वापरले जाते.
6.	getObject(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून कोणताही ऑब्जेक्ट (Object) प्राप्त करण्यासाठी वापरले जाते.
7.	getArray(int columnIndex)	निर्दिष्ट केलेल्या कॉलममधून SQL array प्राप्त करण्यासाठी वापरले जाते.

उदाहरण

खालीलप्रमाणे **Student** डेटा TABLE**(Data Table)** विचारात घ्या.

Table 6.8.2.1: Student

```
MariaDB [test]> select * from student;
+--------+--------------+---------------+---------+
| id     | name         | city          | sscper  |
+--------+--------------+---------------+---------+
|      1 | Shraddha     | NasikCity     |   97.22 |
|      2 | Ajinkya      | Pune          |   88.76 |
|      3 | Sai          | NasikRural    |   98.76 |
|      4 | SHrushti     | NasikRural    |   94.76 |
|      5 | Krushna      | Mumbai        |   74.79 |
+--------+--------------+---------------+---------+
5 rows in set (0.000 sec)
```

रिज्ल्टसेट (ResultSet) ऑब्जेक्ट वापरून विद्यार्थ्यांच्या डेटाचे पुनर्प्राप्ती करणारा प्रोग्राम दर्शविला आहे.

(Program shows retrival of students data using ResultSet object.)

```
import java.sql.*;
class ResultSetDemo
{
   public static void main(String args[])
```

```
{
  try
      {
        Class.forName("com.mysql.cj.jdbc.Driver");
        String db_name = "test"; //test is database name String username = "root";
        String password = "";
        String url = "jdbc:mysql://localhost:3306/"+db_name;
        Connection con=DriverManager.getConnection(url,username,password);
       Statement st=con.createStatement();
      String query="Select * from student";
       ResultSet rs=st.executeQuery(query);
      System.out.println("*****Student Table Data********");
        while(rs.next())
           {    System.out.println( +rs.getInt("id") + " " + rs.getString("Name") +" "
                 +rs.getString("city")+  " "+ rs.getDouble("sscper"))
                    }
        con.close();
      }
  catch(Exception e)  {
 System.out.println(e.getMessage());
  }
 }
}
```

//Output

```
*****Student Table Data********
1 Shraddha NasikCity 97.22
Sai NasikRural 98.76
SHrushti NasikRural 94.76
Krushna Mumbai 74.79
```

Book References:

1. Balagurusamy, E. (2019). Programming with Java (6th ed.). McGraw Hill.
2. Schildt, H. (2022). Java: The complete reference (12th ed.). McGraw Hill.
3. Horstmann, C. S. (2020). Core Java, Volume I: Fundamentals (12th ed.). Pearson.

संदर्भ (Web References):

1. https://www.geeksforgeeks.org/introduction-to-jdbc/
2. https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html
3. https://www.w3schools.blog/jdbc-tutorial
4. https://www.javatpoint.com/example-to-connect-to-the-mysql-database.