Government Girls Polytechnic Bilaspur

Subject Name: Java Programming
Subject Code: 2022572(022)
Semester : 5th

Prepared By:
Mr. Nuresh Kumar Dewangan
Lecturer
Department of Computer Science & Engineering

Unit-1
Introduction of Java & Programming Structure

Introduction to Java - (जावा का परिचय)

1. जावा को James Gosling ने Sun Microsystems Inc में सन 1995 में विकसित किया था, बाद में Oracle Corporation ने इसे अधिग्रहित कर लिया।
2. यह एक सरल programming language है। जावा writing, compiling, और debugging programming को आसान बनाता है।
3. यह reusable code और modular programs बनाने में मदद करता है।
4. जावा एक class-based, object-oriented programming language है और इसे few implementation dependencies के साथ डिजाइन किया गया है।
5. Java applications को byte-code में compile किया जाता है जो किसी भी Java Virtual Machine पर चल सकता है।
6. जावा का syntax C/C++ जैसा ही है।

History of Java (जावा का इतिहास)

1. जावा का इतिहास बहुत दिलचस्प है। James Gosling, Mike Sheridan, और Patrick Naughton ने 1991 में जावा भाषा प्रोजेक्ट शुरू किया। सन इंजीनियर्स की छोटी टीम को Green Team कहा जाता था।
2. शुरू में इसे छोटे, embedded systems के लिए डिजाइन किया गया था, जैसे set-top boxes।
3. सबसे पहले इसे James Gosling द्वारा "Greentalk" कहा जाता था, और इसकी file extension .gt थी।
4. उसके बाद, इसे Oak कहा गया और इसे Green project के एक हिस्से के रूप में विकसित किया गया।
5. Oak क्यों? Oak शक्ति का प्रतीक है और इसे कई देशों जैसे U.S.A., France, Germany, Romania आदि का राष्ट्रीय वृक्ष चुना गया है।
6. 1995 में, Oak का नाम बदलकर "Java" रखा गया क्योंकि यह नाम पहले से ही Oak Technologies का एक trademark था।
7. उन्होंने जावा भाषा के लिए Java नाम क्यों चुना? टीम एक नया नाम चुनने के लिए इकट्ठा हुई। सुझाए गए शब्द थे "dynamic", "revolutionary", "Silk", "Jolt", "DNA" आदि। वे कुछ ऐसा चाहते थे जो technology के सार को दर्शाता हो: क्रांतिकारी, गतिशील, जीवंत, अद्वितीय, और जिसे spell करना और बोलना आसान हो।

   James Gosling के अनुसार, "Java और Silk शीर्ष विकल्पों में से थे! चूंकि Java बहुत unique था, टीम के कई सदस्यों ने अन्य नामों की तुलना में Java को पसंद किया।"
8. Java इंडोनेशिया में एक द्वीप है जहाँ पहली कॉफी का उत्पादन हुआ (जिसे Java coffee कहा जाता है)। यह एक प्रकार का espresso bean है। Java नाम James Gosling ने अपने office के पास एक कप कॉफी पीते हुए चुना था।
9. ध्यान दें कि Java सिर्फ एक नाम है, कोई acronym (संक्षिप्त रूप) नहीं है।
10. जावा को शुरू में James Gosling द्वारा Sun Microsystems में विकसित किया गया था, जिसे अब Oracle Corporation की subsidiary है और इसे 1995 में release किया गया था। (Note: PDF में 1835 गलत है, सही वर्ष 1995 है)
11. 1995 में, Time magazine ने Java को 1995 के दस best products में से एक कहा।
12. JDK 1.0 23 जनवरी, 1996 को release हुआ। Java के पहले release के बाद, language में कई additional features जोड़े गए हैं। नया Java Windows applications, Web applications, enterprise applications, mobile applications, आदि में use किया जा रहा है। हर नया version Java में नई feature जोड़ता है।

Features of Java (जावा की विशेषताएं)

Java programming language creation का primary objective था इसे एक portable, simple और secure programming language बनाना। इसके अलावा, कुछ excellent features भी हैं जो इस language की popularity में important role निभाते हैं। Java की features को Java buzzwords के नाम से भी जाना जाता है।

Java language की सबसे important features की list नीचे दी गई है:

1. Simple (सरल)
2. Object-oriented (वस्तु–उन्मुख)
3. Portable (पोर्टेबल)
4. Platform independent (प्लेटफॉर्म स्वतंत्र)
5. Secured (सुरक्षित)
6. Robust (मजबूत)
7. Architecture neutral (आर्किटेक्चर न्यूट्रल)
8. Interpreted (इंटरप्रेटेड)
9. High Performance (उच्च प्रदर्शन)
10. Multithreaded (मल्टीथ्रेडेड)
11. Distributed (वितरित)
12. Dynamic (डायनामिक)

1. Simple: Java सीखना बहुत आसान है, और इसका syntax simple, clean और easy to understand है।

- Java syntax C/C++ पर आधारित है इसलिए C++ सीखने के बाद programmers के लिए इसे सीखना आसान है।
- Java ने many complicated और rarely used features को हटा दिया है, उदाहरण के लिए, explicit pointers, operator overloading आदि।

- Unreferenced objects को remove करने की need नहीं है क्योंकि Java में automatic garbage collection होता है।

2. Object-oriented : Java एक object-oriented programming language है। Java में everything एक object है। Object-oriented का मतलब है कि हम अपने software को different types of objects के combination के रूप में organize करते हैं जो data और behaviour दोनों को incorporate करते हैं।

Object-oriented programming (OOP) एक methodology है जो software development और maintenance को कुछ rules provide करके simplify करती है।

OOPs के basic concepts हैं:

A. Object (वस्तु)

B. Class (वर्ग)

C. Inheritance (विरासत)

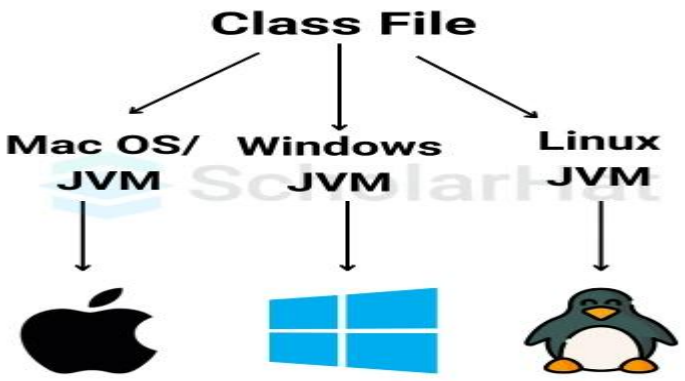
D. Polymorphism (बहुरूपता)

E. Abstraction (अमूर्तता)

F. Encapsulation (एनकैप्सुलेशन)

3. Portable : Java portable है क्योंकि यह आपको Java bytecode को किसी भी platform पर carry करने में सुविधा प्रदान करता है। इसके लिए किसी implementation की आवश्यकता नहीं होती है।

4. Platform Independent :



- Java एक platform independent language है क्योंकि यह C/C++ जैसी अन्य languages से अलग है जिन्हें platform specific machine code में compile किया जाता है जबकि Java एक "write once, run anywhere" program language है। एक platform hardware या software environment होता है जिसमें एक program run होता है।
- दो प्रकार के platforms होते हैं: software-based और hardware-based platform।
- Java platform अधिकांश अन्य platforms से इस मायने में भिन्न है कि यह एक software-based platform है जो अन्य hardware-based platforms के ऊपर चलता है। इसके दो components हैं:

    A. Runtime Environment (रनटाइम वातावरण)

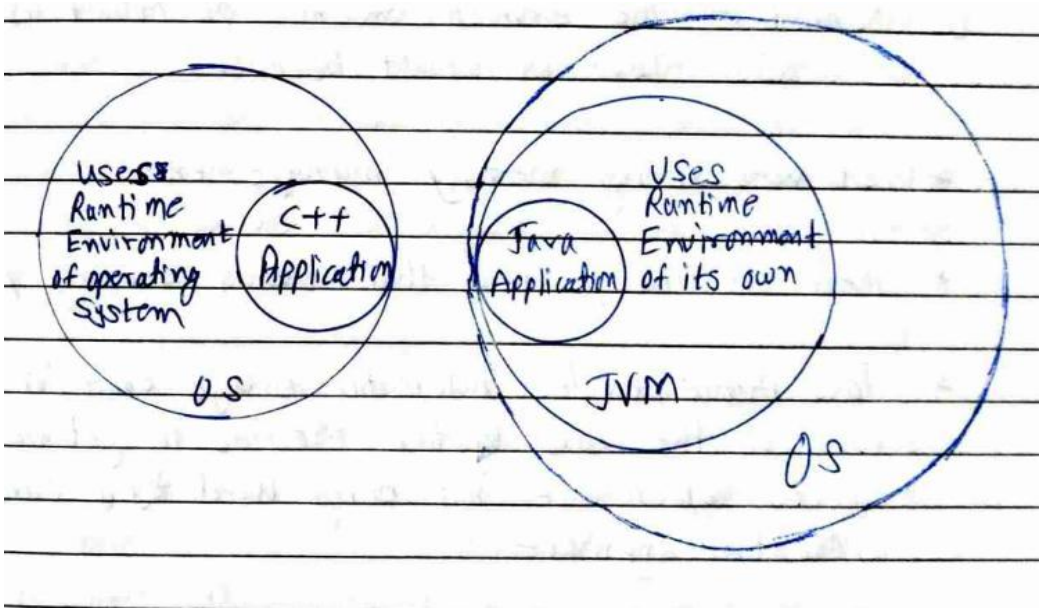
    B. API (Application Programming Interface) (एप्लिकेशन प्रोग्रामिंग इंटरफेस)

- Java code multiple platforms पर execute किया जा सकता है, उदाहरण के लिए, Windows, Linux, Sun Solaris, MacOS, आदि। Java code compiler द्वारा compile किया जाता है और bytecode में converted किया जाता है। यह bytecode एक platform-independent code है क्योंकि यह multiple platforms पर run हो सकता है, यानी, Write Once and Run Anywhere (WORA).

5. Secured (सुरक्षित) : Java अपनी security के लिए सबसे ज्यादा जाना जाता है। Java के साथ, हम virus-free systems develop कर सकते हैं।

Java secured है क्योंकि:

- No explicit pointer (कोई पॉइंटर नहीं)
- Java programs एक virtual machine sandbox के अंदर चलते हैं।



- Classloaders: Java में Classloaders, Java Runtime Environment (JRE) का एक हिस्सा है जिसका उपयोग Java classes को Java Virtual Machine में dynamically load करने के लिए किया जाता है। यह local file system की classes के package को network sources से imported those classes से अलग करके security add करता है।
- Bytecode Verifier: यह code fragments की illegal code के लिए जाँच करता है जो objects के access rights का उल्लंघन कर सकते हैं।
- Security Manager: यह निर्धारित करता है कि एक class किन resources तक access कर सकती है जैसे local disk पर reading और writing।

Java language ये securities by default प्रदान करती है। कुछ security एक application developer द्वारा explicitly भी प्रदान की जा सकती है, जैसे SSL, JAAS, JCE Cryptography आदि के through।

6. Robust : Robust का अंग्रेजी अर्थ है strong (मजबूत)।

Java robust है क्योंकि:

- यह strong memory management का use करता है।
- कोई pointers नहीं हैं जो security problems से बचाते हैं।
- Java provide करता है automatic garbage collection जो Java Virtual Machine पर चलता है ताकि उन objects से छुटकारा मिल सके जिनका अब Java application द्वारा उपयोग नहीं किया जा रहा है।
- Java में exception handling और type checking mechanism हैं। और ये सभी points Java को robust बनाते हैं।

7. Architecture-neutral: Java architecture neutral है क्योंकि इसमें कोई implementation dependent features नहीं हैं, उदाहरण के लिए, primitive types का size fixed है।

एक C programming में, int data type 32-bit architecture के लिए 2 bytes memory occupy करता है और 64-bit architecture के लिए 4 bytes memory occupy करता है। हालाँकि, Java में यह 32 और 64-bit architecture दोनों के लिए 4 bytes memory occupy करता है।

8. Interpreted : Programming languages में, आपने सीखा है कि वे या तो compiler या interpreter का use करती हैं, लेकिन Java programming language एक compiler और interpreter दोनों का use करती है। Java programs bytecode files generate करने के लिए compiled होते हैं, फिर JVM execution के दौरान bytecode file को interpret करता है। इसके साथ ही JVM एक JIT compiler का भी use करता है (यह execution की speed को बढ़ाता है)।

9. High-Performance : Java अन्य traditional interpreted programming languages की तुलना में faster है क्योंकि Java bytecode native machine code के "close" है। यह still एक compiled language (जैसे C++) की तुलना में थोड़ा slower है। Java bytecode को carefully designed किया गया था ताकि इसे directly native machine code में translate करना easy हो Just-in-time compiler का use करके very high performance के लिए।

10. Multi-threaded: एक thread एक अलग program की तरह होता है, जो concurrently execute होता है। हम Java programs लिख सकते हैं जो एक साथ many tasks को handle करते हैं multiple threads को define करके। Multi-threading का main advantage यह है कि यह प्रत्येक thread के लिए memory occupy नहीं करता है। यह एक common memory area share करता है। Threads multimedia, web applications, आदि के लिए important हैं।

11. Distributed: Java distributed है क्योंकि यह user को Java में distributed application create करने में सुविधा प्रदान करता है। RMI (Remote Method Invocation) और EJB (Enterprise Java Beans) distributed applications create करने के लिए use किए जाते हैं। Java की यह feature हमें internet पर किसी भी machine से methods को call करके files access करने में सक्षम बनाती है।

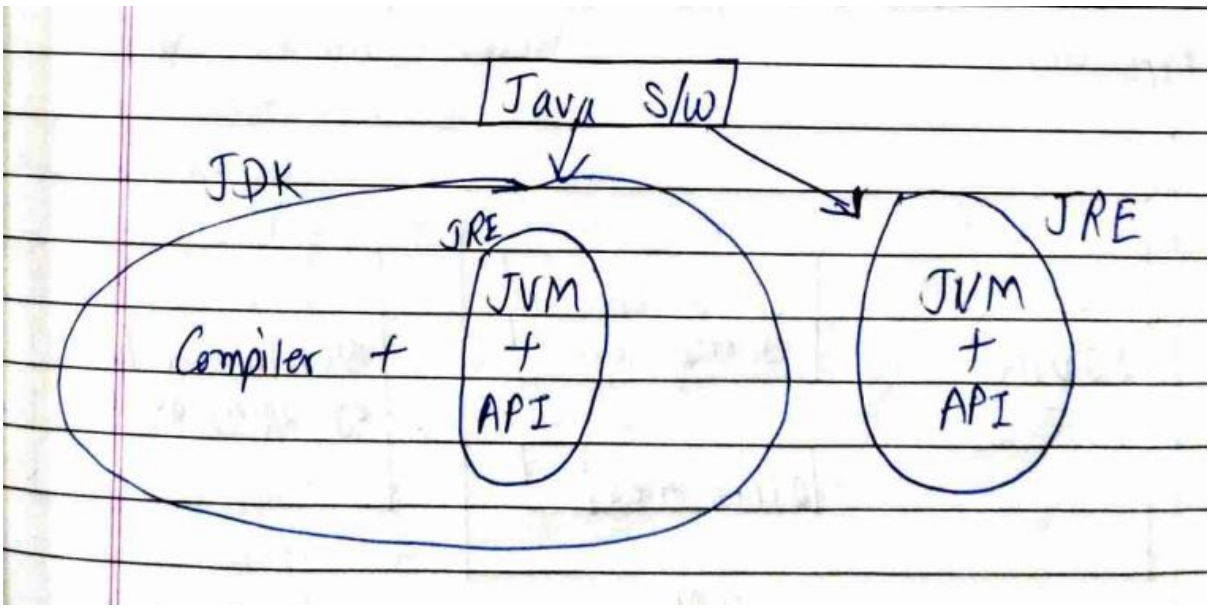
12. Dynamic: Java एक dynamic language है। यह classes के dynamic loading को support करती है। इसका मतलब है कि classes on demand loaded होती हैं। यह अपनी native language के functions को भी support करती है; यानी C और C++।

Java dynamic compilation और automatic memory management (garbage collection) को support करती है।

Java Software : Sun Microsystems (जिसे अब Oracle corporation ने acquire कर लिया है) ने user requirements के आधार पर Java software को मुख्य रूप से दो parts में बांटा। यदि आप देखें, तो मुख्य रूप से दो प्रकार के user हैं – 1. Developer और 2. Customer।

1. **Developer :** वह person जो एक नया program develop करता है, mostly all programs, उसे compile करता है और execute करता है। यानी एक developer को development kit, compiler और program execute करने के लिए एक environment की आवश्यकता होती है।
2. **Customer :** Customer को finalized product मिलेगा, इसलिए उसे केवल program execute करने के लिए एक environment की आवश्यकता होती है।

इस हिसाब से Java Software को दो parts में बांटा गया है



JDK (Java Development Kit) - Java Development Kit (JDK) एक software development environment है जिसका उपयोग Java applications और applets develop करने के लिए किया जाता है। यह physically exist करता है। इसमें JRE + development tools होते हैं।

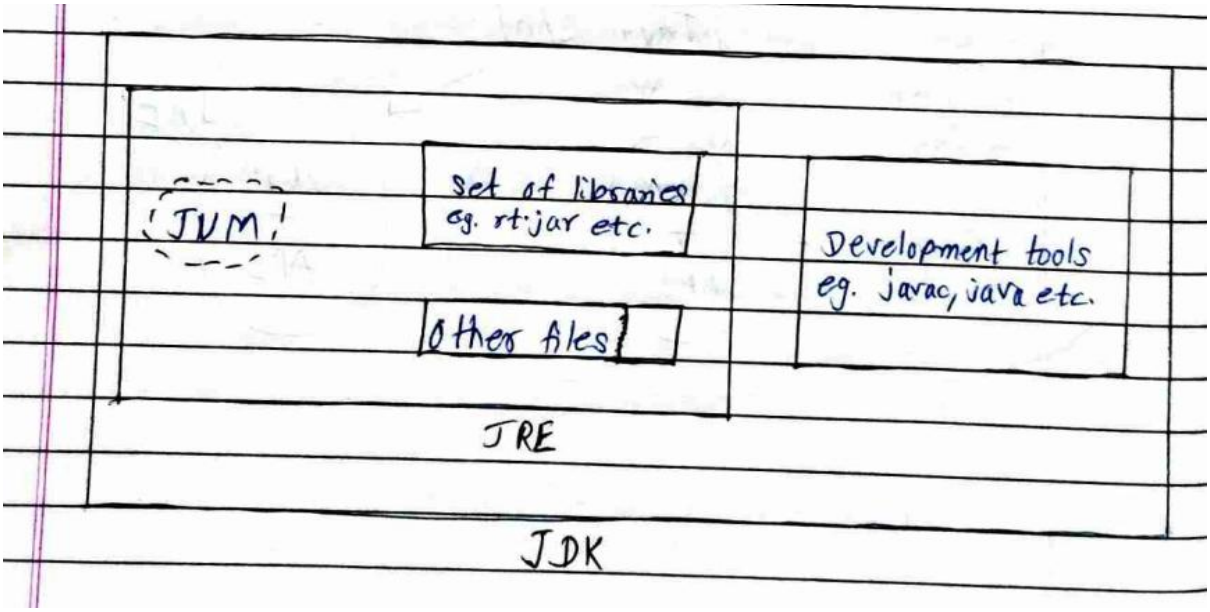
JDK, Oracle Corporation द्वारा release किए गए नीचे दिए गए Java platforms में से किसी एक का implementation है :

a. Standard Edition Java Platform

b. Enterprise Edition Java Platform

c. Micro Edition Java Platform

JDK में एक private Java Virtual Machine (JVM) और कुछ अन्य resources होते हैं जैसे एक interpreter (java), एक compiler (javac), एक archiver (jar), एक documentation generator (javadoc), आदि एक Java Application के development को complete करने के लिए।



JRE - JRE, Java Runtime Environment के लिए एक acronym (संक्षिप्त नाम) है। इसे Java RTE भी लिखा जाता है। Java Runtime Environment software tools का एक सेट है जिसका उपयोग Java applications develop करने के लिए किया जाता है। यह JVM का implementation है। यह physically exist करता है। इसमें libraries + other files का एक सेट होता है जिनका उपयोग JVM runtime पर करता है।

JVM का implementation Sun Micro Systems के अलावा अन्य companies द्वारा भी actively release किया जाता है।

JVM -JVM (Java Virtual Machine) एक abstract machine है। इसे virtual machine इसलिए कहा जाता है क्योंकि यह physically exist नहीं करता है। यह एक specification (विनिर्देश) है जो एक runtime environment provide करता है जिसमें Java bytecode executed किया जा सकता है। यह उन programs को भी run कर सकता है जो other languages में लिखे गए हैं और Java bytecode में compiled किए गए हैं।

JVMs many hardware और software platforms के लिए available हैं। JVM, JRE और JDK platform dependent हैं क्योंकि प्रत्येक OS का configuration एक दूसरे से different होता है। हालाँकि, Java platform independent है।

JVM निम्नलिखित main tasks perform करता है:

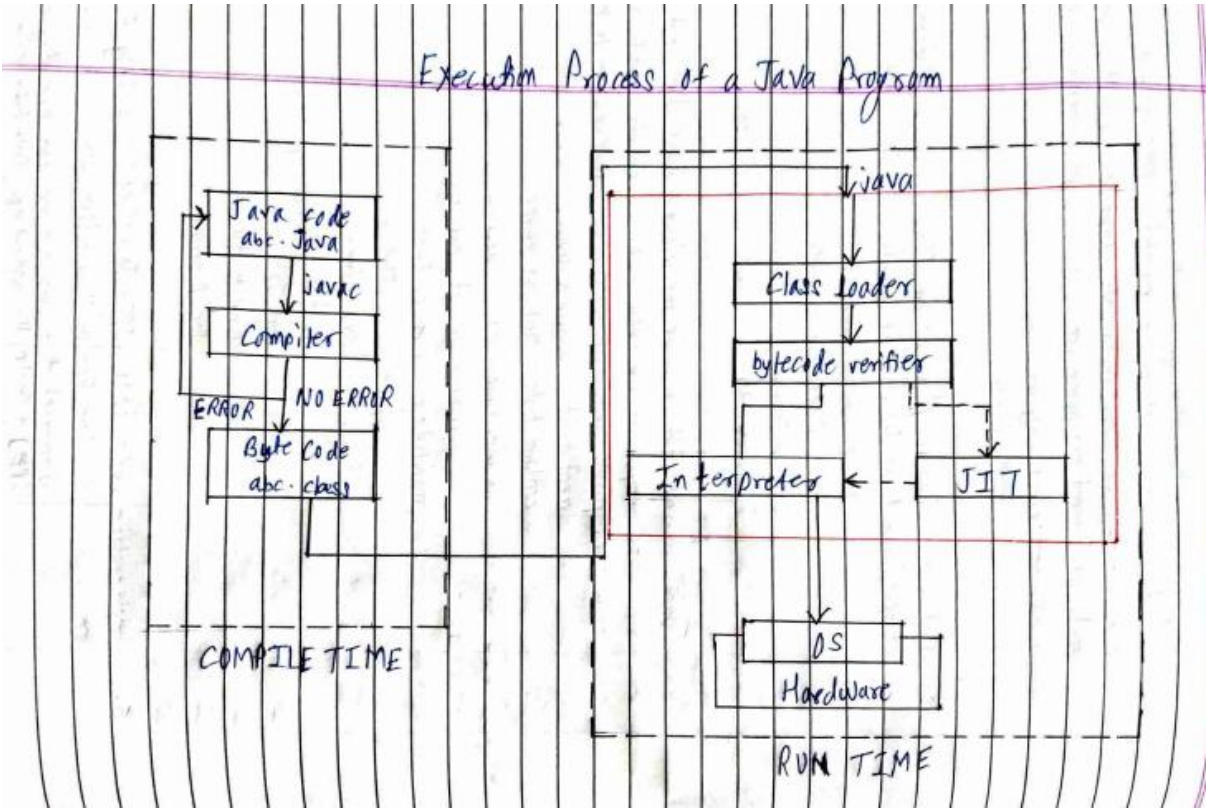
- Loads Code (कोड लोड करता है)
- Verifies code (कोड verifies करता है)
- Executes code (कोड execute करता है)
- Provides a runtime environment (एक runtime environment प्रदान करता है)

Difference between JDK, JRE and JVM: (JDK, JRE और JVM के बीच अंतर)

S.No.	Key	JDK	JRE	JVM
1	Definition	JDK (Java Development Kit) Java में	JRE (Java Runtime Environment) JVM का implementation	JVM (Java Virtual Machine) एक abstract machine है जो platform-dependent है। इसे तीन notions के

		application develop करने के लिए एक software development kit है। JRE के अलावा, JDK में number of development tools (compilers, JavaDoc, etc.) भी होते हैं।	है और इसे एक software package के रूप में designed किया गया है जो Java class libraries, Java Virtual Machine (JVM), और other components provide करता है Java programs में applications को run करने के लिए।	लिए एक specification के रूप में माना जाता है: एक document जो JVM implementation की requirements describe करता है, एक computer program जो JVM requirements को पूरा करता है, और एक instance जो Java byte code execute करता है और Java byte code execute करने के लिए एक runtime environment provide करता है।
2	Prime functionality	JDK primarily code execution के environment के लिए use किया जाता है और इसकी prime functionality development की होती है	दूसरी ओर, JRE mainly code execution के environment create करने के लिए responsible है	JVM दूसरी ओर सभी implementations specify करता है और JRE को ये implementations provide करने के लिए responsible है।
3	Platform independence	JDK platform dependent है यानी different platforms के लिए different JDK required होते हैं	JDK की तरह, JRE भी platform dependent है।	JVM भी platform dependent है।
4	Task	चूंकि JDK prime development के लिए responsible है इसलिए इसमें Java applications को developing, debugging और monitoring के tools होते हैं।	दूसरी ओर JRE में tools जैसे compiler या debugger आदि नहीं होते हैं। बल्कि इसमें class libraries और other supporting files होते हैं जिनकी JVM को program run करने के लिए आवश्यकता होती है।	JVM में software development tools include नहीं होते हैं।
5	Implementation	JDK = Java Runtime Environment + Development Tools	JRE = Java Virtual Machine (JVM) + Libraries to run the applications	JVM = Only Run-time environment for executing the Java byte code

Execution Process of a Java Program (एक जावा प्रोग्राम की निष्पादन प्रक्रिया) –



JIT - JIT stands for Just-in-time compiler। Java में JIT, JVM का एक integral part है। यह execution performance को previous level से many times over accelerate करता है। दूसरे शब्दों में, यह एक key-running, computes-intensive program है जो best performance environment provide करता है। यह Java application की performance को compile time या runtime पर optimize करता है।

JIT compilation में code को machine code में translate करने के लिए दो approaches शामिल हैं AOT (Ahead-of-time compilation) और interpretation। AOT code को native machine language में compile करता है (normal compiler की तरह)। यह VM के bytecode को machine code में transform करता है।

JIT compilers द्वारा निम्नलिखित optimizations की जाती हैं:

- Method In-lining
- Local Optimization
- Control Flow optimization
- Constant Folding
- Dead Code Elimination
- Global Optimizations
- Heuristics for optimizing all sites

Advantages of JIT Compiler:

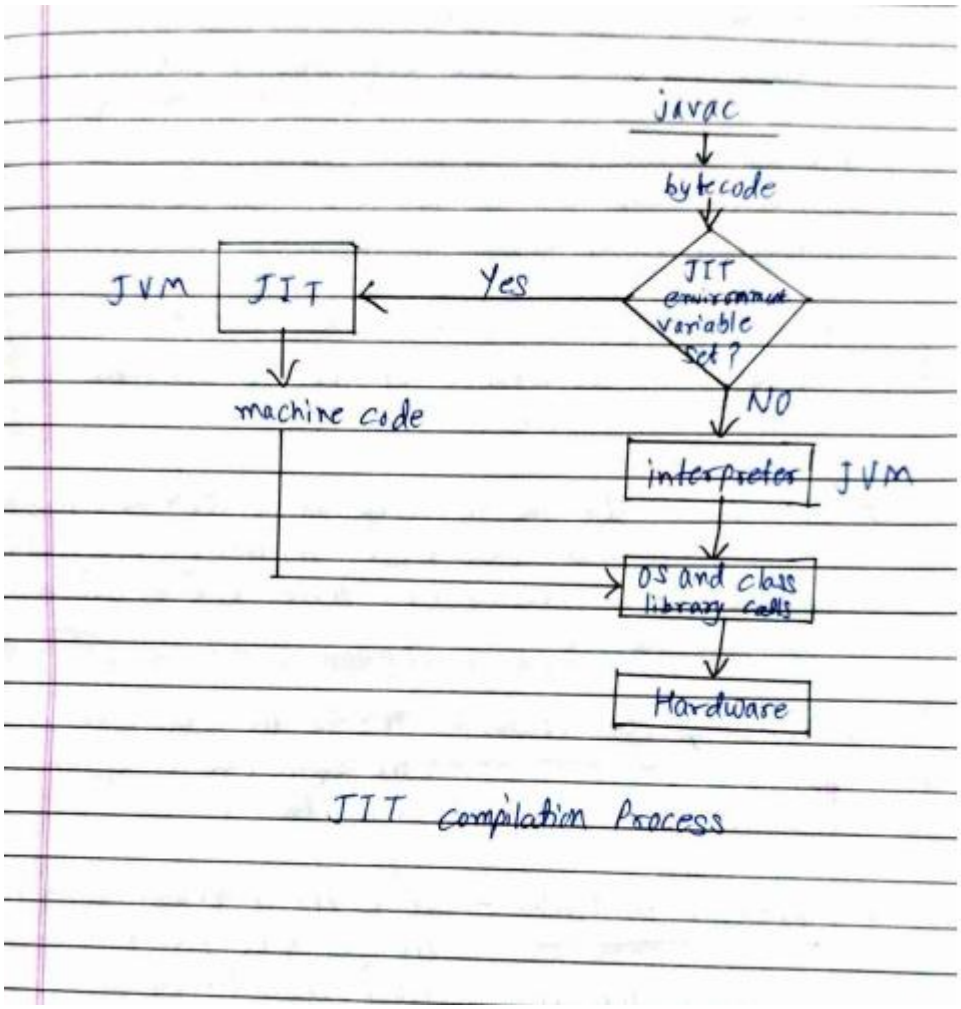
- इसे less memory usage की आवश्यकता होती है।
- Code optimization run time पर की जाती है।
- यह different levels of optimization का use करता है।
- यह page faults को कम करता है।

Disadvantages of JIT Compiler:

- यह program की complexity को बढ़ाता है।
- Less code वाले program JIT Compilation का benefit नहीं लेते हैं।
- यह lots of cache memory का use करता है।

हम जानते हैं कि Java bytecode का interpretation native application के performance को कम कर देता है। JIT Compiler को implement करने का यही कारण है। JIT Compiler bytecode को native machine code में compile करके application के performance को accelerate करता है।

हम निम्नलिखित flow chart की मदद से JIT Compiler के working को समझ सकते हैं।



JVM Architecture in Java (जावा में जेवीएम आर्किटेक्चर)

JVM Architecture में three main subsystems होते हैं:

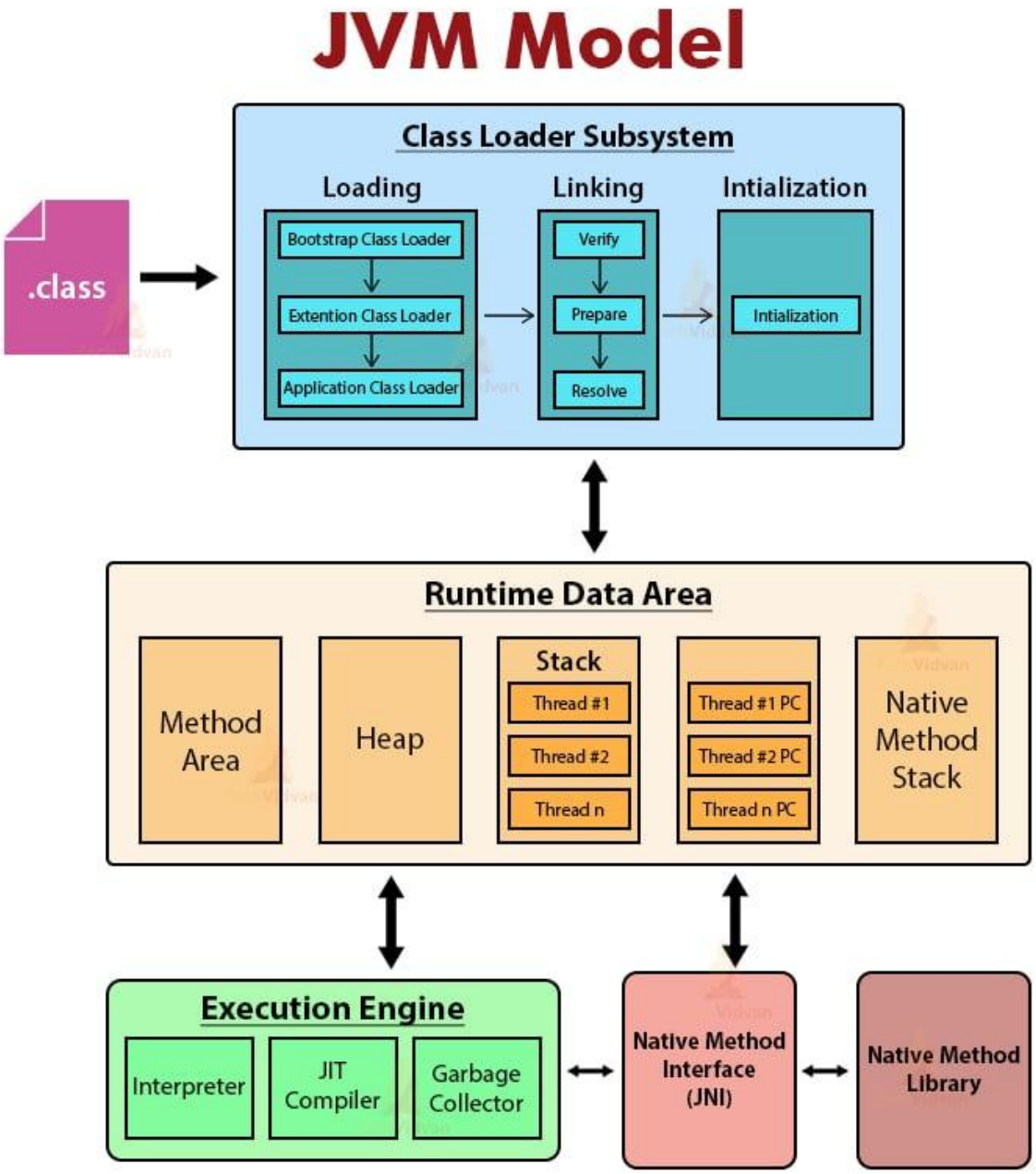
- Class Loader Subsystem (क्लास लोडर उपतंत्र)
- Runtime Data Area (रनटाइम डेटा एरिया)
- Execution Engine (एक्जिक्यूशन इंजन)

Class Loader Subsystem : इसके 3 components हैं:

A. Loading: यह JVM Architecture का वह component है जो classes को memory में load करता है। प्रत्येक JVM में एक class loader होता है। Java में three built-in classloaders हैं:

1. **Bootstrap Class Loader:** यह वह classloader है जो Extension classloader का super class है। यह rt.jar जैसे core Java libraries load करता है।
2. **Extension Class Loader:** यह वह classloader है जो jre/lib/ext directory में मौजूद jar files को load करता है। यह Bootstrap classloader का child है और System classloader का parent है।
3. **System/Application Class Loader:** यह वह classloader है जो classpath से classfiles को load करता है। यह Extension classloader का child है।

Architecture of JVM (जेवीएम का आर्किटेक्चर)



B. Linking - यह operation different files को main program में एक साथ combine करती है। यह Verification, Preparation, और Symbolic reference resolution perform करती है।

1. **Verification -** Verification phase class file की correctness check करती है। इसका मतलब है कि यह check करता है कि file format valid compiler द्वारा generate किया गया है या नहीं। यदि verification fail हो जाता है तो हमें एक java.lang.VerifyError मिलता है।
2. **Preparation -** JVM, class variables के लिए memory allocate करता है और memory को default values में initialize करता है।
3. **Resolution -** Resolution symbolic references को direct references के साथ replace करने की process है। यह returned entity locate करने के लिए method area में searching का use करता है।

C. Initialization - इस operation में सभी static variables को program block में उनके specific values के साथ assign करना शामिल है।

Runtime Data Area - इसके 5 components हैं!

A. Method Area :- यह प्रत्येक class की structure store करता है जैसे method data, field data, runtime constant pool, metadata।

B. Heap :- Heap runtime area है जहाँ object allocation होता है।

C. Stacks :- Stacks partial results और local variables of a program store करते हैं। जब भी एक thread create की जाती है, तो simultaneously एक JVM stack का creation होता है। जब हम एक method invoke करते हैं, तो एक new frame create होता है और destroy उसी समय हो जाता है जब invocation process complete हो जाता है।

D. PC Registers :- यह currently executing JVM instruction का address store करता है।

E. Native Method Stacks :- इसमें किसी भी application में required सभी native methods शामिल होते हैं।

Execution Engine :- यह JVM का वह component है जो memory locations से data read करता है और instructions execute करता है। इसके three major components हैं namely एक interpreter, एक JIT compiler और garbage collector।

A. Interpreter :- Bytecode stream read करता है फिर instruction execute करता है।

B. Just In Time (JIT compiler) :- यह performance improve करता है। JIT similar functionality वाले byte code के parts को एक साथ compile करता है जिससे compilation के लिए needed time की amount reduce हो जाती है।

C. Garbage Collector :- यह automatic memory management perform करता है। Garbage collector का goal unused और unreferenced objects find करना और memory free up करने के लिए उन्हें delete करना है।

Native Method Interface :- यह एक framework है जो different languages जैसे C, C++, आदि में लिखे गए different applications के बीच communication में help करता है।

Native Method Libraries :- Native Libraries, Native Libraries (C, C++) का एक collection है जो Execution Engine के लिए essential हैं।

Object Oriented concepts in Java :- चार main concepts हैं : –

1. Abstraction :- Abstraction का aim users से underlying complexity को hide करना और उन्हें केवल relevant information show करना है। उदाहरण के लिए, यदि आप car drive कर रहे हैं, तो आपको इसके internal workings के बारे में जानने की need नहीं है।

Java classes में भी यही सच है। आप abstract classes या interfaces का use करके internal implementation details hide कर सकते हैं। Abstract level पर, आपको method signature (name और parameter list) define करने की need हो सकती है और प्रत्येक class को उन्हें अपने own way में implement करने दें।

- Abstraction in Java (जावा में अमूर्तता)
- Underlying complexity of data को hide करता है।
- Repetitive code से बचने में help करता है।
- Internal functionality के केवल signature present करता है।
- Programmers को abstract behaviour पर implementation change करने के लिए flexibility देता है।
- Partial abstraction (0-100%) abstract class के साथ achieve की जा सकती है।
- Total abstraction (100%) interfaces के साथ achieve की जा सकती है।

2. Encapsulation:- Encapsulation data security में help करता है, जिससे आप एक class में stored data को system-wide access से protect कर सकते हैं। जैसा कि name suggest करता है, यह एक capsule की तरह class की internal contents को safeguard करता है।

आप Java में fields (class variables) को private बनाकर और उन्हें उनके public getter और setter methods के via access करके encapsulation implement कर सकते हैं।

Encapsulation in Java:

- एक class के data members (fields) तक direct access restrict करता है।
- Fields को private set किया जाता है।
- प्रत्येक field का एक getter और setter method होता है।
- Getter methods field return करते हैं।
- Setter methods हमें field का value change करने देते हैं।

3. Inheritance :- Inheritance एक child class create करना possible बनाता है जो parent class के fields और methods inherit करती है। Child class parent class के values और methods को override कर सकती है, लेकिन यह necessary नहीं है। यह अपने parent में new data और functionality भी add कर सकती है।

Parent classes को superclasses या base classes भी कहा जाता है जबकि child classes को subclasses या derived classes के रूप में भी जाना जाता है। Java extends keyword का use करता है code में inheritance के principle को implement करने के लिए।

Inheritance in Java:

- एक class (child class) किसी अन्य class (parent class) को extend कर सकती है उसकी features inheriting करके।
- DRY (Do Not Repeat Yourself) programming principle को implement करता है।
- Code reusability को improve करता है
- Java में multi-level inheritance allowed है (एक child class का अपना child class भी हो सकता है)
- Java में multiple inheritance allowed नहीं है (एक class एक से अधिक classes extend नहीं कर सकती है)

4. Polymorphism :- Polymorphism एक certain action को different ways में perform करने की ability को refer करता है। Java में, polymorphism दो forms ले सकता है: method overloading और method overriding।

Method overloading तब होता है जब एक class में same name के various methods present होते हैं। जब उन्हें call किया जाता है, तो उन्हें उनके parameters की number, order, या types द्वारा differentiate किया जाता है। Method overriding तब होता है जब एक child class अपने parent के एक method को override करती है।

Polymorphism in Java:

- Same method name का use several times किया जाता है।
- Same name के different methods को एक object से call किया जा सकता है।
- सभी Java objects polymorphic माने जा सकते हैं (कम से कम) वे अपने own type के होते हैं और object class के instances होते हैं।
- Java में Static polymorphism method overloading द्वारा implemented होता है।
- Java में Dynamic polymorphism method overriding द्वारा implemented होता है।

Data Types in Java - Data type variables में store होने वाले different sizes और values specify करते हैं। Java में two types के data types होते हैं।

1. Primitive data types: Primitive data types में boolean, char, byte, short, int, long, float और double शामिल हैं।
2. Non-primitive data types: Non-primitive data types में Classes, Interfaces, और Arrays शामिल हैं।

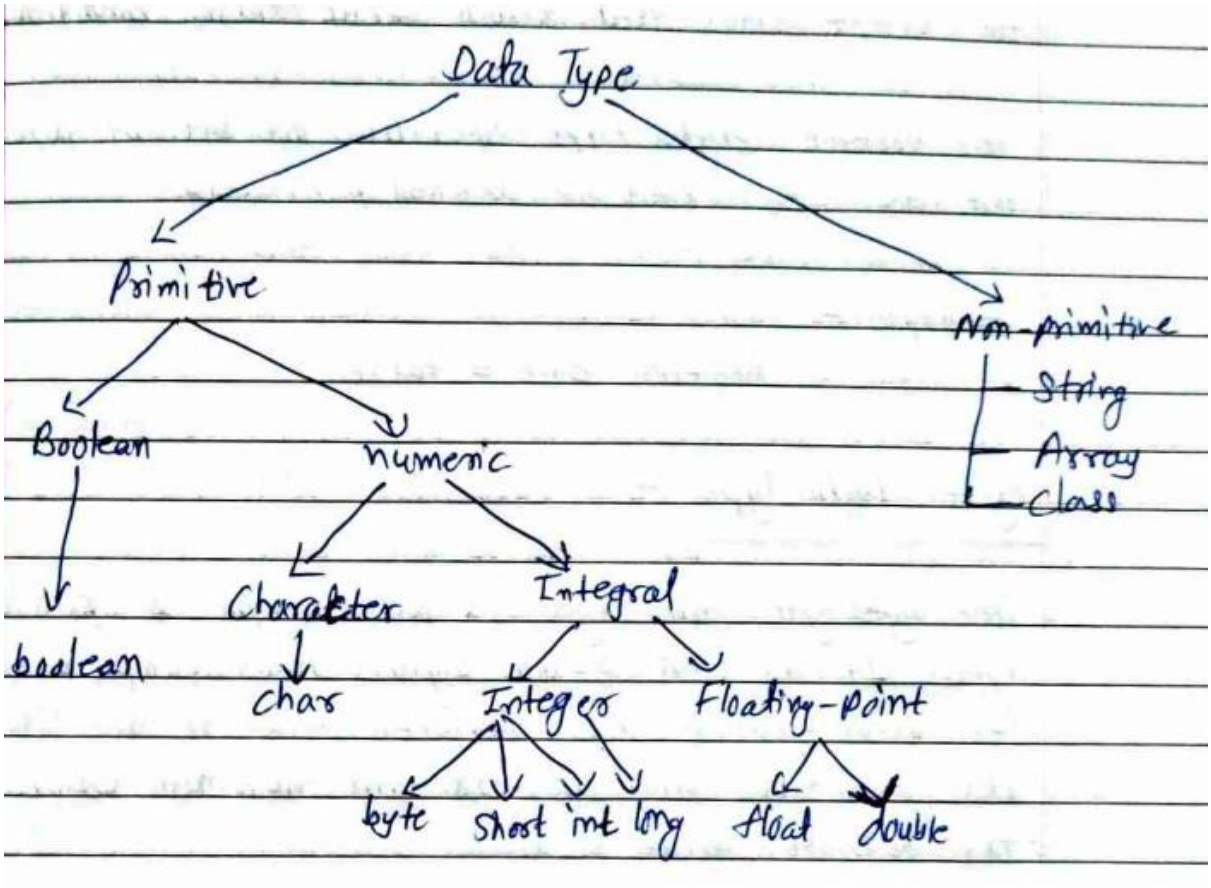
Java Primitive Data Types:

Java language में, primitive data types data manipulation की building blocks हैं। ये Java language में available सबसे basic data types हैं।

Java एक statically-typed programming language है। इसका मतलब है कि सभी variables को इसके use से पहले declared किया जाना चाहिए। इसीलिए हमें variable के type और name declare करने की need होती है।

8 प्रकार के primitive data types हैं:

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



Data Type	Default value	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

Boolean Data Type (बूलियन डेटा टाइप) - boolean data type का use केवल दो possible values: true और false store करने के लिए किया जाता है। इस data type का use simple flags के लिए किया जाता है जो true/false conditions track करते हैं।

boolean data type one bit of information specify करता है लेकिन इसके "size" को precisely defined नहीं किया जा सकता है।

Example :-

boolean one = false;

Byte Data Type :- (बाइट डेटा टाइप) byte data type primitive data type का एक example है। यह एक 8-bit signed two's complement integer है। इसका value-range -128 से 127 (inclusive) के बीच होता है। इसका minimum value -128 और maximum value 127 है। इसका default value 0 है।

byte data type का use memory save करने के लिए large arrays में किया जाता है जहाँ memory saving सबसे required होती है। यह space save करता है क्योंकि एक byte integer से 4 times smaller होता है। इसका use "int" data type के place में भी किया जा सकता है।

Example :-

```
byte a = 10;

byte b = -20;
```

Short Data Type : (शॉर्ट डेटा टाइप) short data type एक 16-bit signed two's complement integer है। इसका value-range -32,768 से 32,767 (inclusive) के बीच होता है। इसका minimum value -32,768 और maximum value 32,767 है। इसका default value 0 है।

short data type का use भी memory save करने के लिए किया जा सकता है जैसे byte data type। एक short data type एक int से 2 times smaller होता है।

Example :

```
short s = 10000;

short r = -5000;
```

Int Data Type : (इंट डेटा टाइप) int data type एक 32-bit signed two's complement integer है। इसका value-range -2,147,483,648 (-2^{31}) से 2,147,483,647 ($2^{31} - 1$) (inclusive) के बीच होता है। इसका minimum value -2,147,483,648 और maximum value 2,147,483,647 है। इसका default value 0 है।

int data type का generally use integral values के लिए default data type के रूप में किया जाता है जब तक कि memory के बारे में कोई problem न हो।

Example :

```
int a = 100000;

int b = -200000;
```

Long Data Type: (लॉन्ग डेटा टाइप) long data type एक 64-bit two's complement integer है। इसका value-range -9,223,372,036,854,775,808 (-2^{63}) से 9,223,372,036,854,775,807 ($2^{63} - 1$) (inclusive) के बीच होता है। इसका minimum value -9,223,372,036,854,775,808 और maximum value 9,223,372,036,854,775,807 है। इसका default value 0 है। long data type का use तब किया जाता है जब आपको int द्वारा provided values की range से अधिक की आवश्यकता होती है।

Example:

```
long a = 100000L;

long b = -200000L;
```

- Negative numbers store करने के लिए 2's complement representation use की जाती है।
- ऐसा लग सकता है कि ऐसी situation में जहाँ int की larger range needed नहीं है, int का use करने की तुलना में byte या short का use करना more efficient होगा, यह case नहीं हो सकता है। इसका reason यह है कि जब byte और short values का use एक expression में किया जाता है तो expression evaluated होने पर उन्हें int में promote कर दिया जाता है।

Float Data Type : float data type single precision 32-bit IEEE 754 floating point है। इसका value range unlimited है। यह recommended है कि floating point numbers के large arrays में memory save करने की need हो तो float का use करें (double के instead) float data type का use कभी भी precise values, जैसे currency, के लिए नहीं किया जाना चाहिए। इसका default value 0.0F है।

Example :-

```
float f = 234.5f;
```

Double Data Type:- (डबल डेटा टाइप) double data type एक double precision 64-bit IEEE 754 floating point है। इसका value range unlimited है। double data type का generally use decimal values के लिए किया जाता है, जैसे float। double data type का use भी precise values, जैसे currency, के लिए नहीं किया जाना चाहिए। इसका default value 0.0d है।

Example :-

```
double d1 = 12.3;
```

Char Data Type:- (चार डेटा टाइप) char data type एक single 16-bit Unicode character है। इसका value-range '\u0000' (or 0) से '\uffff' (or 65,535 inclusive) के बीच होता है। char data-type का use characters store करने के लिए किया जाता है।

Example :-

```
char letterA = 'A';
```

Java 2 bytes का use करता है क्योंकि Java ASCII code system का नहीं बल्कि Unicode system का use करता है। \u0000 Unicode system की lowest range है।

Classes in Java (जावा में कक्षाएं) :- एक class objects का एक group होता है जिसमें common properties होती हैं। यह एक template या blueprint होता है जिससे objects create किए जाते हैं। यह एक logical entity है। यह physical नहीं हो सकता।

Java में एक class निम्नलिखित contain कर सकती है:

a) Fields (क्षेत्र)

b) Methods (विधियाँ)

c) Constructors (कंस्ट्रक्टर्स)

d) Blocks (ब्लॉक्स)

e) Nested class and interface (नेस्टेड क्लास और इंटरफेस)

Example :-

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

String in Java :- (जावा में स्ट्रिंग) Strings को characters के array के रूप में defined किया गया है। Java में एक character array और एक string के बीच difference यह है कि string को एक single variable में characters के sequence को hold करने के लिए designed किया गया है; जबकि एक character array अलग-अलग char type entities का collection है। C/C++ के unlike, Java strings एक null character के साथ terminate नहीं होती हैं।

Example :

```
<string_type> <string_variable> = "Sequence of String";
String s = "Geeks for Geeks";
```

Array Data Type :- (ऐरे डेटा टाइप) एक array same-typed variables का एक group होता है जिसे एक common name द्वारा referred to किया जाता है। Java में arrays C/C++ की तुलना में differently work करते हैं। Java arrays के बारे में कुछ important points निम्नलिखित हैं :

a) Java में, सभी arrays dynamically allocated होते हैं;
b) चूंकि arrays Java में objects हैं, हम उनकी length length member का use करके find कर सकते हैं। यह C/C++ से different है जहाँ हम length sizeof का use करके find करते हैं।
c) एक Java array variable को भी other variables की तरह declared किया जा सकता है with [ ] data type के बाद।
d) Array में variables ordered होते हैं और प्रत्येक का index 0 से beginning होता है।
e) Java array का use एक static field, एक local variable, या एक method parameter के रूप में भी किया जा सकता है।
f) Array का size एक int value द्वारा specified होना चाहिए न कि long या short द्वारा।
g) एक array type का direct superclass Object होता है।

Example:

```
type var-name[ ];
or
type[ ] var-name;

int intArray[ ];
```

or

int[] intArray;

Variable in Java: (जावा में चर) Java में variable एक data container होता है जो Java program execution के दौरान data values store करता है। प्रत्येक variable को एक data type assigned किया जाता है जो type और quantity of value designate करता है जिसे यह hold कर सकता है। Variable data का एक memory location name होता है।

Java में, variable के use से पहले इसे declare करना mandatory होता है।

Variable declaration: (चर घोषणा) एक variable declare करने के लिए, आपको data type specify करना होगा और variable को एक unique name देना होगा।

[Type] Name

int count;

Example of other valid declarations are:

int a, b, c;

float pi;

double d;

char ch;

Operators in Java: (जावा में ऑपरेटर्स) Java में operator एक symbol होता है जिसका use operations perform करने के लिए किया जाता है। उदाहरण के लिए : +, -, *, / आदि।

Java में many types के operators होते हैं जो नीचे दिए गए हैं :

Unary Operator: (एकल ऑपरेटर) वे operators होते हैं जो single operand पर apply होते हैं।

1. + (Unary plus)
2. - (Unary minus)
3. ++ (Increment)
4. -- (Decrement)
5. ! (Logical complement)

Assignment operator: (असाइनमेंट ऑपरेटर) Java assignment operator सबसे common operators में से एक है। इसका use अपनी right side के value को अपनी left side के operand पर assign करने के लिए किया जाता है। =, +=, -=, *=, /=, %=

Example :-

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

```
        int a = 10;
        int b = 20;
        a = b; // Assignment
        System.out.println(a);
        System.out.println(b);
    }
}
```

Output:

20

20

3. Arithmetic Operator :- (अंकगणितीय ऑपरेटर) Java arithmetic operators का use addition, subtraction, multiplication, और division perform करने के लिए किया जाता है। They act as basic mathematical operations.

Example :-

```
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
        System.out.println(a++);   // 10 (a becomes 11)
        System.out.println(--b);   // 4
        System.out.println(a++ + ++a); // 11 + 13 = 24
        System.out.println(b++ + b++); // 4 + 5 = 9
    }
}
```

Output:

15

5

50

2

0

10

4

24

9

- Arithmetic operators को assignment operator के साथ combine किया जा सकता है और combinational operators बना सकते हैं जैसे +=, -=, *= आदि।

Example :-

```
class OperatorExample {
   public static void main(String args[]) {
      int a = 10, b = 5, c = 3;
      System.out.println(a += 2); // a = a + 2 -> 12
      System.out.println(b += 3); // b = b + 3 -> 8
      System.out.println(c += 5); // c = c + 5 -> 8
   }
}
```

Output :-

12

8

8

- यहाँ a += 2, a = a + 2 की तरह behave करता है, b += 3, b = b + 3 की तरह behave करता है और c += 5, c = c + 5 की तरह behave करता है।

4. Relational Operators: (रिलेशनल ऑपरेटर्स) इन operators का use relations जैसे equality, greater than और less than check करने के लिए किया जाता है। They perform boolean results after the comparison और extensively use किए जाते हैं looping statements में और conditional if-else statements में।

Example :-

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

Output:

```
false
true
true
true
false
false
```

5. Logical Operator: (लॉजिकल ऑपरेटर्स) इन operators का use logical AND, logical OR और logical NOT operations perform करने के लिए किया जाता है; यानी function similar to AND gate, OR gate और NOT gate। इनके साथ-साथ Java में short-circuit AND (&&) और short-circuit OR (||) operator भी होते हैं, इसमें यदि output first operand से evaluate करना possible हो तो यह second operand check नहीं करेगा।

"logical operator works only upon boolean values" (लॉजिकल ऑपरेटर्स केवल boolean values पर काम करते हैं)

Example :-

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

```
        boolean a = true;
        boolean b = false;
        int denom = 0, num = 20;
        System.out.println(a && b); // AND -> false
        System.out.println(a || b); // OR -> true
        System.out.println( !a );   // NOT -> false
        System.out.println(a ^ b);  // XOR -> true
        System.out.println(a || b); // short circuit OR -> true
        System.out.println(denom != 0 && num / denom > 10); /* short circuit AND -> false
                                                (denom!=0 is false, so second part not evaluated)  */
    }
}
```

Output:

```
false
true
false
true
true
false
```

6. Conditional Operator :- (कंडीशनल ऑपरेटर) Java includes एक special ternary operator है जो certain types के if-then-else statements को replace कर सकता है।

Syntax:

```
Expression1 ? Expression2 : Expression3
```

यहाँ, Expression1 कोई भी expression हो सकता है जो एक boolean value evaluate करता है। यदि Expression1 true है, तो Expression2 evaluate होता है; अन्यथा Expression3 evaluate होता है।

Example:

```
class OperatorExample {
   public static void main(String args[]) {
      System.out.println(5 < 10 ? 20 : 30); // Condition is true, so prints 20
```

```
    }
}
```

Output:

```
20
```

7. Increment - Decrement & Unary minus :- (इंक्रीमेंट–डिक्रीमेंट और यूनरी माइनस)

- Unary minus values को negate करने के लिए use किया जाता है।
- Increment (++) और decrement (--) operators values को 1 से increment और decrement करने के लिए use किए जाते हैं।
- Increment और decrement operators के two variants होते हैं: Post (a++) और Pre (++a).

Example:

```
class OperatorExample {
    public static void main(String args[]) {
        int a = 5, b = 10, c;
        System.out.println(-a);      // Unary minus -> -5
        System.out.println(++a);     // Pre-increment -> 6 (a becomes 6)
        System.out.println(--b);     // Pre-decrement -> 9 (b becomes 9)
        System.out.println(c = a--); /* Post-decrement: assign then decrement -> c=6,
                                        a becomes 5  */
        System.out.println(c = ++b); /* Pre-increment: increment then assign -> b becomes 10,
                                        c=10  */
    }
}
```

Output:

```
-5
6
9
6
10
```

8. Bitwise Operator : (बिटवाइज़ ऑपरेटर) Java several bitwise operators defines करता है जिन्हें integer types long, int, short, char और byte पर applied किया जा सकता है। ये operators अपने operands के individual bits पर act करते हैं। They are summarized in the following table:

Operators	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right (Signed right shift)
>>>	Shift right zero fill (Unsigned right shift)
<<	Shift left

Example :

```
class OperatorExample {
   public static void main(String args[]){
      int a = 10, b = 20; // 10 = 1010, 20 = 10100 in binary
      System.out.println(~a);     /* Bitwise NOT (~1010 = ...11110101 which is -11 in decimal
                                     for int)   */
      System.out.println(a | b);    // OR (1010 | 10100 = 11110 -> 30)
      System.out.println(a & b);    // AND (1010 & 10100 = 0)
      System.out.println(a ^ b);    // XOR (1010 ^ 10100 = 11110 -> 30)
      System.out.println(30 >> 2);  // Signed right shift (11110 -> 111 = 7)
      System.out.println(20 >>> 2); // Unsigned right shift (10100 -> 101 = 5)
      System.out.println(-30 >>> 2); // Unsigned right shift of negative number (large positive)
      System.out.println(-20 >>> 2); // Unsigned right shift of negative number (large positive)
      System.out.println(10 << 2);  // Shift left (1010 -> 101000 = 40)
   }
}
```

Output:

```
-11
30
```

0

30

7

5

1073741817

1073741819

40

Operator Precedence: — (ऑपरेटर प्रीसीडेंस) Below table shows the order of precedences for Java Operators.

यदि एक expression में एक से अधिक operator हों having same precedence, तो expression को left to right solve किया जाता है।

Precedence	Operators	Description
Highest	(), [], .	Parentheses, Array subscript, Member access
	++, --, ~, !	Unary (Postfix/Prefix), Bitwise NOT, Logical NOT
	*, /, %	Multiplicative
	+, -	Additive
	>>, >>>, <<	Shift
	>, >=, <, <=, instanceof	Relational
	==, !=	Equality
	&	Bitwise AND
	^	Bitwise XOR
	|	Bitwise OR
	&&	Logical AND
	||	Logical OR
	?:	Ternary (Conditional)
Lowest	=, +=, -=, etc.	Assignment

Java Control Flow Statement / Control flow in Java (जावा कंट्रोल फ्लो स्टेटमेंट) : Java compiler code को top से bottom तक execute करता है। Code में statements उस order के according to execute होती हैं जिसमें वे हैं। However, Java statements provide करता है जिनका use Java code के flow को control करने के लिए किया जा सकता है। ऐसे statements को control flow statements कहा जाता है। यह Java की fundamental features में से एक है, जो program का smooth flow provide करता है।

Java three types of control flow statements provide करता है।

1. Decision Making Statements (निर्णय लेने वाले कथन)
    - if statement
    - switch statement
2. Loop Statements (लूप कथन)
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump Statements (जंप कथन)
    - break statement
    - continue statement

Decision-making statement: जैसा कि name suggest करता है, decision-making statements decide करते हैं कि कौन सा statement execute करना है और कब। Decision-making statements boolean expression evaluate करते हैं और provided condition के result के आधार पर program flow control करते हैं।

Java में दो प्रकार के decision-making statements हैं; यानी if statements और switch statements।

A. If Statement: Java में, "if" statement का use एक condition evaluate करने के लिए किया जाता है। Program का control specific condition पर depend करता है। If statement की condition एक Boolean value देती है; या तो true या false। Java में, नीचे दिए गए चार प्रकार के if-statements हैं:

a. Simple if statement (सरल यदि कथन)

b. if-else statement (यदि–अन्यथा कथन)

c. if-else-if statement ladder (यदि–अन्यथा–यदि सीढ़ी कथन)

d. Nested if-statement (नेस्टेड यदि कथन)

आइए if-statements को एक–एक करके समझते हैं।

a. Simple if statement:- यह Java के सभी control flow statements में सबसे basic statement है। यह एक Boolean expression evaluate करता है और program को code के block में enter करने में सक्षम बनाता है यदि expression true evaluate होता है।

Syntax :-

```
if (condition) {
   statement(s); // executes when the condition is true
}
```

Example :-

```
public class Student {
   public static void main(String args[]) {
      int x = 10;
      int y = 12;
      if (x + y > 20) {
         System.out.println("x + y is greater than 20");
      }
   }
}
```

Output :

```
x + y is greater than 20
```

b. if - else statement :- if - else statement, if - statement का एक extension है, जो code के another block यानी else block का use करता है। Else block execute होता है यदि if - block की condition false evaluated होती है।

Syntax :-

```
if (condition) {
   Statement1; // executes when condition is true
} else {
   Statement2; // executes when condition is false
}
```

Example :-

```
class Student {
   public static void main(String args[]) {
      int x = 10;
      int y = 13;
      if (x + y < 10) {
         System.out.println("x + y is less than 10");
      } else {
         System.out.println("x + y is greater than 10");
      }
   }
```

```
}
```

Output :-

```
x + y is greater than 10
```

C. If -else-if ladder : if -else-if statement में if-statement के बाद multiple else-if statements होते हैं। दूसरे शब्दों में, हम कह सकते हैं कि यह if-else statements की chain है जो एक decision tree create करती है जहाँ program code के उस block में enter हो सकता है जहाँ condition true है। हम chain के end में एक else statement भी define कर सकते हैं।

Syntax :-

```
if (Condition1) {
   Statement1; // execute when condition 1 is true
} else if (Condition2) {
   Statement2; // executes when condition 2 is true
}
// ... more else if blocks
else {
   Statement; // executes when all the conditions are false
}
```

Example :-

```
class Student {
   public static void main(String args[]) {
      String city = "Delhi";
      if (city == "Meerut") {
         System.out.println("city is meerut");
      } else if (city == "Noida") {
         System.out.println("city is noida");
      } else if (city == "Agra") {
         System.out.println("city is agra");
      } else {
         System.out.println(city); // This will execute
      }
   }
```

```
}
```

Output :

```
Delhi
```

D. Nested if statement :- Nested if statement में, if statement के अंदर another if or if-else statement contain हो सकता है।

Syntax :-

```
if (condition 1) {
   Statement 1; // executes when condition 1 is true.
   if (condition 2) {
      Statement 2; // executes when condition 2 is true.
   } else {
      Statement 3; // executes when condition 2 is false.
   }
}
```

Example :-

```
class Student {
   public static void main(String args[]) {
      String address = "Delhi, India";
      if (address.endsWith("India")) {
         if (address.contains("Meerut")) {
            System.out.println("Your city is Meerut");
         } else if (address.contains("Noida")) {
            System.out.println("Your city is Noida");
         } else {
            System.out.println(address.split(",")[0]); // Prints "Delhi"
         }
      } else {
         System.out.println("You are not living in India");
      }
   }
```

```
}
```

Output :

```
Delhi
```

B. Switch Statement : Java में, Switch statements if - else if statements के similar होते हैं। Switch statement में multiple blocks of code होते हैं called cases और एक single case executed होता है based on the variable which is being switched। Switch statement का use if - else if statement के instead easier होता है। यह program की scalability को भी enhance करता है।

Switch statement के बारे में noted होने वाले points:

- Case variable int, short, byte, char या enumeration हो सकता है। Java के version 7 से String type भी supported है।
- Cases duplicate नहीं हो सकते हैं।
- Default statement executed होता है जब कोई भी case expression के value से match नहीं करता है। यह optional है।
- Break statement switch block terminate कर देता है जब condition satisfied हो जाती है।
- यह optional है, अगर use नहीं किया जाता है, तो next case executed होता है (fall-through)।
- Switch statements use करते समय, हम must notice करें कि case expression variable के same type का होगा और यह एक constant value भी होगा।

Syntax :-

```
switch (expression) {
   case value1:
      Statement1;
      break;
   case value2:
      statement2;
      break;
   // ...
   case valueN:
      StatementN;
      break;
   default:
      default statement;
}
```

Example :-

```
class Student {
   public static void main(String args[]) {
      int num = 2;
      switch (num) {
         case 0:
            System.out.println("number is 0");
            break;
         case 1:
            System.out.println("number is 1");
            break;
         default:
            System.out.println("num is " + num); // This will execute
      }
   }
}
```

Output :-

```
num is 2
```

Loop Statement: (लूप कथन) Programming में, sometimes हमें code के block को repeatedly execute करने की need होती है जबकि some condition true evaluate होती है। However, loop statements का use instructions के set को repeated order में execute करने के लिए किया जाता है। Instructions के set का execution एक particular condition पर depend करता है।

Java में, हमारे पास three types के loops हैं जो similarly execute करते हैं। However, उनके syntax और condition checking time में differences हैं:

a. for loop

b. while loop

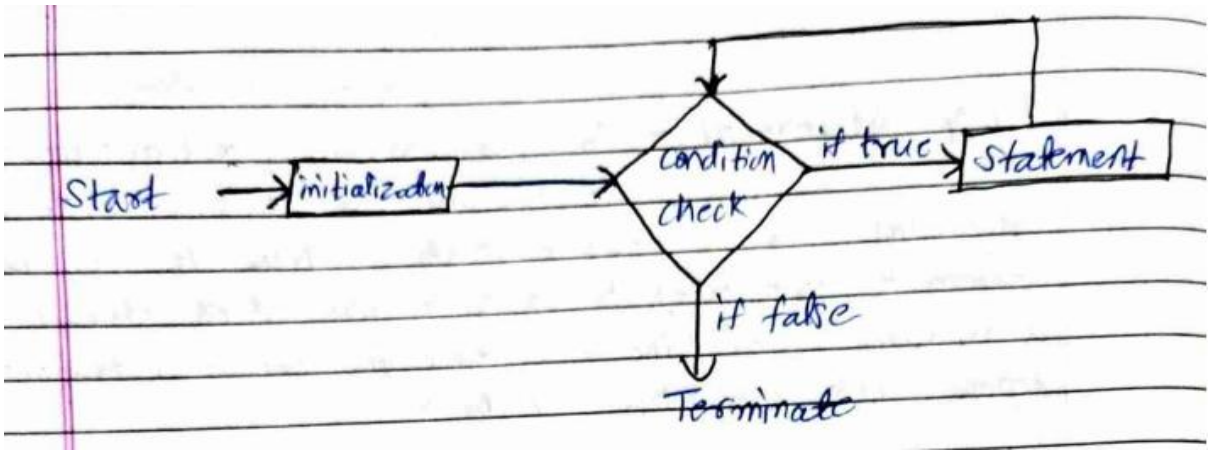
c. do while loop

Java for loop: Java में, for loop C और C++ के similar है। यह हमें loop variable initialize करने, condition check करने, और increment/decrement single line of code में करने में सक्षम बनाता है। हम for loop का use only तब करते हैं जब हम exactly know the number of times, जितनी बार हम code के block को execute करना चाहते हैं।

Syntax:

```
for (initialization; condition; increment/decrement) {
   // block of statements
}
```

Flow chart (फ्लो चार्ट)



Example:

```
class Calculation {
   public static void main(String args[]) {
      int sum = 0;
      for (int j = 1; j <= 10; j++) {
         sum = sum + j;
      }
      System.out.println("The sum of first 10 natural numbers is " + sum);
   }
}
```

Output:

```
The sum of first 10 natural numbers is 55
```

Java for-each loop :- Java एक enhanced for loop provide करता है data structures जैसे array या collection को traverse करने के लिए।

For-each loop में, हमें loop variable update करने की need नहीं होती है।

Syntax :-

```
for (data_type variable : array_name/collection_name) {
   // block of statement
}
```

Example :-

```
class Calculation {
   public static void main(String args[]) {
      String names[] = {"Java", "C", "C++", "Python", "Javascript"};
      System.out.println("Printing the content of the array names:\n");
      for (String name : names) {
         System.out.println(name);
      }
   }
}
```

Output :-

```
Printing the content of the array names:
Java
C
C++
Python
Javascript
```

B. Java while loop :- While loop का use भी multiple times statements के set को iterate करने के लिए किया जाता है। However, यदि हम number of iterations advance में नहीं जानते हैं, तो while loop use करना recommended है। For loop के unlike, initialization और increment/decrement while loop statement के inside नहीं होता है।

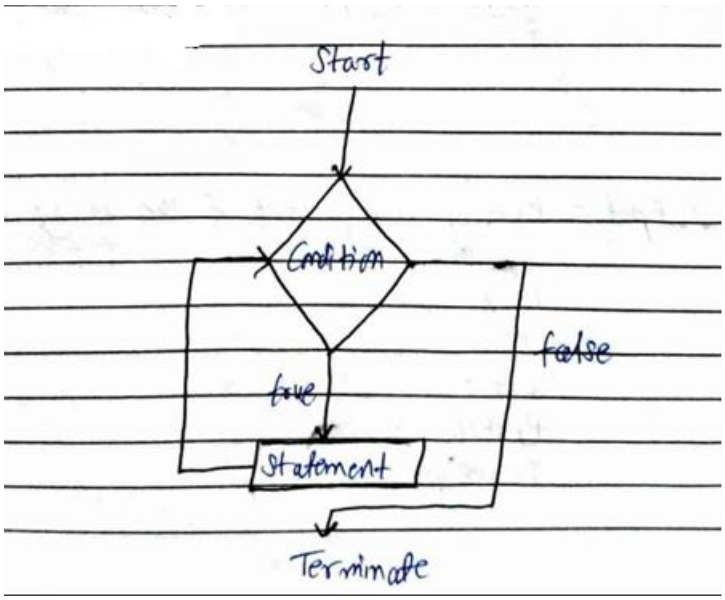
यह entry-controlled loop के रूप में भी जाना जाता है क्योंकि condition loop की start पर checked होती है। यदि condition true है, तो loop body executed होगी; अन्यथा, loop के बाद की statements executed होंगी।

Syntax :-

```
initialization;
while (condition) {
   // looping statement(s)
```

```
    increment/decrement;
}
```

Flow - chart



Example :-

```
class Calculation {
    public static void main(String args[]) {
        int i = 2;
        System.out.println("Printing the list of first 10 even numbers\n");
        while (i <= 20) {
            System.out.println(i);
            i = i + 2;
        }
    }
}
```

Output :

```
Printing the list of first 10 even numbers
2
4
6
8
```

10

12

14

16

18

20

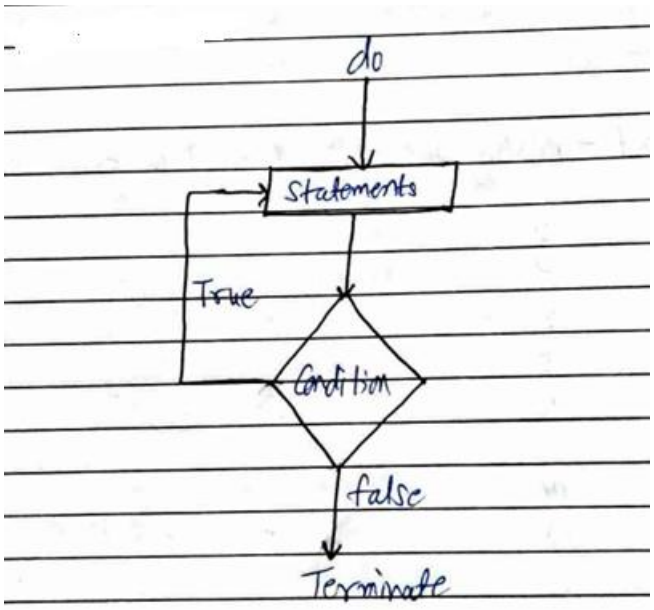
C) Java do-while loop : Do-while loop condition को loop के end में check करता है after executing the loop statements। जब number of iterations known नहीं होती है और हमें loop को कम से कम एक बार execute करना होता है, तो हम do-while loop use कर सकते हैं।

यह exit-controlled loop के रूप में भी जाना जाता है क्योंकि condition advance में checked नहीं की जाती है।

Syntax:

```
initialization;
do {
   // body statements
   increment/decrement;
} while (condition);
```

Flow - chart



Example :

```
class Calculation {
   public static void main(String args[]) {
      int i = 1;
      System.out.println("Printing the list of first 10 odd numbers\n");
      do {
         System.out.println(i);
         i = i + 2;
      } while (i <= 20);
   }
}
```

Output :-

```
Printing the list of first 10 odd numbers
1
3
5
7
9
11
13
15
17
19
```

Jump Statement :- Jump statements का use program के control को specific statements में transfer करने के लिए किया जाता है।

दूसरे शब्दों में, jump statements execution control को program के other part में transfer करते हैं। Java में दो types के jump statements हैं, यानी break और continue।

A) Java break statement :- जैसा कि name suggests, break statement का use program के control flow को break करने और control को एक loop या switch statement के बाहर next statement में transfer करने के लिए किया जाता है। However, यह nested loop के case में only the inner loop break करता है।

Break statement independently Java program में use नहीं किया जा सकता है; यानी, इसे केवल loop के inside या switch statement में लिखा जा सकता है।

Example:

```
class BreakExample {
   public static void main(String args[]) {
      for (int i = 0; i <= 10; i++) {
         System.out.println(i);
         if (i == 6) {
            break; // Loop terminates when i becomes 6
         }
      }
   }
}
```

Output:

```
0
1
2
3
4
5
6
```

B. Java Continue Statement :- Break statement के unlike, continue statement loop break नहीं करता है बल्कि यह loop के specific part को skip करता है और immediately loop की next iteration पर jump करता है।

Example:

```
class ContinueExample {
   public static void main(String args[]) {
      for (int i = 0; i <= 10; i++) {
         if (i > 4 && i < 7) { // Skip printing 5 and 6
```

```
            continue;
        }
        System.out.println(i);
      }
    }
}
```

Output:

```
0
1
2
3
4
7
8
9
10
```

Input from Keyboard in Java (जावा में कीबोर्ड से इनपुट) :

A. Scanner class in Java :- Scanner Java में java.util package की एक class है जिसका use primitive types जैसे int, float, double, आदि और strings का input obtain करने के लिए किया जाता है। यह Java program में input read करने का easiest way है; हालाँकि

a. Scanner class का object create करने के लिए, हम usually predefined object System.in pass करते हैं, जो Standard input stream represent करता है। यदि हम file से input read करना चाहते हैं तो हम class File का object pass कर सकते हैं।

b. किसी certain data type XYZ का numerical value read करने के लिए, function nextXYZ() use करना होता है। उदाहरण के लिए short type का value read करने के लिए, हम nextShort() use कर सकते हैं।

c. एक string read करने के लिए, हम nextLine() use करते हैं।

d. एक single character read करने के लिए, हम next().charAt(0) use करते हैं। next() function input में next token/word को string के रूप में return करता है और charAt(0) function उस string में first character return करता है।

Example :-

```
import java.util.Scanner;
```

```
public class ScannerDemo1 {
   public static void main(String[] args) {
      // Declare the object and initialize with predefined standard input object
      Scanner sc = new Scanner(System.in);
      // String input
      String name = sc.nextLine();
      // Character input
      char gender = sc.next().charAt(0);
      /* Numerical data input byte, short, int, long, float, double can be read using similar-
         named functions.  */
      int age = sc.nextInt();
      long mobileNo = sc.nextLong();
      double cgpa = sc.nextDouble();
      // Print the values to check if the input was correctly obtained
      System.out.println("Name: " + name);
      System.out.println("Gender: " + gender);
      System.out.println("Age: " + age);
      System.out.println("Mobile Number: " + mobileNo);
      System.out.println("CGPA: " + cgpa);
   }
}
```

Input (User enters):

```
Geek
F
20
9876543210
9.9
```

Output :-

```
Name: Geek
Gender: F
```

Age: 20

Mobile Number: 9876543210

CGPA: 9.9

Sometimes, हमें check करना होता है कि next value जिसे हम read करने वाले हैं वह certain type का है या input ended हो गया है (EOF marker encountered)।

तब, हम scanner के input की check करते हैं कि क्या वह type है जिसमें हम interested हैं hasNextXYZ() methods की help से, जहाँ XYZ type है जिसमें हम interested हैं।

Function true return करता है यदि scanner के पास उस type का token है; अन्यथा false। उदाहरण के लिए,

Example:

```
import java.util.Scanner;
class ScannerDemo2 {
   public static void main(String args[]) {
      // Declare an object and initialize with predefined standard input object
      Scanner sc = new Scanner(System.in);

      // Initialize sum and count of input elements
      int sum = 0, count = 0;

      // Check if an int value is available
      while (sc.hasNextInt()) {
         // read an int value
         int num = sc.nextInt();
         sum += num;
         count++;
      }
      int mean = sum / count;
      System.out.println("Mean: " + mean);
   }
}
```

Input (User enters numbers and then a non-integer to stop):

101

223

238

892

39

500

728

stop

Output :-

Mean: 357 // (101+223+238+892+39+500+728) / 7 = 2721 / 7 = ~357

B. Command Line Arguments in Java :- Java command-line argument एक argument है जो Java program run करते समय pass किया जाता है। Command line में, console से passed arguments Java program में receive किए जा सकते हैं और उन्हें input के रूप में use किया जा सकता है। User main() method के inside command-line arguments pass करके execution के during arguments pass कर सकता है।

हमें arguments को space-separated values के रूप में pass करना होता है।

हम strings और primitive data types (int, double, float, char, etc) दोनों को command-line arguments के रूप में pass कर सकते हैं। ये arguments एक string array में convert हो जाते हैं और main() function को एक string array argument के रूप में provided किए जाते हैं।

जब command-line arguments JVM को supplied किए जाते हैं, JVM इन्हें wrap करता है और args[] को supply करता है। यह confirmed किया जा सकता है कि वे एक args array में wrapped up हैं args.length का use करके check करके।

Internally, JVM इन command-line arguments को args[] array में wrap up करता है जिसे हम main() function में pass करते हैं। हम args.length method का use करके इन arguments को check कर सकते हैं। JVM first command-line argument को args[0] पर, second को args[1] पर, third को args[2] पर, store करता है और so on।

Example:

```
class GFG {

   // Main driver method

   public static void main(String args[]) {

      // Printing the first argument

      System.out.println(args[0]);

   }

}
```

Running and Output:

```
> javac GFG.java
> java GFG
// Error: java.lang.ArrayIndexOutOfBoundsException (No arguments provided)

> java GFG Hello
Hello // (args[0] is "Hello")
```

Example:-

```
class GfG {
    // main driver method
    public static void main(String args[]) {
        // checking if length of args array is greater than 0
        if (args.length > 0) {
            // print statements
            System.out.println("The command line arguments are:");

            // Iterating the args array using for each loop
            for (String val : args) {
                // printing command line arguments
                System.out.println(val);
            }
        } else {
            System.out.println("No command line arguments found");
        }
    }
}
```

Output:

```
> java GfG
No command line arguments found

> java GfG Hello World
```

The command line arguments are:

Hello

World

Unit-2
Class, Object & Inheritance

Classes and Objects in Java - Classes and Objects Object-Oriented Programming के basic concepts हैं जो real-life entities के around revolve करते हैं।

Class (कक्षा) :

- Class objects का एक set होता है जो common characteristics और common properties/attributes share करते हैं।
- Class एक real-world entity नहीं है। यह just एक template या blueprint या prototype होता है जिससे objects create किए जाते हैं।
- Class memory occupy नहीं करती है।
- Class different data types के variables और methods के group का एक समूह होता है।

Java में एक class निम्नलिखित contain कर सकती है:

- data member (डेटा सदस्य)
- method (विधि)
- constructor (कंस्ट्रक्टर)
- nested class (नेस्टेड क्लास)
- interface (इंटरफेस)

Syntax :

```
access_modifier class <class_name>
{
   data member;
   method;
   constructor;
   nested class;
   interface;
}
```

Example (उदाहरण) :

- Animal (जानवर)
- Student (छात्र)
- Bird (चिड़िया)
- Vehicle (वाहन)
- Company (कंपनी)

```
class Student
{
   int id;       //data member (also instance variable)
   String name;  //data member (also instance variable)

   public static void main(String args[])
   {
      Student s1 = new Student();  //creating an object of Student
      System.out.println(s1.id);
      System.out.println(s1.name);
   }
}
```

एक class एक user-defined blueprint या prototype होता है जिससे objects create किए जाते हैं। यह properties या methods के set को represent करता है जो एक type के सभी objects के लिए common होते हैं। General तौर पर, class declarations में निम्नलिखित components शामिल हो सकते हैं, क्रम में:

- **Modifiers (संशोधक) :** एक class public हो सकती है या इसकी default access हो सकती है।
- **Class keyword:** class keyword का use एक class create करने के लिए किया जाता है।
- **Class name:** Name एक initial letter से शुरू होना चाहिए (convention के अनुसार capitalized)।
- **Superclass (यदि कोई हो) :** Class के parent (superclass) का नाम, यदि कोई हो, extends keyword से पहले। एक class केवल एक parent को extend (subclass) कर सकती है।
- **Interfaces (यदि कोई हो) :** Class द्वारा implemented interfaces की comma-separated list, यदि कोई हो, implements keyword से पहले। एक class एक से अधिक interface implement कर सकती है।
- **Body :** Class body braces {} से surrounded होती है।

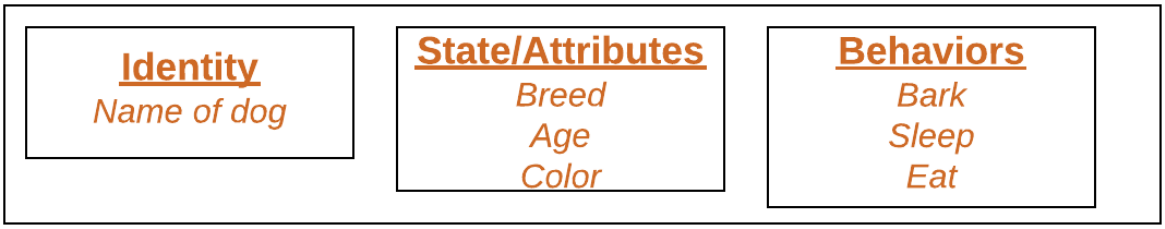
Object: यह Object-Oriented Programming की एक basic unit है और real-life entities को represent करता है। एक typical Java program कई objects create करता है, जो जैसा कि आप जानते हैं, methods invoke करके interact करते हैं। एक object में निम्नलिखित शामिल होते हैं:

State (स्थिति) : यह किसी object की attributes द्वारा represent की जाती है। यह object की properties को भी reflect करती है।

Behaviour (व्यवहार) : यह object की methods द्वारा represent किया जाता है। यह other objects के साथ object की response को भी reflect करता है।

Identity (पहचान) : यह एक object को unique name देती है और एक object को other objects के साथ interact करने में सक्षम बनाती है।

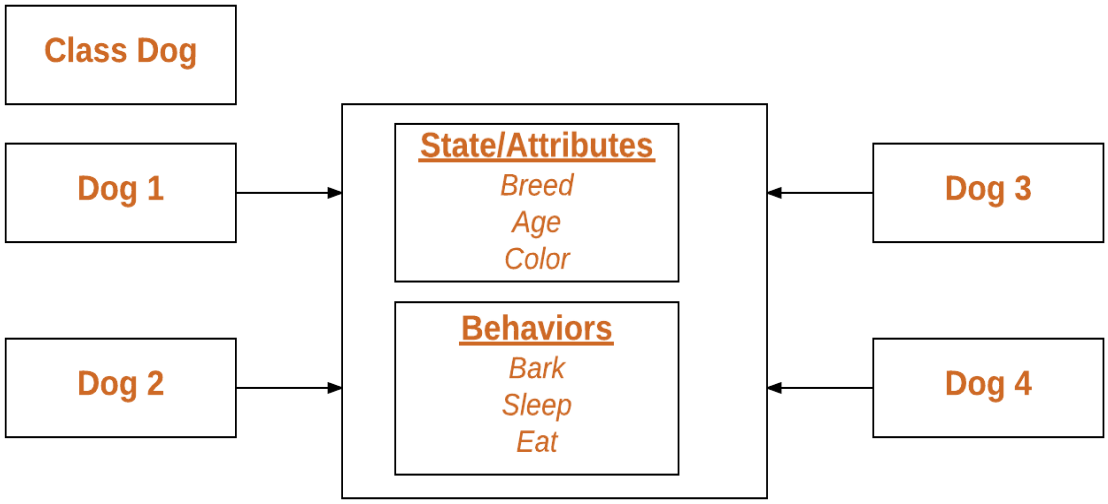
किसी Object का उदाहरण: dog (कुत्ता)



Objects real world में पाई जाने वाली चीजों के correspond करते हैं। उदाहरण के लिए, एक graphics program में objects हो सकते हैं जैसे "circle", "square", और "menu"। एक online shopping system में objects हो सकते हैं जैसे "shopping cart", "customer", और "product"।

Declaring Objects (Objects घोषित करना) (इसे class को instantiate करना भी कहा जाता है): जब किसी class का object create किया जाता है, तो class को instantiated कहा जाता है। सभी instances, class के attributes और behaviour को share करते हैं। लेकिन उन attributes के values, यानी state, प्रत्येक object के लिए unique होते हैं। एक single class के any number of instances हो सकते हैं।

Example:



Initializing an object (किसी Object को प्रारंभ करना) - new operator एक class को instantiate करता है एक new object के लिए memory allocate करके और उस memory का reference return करके। new operator class constructor को भी invoke करता है।

Example:

```
public class Dog
{
   String name;
   String breed;
   int age;
   String color;

   public Dog(String name, String breed, int age, String color)
   {
      this.name = name;
      this.breed = breed;
      this.age = age;
      this.color = color;
   }

   public String getName()
   {
      return name;
   }

   public String getBreed()
   {
      return breed;
   }
   public int getAge()
   {
      return age;
   }

   public String getColor()
   {
```

```
        return color;
    }

    @Override
    public String toString()
    {
        return("Hi my name is "+ this.getName()+
               ".\nMy breed,age and color are " +
               this.getBreed()+"," + this.getAge()+
               ","+ this.getColor());
    }

    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        System.out.println(tuffy.toString());
    }
}
```

Output:

```
Hi my name is tuffy.
My breed,age and color are papillon,5,white
```

Initialize by using method/function (मेथड/फंक्शन का उपयोग करके प्रारंभ करना)

```
public class GFG {
    static String sw_name;
    static float sw_price;

    void set(String n, float p) {
        sw_name = n;
        sw_price = p;
    }
```

```
    void get() {
        System.out.println("Software name is: " + sw_name);
        System.out.println("Software price is: " + sw_price);
    }

    public static void main(String args[]) {
        GFG G = new GFG();
        G.set("Visual studio", 0.0f);
        G.get();
    }
}
```

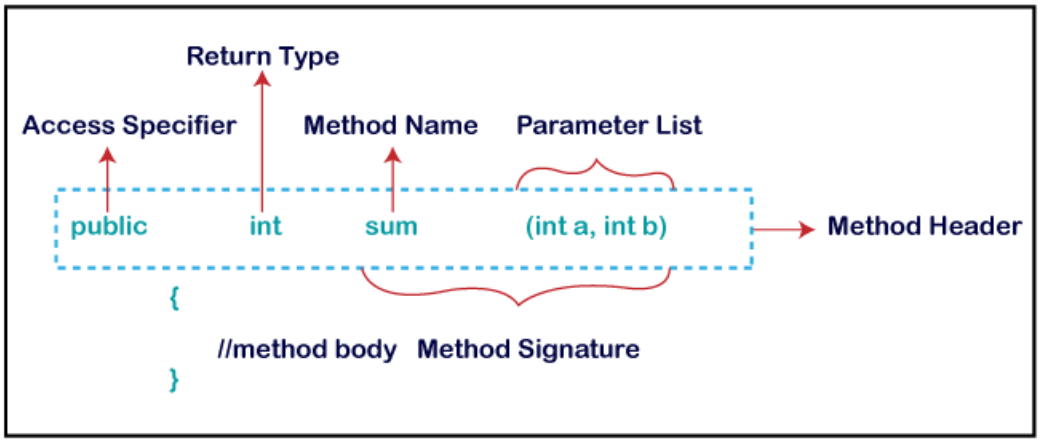
Output:

```
Software name is: Visual studio
Software price is: 0.0
```

Methods in Java (जावा में विधियाँ) - एक method code का एक block या statements का collection या code का एक set होता है जिसे एक certain task या operation perform करने के लिए एक साथ grouped किया गया होता है। इसका use code की reusability (पुन: प्रयोज्यता) achieve करने के लिए किया जाता है। हम एक method को एक बार लिखते हैं और इसे many times use करते हैं। हमें code को again and again लिखने की आवश्यकता नहीं होती है। यह code के easy modification (आसान संशोधन) और readability (पठनीयता) भी provide करता है, बस code का एक chunk adding या removing करके। Method तभी executed होती है जब हम इसे call या invoke करते हैं।

Method Declaration (विधि घोषणा) - Method declaration, method के attributes के बारे में information provide करती है, जैसे visibility, return-type, name, और arguments। इसके six components होते हैं जिन्हें method header के रूप में जाना जाता है, जैसा कि हमने निम्नलिखित figure में दिखाया है।

Method Declaration



Method Signature (विधि हस्ताक्षर) : प्रत्येक method का एक method signature होता है। यह method declaration का एक part होता है। इसमें method name और parameter list शामिल होती है।

Access Specifier (एक्सेस विशिष्टक) : Access specifier या modifier method का access type होता है। यह method की visibility specify करता है। Java four types के access specifier provide करता है:

- **Public:** जब हम अपने application में public specifier का use करते हैं, तो method सभी classes द्वारा accessible होती है।
- **Private:** जब हम एक private access specifier का use करते हैं, तो method केवल उन classes में accessible होती है जिनमें इसे defined किया गया है।
- **Protected:** जब हम protected access specifier का use करते हैं, तो method same package के within या different package की subclasses में accessible होती है।
- **Default:** जब हम method declaration में किसी भी access specifier का use नहीं करते हैं, तो Java default रूप से default access specifier का use करता है। यह केवल same package से ही visible होती है।

Return Type (वापसी प्रकार) : Return type एक data type होता है जिसे method return करता है। इसमें एक primitive data type, object, collection, void, आदि हो सकता है। यदि method कुछ return नहीं करती है, तो हम void keyword का use करते हैं।

Method Name (विधि नाम) : यह एक unique name होता है जिसका use किसी method का name define करने के लिए किया जाता है। यह method की functionality के corresponding होना चाहिए। Suppose, यदि हम दो numbers के subtraction के लिए एक method create कर रहे हैं, तो method name subtraction() होना चाहिए। एक method को उसके name द्वारा invoke किया जाता है।

Parameter List (पैरामीटर सूची) : यह parameters की list होती है जो comma द्वारा separated होती है और parentheses की pair में enclosed होती है। इसमें data type और variable name शामिल होता है। यदि method का कोई parameter नहीं है, तो parentheses को blank छोड़ दें।

Method Body (विधि बॉडी) : यह method declaration का एक part होता है। इसमें performed किए जाने वाले सभी actions शामिल होते हैं। यह curly braces की pair के within enclosed होता है।

Method types (विधि प्रकार) – Return type और parameter के आधार पर methods को four types में classified किया जा सकता है।

1. Method with no argument and no return value–

```
public class ABC {
   public static void greet()
   {
      System.out.println("Hey Geeks! Welcome to No Where.");
   }
   public static void main(String args[])
   {
      greet();
   }
}
```

2. Method with arguments but no return value –

```
public class ABC {
   public static void calc(int x, int y)
   {
      int sum = x + y;
      System.out.print("Sum of two numbers is :" + sum);
   }
   public static void main(String args[])
   {
      int a = 4;
      int b = 5;
      calc(a, b);
   }
}
```

3. Method with no arguments but returns a value –

```
public class ABC {
```

```
    public static int calc()
    {
        int a = 4;
        int b = 5;
        int sum = a + b;
        return sum;
    }
    public static void main(String args[]) {
        int sum = calc();
        System.out.print("Sum of two numbers is :" + sum);
    }
}
```

4. Method with arguments and return value -

```
public class ABC {
    public static int calc(int x, int y)
    {
        int sum = x + y;
        return sum;
    }
    public static void main(String args[]) {
        int a = 4;
        int b = 5;
        int sum = calc(a, b);
        System.out.print("Sum of two numbers is :" + sum);
    }
}
```

Method overloading (विधि ओवरलोडिंग) - यदि किसी class में multiple methods हों जिनका name same हो लेकिन parameters में different हो, तो इसे Method Overloading के रूप में जाना जाता है।

यदि हमें only one operation perform करना है, तो methods का same name होना program की readability को बढ़ाता है।

Suppose आपको given numbers का addition perform करना है लेकिन any number of arguments हो सकते हैं, यदि आप method लिखते हैं जैसे a(int,int) two parameters के लिए, और b(int,int,int) three parameters के लिए तो आपके लिए और other programmers के लिए method के behaviour को understand करना difficult हो सकता है क्योंकि इसका name differs होता है।

Method overloading को Java में Compile-time Polymorphism, Static Polymorphism, या Early binding के रूप में भी जाना जाता है।

Example:

```
public class Sum {
   public int sum(int x, int y)
   {
      return (x + y);
   }
   public int sum(int x, int y, int z)
   {
      return (x + y + z);
   }
   public double sum(double x, double y)
   {
      return (x + y);
   }
   public static void main(String args[]) {
      Sum s = new Sum();
      System.out.println(s.sum(10, 20));
      System.out.println(s.sum(10, 20, 30));
System.out.println(s.sum(10.5, 20.5));
}
}
```

Different Ways of Method Overloading in Java (जावा में विधि ओवरलोडिंग के विभिन्न तरीके)

1. Changing the Number of Parameters. (पैरामीटरों की संख्या बदलना)
2. Changing Data Types of the Arguments. (तर्कों के डेटा प्रकार बदलना)

1. Changing the Number of Parameters (पैरामीटरों की संख्या बदलना) –

Method overloading को different methods को pass करते समय parameters की number changing करके achieved किया जा सकता है।

```
class Product {
   public int multiply(int a, int b)
   {
      int prod = a * b;
      return prod;
   }
   public int multiply(int a, int b, int c)
   {
      int prod = a * b * c;
      return prod;
   }
}

class GFG {
   public static void main(String[] args)
   {
      Product ob = new Product();
      int prod1 = ob.multiply(1, 2);
      System.out.println("Product of the two integer value :" + prod1);

      int prod2 = ob.multiply(1, 2, 3);
      System.out.println("Product of the three integer value :" + prod2);
   }
}
```

2. Changing Data Types of the Arguments (तर्कों के डेटा प्रकार बदलना)

Many cases में, methods को Overloaded considered किया जा सकता है यदि उनका same name हो लेकिन different parameter types हों, methods को overloaded considered किया जाता है।

```
class Product {
```

```
    public int Prod(int a, int b, int c)
    {
        int prod1 = a * b * c;
        return prod1;
    }
    public double Prod(double a, double b, double c)
    {
        double prod2 = a * b * c;
        return prod2;
    }
}

class GFG {
    public static void main(String[] args)
    {
        Product obj = new Product();
        int prod1 = obj.Prod(1, 2, 3);
        System.out.println("Product of the three integer value :"+ prod1);

        double prod2 = obj.Prod(1.0, 2.0, 3.0);
        System.out.println("Product of the three double value :"+ prod2);
    }
}
```

Constructor (कंस्ट्रक्टर) - Java में, एक constructor codes का एक block होता है जो method के similar होता है। इसे तब called किया जाता है जब class का एक instance create किया जाता है। Constructor को call करने के time पर, object के लिए memory allocate की जाती है।

- यह एक special type की method होती है जिसका use object को initialize करने के लिए किया जाता है।
- Every time एक object new() keyword का use करके create किया जाता है, at least one constructor called होता है।
- यह एक default constructor call करता है यदि class में कोई constructor available नहीं है। ऐसे case में, Java compiler default रूप से एक default constructor provide करता है।

Java में दो types के constructors होते हैं: no-argument/default constructor, और parameterized constructor।

Java constructor create करने के Rules (नियम) –

- Constructor name उसकी class name के same होना चाहिए
- एक Constructor का no explicit return type होना चाहिए
- एक Java constructor abstract, static, final, और synchronized नहीं हो सकता

Java constructors के Types (प्रकार) – Java में दो types के constructors होते हैं:

1. Default constructor (no-argument constructor) (डिफ़ॉल्ट कंस्ट्रक्टर)
2. Parameterized constructor (पैरामीटराइज्ड कंस्ट्रक्टर)

1. Java Default Constructor (जावा डिफ़ॉल्ट कंस्ट्रक्टर) – एक constructor को "Default Constructor" called किया जाता है जब इसका कोई parameter नहीं होता है।

Default constructor का Syntax:

```
<class_name>(){ }
```

Default constructor का Example:

```
class Bike1{
  Bike1()
  {
    System.out.println("Bike is created");
  }
  public static void main(String args[]){
    Bike1 b=new Bike1();
  }
}
```

Output:

```
Bike is created
```

Default constructor का Example जो default values display करता है:

```
class Student3{
  int id;
  String name;

  void display()
  {
    System.out.println(id+" "+name);
  }
  public static void main(String args[]){
    Student3 s1=new Student3();
    Student3 s2=new Student3();
    s1.display();
    s2.display();
  }
}
```

Output:

```
0 null
0 null
```

2. Java Parameterized Constructor (जावा पैरामीटराइज्ड कंस्ट्रक्टर) – एक constructor जिसमें specific number of parameters होते हैं, उसे parameterized constructor called किया जाता है।

Parameterized constructor का use क्यों करें?

Parameterized constructor का use distinct objects को different values provide करने के लिए किया जाता है। However, आप same values भी provide कर सकते हैं।

Parameterized constructor का Example:

```
class Student4{
  int id;
  String name;
  Student4(int i,String n)
  {
```

```
        id = i;
        name = n;
    }
    void display()
    {
        System.out.println(id+" "+name);
    }
    public static void main(String args[]){
        Student4 s1 = new Student4(111,"Karan");
        Student4 s2 = new Student4(222,"Aryan");
        s1.display();
        s2.display();
    }
}
```

Output:

```
111 Karan
222 Aryan
```

Constructor Overloading in Java (जावा में कंस्ट्रक्टर ओवरलोडिंग) – Java में, एक constructor एक method की तरह होता है लेकिन return type के बिना। इसे Java methods की तरह overloaded भी किया जा सकता है।

Java में Constructor overloading एक technique है जिसमें एक से अधिक constructors होते हैं जिनकी different parameter lists होती हैं। उन्हें इस तरह arranged किया जाता है कि प्रत्येक constructor एक different task perform करता है। उन्हें compiler द्वारा list में parameters की number और उनके types द्वारा differentiated किया जाता है।

Constructor Overloading का Example:

```
class Student5{
    int id;
    String name;
    int age;
    Student5(int i,String n)
    {
```

```
    id = i;
    name = n;
  }

  Student5(int i,String n,int a)
  {
    id = i;
    name = n;
    age=a;
  }
  void display()
  {
    System.out.println(id+" "+name+" "+age);
  }

  public static void main(String args[]) {
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
  }
}
```

Output:

```
111 Karan 0
222 Aryan 25
```

This Keyword (दिस कीवर्ड) - Java में this keyword एक reference variable होता है जो current object को refer करता है, चाहे वह method का हो या constructor का। Java में this keyword का उपयोग करने का main purpose class attributes और parameters के बीच confusion को दूर करना है जिनके same names होते हैं।

यहाँ java this keyword का usage दिया गया है।

- this का use current class instance variable को refer करने के लिए किया जा सकता है।
- this का use current class method को invoke करने के लिए किया जा सकता है (implicitly)।

- this() का use current class constructor को invoke करने के लिए किया जा सकता है।

1) this: to refer current class instance variable (वर्तमान कक्षा इंस्टेंस चर को संदर्भित करने के लिए)

this keyword का use current class instance variable को refer करने के लिए किया जा सकता है। यदि instance variables और parameters के बीच ambiguity (अस्पष्टता) होती है, तो this keyword ambiguity की problem को resolve करता है।

this keyword के बिना problem को समझना – आइए उस problem को understand करते हैं यदि हम this keyword का use नहीं करते हैं, नीचे दिए गए example द्वारा:

```
class Student{
   int rollno;
   String name;
   float fee;
   Student(int rollno,String name,float fee){
      rollno=rollno;
      name=name;
      fee=fee;
   }

   void display(){
      System.out.println(rollno+" "+name+" "+fee);
   }
}

class TestThis1{
   public static void main(String args[]){
      Student s1=new Student(111,"ankit",5000f);
      Student s2=new Student(112,"sumit",6000f);
      s1.display();
      s2.display();
   }
}
```

Output:

0 null 0.0

0 null 0.0

ऊपर दिए गए example में, parameters (formal arguments) और instance variables same हैं। इसलिए, हम local variable और instance variable को distinguish करने के लिए this keyword का use कर रहे हैं।

this keyword द्वारा उपरोक्त समस्या का समाधान:

```
class Student{
   int rollno;
   String name;
   float fee;
   Student(int rollno,String name,float fee){
      this.rollno=rollno;
      this.name=name;
      this.fee=fee;
   }
   void display(){
      System.out.println(rollno+" "+name+" "+fee);
   }
}

class TestThis2{
   public static void main(String args[]) {
      Student s1=new Student(111,"ankit",5000f);
      Student s2=new Student(112,"sumit",6000f);
      s1.display();
      s2.display();
   }
}
```

Output:

111 ankit 5000.0

112 sumit 6000.0

यदि local variables (formal arguments) और instance variables different हैं, तो this keyword का use करने की कोई आवश्यकता नहीं है जैसा कि निम्नलिखित program में है:

Program जहाँ this keyword required नहीं है:

```
class Student{
   int rollno;
   String name;
   float fee;
   Student(int r,String n,float f){
      rollno=r;
      name=n;
      fee=f;
   }
   void display(){
      System.out.println(rollno+" "+name+" "+fee);
   }
}

class TestThis3{
   public static void main(String args[]){
      Student s1=new Student(111,"ankit",5000f);
      Student s2=new Student(112,"sumit",6000f);
      s1.display();
      s2.display();
   }
}
```

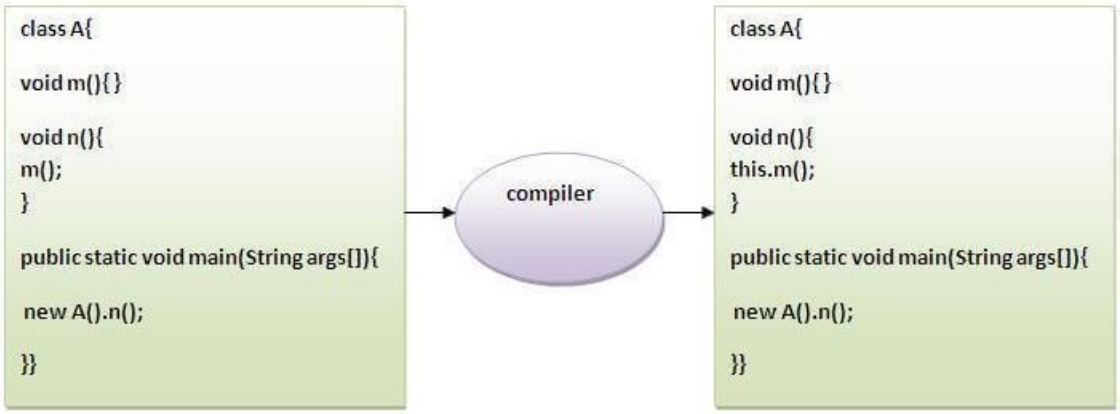
Output:

111 ankit 5000.0

112 sumit 6000.0

2) this: to invoke current class method (वर्तमान कक्षा विधि को आमंत्रित करने के लिए)

आप this keyword का use करके current class की method invoke कर सकते हैं। यदि आप this keyword का use नहीं करते हैं, तो compiler method invoke करते समय automatically this keyword add कर देता है। आइए example देखें




```
class A{
   void m(){
      System.out.println("hello m");
   }

   void n(){
      System.out.println("hello n");
      //m();//same as this.m()
      this.m(); // explicitly using 'this'
   }
}

class TestThisA{
   public static void main(String args[]) {
      A a=new A();
      a.n();
   }
```

```
}
```

Output:

```
hello n
hello m
```

3) this() : to invoke current class constructor (वर्तमान कक्षा कंस्ट्रक्टर को आमंत्रित करने के लिए)

this() constructor call का use current class constructor को invoke करने के लिए किया जा सकता है। इसका use constructor को reuse करने के लिए किया जाता है। दूसरे शब्दों में, इसका use constructor chaining के लिए किया जाता है।

Calling default constructor from parameterized constructor (पैरामीटराइज्ड कंस्ट्रक्टर से डिफ़ॉल्ट कंस्ट्रक्टर को कॉल करना) :

```
class A{
   A(){
      System.out.println("hello a");
   }
   A(int x){
      this(); // calling default constructor
      System.out.println(x);
   }
}

class TestThis5{
   public static void main(String args[]) {
      A a=new A(10);
   }
}
```

Output:

```
hello a
10
```

Calling parameterized constructor from default constructor (डिफ़ॉल्ट कंस्ट्रक्टर से पैरामीटराइज्ड कंस्ट्रक्टर को कॉल करना) :

```
class A{
   A(){
      this(5); // calling parameterized constructor
      System.out.println("hello a");
   }
   A(int x){
      System.out.println(x);
   }
}

class TestThis6{
   public static void main(String args[]){
      A a=new A();
   }
}
```

Output:

```
5
hello a
```

Real usage of this() constructor call (this() कंस्ट्रक्टर कॉल का वास्तविक उपयोग)

this() constructor call का use constructor से constructor को reuse करने के लिए किया जाना चाहिए। यह constructors के बीच chain को maintain करता है यानी इसका use constructor chaining के लिए किया जाता है। आइए नीचे दिया गया example देखें जो this keyword के actual use को display करता है।

```
class Student{
   int rollno;
```

```
    String name,course;
    float fee;
    Student(int rollno,String name,String course){
      this.rollno=rollno;
      this.name=name;
      this.course=course;
    }
    Student(int rollno,String name,String course,float fee){
      this(rollno,name,course);//reusing constructor
      this.fee=fee;
    }
    void display(){
      System.out.println(rollno+" "+name+" "+course+" "+fee);
    }
}

class TestThis7{
    public static void main(String args[]){
      Student s1=new Student(111,"ankit","java");
      Student s2=new Student(112,"sumit","java",6000f);
      s1.display();
      s2.display();
    }
}
```

Output:

```
111 ankit java 0.0
112 sumit java 6000.0
```

- Rule (नियम)： Call to this() must be the first statement in constructor. (कंस्ट्रक्टर में this() को कॉल पहला स्टेटमेंट होना चाहिए।)

Garbage Collector (गार्बेज कलेक्टर)

Garbage Collection in Java एक process है जिसके द्वारा programs memory management automatically perform करते हैं। Garbage Collector(GC) unused objects को find करता है और memory reclaim करने के लिए उन्हें delete कर देता है। Java में, objects की dynamic memory allocation new operator का use करके achieved की जाती है जो कुछ memory का use करता है और memory तब तक allocated रहती है जब तक object के use के लिए references हैं।

जब किसी object का कोई reference नहीं होता है, तो इसे no longer needed माना जाता है, और object द्वारा occupied memory को reclaim किया जा सकता है। किसी object को explicitly destroy करने की कोई आवश्यकता नहीं है क्योंकि Java de-allocation automatically handles करता है।

जो technique इसे accomplish करती है उसे Garbage Collection के रूप में जाना जाता है।

C/C++ में, एक programmer objects के both creation और destruction के लिए responsible होता है। Usually, programmer useless objects के destruction को neglect करता है। इस negligence के कारण, एक certain point पर, new objects create करने के लिए sufficient memory available नहीं हो सकती है, और entire program abnormally terminate हो जाएगी, जिससे OutOfMemoryErrors होती हैं।

लेकिन Java में, programmer को उन सभी objects के लिए care करने की need नहीं है जो no longer in use हैं।

Garbage collector इन objects को destroy कर देता है। Garbage Collector का main objective unreachable objects को destroy करके heap memory को free करना है।

Garbage Collection के Advantages (लाभ) :

- यह java memory को efficient बनाता है क्योंकि garbage collector unreferenced objects को heap memory से remove कर देता है।
- यह automatically garbage collector (JVM का एक part) द्वारा done किया जाता है इसलिए हमें extra efforts करने की need नहीं है।

Object as Parameter (पैरामीटर के रूप में वस्तु)

Java में, जब किसी primitive type को किसी method में passed किया जाता है, तो यह call-by-value के use द्वारा done किया जाता है। Objects implicitly call-by-reference के use द्वारा passed होते हैं।

इसका मतलब है कि जब हम primitive data types को method में pass करते हैं तो यह function parameters के लिए only values pass करेगा इसलिए parameter में made कोई भी change actual parameters के value को affect नहीं करेगा।

जबकि java में Objects reference variables होते हैं, इसलिए objects के लिए एक value जो object का reference है, passed होती है। इसलिए whole object passed नहीं होता है बल्कि इसका reference passed हो जाता है। Method में object के सभी modifications Heap में object को modify करेंगे।

Function में Parameter के रूप में Object Passing (वस्तु पास करना) :

```
class ObjectPassDemo {
   int a, b;
   ObjectPassDemo(int i, int j)
   {
      a = i;
      b = j;
   }
   boolean equalTo(ObjectPassDemo o)
   {
      return (o.a == a && o.b == b);
   }
}

public class GFG {
   public static void main(String args[])
   {
      ObjectPassDemo ob1 = new ObjectPassDemo(100, 22);
      ObjectPassDemo ob2 = new ObjectPassDemo(100, 22);
      ObjectPassDemo ob3 = new ObjectPassDemo(-1, -1);

      System.out.println("ob1 == ob2: "+ ob1.equalTo(ob2));
      System.out.println("ob1 == ob3: "+ ob1.equalTo(ob3));
   }
}
```

Output:

ob1 == ob2: true

ob1 == ob3: false

Defining a constructor that takes an object of its class as a parameter (एक कंस्ट्रक्टर को परिभाषित करना जो अपनी कक्षा की एक वस्तु को पैरामीटर के रूप में लेता है) :

Object parameters के सबसे common uses में से एक instructors को involve करता है। Frequently, practice में, एक new object construct करने की need होती है ताकि यह initially किसी existing object के same हो। ऐसा करने के लिए, या तो हम Object.clone() method का use कर सकते हैं या एक constructor define कर सकते हैं जो अपनी class की object को parameter के रूप में लेता है।

Example:

```
class Box {
   double width, height, depth;
   Box(Box ob) // Constructor taking an object as parameter
   {
      width = ob.width;
      height = ob.height;
      depth = ob.depth;
   }
   Box(double w, double h, double d)
   {
      width = w;
      height = h;
      depth = d;
   }
   double volume() {
      return width * height * depth;
   }
}

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

```
        Box mybox = new Box(10, 20, 15);
        Box myclone = new Box(mybox); // Using the object parameter constructor

        double vol;
        vol = mybox.volume();
        System.out.println("Volume of mybox is " + vol);

        vol = myclone.volume();
        System.out.println("Volume of myclone is " + vol);
    }
}
```

Output:

```
Volume of mybox is 3000.0
Volume of myclone is 3000.0
```

Returning Objects (वस्तुओं को लौटाना) :

Java में, एक method किसी भी type का data return कर सकती है, including objects। उदाहरण के लिए, निम्नलिखित program में, incrByTen() method एक object return करती है जिसमें a (एक integer variable) का value invoking object की तुलना में ten greater होता है।

Example:

```
class ObjectReturnDemo {
    int a;
    ObjectReturnDemo(int i) {
        a = i;
    }

    ObjectReturnDemo incrByTen(){
        ObjectReturnDemo temp = new ObjectReturnDemo(a + 10);
        return temp;
    }
}
```

```
public class GFG {
   public static void main(String args[]){
      ObjectReturnDemo ob1 = new ObjectReturnDemo(2);
      ObjectReturnDemo ob2;

      ob2 = ob1.incrByTen();

      System.out.println("ob1.a: " + ob1.a);
      System.out.println("ob2.a: " + ob2.a);
   }
}
```

Output:

```
ob1.a: 2
ob2.a: 12
```

Array in Java (जावा में सरणी) - Array एक object होता है जिसमें similar data type के elements होते हैं। Additionally, किसी array के elements एक contiguous memory location में stored होते हैं। यह एक data structure होती है जहाँ हम similar elements store करते हैं। हम एक Java array में only a fixed set of elements store कर सकते हैं।

Java में Array index-based होती है, array का first element 0th index पर stored होता है, 2nd element 1st index पर stored होता है और so on।

हम length member का use करके array की length प्राप्त कर सकते हैं।

Java में Array के Types (प्रकार)

दो types की arrays होती हैं।

- Single Dimensional Array (एकल-आयामी सरणी)
- Multidimensional Array (बहु-आयामी सरणी)

Java में Single Dimensional Array (एकल-आयामी सरणी) :

Java में Array Declare करने का Syntax (वाक्य-विन्यास) :

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

```
dataType arr[];
```

Java में Array की Instantiation (अभिव्यक्ति) :

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

Java Array का Example (उदाहरण) :

आइए java array का simple example देखें, जहाँ हम एक array को declare, instantiate, initialize और traverse करने जा रहे हैं।

```
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]=70;
      a[3]=40;
      a[4]=50;

      for(int i=0;i<a.length;i++){
         System.out.println(a[i]);
      }
   }
}
```

Output:

```
10
20
70
40
50
```

Java Array का Declaration, Instantiation और Initialization (घोषणा, अभिव्यक्ति और प्रारंभीकरण) :

हम java array को एक साथ declare, instantiate और initialize कर सकते हैं:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

आइए इस array को print करने का simple example देखें।

```
class Testarray1{
   public static void main(String args[]){
      int a[]={33,3,4,5};
      for(int i=0;i<a.length;i++){
         System.out.println(a[i]);
      }
   }
}
```

Output:

```
33
3
4
5
```

For-each Loop for Java Array (जावा सरणी के लिए फॉर-इच लूप) – हम for-each loop का use करके Java array को भी print कर सकते हैं। Java for-each loop array elements को one by one print करती है। यह एक variable में array element hold करती है, फिर loop के body को execute करती है।

for-each loop का syntax नीचे दिया गया है:

```
for(data_type variable:array){
//body of the loop
}
```

आइए हम for-each loop का use करके Java array के elements को print करने का example देखें।

```
class Testarray1{
   public static void main(String args[]){
      int arr[]={33,3,4,5};
      for(int i:arr){
         System.out.println(i);
      }
```

```
    }
}
```

Output:

```
33
3
4
5
```

Passing Array to a Method in Java (जावा में एक विधि को सरणी पास करना) :

हम java array को method में pass कर सकते हैं ताकि हम किसी भी array पर same logic का reuse कर सकें। आइए किसी array की minimum number प्राप्त करने के लिए method का use करने का simple example देखें।

```
class Testarray2{
  static void min(int arr[]){
    int min=arr[0];
    for(int i=1;i<arr.length;i++){
      if(min>arr[i])
        min=arr[i];
    }
    System.out.println(min);
  }

  public static void main(String args[]){
    int a[]={33,3,4,5};//declaring and initializing an array
    min(a);//passing array to method
  }
}
```

Output:

```
3
```

Multidimensional Array in Java (जावा में बहु-आयामी सरणी) :

ऐसे case में, data row और column based index में stored होता है (जिसे matrix form के रूप में भी जाना जाता है)।

Java में Multidimensional Array Declare करने का Syntax (वाक्य-विन्यास):

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

Java में Multidimensional Array Instantiate करने का Example (उदाहरण):

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

Java में Multidimensional Array Initialize करने का Example (उदाहरण):

```
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;
```

Multidimensional Java Array का Example (उदाहरण):

आइए 2Dimensional array को declare, instantiate, initialize और print करने का simple example देखें।

```
class Testarray3{
  public static void main(String args[]) {
    int arr[][]={{1,2,3},{2,4,5},{4,4,5}};

    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
```

Addition of 2 Matrices in Java (जावा में 2 मैट्रिक्स का जोड़) :

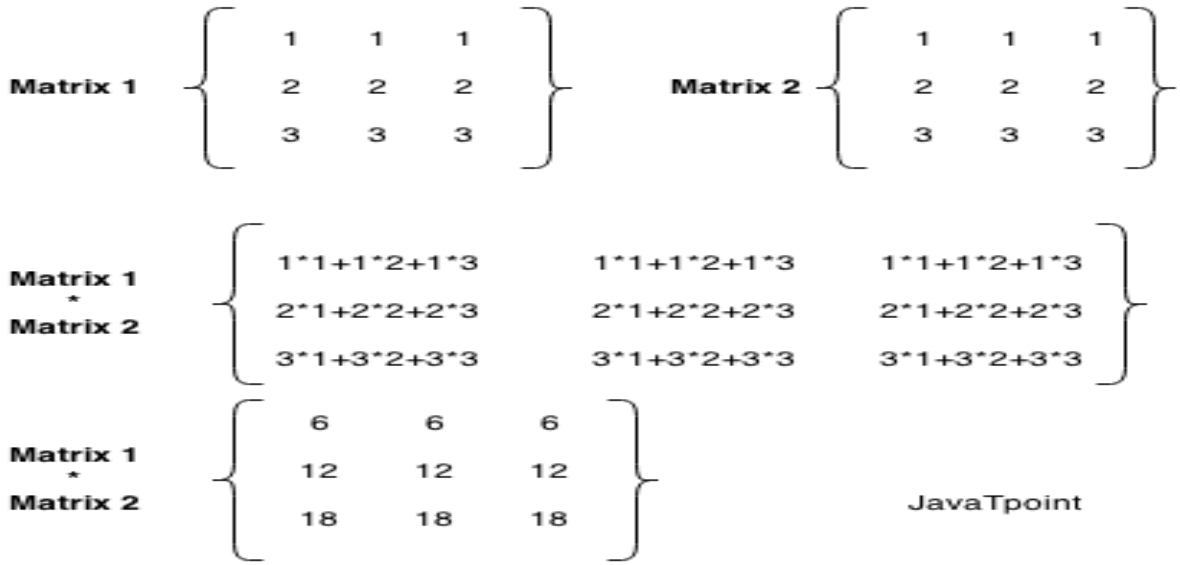
```
class Testarray5{
    public static void main(String args[]) {
        int a[][]={{1,3,4},{3,4,5}};
        int b[][]={{1,3,4},{3,4,5}};

        int c[][]=new int[2][3];

        for(int i=0;i<2;i++) {
            for(int j=0;j<3;j++) {
                c[i][j]=a[i][j]+b[i][j];
                System.out.print(c[i][j]+" ");
            }
            System.out.println();//new line
        }
    }
}
```

Output:

```
2 6 8
6 8 10
```

Multiplication of 2 Matrices in Java (जावा में 2 मैट्रिक्स का गुणन) :

Matrix multiplication के case में, first matrix का one-row element, second matrix के सभी columns से multiplied होता है जिसे नीचे दी गई image द्वारा understood किया जा सकता है।



आइए 3 rows और 3 columns के दो matrices को multiply करने का एक simple example देखें।

```
public class MatrixMultiplicationExample{
  public static void main(String args[]) {
    int a[][]={{1,1,1},{2,2,2},{3,3,3}};
    int b[][]={{1,1,1},{2,2,2},{3,3,3}};

    int c[][]=new int[3][3]; //3 rows and 3 columns

    //multiplying and printing multiplication of 2 matrices
    for(int i=0;i<3;i++) {
      for(int j=0;j<3;j++) {
        c[i][j]=0;
        for(int k=0;k<3;k++) {
          c[i][j]+=a[i][k]*b[k][j];
        }//end of k loop
        System.out.print(c[i][j]+" "); //printing matrix element
```

```
        }//end of j loop
        System.out.println();//new line
      }
    }
}
```

Output:

```
6 6 6
12 12 12
18 18 18
```

Wrapper Class (रैपर क्लास) - Java में wrapper class primitive को object में और object को primitive में convert करने की mechanism provide करती है।

Java.lang package की eight classes को Java में wrapper classes के रूप में जाना जाता है। Eight wrapper classes की list नीचे दी गई है:

Primitive Type (आदिम प्रकार)	Wrapper class (रैपर क्लास)
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Autoboxing (ऑटोबॉक्सिंग) : Primitive data type का अपने corresponding wrapper class में automatic conversion, autoboxing के रूप में जाना जाता है, उदाहरण के लिए, byte से Byte, char से Character, int से Integer, long से Long, float से Float, boolean से Boolean, double से Double, और short से Short।

Wrapper class Example: Primitive to Wrapper (आदिम से रैपर)

```
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 20 20
```

Unboxing (अनबॉक्सिंग) : Wrapper type का अपने corresponding primitive type में automatic conversion, unboxing के रूप में जाना जाता है। यह autoboxing की reverse process है। Java 5 से, हमें wrapper type को primitives में convert करने के लिए wrapper classes की intValue() method का use करने की need नहीं है।

Wrapper class Example: Wrapper to Primitive (रैपर से आदिम)

```
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);
  }
}
```

Output:

```
3 3 3
```

Inheritance in Java (जावा में इनहेरिटेंस) : Java में Inheritance एक mechanism है जिसमें एक object parent object की सभी properties और behaviors acquire कर लेता है। यह OOPs (Object Oriented programming system) का एक important part है।

Java में inheritance के पीछे idea यह है कि आप new classes create कर सकते हैं जो existing classes पर built upon हैं। जब आप किसी existing class से inherit करते हैं, तो आप parent class के methods और fields का reuse कर सकते हैं। Moreover, आप अपनी current class में new methods और fields भी add कर सकते हैं।

Java में inheritance का use क्यों करें

- For Method Overriding (इसलिए runtime polymorphism achieved की जा सकती है)।
- For Code Reusability (कोड पुन: प्रयोज्यता के लिए)।

Inheritance में used Terms (शब्द)

- Class (कक्षा): एक class objects का एक group होता है जिसमें common properties होती हैं। यह एक template या blueprint होता है जिससे objects create किए जाते हैं।
- Sub Class/Child Class (उप कक्षा/बाल कक्षा): Subclass एक class होती है जो other class को inherits करती है। इसे derived class, extended class, या child class भी called किया जाता है।
- Super Class/Parent Class (सुपर क्लास/पैरेंट क्लास): Superclass वह class होती है जहाँ से एक subclass features inherits करती है। इसे base class या parent class भी called किया जाता है।
- Reusability (पुन: प्रयोज्यता): जैसा कि name specifies करता है, reusability एक mechanism है जो आपको existing class के fields और methods का reuse करने में facilitates करती है जब आप एक new class create करते हैं। आप same fields और methods का use कर सकते हैं जो पहले से ही previous class में defined हैं।

Java Inheritance का Syntax (वाक्य-विन्यास):

```
class Subclass-name extends Superclass-name
{
//methods and fields
}
```

Example:

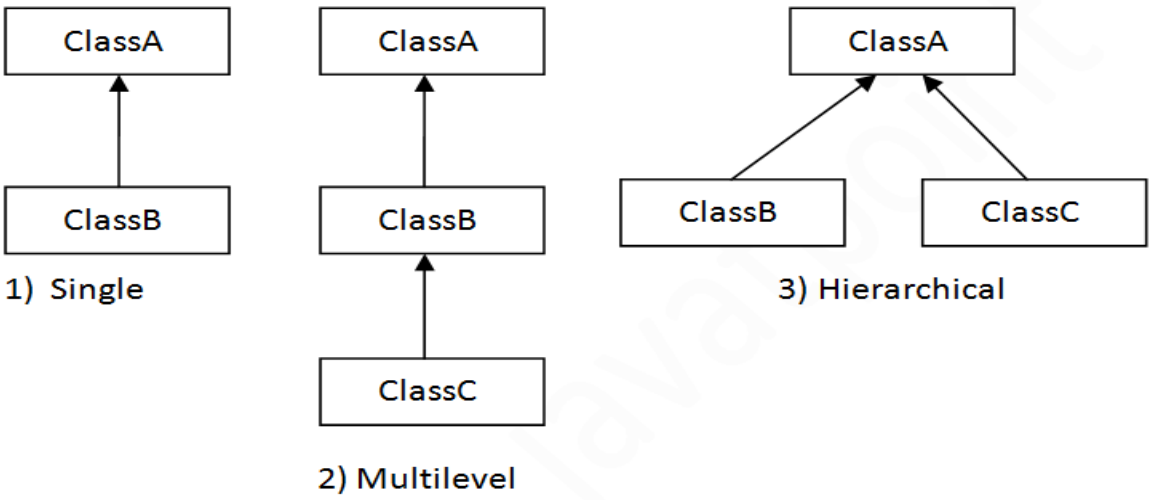
```
class Employee{
   float salary=40000;
}

class Programmer extends Employee{
   int bonus=10000;
   public static void main(String args[]) {
      Programmer 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
```

Types of inheritance in java (जावा में इनहेरिटेंस के प्रकार) :

Class के basis पर, java में inheritance के three types हो सकते हैं: single, multilevel और hierarchical।



Single Inheritance Example (सिंगल इनहेरिटेंस उदाहरण) :

जब एक class another class inherits करती है, तो इसे single inheritance के रूप में जाना जाता है। नीचे दिए गए example में, Dog class Animal class inherits करती है, इसलिए single inheritance है।

```
class Animal{
  void eat(){
    System.out.println("eating...");
  }
}

class Dog extends Animal{
  void bark(){
    System.out.println("barking...");
  }
}
class TestInheritance{
  public static void main(String args[]) {
    Dog d=new Dog();
```

```
        d.bark();
        d.eat();
    }
}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example (मल्टीलेवल इनहेरिटेंस उदाहरण) :

जब inheritance की एक chain होती है, तो इसे multilevel inheritance के रूप में जाना जाता है। जैसा कि आप नीचे दिए गए example में देख सकते हैं, BabyDog class, Dog class inherits करती है जो फिर से Animal class inherits करती है, इसलिए multilevel inheritance है।

```
class Animal{
  void eat(){
    System.out.println("eating...");
  }
}

class Dog extends Animal{
  void bark(){
    System.out.println("barking...");
  }
}

class BabyDog extends Dog{
  void weep(){
    System.out.println("weeping...");
  }
}
class TestInheritance2{
```

```
    public static void main(String args[]){
      BabyDog d=new BabyDog();
      d.weep();
      d.bark();
      d.eat();
    }
}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example (हाइरार्किकल इनहेरिटेंस उदाहरण) :

जब दो या दो से अधिक classes एक single class inherits करती हैं, तो इसे hierarchical inheritance के रूप में जाना जाता है। नीचे दिए गए example में, Dog और Cat classes Animal class inherits करती हैं, इसलिए hierarchical inheritance है।

```
class Animal{
  void eat(){
    System.out.println("eating...");
  }
}

class Dog extends Animal{
  void bark(){
    System.out.println("barking...");
  }
}
class Cat extends Animal{
  void meow(){
    System.out.println("meowing...");
  }
```

```
}
class TestInheritance3{
  public static void main(String args[]) {
    Cat c=new Cat();
    c.meow();
    c.eat();
    //c.bark();//C.T.Error - Cat class doesn't have bark()
  }
}
```

Output:

```
meowing...
eating...
```

Super Keyword in Java (जावा में सुपर कीवर्ड) :

Java में super keyword एक reference variable होता है जिसका use immediate parent class object को refer करने के लिए किया जाता है।

Whenever आप subclass का instance create करते हैं, parent class का एक instance implicitly created होता है जिसे super reference variable द्वारा referred to किया जाता है।

Java super Keyword का Usage (उपयोग) :

- super का use immediate parent class instance variable को refer करने के लिए किया जा सकता है।
- super का use immediate parent class method को invoke करने के लिए किया जा सकता है।
- super() का use immediate parent class constructor को invoke करने के लिए किया जा सकता है।

Example:

```
class Person{
  int id;
  String name;
  Person(int id,String name){
    this.id=id;
    this.name=name;
  }
}
class Emp extends Person{
```

```
    float salary;
    Emp(int id,String name,float salary){
       super(id,name);//reusing parent constructor
       this.salary=salary;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}

class TestSuper5{
    public static void main(String[] args){
       Emp e1=new Emp(1,"ankit",45000f);
       e1.display();
    }
}
```

Output:

```
1 ankit 45000
```

Method Overriding in Java (जावा में विधि ओवरराइडिंग) :

यदि subclass (child class) में parent class में declared method के same method हों, तो इसे Java में method overriding के रूप में जाना जाता है।

दूसरे शब्दों में, यदि कोई subclass उस method की specific implementation provide करती है जिसे उसकी parent classes में से एक द्वारा declare किया गया है, तो इसे method overriding के रूप में जाना जाता है।

Java Method Overriding का Usage (उपयोग)

- Method overriding का use किसी method की specific implementation provide करने के लिए किया जाता है जो पहले से ही उसकी superclass द्वारा provided की जा चुकी है।
- Method overriding का use runtime polymorphism के लिए किया जाता है।

Java Method Overriding के Rules (नियम)

1. Method का name parent class में जैसा है वैसा ही होना चाहिए।
2. Method का parameter parent class में जैसा है वैसा ही होना चाहिए।
3. एक IS-A relationship (inheritance) होनी चाहिए।

Method overriding के बिना problem को समझना :

आइए उस problem को understand करते हैं जिसका हम program में सामना कर सकते हैं यदि हम method overriding का use नहीं करते हैं।

```
class Vehicle{
  void run(){
    System.out.println("Vehicle is running");
  }
}

class Bike extends Vehicle{
  public static void main(String args[]){
    Bike obj = new Bike();
    obj.run();
  }
}
```

Output:

```
Vehicle is running
```

Problem यह है कि मुझे subclass में run() method की एक specific implementation provide करनी है that is why we use method overriding.

Method overriding का Example (उदाहरण) :

इस example में, हमने subclass में run method को defined किया है जैसा कि parent class में defined है लेकिन इसकी कुछ specific implementation है। Method का name और parameter same हैं, और classes के बीच IS-A relationship है, इसलिए method overriding है।

```
class Vehicle{
  void run(){
    System.out.println("Vehicle is running");
  }
}
class Bike2 extends Vehicle{
  void run(){
```

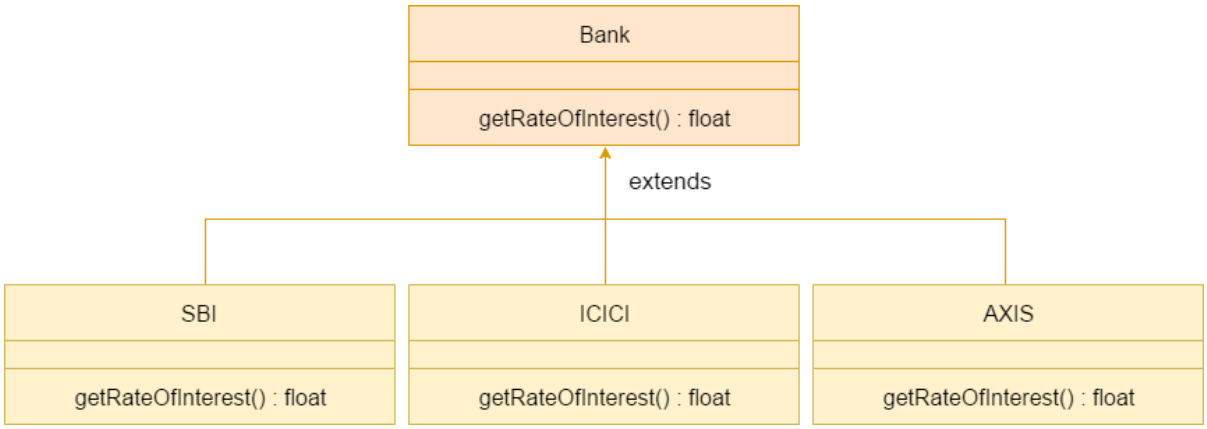
```
    System.out.println("Bike is running safely");
  }
  public static void main(String args[]){
    Bike2 obj = new Bike2();//creating object
    obj.run();//calling method
  }
}
```

Output:

Bike is running safely

A real example of Java Method Overriding (जावा विधि ओवरराइडिंग का एक वास्तविक उदाहरण) :

एक scenario consider करें जहाँ Bank एक class है जो interest की rate प्राप्त करने की functionality provide करती है। However, interest की rate banks के according varies होती है। उदाहरण के लिए, SBI, ICICI और AXIS banks क्रमशः 8%, 7%, और 9% interest की rate provide कर सकते हैं।




```
class Bank{
  int getRateOfInterest(){
    return 0;
  }
}
class SBI extends Bank{
  int getRateOfInterest(){
    return 8;
```

```
    }
}
class ICICI extends Bank{
    int getRateOfInterest(){
        return 7;
    }
}

class AXIS extends Bank{
    int getRateOfInterest(){
        return 9;
    }
}

class Test2{
    public static void main(String args[]){
        SBI s=new SBI();
        ICICI i=new ICICI();
        AXIS a=new AXIS();
        System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
        System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
        System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
    }
}
```

Output:

```
SBI Rate of Interest: 8
ICICI Rate of Interest: 7
AXIS Rate of Interest: 9
```

Dynamic Method Dispatch (डायनामिक मेथड डिस्पैच) :

Method overriding उन ways में से एक है जिनमें Java Runtime Polymorphism को support करता है। Dynamic method dispatch वह mechanism है जिसके द्वारा किसी overridden method के call को run time पर resolve किया जाता है, बजाय compile time के।

- जब किसी overridden method को superclass reference के through call किया जाता है, तो Java determine करता है कि उस method का which version (superclass/subclasses) executed होना है, उस object के type के based upon जिसे call occur होने के time पर referred to किया जा रहा है। Thus, यह determination run time पर made किया जाता है।
- At run-time, यह उस object के type पर depend करता है जिसे referred to किया जा रहा है (reference variable के type पर नहीं) जो determine करता है कि overridden method का which version executed होगा।
- एक superclass reference variable किसी subclass object को refer कर सकती है। इसे upcasting के रूप में भी जाना जाता है। Java इस fact का use overridden methods के calls को run time पर resolve करने के लिए करता है।

Therefore, यदि कोई superclass एक method contain करती है जिसे subclass द्वारा override किया गया है, तो जब different types के objects को superclass reference variable के through referred to किया जाता है, तो method के different versions executed होते हैं। यहाँ एक example है जो dynamic method dispatch को illustrate करता है:

```
class A{
  void m1(){
    System.out.println("Inside A's m1 method");
  }
}
class B extends A{
  // overriding m1()
  void m1(){
    System.out.println("Inside B's m1 method");
  }
}
class C extends A
{
  // overriding m1()
  void m1(){
    System.out.println("Inside C's m1 method");
```

```
    }
}


class Dispatch
{
    public static void main(String args[]) {
        // object of type A
        A a = new A();
        // object of type B
        B b = new B();
        // object of type C
        C c = new C();
        // obtain a reference of type A
        A ref;
        // ref refers to an A object
        ref = a;
        // calling A's version of m1()
        ref.m1();
        // now ref refers to a B object
        ref = b;
        // calling B's version of m1()
        ref.m1();
        // now ref refers to a C object
        ref = c;
        // calling C's version of m1()
        ref.m1();
    }
}
```

Output:

```
Inside A's m1 method
```

Inside B's m1 method

Inside C's m1 method

Unit-3
Packages, Interfaces & Exception Handling

Abstract class in Java

Java में एक class जो abstract keyword के साथ declare की जाती है, उसे abstract class कहते हैं। इसमें abstract और non-abstract methods (method with body) दोनों हो सकते हैं।

Java abstract class सीखने से पहले, पहले Java में abstraction को समझते हैं।

Abstraction in Java

Abstraction एक process है जिसमें implementation details को hide करके केवल functionality को user को दिखाया जाता है।

दूसरे शब्दों में, यह user को केवल essential things दिखाता है और internal details को hide करता है, उदाहरण के लिए, SMS भेजना जहां आप text type करके message send करते हैं। आपको message delivery के internal processing के बारे में पता नहीं होता।

Abstraction आपको object "क्या करता है" उस पर focus करने देती है, "यह कैसे करता है" उस पर नहीं।

Abstraction achieve करने के तरीके

Java में abstraction दो तरीकों से achieve की जा सकती है:

- Abstract class (0 to 100%)
- Interface (100%)

Abstract class in Java

एक class जिसे abstract declare किया जाता है, उसे abstract class कहते हैं। इसमें abstract और non-abstract methods हो सकते हैं। इसे extend किया जाना चाहिए और इसके method implemented किए जाने चाहिए। इसका object नहीं बनाया जा सकता (instantiate)।

Points to Remember

- एक abstract class को abstract keyword के साथ declare किया जाना चाहिए।
- इसमें abstract और non-abstract methods हो सकते हैं।
- इसका object नहीं बनाया जा सकता (instantiate)।
- इसमें constructors और static methods भी हो सकते हैं।
- इसमें final methods हो सकते हैं जो subclass को method के body को change करने से रोकेंगे।

Abstract class का syntax

abstract class A{}

Abstract Method in Java

एक method जिसे abstract declare किया जाता है और जिसकी कोई implementation नहीं होती, उसे abstract method कहते हैं।

Abstract method का syntax

abstract void printStatus(); //no method body and abstract

Abstract class का Example जिसमें एक abstract method है

इस example में, Bike एक abstract class है जिसमें केवल एक abstract method run है। इसकी implementation Honda class द्वारा provide की गई है।

```
abstract class Bike{
   abstract void run();
}
class Honda4 extends Bike{
   void run(){
      System.out.println("running safely");
   }
   public static void main(String args[]){
      Bike obj = new Honda4();
      obj.run();
   }
}
```

Output:

running safely

Abstract class का real scenario

इस example में, Shape abstract class है, और इसकी implementation Rectangle और Circle classes द्वारा provide की गई है।

```
abstract class Shape{
    abstract void draw();
}
class Rectangle extends Shape{
    void draw(){
        System.out.println("drawing rectangle");
    }
}
class Circle1 extends Shape{
    void draw(){
        System.out.println("drawing circle");
    }
}
class TestAbstraction1{
    public static void main(String args[]){
        Shape s=new Circle1();
        s.draw();
    }
}
```

Output:

```
drawing circle
```

Abstract class जिसमें constructor, data member और methods हों:

एक abstract class में data member, abstract method, method body (non-abstract method), constructor, और यहां तक कि main() method भी हो सकते हैं।

```
abstract class Bike{
    Bike(){
        System.out.println("bike is created");
    }
 abstract void run();

    void changeGear(){
        System.out.println("gear changed");
    }
}
class Honda extends Bike{
    void run(){
        System.out.println("running safely...");
    }
}
class TestAbstraction2{
    public static void main(String args[]){
        Bike obj = new Honda();
        obj.run();
        obj.changeGear();
    }
}
```

Output:

bike is created

running safely..

gear changed

Interface in Java

Java में एक interface, class का blueprint होता है। इसमें static constants और abstract methods होते हैं।

Java में interface abstraction achieve करने का एक mechanism है। Java interface में केवल abstract methods हो सकते हैं, method body नहीं। इसका use Java में abstraction और multiple inheritance achieve करने के लिए किया जाता है।

दूसरे शब्दों में, आप कह सकते हैं कि interfaces में abstract methods और variables हो सकते हैं। इसकी method body नहीं हो सकती।

इसका object नहीं बनाया जा सकता, ठीक abstract class की तरह।

Java interface क्यों use करें?

Interface use करने के मुख्यतः तीन reasons हैं। वे नीचे दिए गए हैं।

- इसका use abstraction achieve करने के लिए किया जाता है।
- Interface के द्वारा, हम multiple inheritance की functionality support कर सकते हैं।
- इसका use loose coupling achieve करने के लिए किया जा सकता है।

Interface कैसे declare करें?

एक interface को interface keyword का use करके declare किया जाता है। यह total abstraction provide करता है; मतलब interface के सभी methods empty body के साथ declared होते हैं, और सभी fields default रूप से public, static और final होते हैं। एक class जो interface को implement करती है, उसे interface में declared सभी methods को implement करना होगा।

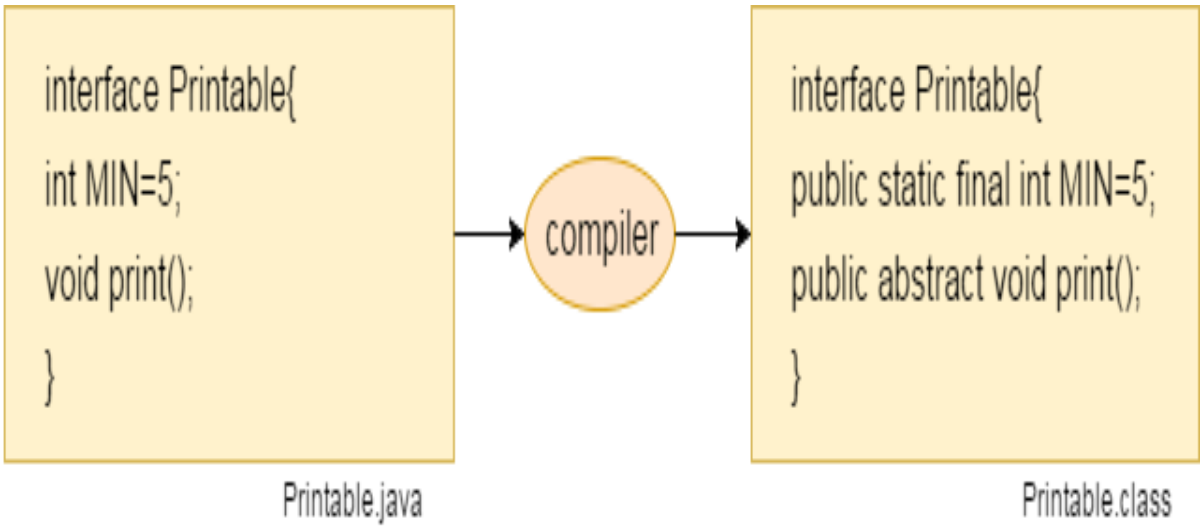
Syntax:

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

Compiler द्वारा internal addition:

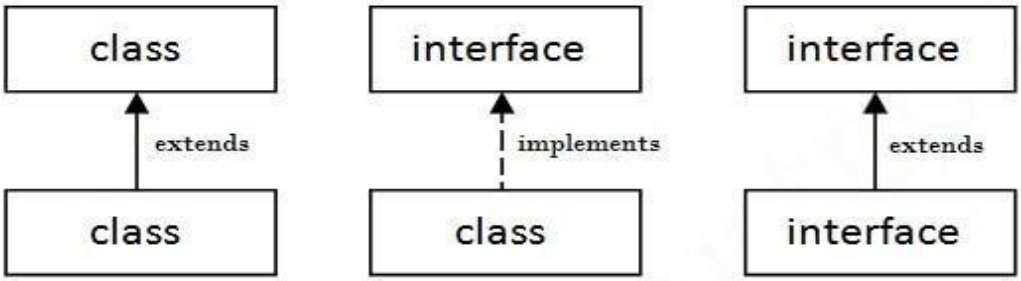
Java compiler interface method से पहले public और abstract keywords add कर देता है। इसके अलावा, यह data members से पहले public, static और final keywords add कर देता है।

दूसरे शब्दों में, Interface fields default रूप से public, static और final होते हैं, और methods public और abstract होते हैं।



Classes और interfaces के बीच relationship:

नीचे दिए गए figure में दिखाए अनुसार, एक class दूसरी class को extend करती है, एक interface दूसरे interface को extend करता है, लेकिन एक class interface को implement करती है।



Java Interface Example 1:

इस example में, Printable interface में केवल एक method है, और इसकी implementation A6 class में provide की गई है।

```
interface printable{
    void print();
}
class A6 implements printable{
    public void print(){
        System.out.println("Hello");
    }
    public static void main(String args[]){
        A6 obj = new A6();
        obj.print();
    }
}
```

Output:

Hello

Java Interface Example 2:

इस example में, Drawable interface में केवल एक method है। इसकी implementation Rectangle और Circle classes द्वारा provide की गई है। एक real scenario में, एक interface किसी और के द्वारा define किया जाता है, लेकिन इसकी implementation अलग-अलग implementation providers द्वारा provide की जाती है। इसके अलावा, इसका use कोई और करता है। Implementation part user द्वारा hide कर दिया जाता है जो interface का use करता है।

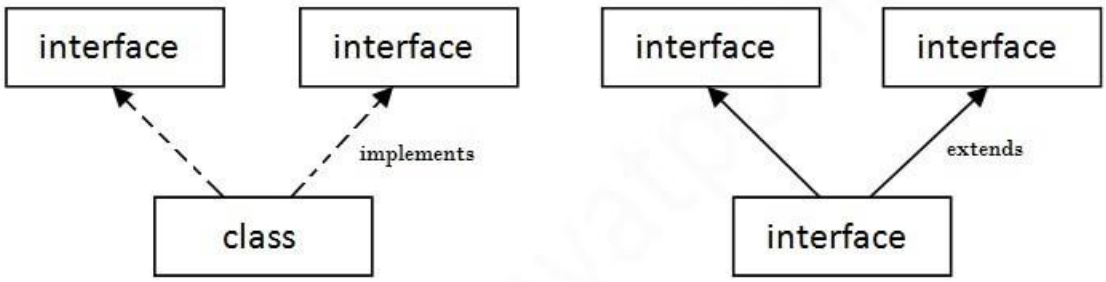
```
interface Drawable{
    void draw();
}
class Rectangle implements Drawable{
    public void draw(){
        System.out.println("drawing rectangle");
    }
}
class Circle implements Drawable{
    public void draw(){
        System.out.println("drawing circle");
    }
}
class TestInterface1{
    public static void main(String args[]){
        Drawable d=new Circle();
        d.draw();
    }
}
```

Output:

drawing circle

Interface द्वारा Java में Multiple inheritance

यदि एक class multiple interfaces को implement करती है, या एक interface multiple interfaces को extend करता है, तो इसे multiple inheritance कहा जाता है।



Multiple Inheritance in Java

```
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
```

क्यों, Java में class के through multiple inheritance supported नहीं है, लेकिन interface द्वारा possible है ?

जैसा कि हमने inheritance chapter में explain किया है, class के case में ambiguity के कारण multiple inheritance supported नहीं है। हालांकि, interface के case में यह supported है क्योंकि यहां कोई ambiguity नहीं है। ऐसा इसलिए है क्योंकि इसकी implementation implementation class द्वारा provide की जाती है। उदाहरण के लिए:

```
interface Printable{
    void print();
}
interface Showable{
    void print();
}
class TestInterface3 implements Printable, Showable{
    public void print(){
        System.out.println("Hello");
    }
```

```
    public static void main(String args[]){
        TestInterface3 obj = new TestInterface3();
        obj.print();
    }
}
```

Output:

```
Hello
```

जैसा कि आप ऊपर के example में देख सकते हैं, Printable और Showable interface में same methods हैं लेकिन इसकी implementation class TestInterface3 द्वारा provide की गई है, इसलिए कोई ambiguity नहीं है।

Interface inheritance:

एक class एक interface को implement करती है, लेकिन एक interface दूसरे interface को extend करता है।

```
interface Printable{
    void print();
}
interface Showable extends Printable{
    void show();
}
class TestInterface4 implements Showable{
    public void print(){
        System.out.println("Hello");
    }
    public void show(){
        System.out.println("Welcome");
```

```
    }
    public static void main(String args[]){
        TestInterface4 obj = new TestInterface4();
        obj.print();
        obj.show();
    }
}
```

Output:

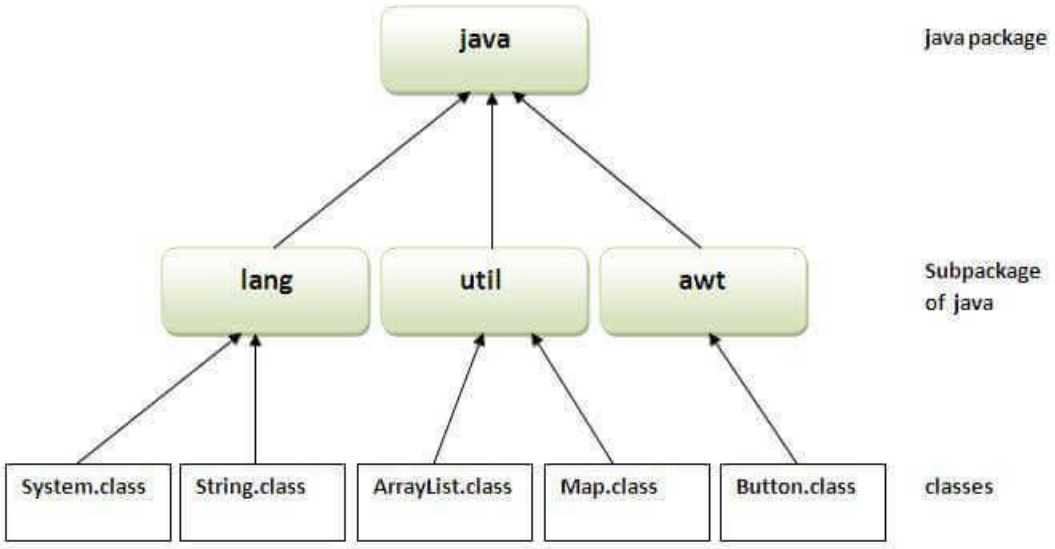
```
Hello
Welcome
```

Java Package:

Java में packages का use naming conflict से बचने और class, interface, sub-classes, आदि के access को control करने के लिए किया जाता है। एक package को similar types की classes, sub-classes, interfaces या enumerations, आदि के group के रूप में define किया जा सकता है। Packages का use करते समय related classes को locate या find करना easier हो जाता है और packages classes और files की huge amount के साथ project की एक good structure या outline provide करते हैं।

Java Package के Advantage:

1. Java package का use classes और interfaces को categorize करने के लिए किया जाता है ताकि उन्हें easily maintained किया जा सके।
2. Java package access protection provide करता है।
3. Java package naming collision को remove करता है।

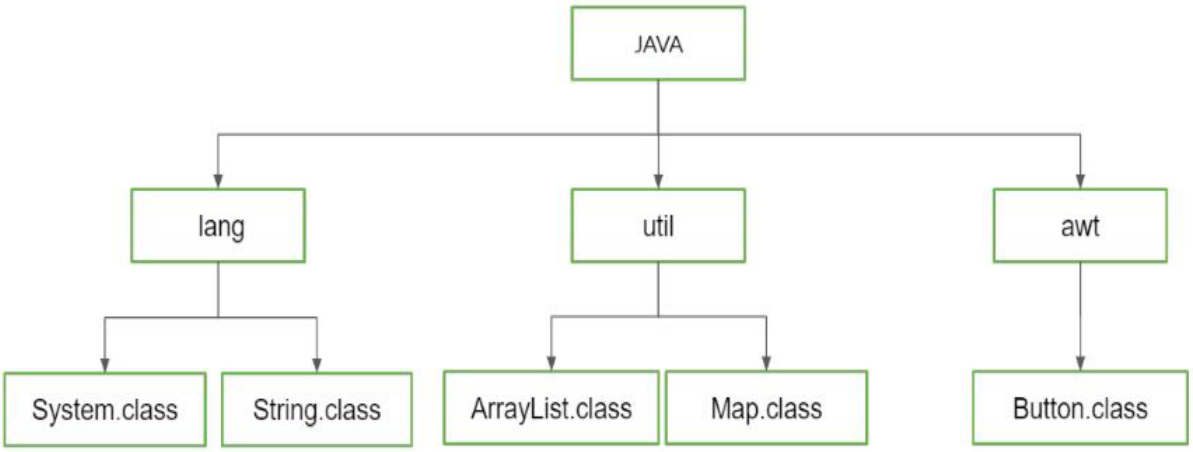


Packages को दो भागों में divided किया गया है:

- Built-in packages: Java में, हमारे पास पहले से ही various pre-defined packages हैं और इन packages में large numbers की classes और interfaces हैं जिन्हें हम Java में use करते हैं, इन्हें Built-in packages कहा जाता है।
- User-defined packages: जैसा कि name से पता चलता है, user-defined packages वह package होता है जिसे user या programmer द्वारा define किया जाता है।

Built-in packages / Java API:

वे packages जो JDK के साथ आते हैं या जिन्हें आप download करते हैं, built-in packages कहलाते हैं। Built-in packages JAR files के रूप में आते हैं और जब हम JAR files को unzip करते हैं तो हम easily JAR files में packages देख सकते हैं, उदाहरण के लिए, lang, io, util, SQL, आदि। Java various built-in packages provide करता है, उदाहरण के लिए java.awt।



Built-in Packages के Examples:

- java.sql: Database में stored data को access और process करने के लिए classes provide करता है। Classes जैसे Connection, DriverManager, PreparedStatement, ResultSet, Statement, आदि इस package का part हैं।
- java.lang: Classes और interfaces contain करता है जो Java programming language के design के लिए fundamental हैं। Classes जैसे String, StringBuffer, System, Math, Integer, आदि इस package का part हैं।
- java.util: Collections framework, some internationalization support classes, properties, random number generation classes contain करता है। Classes जैसे ArrayList, LinkedList, HashMap, Calendar, Date, Time Zone, आदि इस package का part हैं।
- java.net: Networking applications implement करने के लिए classes provide करता है। Classes जैसे Authenticator, HTTP Cookie, Socket, URL, URLConnection, URLEncoder, URLDecoder, आदि इस package का part हैं।
- java.io: System input/output operations के लिए classes provide करता है। Classes जैसे BufferedReader, BufferedWriter, File, InputStream, OutputStream, PrintStream, Serializable, आदि इस package का part हैं।
- java.awt: User interfaces create करने और graphics और images paint करने के लिए classes contain करता है। Classes जैसे Button, Color, Event, Font, Graphics, Image, आदि इस package का part हैं।

User-Defined packages:

Example:

package keyword का use Java में package create करने के लिए किया जाता है।

```
//save as Simple.java
package mypack;
public class Simple{
    public static void main(String args[]) {
        System.out.println("Welcome to package");
    }
}
```

Output:

Welcome to package

Java package को कैसे compile करें

javac -d directory javafilename

For example:

javac -d . Simple.java

-d switch destination specify करती है कि generated class file कहां रखी जाए। आप कोई भी directory name use कर सकते हैं जैसे /home (Linux के case में), d:/abc (windows के case में) आदि। यदि आप package को same directory के within रखना चाहते हैं, तो आप . (dot) use कर सकते हैं।

Java package program को कैसे run करें

Class को run करने के लिए आपको fully qualified name use करने की need होती है, जैसे mypack.Simple आदि।

For example:

java mypack.Simple

किसी दूसरे package से package को कैसे access करें?

Package के outside से package को access करने के three ways हैं।

1. import package.*;
2. import package.classname;
3. fully qualified name.

1) packagename.* का use करके

यदि आप package.* use करते हैं तो इस package की सभी classes और interfaces accessible होंगी लेकिन subpackages नहीं।

import keyword का use किसी दूसरे package की classes और interface को current package accessible बनाने के लिए किया जाता है।

packagename.* import करने वाले package का Example

```
//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
```

2) packagename.classname का use करके

यदि आप import package.classname करते हैं तो केवल इस package की declared class ही accessible होगी।

package.classname import करने वाले package का Example

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

```
//save by B.java
package mypack;
import pack.A;

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

Output:

```
Hello
```

3) Fully qualified name का use करके

यदि आप fully qualified name use करते हैं तो केवल इस package की declared class ही accessible होगी। अब import करने की कोई need नहीं है। लेकिन हर बार जब आप class या interface access कर रहे हों तो आपको fully qualified name use करने की need होगी।

इसका use generally तब किया जाता है जब दो packages का same class name हो, उदाहरण के लिए java.util और java.sql packages में Date class होती है।

Fully qualified name import करने वाले package का Example

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

```
    }
}

//save by B.java
package mypack;

class B{
    public static void main(String args[]){
        pack.A obj = new pack.A();//using fully qualified name
        obj.msg();
    }
}
```

Output:

```
Hello
```

Java में Subpackage

Package के inside package को subpackage कहा जाता है। इसे package को और categorize करने के लिए create किया जाना चाहिए।

Let's take an example, Sun Microsystem ने java नाम का एक package define किया है जिसमें many classes हैं जैसे System, String, Reader, Writer, Socket आदि। ये classes एक particular group represent करती हैं, उदाहरण के लिए Reader और Writer classes Input/Output operation के लिए हैं, Socket और ServerSocket classes networking के लिए हैं, आदि। इसलिए, Sun ने java package को subpackages में subcategorized किया है जैसे lang, net, io आदि, और Input/Output related classes को io package में, Server और ServerSocket classes को net packages में डाल दिया है, आदि।

Subpackage का Example

```
package com.javatpoint.core;
class Simple{
   public static void main(String args[]) {
      System.out.println("Hello subpackage");
   }
}
```

Output:

```
Hello subpackage
```

To Compile: javac -d . Simple.java

To Run: java com.javatpoint.core.Simple

Exception Handling in Java

Java में Exception Handling runtime errors को handle करने का एक powerful mechanism है ताकि application का normal flow maintain किया जा सके।

Java में, exception एक event है जो program के normal flow को disrupt करता है। यह एक object होता है जिसे runtime पर throw किया जाता है।

Exception Handling क्या है?

Exception Handling runtime errors जैसे ClassNotFoundException, IOException, SQLException, RemoteException, आदि को handle करने का एक mechanism है।

Exception Handling का Advantage

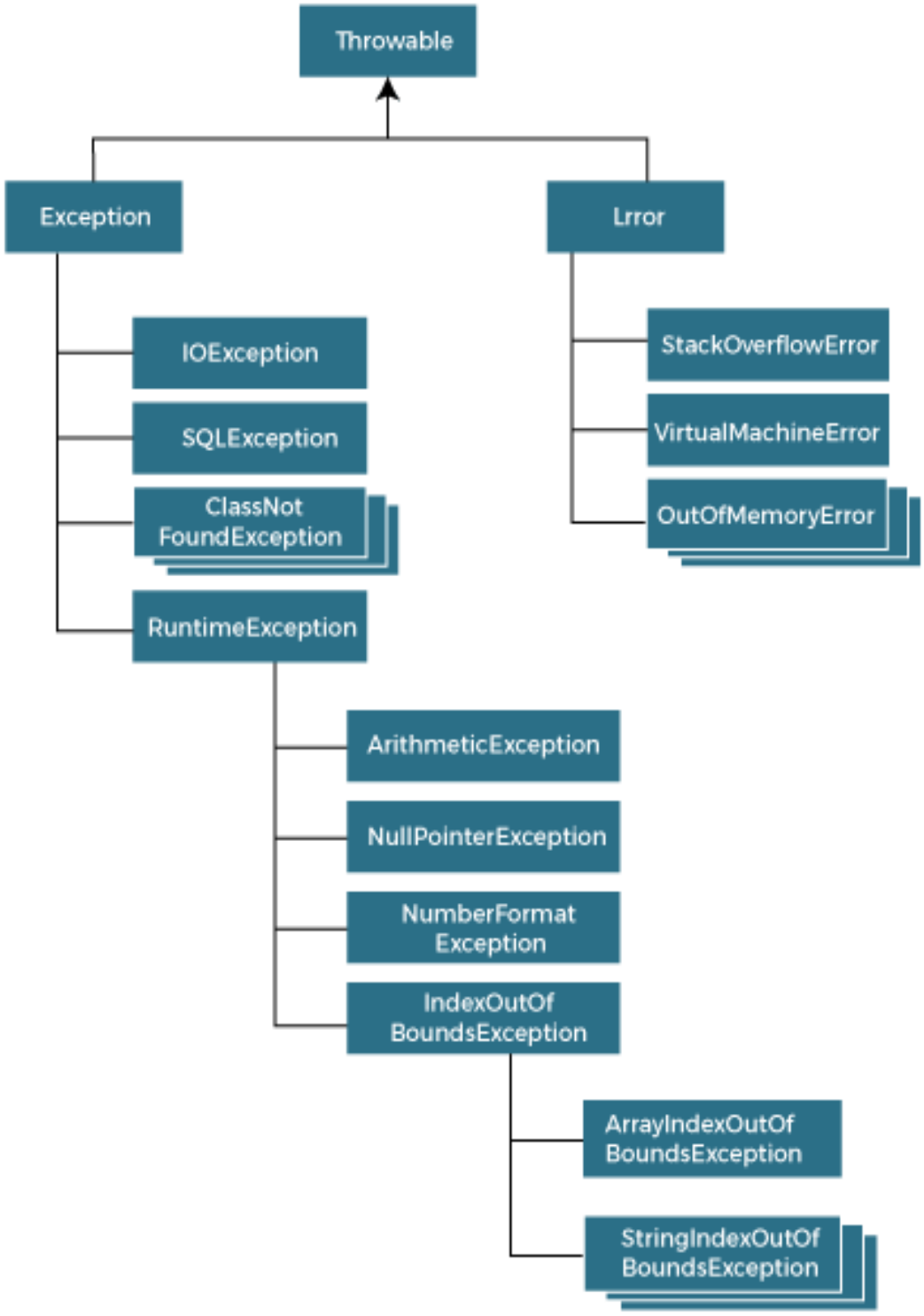
Exception handling का core advantage application के normal flow को maintain करना है। एक exception normally application के normal flow को disrupt कर देता है; इसीलिए हमें exceptions को handle करने की need होती है। Let's consider a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5; //exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose एक Java program में 10 statements हैं और statement 5 पर एक exception occur होता है; rest of the code execute नहीं होगा, यानी statements 6 से 10 execute नहीं होंगे। However, जब हम exception handling perform करते हैं, तो rest of the statements execute हो जाएंगे। That is why we use exception handling in Java.

Java Exception classes की Hierarchy

java.lang.Throwable class Java Exception hierarchy की root class है जिसे two subclasses द्वारा inherit किया जाता है: Exception और Error। Java Exception classes की hierarchy नीचे दी गई है:



Java Exceptions के Types:

मुख्य रूप से दो types के exceptions होते हैं: checked और unchecked। एक error को unchecked exception माना जाता है। However, Oracle के according, तीन types के exceptions हैं namely:

1. Checked Exception
2. Unchecked Exception
3. Error

Checked और Unchecked Exceptions के बीच Difference:

1) Checked Exception

वे classes जो सीधे Throwable class को inherit करती हैं, RuntimeException और Error को छोड़कर, checked exceptions कहलाती हैं। उदाहरण के लिए, IOException, SQLException, आदि। Checked exceptions compile-time पर checked होते हैं।

2) Unchecked Exception

वे classes जो RuntimeException को inherit करती हैं, unchecked exceptions कहलाती हैं। उदाहरण के लिए, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, आदि। Unchecked exceptions compile-time पर checked नहीं होते हैं, लेकिन वे runtime पर checked होते हैं।

3) Error

Error irrecoverable होता है। Errors के कुछ examples हैं OutOfMemoryError, VirtualMachineError, AssertionError आदि।

Java Exception Keywords

Java पांच keywords provide करता है जिनका use exception handle करने के लिए किया जाता है। निम्न तालिका प्रत्येक का description करती है।

Keyword	Description
try	"try" keyword का use एक block specify करने के लिए किया जाता है जहां हमें exception code place करना चाहिए। इसका मतलब है कि हम try block का use अकेले नहीं कर सकते। Try block के बाद या तो catch या finally होना चाहिए।
catch	"catch" block का use exception handle करने के लिए किया जाता है। इससे पहले try block होना चाहिए जिसका मतलब है कि हम catch block का use अकेले नहीं कर सकते। इसके बाद finally block हो सकता है।
finally	"finally" block का use program के necessary code को execute करने के लिए किया जाता है। यह execute होता है चाहे exception handle हुआ हो या नहीं।
throw	"throw" keyword का use exception throw करने के लिए किया जाता है।
throws	"throws" keyword का use exceptions declare करने के लिए किया जाता है। यह specify करता है कि method में एक exception occur हो सकता है। यह exception throw नहीं करता। यह हमेशा method signature के साथ use किया जाता है।

Java Exception Handling Example

Let's see an example of Java Exception Handling in which we are using a try-catch statement to handle the exception.

```
public class JavaExceptionExample{
   public static void main(String args[]) {
      try{
         //code that may raise exception
         int data=100/0;
      }
      catch(ArithmeticException e){
         System.out.println(e);
      }
```

```
        //rest code of the program
        System.out.println("rest of the code...");
    }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Java Exceptions के Common Scenarios

यहां कुछ scenarios दिए गए हैं जहां unchecked exceptions occur हो सकती हैं। They are as follows:

1) वह scenario जहां ArithmeticException occur होती है:

यदि हम किसी number को zero से divide करते हैं, तो ArithmeticException occur होती है।

```
int a=50/0; //ArithmeticException
```

2) वह scenario जहां NullPointerException occur होती है:

यदि हमारे पास किसी variable में null value है, तो variable पर कोई operation perform करने से NullPointerException throw होती है।

```
String s=null; System.out.println(s.length()); //NullPointerException
```

3) वह scenario जहां NumberFormatException occur होती है:

यदि किसी variable या number का formatting mismatched है, तो इसके परिणामस्वरूप NumberFormatException हो सकती है। Suppose हमारे पास एक string variable है जिसमें characters हैं; इस variable को digit में convert करने से NumberFormatException cause होगी।

```
String s="abc"; int i=Integer.parseInt(s); //NumberFormatException
```

4) वह scenario जहां ArrayIndexOutOfBoundsException occur होती है

जब एक array अपने size से exceed कर जाता है, तो ArrayIndexOutOfBoundsException occur होती है। ArrayIndexOutOfBoundsException occur होने के other reasons हो सकते हैं। निम्न statements पर consider करें।

```
int a[]=new int[5]; a[10]=50; //ArrayIndexOutOfBoundsException
```

Java try-catch block

Java try block

Java try block का use उस code को enclose करने के लिए किया जाता है जो exception throw कर सकता है। इसका use method के within होना चाहिए।

यदि try block के particular statement पर exception occur होता है, तो block का rest of the code execute नहीं होगा। So, it is recommended not to keep the code in try block that will not throw an exception.

Java try block के बाद या तो catch या finally block होना चाहिए।

Java try-catch का Syntax

```
try{
   //code that may throw an exception
}
catch(Exception_class_Name ref) {
   // Exception Handling Code
}
```

try-finally block का Syntax

```
try{
   //code that may throw an exception
}
finally{
   // Code that must be executed
}
```

Java catch block

Java catch block का use parameter के within exception के type को declare करके Exception handle करने के लिए किया जाता है। Declared exception parent class exception (यानी Exception) या generated exception type होना चाहिए। However, good approach यह है कि generated type of exception declare किया जाए।

Catch block का use केवल try block के बाद ही किया जाना चाहिए। आप single try block के साथ multiple catch block use कर सकते हैं।

Exception handling के बिना Problem

Let's try to understand the problem if we don't use a try-catch block.

```
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
```

जैसा कि ऊपर के example में display किया गया है, rest of the code execute नहीं हुआ (ऐसे case में, rest of the code statement print नहीं हुआ)।

Exception के बाद 100 lines का code हो सकता है। यदि exception handle नहीं किया जाता है, तो exception के नीचे का सारा code execute नहीं होगा।

Exception handling द्वारा Solution

Let's see the solution of the above problem by a java try-catch block.

```
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
```

Java Multi-catch block

एक try block के बाद एक या अधिक catch blocks हो सकते हैं। प्रत्येक catch block में एक different exception handler contain होना चाहिए। So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

Points to remember

- At a time only one exception occurs and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general, i.e. catch for ArithmeticException must come before catch for Exception.

Example:

```
public class MultipleCatchBlock1 {
    public static void main(String[] args) {
        try{
            int a[]=new int[5];
            a[5]=30/0;
        }
        catch(ArithmeticException e){
            System.out.println("Arithmetic Exception occurs");
        }
        catch(ArrayIndexOutOfBoundsException e){
            System.out.println("ArrayIndexOutOfBounds Exception occurs");
        }
        catch(Exception e){
            System.out.println("Parent Exception occurs");
        }
        System.out.println("rest of the code");
```

```
    }
}
```

Output:

```
Arithmetic Exception occurs
rest of the code
```

Java Nested try block

Java में, एक try block के inside another try block का use permitted है। इसे nested try block कहा जाता है। Every statement that we enter a statement in try block, context of that exception is pushed onto the stack.

For example, inner try block का use ArrayIndexOutOfBoundsException handle करने के लिए किया जा सकता है जबकि outer try block ArithmeticException (division by zero) handle कर सकता है।

Nested try block क्यों use करें

Sometimes एक situation arise हो सकती है जहां एक block का एक part एक error cause कर सकता है और entire block itself another error cause कर सकता है। ऐसे cases में, exception handlers को nested होना पड़ता है।

Java Nested try Example

चलिए निम्नलिखित example को consider करते हैं। यहाँ nested try block के अंदर वाला try block (inner try block 2) exception को handle नहीं करता है। फिर control उसके parent try block (inner try block 1) में transfer हो जाता है। यदि वह exception को handle नहीं करता, तो control main try block (outer try block) में transfer हो जाता है जहाँ appropriate catch block exception को handle करता है। इसे nesting कहा जाता है।

```
public class NestedTryBlock2 {
    public static void main(String args[])
    {
```

```
// outer (main) try block
try {
    //inner try block 1
    try {
        // inner try block 2
        try {
            int arr[] = { 1, 2, 3, 4 };

            //printing the array element out of its bounds
            System.out.println(arr[10]);
        }

        // to handles ArithmeticException
        catch (ArithmeticException e) {
            System.out.println("Arithmetic exception");
            System.out.println(" inner try block 2");
        }
    }
    // to handle ArithmeticException
    catch (ArithmeticException e) {
        System.out.println("Arithmetic exception");
        System.out.println("inner try block 1");
    }
}
// to handle ArrayIndexOutOfBoundsException
catch (ArrayIndexOutOfBoundsException e4) {
    System.out.print(e4);
    System.out.println(" outer (main) try block");
```

```
        }
        catch (Exception e5) {
            System.out.print("Exception");
            System.out.println(" handled in main try-block");
        }
    }
}
```

Output:

```
java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 4 Outer (main) try block
```

Java finally block

Java finally block एक block है जिसका use important code जैसे connection को close करना, आदि execute करने के लिए किया जाता है।

Java finally block हमेशा execute होता है चाहे exception handle हुआ हो या नहीं। Therefore, it contains all the necessary statements that need to be printed regardless of the exception occurs or not.

Java finally block क्यों use करें?

- Java में finally block का use "cleanup" code जैसे file close करना, connection close करना, आदि put करने के लिए किया जा सकता है।
- Printed किए जाने वाले important statements को finally block में place किया जा सकता है।

Example:

```
public class TestFinallyBlock2{
    public static void main(String args[]){
        try {
```

```
            System.out.println("Inside try block");
            int data=25/0;
            System.out.println(data);
        }

        //handles the Arithmetic Exception / Divide by zero exception
        catch(ArithmeticException e){
            System.out.println("Exception handled");
            System.out.println(e);
        }

        //executes regardless of exception occured or not
        finally {
            System.out.println("finally block is always executed");
        }
        System.out.println("rest of the code...");
    }
}
```

Output:

```
Inside try block
Exception handled
Java.lang.ArithmeticException: / by zero
finally block is always executed
Rest of the code...
```

Java throw keyword

Java throw keyword का use exception को explicitly throw करने के लिए किया जाता है।

हम उस exception object को specify करते हैं जिसे throw किया जाना है। Exception के साथ कुछ message होता है जो error का description provide करता है। ये exceptions user inputs, server, आदि से related हो सकती हैं।

हम Java में throw keyword का use करके checked या unchecked exceptions दोनों में से किसी को भी throw कर सकते हैं। इसका use मुख्य रूप से custom exception throw करने के लिए किया जाता है।

Syntax:

```
throw new exception_class("error message");
```

Example:

```
public class TestThrow1 {
   //function to check if person is eligible to vote or not
   public static void validate(int age) {
      if(age<18) {
         //throw Arithmetic exception if not eligible to vote
         throw new ArithmeticException("Person is not eligible to vote");
      }
      else {
         System.out.println("Person is eligible to vote!!");
      }
   }

   //main method
   public static void main(String args[]){
```

```
        //calling the function
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

Output:

Exception in thread “main” java.lang.ArithmeticException: Person is not eligible to vote

Java throws keyword

Java throws keyword का use exception declare करने के लिए किया जाता है। यह programmer को एक information देता है कि वहां एक exception occur हो सकता है। So, it is better for the programmer to provide the exception handling code so that the normal flow of the program can be maintained.

Java throws का Syntax

return_type method_name() throws exception_class_name{ //method code }

Java throws Example:

```
import java.io.IOException;
class Testthrows1{
    void m() throws IOException{
        throw new IOException("device error");//checked exception
    }
    void n() throws IOException{
        m();
    }
```

```
    void p(){
        try{
            n();
        }
        catch(Exception e){
            System.out.println("exception handled");
        }
    }

    public static void main(String args[]){
        Testthrows1 obj=new Testthrows1();
        obj.p();
        System.out.println("normal flow...");
    }
}
```

Output:

```
exception handled
normal flow...
```

Java में throw और throws के बीच Difference:

throw और throws exception handling की concept है जहां throw keyword किसी method या code block से exception explicitly throw करता है, जबकि throws keyword method के signature में use किया जाता है।

throw और throws keywords के बीच many differences हैं। throw और throws के बीच differences की एक list नीचे दी गई है:

Sr. no.	Basis of Differences	throw	throws
1.	Definition	Java throw keyword का use code में, function के inside या code block में exception explicitly throw करने के लिए किया जाता है।	Java throws keyword का use method signature में एक exception declare करने के लिए किया जाता है जो code के execution के दौरान function द्वारा throw किया जा सकता है।
2.	Type of exception	throw keyword का use करके, हम केवल unchecked exception propagate कर सकते हैं, यानी checked exception को केवल throw का use करके propagate नहीं किया जा सकता।	throws keyword का use करके, हम both checked और unchecked exceptions declare कर सकते हैं। However, throws keyword का use केवल checked exceptions propagate करने के लिए किया जा सकता है।
3.	Syntax	throw keyword के बाद Exception का एक instance आता है जिसे throw किया जाना है।	throws keyword के बाद Exceptions के class names आते हैं जिन्हें throw किया जाना है।
4.	Declaration	throw का use method के within किया जाता है।	throws का use method signature के साथ किया जाता है।
5.	Internal implementation	हमें एक समय में केवल एक exception throw करने की allowed है, यानी हम multiple exceptions throw नहीं कर सकते।	हम throws keyword का use करके multiple exceptions declare कर सकते हैं जिन्हें method द्वारा throw किया जा सकता है। For example, main() throws IOException, SQLException।

Java Custom Exception

Java में, हम अपने own exceptions create कर सकते हैं जो Exception class की derived classes हैं। अपना own Exception create करना custom exception या user-defined exception कहलाता है। Basically, Java custom exceptions का use user need के according exception को customize करने के लिए किया जाता है।

Custom exception का use करके, हमारे पास अपना own exception और message हो सकता है। Here, we have passed a string to the constructor of superclass i.e. Exception class that can be obtained using getMessage() method on the object we have created.

Custom exceptions क्यों use करें?

Java exceptions लगभग सभी general type के exceptions को cover करते हैं जो programming में occur हो सकते हैं। However, हमें sometimes custom exceptions create करने की need होती है।

Custom exceptions use करने के few reasons निम्नलिखित हैं:

- Existing Java exceptions के एक subset को catch और specific treatment provide करने के लिए।
- Business logic exceptions: ये exceptions business logic और workflow से related होते हैं। Application users या developers के लिए exact problem को समझना useful होता है।

Custom exception create करने के लिए, हमें Exception class को extend करने की need होती है जो java.lang package से belongs होती है।

Syntax:

```
public class WrongFileNameException extends Exception {
    public WrongFileNameException(String errorMessage) {
        super(errorMessage);
    }
}
```

Example:

चलिए Java custom exception का एक simple example देखते हैं। नीचे दिए गए code में, InvalidAgeException का constructor एक string को argument के रूप में लेता है। यह string super() method का use करके parent class Exception के constructor को pass की जाती है।

इसके अलावा, Exception class के constructor को बिना parameter के भी call किया जा सकता है और super() method को call करना mandatory नहीं है।

```
// class representing custom exception
class InvalidAgeException extends Exception{
    public InvalidAgeException (String str){
        // calling the constructor of parent Exception
        super(str);
    }
}

// class that uses custom exception InvalidAgeException
public class TestCustomException1
{
    // method to check the age
    static void validate (int age) throws InvalidAgeException{
        if(age < 18){
            // throw an object of user defined exception
            throw new InvalidAgeException("age is not valid to vote");
        }
        else {
            System.out.println("welcome to vote");
        }
    }
```

```
    // main method
    public static void main(String args[]){
        try{
            // calling the method
            validate(13);
        }
        catch (InvalidAgeException ex){
            System.out.println("Caught the exception");
            // printing the message from InvalidAgeException object
            System.out.println("Exception occured: " + ex);
        }
        System.out.println("rest of the code...");
    }
}
```

Output:

```
Caught the exception
Exception occurred: InvalidAgeException: age is not valid to vote
rest of the code...
```

Unit-4
Multi-Threading, String & Swing

Java String

Generally, String characters का एक sequence होता है। But in Java, string एक object होता है जो characters के sequence को represent करता है। java.lang.String class का use string object create करने के लिए किया जाता है।

String object कैसे create करें?

String object create करने के दो तरीके हैं:

1. By string literal
2. By new keyword

1) String Literal

Java String literal double quotes का use करके create किया जाता है।

For Example:

```
String s="welcome";
```

2) By new keyword

```
String s=new String("Welcome"); //यह दो objects और एक reference variable
                                  create करता है
```

Java String Example

```
public class StringExample{
  public static void main(String args[]){
    String s1="java"; //creating string by Java string literal
    char ch[]={'s','t','r','i','n','g','s'};
    String s2=new String(ch); //converting char array to string
    String s3=new String("example"); //creating Java string by new
                                       keyword
```

```
        System.out.println(s1);
        System.out.println(s2);
        System.out.println(s3);
    }
}
```

Output:

```
java
strings
example
```

Java String class operations perform करने के लिए बहुत सारे methods provide करती है जैसे compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() आदि।

Length method - length() method specified string की length return करता है। एक empty string की length 0 होती है।

Syntax:

```
public int length()
```

Parameters:

None

Return Value:

String की length

Example:

```
public class StringExample1{
    public static void main(String args[]) {
```

```
        String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        System.out.println(txt.length());
    }
}
```

Output:

```
26
```

Concat method - यह method एक String को दूसरी String के end में append करता है। Method एक String return करता है जिसमें method में pass की गई String का value, उस String के end में append होता है जिसका use इस method को invoke करने के लिए किया गया था।

Syntax:

```
public String concat(String s)
```

Parameters:

s – वह String जो इस String के end में concatenated होती है।

Return Value:

यह methods एक string return करता है जो इस object's characters के concatenation को represent करता है जिसके बाद string argument's characters आते हैं।

Example:

```
public class Test {
    public static void main(String args[]) {
        String s = "Strings are immutable";
        s = s.concat(" all the time");
        System.out.println(s);
    }
}
```

Output:

Strings are immutable all the time

toString method - यदि आप किसी भी object को string के रूप में represent करना चाहते हैं, तो toString() method existence में आती है। toString() method object का String representation return करती है।

यदि आप कोई object print करते हैं, तो Java compiler internally object पर toString() method को invoke करता है। So overriding the toString() method, desired output return करता है, यह object की state आदि हो सकती है, आपके implementation पर depend करता है।

Object class की toString() method को overriding करके, हम object के values return कर सकते हैं, so we don't need to write much code.

toString() method के बिना problem को समझना:

```
class Student{
    int rollno;
    String name;
    String city;
    Student(int rollno, String name, String city){
        this.rollno=rollno;
        this.name=name;
        this.city=city;
    }

    public static void main(String args[]) {
        Student s1=new Student(101,"Raj","lucknow");
        Student s2=new Student(102,"Vijay","ghaziabad");
        System.out.println(s1); //compiler writes here s1.toString()
        System.out.println(s2); //compiler writes here s2.toString()
```

```
    }
}
```

Output:

```
Student@1fee6fc
Student@1eed786
```

जैसा कि आप ऊपर के example में देख सकते हैं, s1 और s2 को print करने पर objects के hashcode values print होते हैं लेकिन मैं इन objects के values print करना चाहता हूं। Since Java compiler internally toString() method call करता है, इस method को overriding करना specified values return करेगा। चलिए इसे नीचे दिए गए example के साथ समझते हैं:

Java toString() method का Example:

```
class Student{
    int rollno;
    String name;
    String city;
    Student(int rollno, String name, String city){
        this.rollno=rollno;
        this.name=name;
        this.city=city;
    }

    public String toString(){ //overriding the toString() method
        return rollno+" "+name+" "+city;
    }

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

```
        Student s1=new Student(101,"Raj","lucknow");
        Student s2=new Student(102,"Vijay","ghaziabad");
        System.out.println(s1); //compiler writes here s1.toString()
        System.out.println(s2); //compiler writes here s2.toString()
    }
}
```

Output:

```
101 Raj lucknow
102 Vijay Ghaziabad
```

ऊपर दिए गए program में, Java compiler internally toString() method call करता है, इस method को overriding करना Student class के s1 और s2 objects के specified values return करेगा।

charAt method - Java String charAt() method specified index पर character return करता है। Index value 0 और length() के बीच होनी चाहिए।

Syntax :

```
public char charAt(int index)
```

Parameter:

index- Return किए जाने वाले character का Index।

Return Value:

specified position पर character return करता है।

Example:

```
class Gfg {
    public static void main(String args[]){
        String s = "Welcome! to Geeksforgeeks Planet";
        char ch = s.charAt(3);
        System.out.println(ch);
        ch = s.charAt(0);
        System.out.println(ch);
    }
}
```

Output:

```
c
W
```

toCharArray method - java string toCharArray() method इस string को character array में convert करती है। यह एक newly created character array return करती है, इसकी length इस string के similar होती है और इसकी contents इस string के characters से initialized होती हैं।

Syntax:

```
public char[] toCharArray()
```

Parameter:

None

Return Value:

यह एक newly allocated character array return करती है।

Example:

```
class Gfg {
    public static void main(String args[]) {
        String s = "GeeksforGeeks";
        char gfg[] = s.toCharArray();
        for (int i = 0; i < gfg.length; i++) {
            System.out.println(gfg[i]);
        }
    }
}
```

Output:

```
G
e
e
k
s
f
```

compareTo method - Java String class का compareTo() method दी गई string की current string के साथ lexicographically compare करता है। यह एक positive number, negative number, या 0 return करता है।

यह strings में each character के Unicode value के basis पर strings की तुलना करता है।

यदि पहली string lexicographically दूसरी string से greater है, तो यह एक positive number (character value का difference) return करती है। यदि पहली string lexicographically दूसरी string से less है, तो यह एक negative number return करती है, और यदि पहली string lexicographically दूसरी string के equal है, तो यह 0 return करती है।

- if s1 > s2, यह positive number return करती है
- if s1 < s2, यह negative number return करती है
- if s1 == s2, यह 0 return करती है

Syntax:

public int compareTo(String anotherString)

Parameter:

obj: वह Object जिसके साथ compare किया जाना है।

Returns:

Value 0 यदि argument इस string के lexicographically equal है; एक value less than 0 यदि argument इस string से lexicographically greater है; और एक value greater than 0 यदि argument इस string से lexicographically less है।

Example:

```
class Gfg{
   public static void main(String args[]){
      String s1="hello";
      String s2="hello";
      String s3="meklo";
      String s4="hemlo";
      String s5="#ag";
      System.out.println(s1.compareTo(s2)); //0 because both are equal
      System.out.println(s1.compareTo(s3)); //-5 because "h" is 5 times lower than "m"
      System.out.println(s1.compareTo(s4)); //-1 because "l" is 1 times lower than "m"
      System.out.println(s1.compareTo(s5)); //2 because "h" is 2 times greater than "#'
   }
}
```

Output:

0

-5

-1

2

equals method - Java String class का equals() method दो दी गई strings की content के basis पर compare करता है। यदि कोई character match नहीं होता है, तो यह false return करता है। यदि सभी characters match हो जाते हैं, तो यह true return करता है।

Syntax:

public boolean equals(Object anotherObject)

Parameter:

anotherObject : another object, i.e., इस string के साथ compared।

Return Value:

true यदि दोनों strings के characters equal हैं otherwise false।

Example:

```
public class EqualsExample{
   public static void main(String args[]){
      String s1="javatpoint";
      String s2="javatpoint";
      String s3="JAVATPOINT";
      String s4="python";
      System.out.println(s1.equals(s2)); //true because content and case is same
      System.out.println(s1.equals(s3)); //false because case is not same
```

```
        System.out.println(s1.equals(s4)); //false because content is not same
    }
}
```

Output:

```
true
false
false
```

indexOf method - indexOf() method के चार variants हैं। यह article उन सभी के बारे में depict करता है, जो इस प्रकार हैं:

1. int indexOf() : यह method specified character की first occurrence के index को इस string के within return करती है या -1, यदि character occur नहीं होता है।

Syntax:

```
int indexOf(char ch )
```

Parameters:

ch : एक character।

Returns value:

Character का Index position

Example:

```
public class Index1 {
    public static void main(String args[]){
        String gfg = new String("Welcome to geeksforgeeks");
        System.out.print("Found g first at position : ");
```

```
        System.out.println(gfg.indexOf('g'));
    }
}
```

Output:

```
Found g first at position : 11
```

2. int indexOf(char ch, int strt) : यह method specified character की first occurrence के index को इस string के within return करती है, specified index से search start करके या -1, यदि character occur नहीं होता है।

Syntax:

```
int indexOf(char ch, int strt)
```

Parameters:

ch :a character।

strt : वह index जहां से search start करना है।

Return value:

Character का Index position

Example:

```
public class Index2 {
    public static void main(String args[]){
        String gfg = new String("Welcome to geeksforgeeks");
        System.out.print("Found g after 13th index at position : ");
        System.out.println(gfg.indexOf('g', 13));
    }
```

Output:

```
Found g after 13th index at position : 19
```

3. int indexOf(String str) : यह method specified substring की first occurrence के index को इस string के within return करती है। यदि यह substring के रूप में occur नहीं होता है, तो -1 return किया जाता है।

Syntax:

int indexOf(String str)

Parameters:

str : एक string।

Return value:

Substring का Index position

Example:

```
public class Index3 {
    public static void main(String args[]){
        String Str = new String("Welcome to geeksforgeeks");
        String subst = new String("geeks");
        System.out.print("Found geeks starting at position : ");
        System.out.print(Str.indexOf(subst));
    }
}
```

Output:

Found geeks starting at position : 11

4. int indexOf(String str, int strt) : यह method specified substring की first occurrence के index को इस string के within return करती है, specified index से start करके। यदि यह occur नहीं होता है, तो -1 return किया जाता है।

Syntax:

int indexOf(String str, int strt)

Parameters:

strt: वह index जहां से search start करना है।

str : एक string।

Return value:

Substring का Index position

Example:

```
public class Index4 {
   public static void main(String args[]){
      String Str = new String("Welcome to geeksforgeeks");
      String subst = new String("geeks");
      System.out.print("Found geeks(after 14th index) starting at position : ");
      System.out.print(Str.indexOf(subst, 14));
   }
}
```

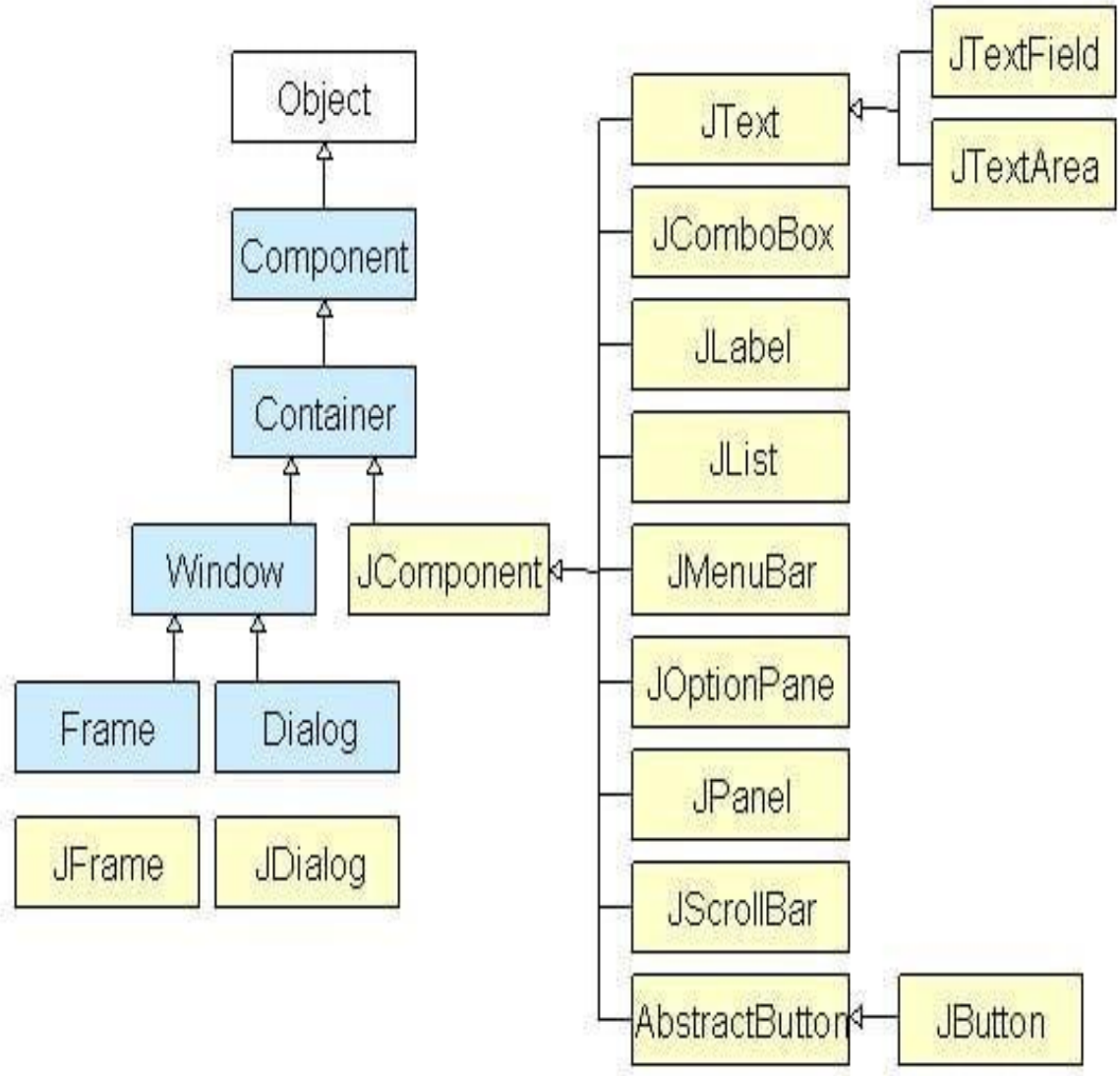
Output:

Found geeks(after 14th index) starting at position : 19

Swing in Java - Java में Swing एक Graphical User Interface (GUI) toolkit है जिसमें GUI components शामिल होते हैं। Swing widgets और packages का एक rich set provide करता है Java applications के लिए sophisticated GUI components बनाने के लिए। Swing Java Foundation Classes(JFC) का एक part है, जो Java GUI programing के लिए एक API है जो GUI provide करता है।

Java Swing library, Java Abstract Widget Toolkit (AWT) के top पर built है, जो एक older, platform dependent GUI toolkit है। आप Java simple GUI programming components जैसे button, textbox, आदि का use library से कर सकते हैं और components को scratch से create नहीं करना पड़ता।

Java Swing class Hierarchy Diagram



Container Class क्या है?

Container classes वे classes हैं जिन पर other components हो सकते हैं। So एक Java Swing GUI create करने के लिए, हमें कम से कम एक container object की need होती है। 3 types के Java Swing containers हैं।

- Panel: यह एक pure container है और अपने आप में एक window नहीं है। एक Panel का sole purpose components को एक window पर organize करना है।
- Frame: यह अपने title और icons के साथ एक fully functioning window है।
- Dialog: इसे एक pop-up window की तरह समझा जा सकता है जो तब pop-out होता है जब एक message display करना होता है। यह Frame की तरह एक fully functioning window नहीं है।

Example :

```
import javax.swing.*;
import java.awt.*;

class gui{
   public static void main(String args[]){
      JFrame frame = new JFrame("My First GUI");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(300,300);
      JButton button1 = new JButton("Press");
      frame.getContentPane().add(button1);
      frame.setVisible(true);
   }
}
```

Output:

Components of Swing Class-

Class	Description
Component	एक Component लगभग SWING के non menu user-interface controls के लिए Abstract base class है। Components एक object को represent करते हैं जिसका graphical representation होता है।
Container	एक Container एक component है जो SWING Components को contain कर सकता है।
JComponent	एक JComponent सभी swing UI Components के लिए base class है, एक swing component का use करने के लिए जो JComponent से inherit करता है, component एक containment hierarchy में होना चाहिए जिसकी root एक top-level Swing container हो।
JLabel	एक JLabel एक container में text place करने के लिए एक object component है।
JButton	यह class एक labeled button create करती है।
JColorChooser	एक JColorChooser controls का एक pane provide करती है designed to allow the user to manipulate and select a color।
JCheckBox	एक JCheckBox एक graphical(GUI) component है जो या तो on-(true) या off-(false) state में हो सकता है।
JRadioButton	JRadioButton class एक graphical(GUI) component है जो या तो on-(true) या off-(false) state में हो सकता है।
JList	एक JList component user को text items की scrolling list के साथ represent करता है।
JComboBox	एक JComboBox component User को choices के एक show up Menu के साथ Present करता है।

JTextField	एक JTextField object एक text component है जो single line of text के editing की allow करेगा।
JPasswordField	एक JPasswordField object एक text component है जो password entry के लिए specialized है।
JTextArea	एक JTextArea object एक text component है जो multiple lines of text के editing की allow करता है।
ImageIcon	एक ImageIcon control, Icon interface का एक implementation है जो Images से Icons paint करता है।
JScrollbar	एक JScrollbar control एक scroll bar component को represent करता है users को range values से Select करने में enable करने के लिए।
JOptionPane	JOptionPane standard dialog boxes का एक set provide करता है जो users को एक value या Something के लिए prompt करता है।
JFileChooser	एक JFileChooser एक dialog window को represent करता है जिससे user एक file select कर सकता है।
JProgressBar	जैसे ही task completion की ओर progresses करता है, progress bar completion पर task का percentage display करता है।
JSlider	एक JSlider यह class user को graphically(GUI) एक value select करने की lets करती है एक bounded interval within एक knob sliding का use करके।
JSpinner	एक JSpinner यह class एक single line input field है जो user को एक number या एक object value को एक ordered sequence से select करने की lets करती है।

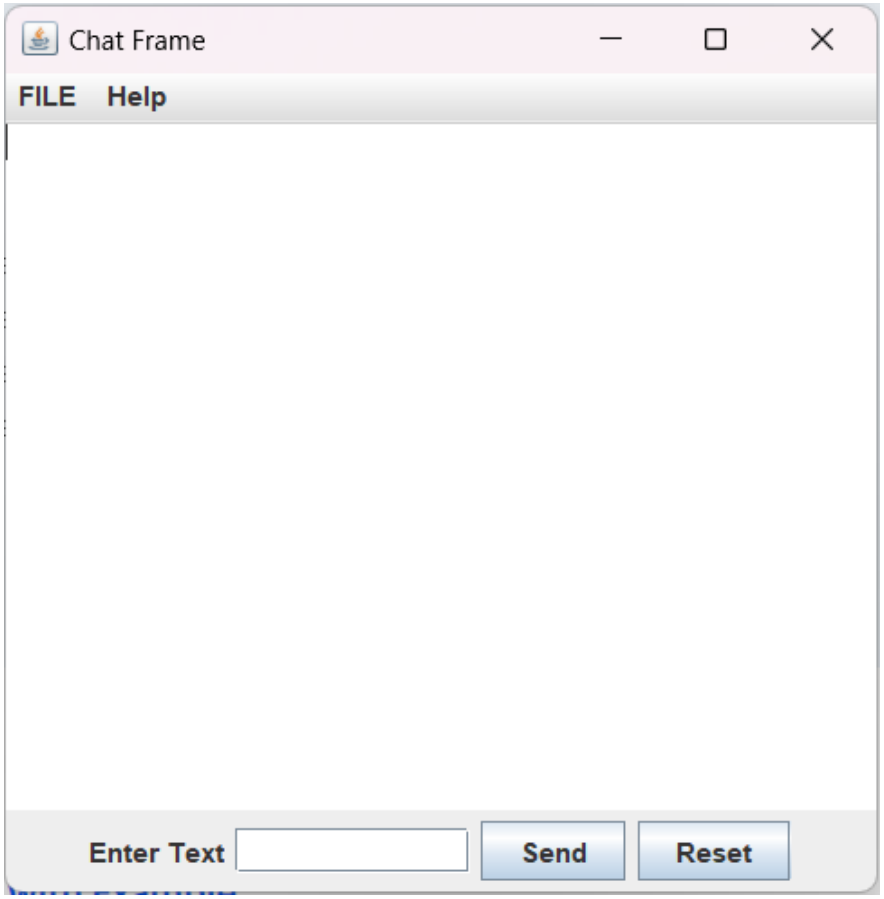
Example:

```
//Usually you will require both swing and awt packages
// even if you are working with just swings.
import javax.swing.*;
import java.awt.*;

class gui {
   public static void main(String args[]) {
      //Creating the Frame
      JFrame frame = new JFrame("Chat Frame");
```

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400, 400);

//Creating the MenuBar and adding components
JMenuBar mb = new JMenuBar();
JMenu m1 = new JMenu("FILE");
JMenu m2 = new JMenu("Help");
mb.add(m1);
mb.add(m2);
JMenuItem m11 = new JMenuItem("Open");
JMenuItem m22 = new JMenuItem("Save as");
m1.add(m11);
m1.add(m22);

//Creating the panel at bottom and adding components
JPanel panel = new JPanel(); // the panel is not visible in output
JLabel label = new JLabel("Enter Text");
JTextField tf = new JTextField(10); // accepts upto 10 characters
JButton send = new JButton("Send");
JButton reset = new JButton("Reset");
panel.add(label); // Components Added using Flow Layout
panel.add(tf);
panel.add(send);
panel.add(reset);

// Text Area at the Center
JTextArea ta = new JTextArea();
```

```
        //Adding Components to the frame.
        frame.getContentPane().add(BorderLayout.SOUTH, panel);
        frame.getContentPane().add(BorderLayout.NORTH, mb);
        frame.getContentPane().add(BorderLayout.CENTER, ta);
        frame.setVisible(true);
    }
}
```

Output:



Multithreading in Java – Java मल्टीथ्रेडिंग एक ऐसा feature है जो multiple threads को concurrently run करने की capability देता है। इससे application better performance देता है क्योंकि multiple tasks एक साथ process होते हैं। हर thread अपना अलग task perform करता है, जिससे CPU resources का efficient use होता है।

Thread एक lightweight process होता है जो within a single program concurrently run हो सकता है। हर thread अपना independent execution flow रखता है, लेकिन वह same memory space share करते हैं।

Without multithreading –

```
class ThreadX{
    public void run(){
        System.out.println("Entering into ThreadX");
        for(int i=1; i<= 5; i++) {
            System.out.println("In ThreadX i ="+ i);
        }
        System.out.println("Exiting from ThreadX");
    }
}

class ThreadY{
    public void run(){
        System.out.println("Entering into ThreadY");
        for(int j=1; j<= 5; j++) {
            System.out.println("In ThreadY j =" + (j*2));
        }
        System.out.println("Exiting from ThreadY");
    }
}
```

```
class ThreadZ{
    public void run(){
        System.out.println("Entering into ThreadZ");
        for(int k=1; k<= 5; k++) {
            System.out.println("In ThreadZ k =" + (-k));
        }
        System.out.println("Exiting from ThreadZ");
    }
}

class WithoutThread{
    public static void main(String args[]) {
        System.out.println("Entering into Main");
        ThreadX X = new ThreadX();
        ThreadY Y = new ThreadY();
        ThreadZ Z = new ThreadZ();
        X.run();
        Y.run();
        Z.run();
        System.out.println("Exiting from Main");
    }
}
```

Multithreading extending Thread class -

```
class ThreadX extends Thread{
    public void run(){
        System.out.println("Entering into ThreadX");
        for(int i=1; i<= 5; i++){
            System.out.println("In ThreadX i ="+ i);
        }
        System.out.println("Exiting from ThreadX");
    }
}

class ThreadY extends Thread{
    public void run(){
        System.out.println("Entering into ThreadY");
        for(int j=1; j<= 5; j++){
            System.out.println("In ThreadY j =" + (j*2));
        }
        System.out.println("Exiting from ThreadY");
    }
}

class ThreadZ extends Thread{
    public void run(){
        System.out.println("Entering into ThreadZ");
        for(int k=1; k<= 5; k++) {
            System.out.println("In ThreadZ k =" + (-k));
        }
```

```
        System.out.println("Exiting from ThreadZ");
    }
}

class UsingThreadClass{
    public static void main(String args[]) {
        System.out.println("Entering into Main");
        ThreadX X = new ThreadX();
        ThreadY Y = new ThreadY();
        ThreadZ Z = new ThreadZ();
        X.start();
        Y.start();
        Z.start();
        System.out.println("Exiting from Main");
    }
}
```

Multithreading implementing runnable class -

```
class ThreadX implements Runnable{
    public void run(){
        System.out.println("Entering into ThreadX");
        for(int i=1; i<= 5; i++){
            System.out.println("In ThreadX i = "+ i);
        }
        System.out.println("Exiting from ThreadX");
    }
}
```

```
class ThreadY implements Runnable{
    public void run(){
        System.out.println("Entering into ThreadY");
        for(int j=1; j<= 5; j++){
            System.out.println("In ThreadY j = " + (j*2));
        }
        System.out.println("Exiting from ThreadY");
    }
}

class ThreadZ implements Runnable{
    public void run(){
        System.out.println("Entering into ThreadZ");
        for(int k=1; k<= 5; k++){
            System.out.println("In ThreadZ k =" + (-k));
        }
        System.out.println("Exiting from ThreadZ");
    }
}

class UsingRunnableInterface{
    public static void main(String args[]){
        System.out.println("Entering into Main");
        ThreadX X = new ThreadX();
        Thread t1 = new Thread(X);
        ThreadY Y = new ThreadY();
        Thread t2 = new Thread(Y);
```

```
        Thread t3 = new Thread(new ThreadZ());
        t1.start();
        t2.start();
        t3.start();
        System.out.println("Exiting from Main");
    }
}
```

Java एक multi-threaded language है जहां multiple threads अपना execution complete करने के लिए parallel चलते हैं। हमें shared resources को synchronize करने की need है यह सुनिश्चित करने के लिए कि एक समय में केवल एक thread ही shared resource तक access करने में सक्षम हो।

यदि एक Object multiple threads द्वारा shared किया जाता है तो Object's state के corrupt होने से बचने के लिए synchronization की need होती है। Synchronization तब needed होती है जब Object mutable हो। यदि shared Object immutable है या जो सभी threads same Object को share करते हैं वे केवल Object's state को read कर रहे हैं modify नहीं कर रहे हैं तो आपको इसे synchronize करने की need नहीं है।

Synchronization के बिना Problem / Data race problem –

```
public class withoutSynchronization extends Thread{
    public static int x;
    public void run(){
        for(int i=0; i<100; i++){
            x = x + 1;
            x = x - 1;
        }
    }
    public static void main(String args[]){
        x = 0;
```

```
        for( int i=0; i<1000; i++){
            new withoutSynchronization().start();
            System.out.println(x);
        }
    }
}
```

Java programming language दो synchronization idioms provide करती है:

- Methods synchronization
- Statement(s) synchronization (Block synchronization)

Method Synchronization –

Synchronized methods thread interference और memory consistency errors को रोकने के लिए एक simple strategy enable करते हैं। यदि एक Object एक से अधिक threads के लिए visible है, तो उस Object's fields के सभी reads या writes synchronized method के through किए जाते हैं।

दो synchronized methods के invocations के लिए interleave करना possible नहीं है। यदि एक thread synchronized method execute कर रहा है, तो अन्य सभी threads जो same Object पर synchronized method invoke करते हैं, उन्हें first thread के Object के साथ done होने तक wait करना होगा।

Example -

```
class Account{
    public int balance;
    public int accountNo;
    void displayBalance(){
        System.out.println("Account No:" + accountNo + " Balance:" + balance);
    }
```

```
    synchronized void deposit(int amount){
        balance = balance + amount;
        System.out.println(amount + " is deposited");
        displayBalance();
    }

    synchronized void withdraw(int amount){
        balance = balance - amount;
        System.out.println(amount + " is withdrawn");
        displayBalance();
    }
}

class TransactionDeposit implements Runnable{
    int amount;
    Account accountX;
    TransactionDeposit(Account x, int amount){
        accountX = x;
        this.amount = amount;
        new Thread(this).start();
    }
    public void run(){
        accountX.deposit(amount);
    }
}

class TransactionWithdraw implements Runnable{
    int amount;
```

```
    Account accountY;
    TransactionWithdraw(Account y, int amount){
        accountY = y;
        this.amount = amount;
        new Thread(this).start();
    }
    public void run(){
        accountY.withdraw(amount);
    }
}

class synchronizationMethod{
    public static void main(String args[]){
        Account ABC = new Account();
        ABC.balance = 1000;
        ABC.accountNo = 111;
        TransactionDeposit t1 = new TransactionDeposit(ABC, 500);
        TransactionWithdraw t2 = new TransactionWithdraw(ABC, 900);
    }
}
```

Block Synchronization-

यदि हमें एक method के within code की सभी lines (instructions) नहीं बल्कि केवल कुछ subsequent lines of code को execute करने की need है, तो हमें केवल code के उस block को synchronize करना चाहिए जिसके within required instructions exist हैं।

For example, मान लीजिए कि एक method है जिसमें 100 lines का code है लेकिन केवल 10 lines (एक के बाद एक) का code है जिसमें code का critical section है यानी ये lines Object's state को modify (change) कर सकती हैं। So हमें Object की state में किसी भी modification से बचने और यह सुनिश्चित करने के लिए कि other threads एक ही method के within rest of the lines को बिना किसी interruption के execute कर सकते हैं, केवल इन 10 lines of code method को synchronize करने की need है।

Syntax:

```
synchronized (object reference)
{
   // Insert code here
}
```

Example -

```
class Table
{
    void printTable(int n){
        synchronized(this){ //synchronized block
            for(int i=1;i<=5;i++){
                System.out.println(n*i);
                try{
                    Thread.sleep(400);
                }
                catch(Exception e){
```

```
                    System.out.println(e);
                }
            }
        }
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(5);
    }
}
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(100);
    }
}

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

```
        Table obj = new Table(); //only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```

Unit-1
Event Handling & JDBC

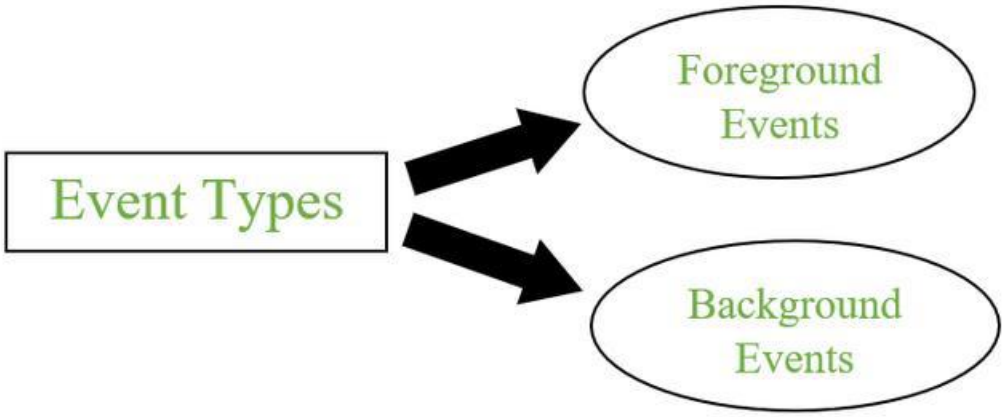
Events क्या होते हैं?

Events objects की state या behavior में change को represent करते हैं। जैसे button click करना, cursor move करना, keyboard से type करना, आदि।

The javax.swing.event package और java.awt.event package various event classes provide करते हैं।

Events के प्रकार:

- Foreground Events - User interaction required होता है (button click, mouse move)
- Background Events - User interaction required नहीं होता (system interrupts, operation complete)



1. Foreground Events

Foreground events वह events होते हैं जिन्हें generate करने के लिए user interaction required होता है, यानी foreground events Graphic User Interface (GUI) में components पर user के interaction के कारण generate होते हैं। Interactions button click करना, scroll bar scroll करना, cursor moments, आदि हैं।

2. Background Events

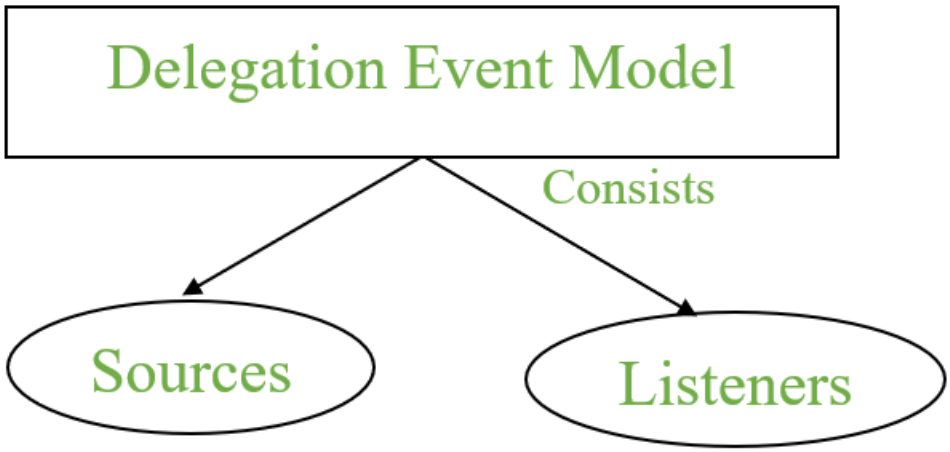
Events जिन्हें generate करने के लिए users के interactions required नहीं होते हैं, उन्हें background events कहा जाता है। इन events के examples हैं operating system failures/interrupts, operation completion, आदि।

Event Handling

यह एक mechanism है जो events को control करने और यह decide करने के लिए कि event occur होने के बाद क्या होना चाहिए के लिए होता है। Events को handle करने के लिए, Java Delegation Event model follow करता है।

Delegation Event model

इसमें Sources और Listeners होते हैं।



Source: Events source से generate होते हैं। Various sources होते हैं जैसे JButtons, JCheckBoxes, JList, JMenuItem, JScrollBar, text components, JWindows, आदि events generate करने के लिए।

Listeners: Listeners source से generate हुए events को handle करने के लिए use किए जाते हैं। इनमें से प्रत्येक listener उस interfaces को represent करता है जो events handle करने के लिए responsible होते हैं।

Event Handling perform करने के लिए, हमें source को listener के साथ register करना होता है।

Registering the Source With Listener

Different Classes different registration methods provide करते हैं।

Syntax:

addTypeListener()

जहाँ Type event के type को represent करता है।

Example 1: KeyEvent के लिए हम register करने के लिए addKeyListener() use करते हैं।

Example 2: ActionEvent के लिए हम register करने के लिए addActionListener() use करते हैं।

Event Classes in Java

Event Class	Listener Interface	Description
ActionEvent	ActionListener	यह event indicate करता है कि कोई component-defined action हुआ है जैसे JButton click करना या JMenuItem list से कोई item select करना
AdjustmentEvent	AdjustmentListener	यह adjustment event JScrollBar जैसे adjustable object से emit होता है
ComponentEvent	ComponentListener	यह event indicate करता है कि कोई component move हुआ, उसका size change हुआ या उसकी visibility change हुई
ContainerEvent	ContainerListener	जब कोई component container में add किया जाता है या remove किया जाता है, तब यह event container object द्वारा generate होता है
FocusEvent	FocusListener	ये focus-related events हैं, जिनमें focus, focusin, focusout, और blur events आते हैं
ItemEvent	ItemListener	यह event indicate करता है कि कोई item selected हुआ या नहीं
KeyEvent	KeyListener	यह event keyboard पर keypresses की sequence के कारण occur होता है

MouseEvent	MouseListener & MouseMotionListener	ये events mouse (Pointing Device) के साथ user interaction के कारण occur होते हैं
MouseWheelEvent	MouseWheelListener	यह event specify करता है कि mouse wheel किसी component में rotate हुई
TextEvent	TextListener	यह event occur होता है जब किसी object का text change होता है
WindowEvent	WindowListener	यह event indicate करता है कि कोई window अपना status change हुई है या नहीं

Different Interfaces के Methods

Listener Interface	Methods
ActionListener	actionPerformed()
AdjustmentListener	adjustmentValueChanged()
ComponentListener	componentResized() componentShown() componentMoved() componentHidden()
ContainerListener	componentAdded() componentRemoved()
FocusListener	focusGained() focusLost()
ItemListener	itemStateChanged()
KeyListener	keyTyped() keyPressed() keyReleased()
MouseListener	mousePressed() mouseClicked() mouseEntered() mouseExited() mouseReleased()
MouseMotionListener	mouseMoved() mouseDragged()
MouseWheelListener	mouseWheelMoved()
TextListener	textChanged()
WindowListener	windowActivated() windowDeactivated() windowOpened() windowClosed() windowClosing() windowIconified() windowDeiconified()

Event Handling का Flow

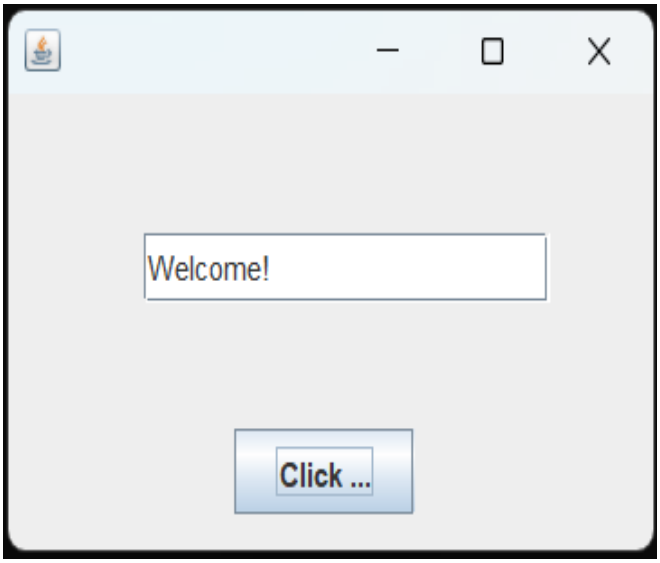
1. Event generate करने के लिए component के साथ User Interaction required होता है।
2. Event generation के बाद respective event class का object automatically create होता है, और यह event source की सभी information hold करता है।
3. Newly created object registered listener के methods में pass होता है।
4. Method execute होता है और result return करता है।

Example

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

class Welcome extends JFrame implements ActionListener {

   JTextField textField;

   Welcome() {
      // Component Creation
      textField = new JTextField();

      // setBounds method component की position और size provide
         करने के लिए use किया जाता है

      textField.setBounds(60, 50, 180, 25);
      JButton button = new JButton("Click Here");
      button.setBounds(100, 120, 80, 30);

      // Registering component with listener
      // this current instance को refer करता है
      button.addActionListener(this);

      // add Components
      add(textField);
      add(button);
```

```
        // set visibility
        setSize(300, 200);
        setLayout(null);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    // implementing method of actionListener
    public void actionPerformed(ActionEvent e) {
        // Setting text to field
        textField.setText("Welcome!");
    }

    public static void main(String[] args) {
        new Welcome();
    }
}
```

Output:



ActionEvent - यह class java.awt.event package में defined है। ActionEvent तब generated होता है जब button clicked किया जाता है या list के किसी item को double clicked किया जाता है।

Example - पिछला example देखें

MouseListener -

The Java MouseListener notify होता है जब भी आप mouse की state change करते हैं। यह MouseEvent के against notify होता है। MouseListener interface java.awt.event package में found होता है। इसके पाँच methods होते हैं।

Methods of MouseListener interface

MouseListener interface में found होने वाले 5 methods की signature नीचे दी गई है:

- public abstract void mouseClicked(MouseEvent e);
- public abstract void mouseEntered(MouseEvent e);
- public abstract void mouseExited(MouseEvent e);
- public abstract void mousePressed(MouseEvent e);
- public abstract void mouseReleased(MouseEvent e);

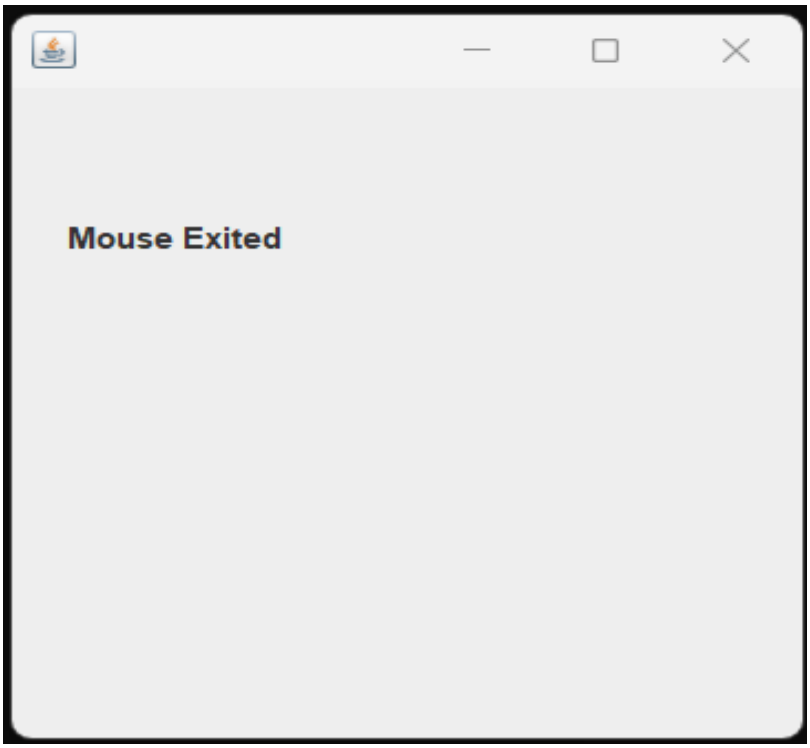
Example –

```
import javax.swing.*;

import java.awt.event.*;


public class MouseListenerExample extends JFrame implements MouseListener
{

   JLabel label;


   MouseListenerExample() {

      addMouseListener(this);
```

```
    label = new JLabel();
    label.setBounds(20, 50, 100, 20);
    add(label);

    setSize(300, 300);
    setLayout(null);
    setVisible(true);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void mouseClicked(MouseEvent e) {
    label.setText("Mouse Clicked");
}

public void mouseEntered(MouseEvent e) {
    label.setText("Mouse Entered");
}

public void mouseExited(MouseEvent e) {
    label.setText("Mouse Exited");
}

public void mousePressed(MouseEvent e) {
    label.setText("Mouse Pressed");
}

public void mouseReleased(MouseEvent e) {
    label.setText("Mouse Released");
```

```
    }

    public static void main(String[] args) {
        new MouseListenerExample();
    }
}
```

Output:



Java Database Connectivity

JDBC का use करके किसी भी java application को database के साथ connect करने के लिए 5 steps होते हैं। ये steps निम्नलिखित हैं:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection

1) Register the driver class

Driver class register करने के लिए Class class का forName() method use किया जाता है। यह method dynamically driver class load करने के लिए use किया जाता है।

Syntax of forName() method

```
public static void forName(String className)throws ClassNotFoundException
```

Example to register the OracleDriver class

यहाँ, Java program database connection establish करने के लिए oracle driver load कर रही है।

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

2) Create the connection object

Database के साथ connection establish करने के लिए DriverManager class का getConnection() method use किया जाता है।

Syntax of getConnection() method

1. public static Connection getConnection(String url)throws SQLException
2. public static Connection getConnection(String url,String name,String password) throws SQLException

Example to establish connection with the Oracle database

```
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"password");
```

3) Create the Statement object

Statement create करने के लिए Connection interface का createStatement() method use किया जाता है। Statement का object database के साथ queries execute करने के लिए responsible होता है।

Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

```
Statement stmt = con.createStatement();
```

4) Execute the query

Database को queries execute करने के लिए Statement interface का executeQuery() method use किया जाता है। यह method ResultSet का object return करता है जिसका use table के सभी records प्राप्त करने के लिए किया जा सकता है।

Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

```
ResultSet rs = stmt.executeQuery("select * from emp");

while(rs.next()) {

   System.out.println(rs.getInt(1) + " " + rs.getString(2));

}
```

5) Close the connection object

Connection object close करने से statement और ResultSet automatically close हो जाएंगे। Connection को close करने के लिए Connection interface का close() method use किया जाता है।

Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
con.close();
```

Example Program –

```
import java.sql.*;

class GFG {
    public static void main(String[] args) throws Exception {
        String url = "jdbc:mysql://localhost:3306/table_name"; // table details
        String username = "rootgfg"; // MySQL credentials
        String password = "gfg123";
        String query = "select * from students"; // query to be run

        Class.forName("com.mysql.cj.jdbc.Driver"); // Driver name
        Connection con = DriverManager.getConnection(url, username, password);

        System.out.println("Connection Established successfully");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery(query); // Execute query
```

```
        rs.next();
        String name = rs.getString("name"); // Retrieve name from db


        System.out.println(name); // Print result on console
        st.close(); // close statement
        con.close(); // close connection
        System.out.println("Connection Closed....");
    }
}
```

JDBC -

JDBC stands for Java Database Connectivity. JDBC एक Java API है database के साथ connect होने और query execute करने के लिए। यह Java Standard Edition का एक part है। JDBC API JDBC drivers का use करता है database के साथ connect होने के लिए। चार types के JDBC drivers होते हैं:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, और
- Thin Driver

1) JDBC-ODBC bridge driver

JDBC-ODBC bridge driver database से connect होने के लिए ODBC driver का use करता है। JDBC-ODBC bridge driver JDBC method calls को ODBC function calls में convert करता है। यह अब discouraged है thin driver के कारण।

Oracle, Java 8 से JDBC-ODBC Bridge support नहीं करता है। Oracle recommends करता है कि आप JDBC-ODBC Bridge के बजाय अपने database के vendor द्वारा provided JDBC drivers का use करें।

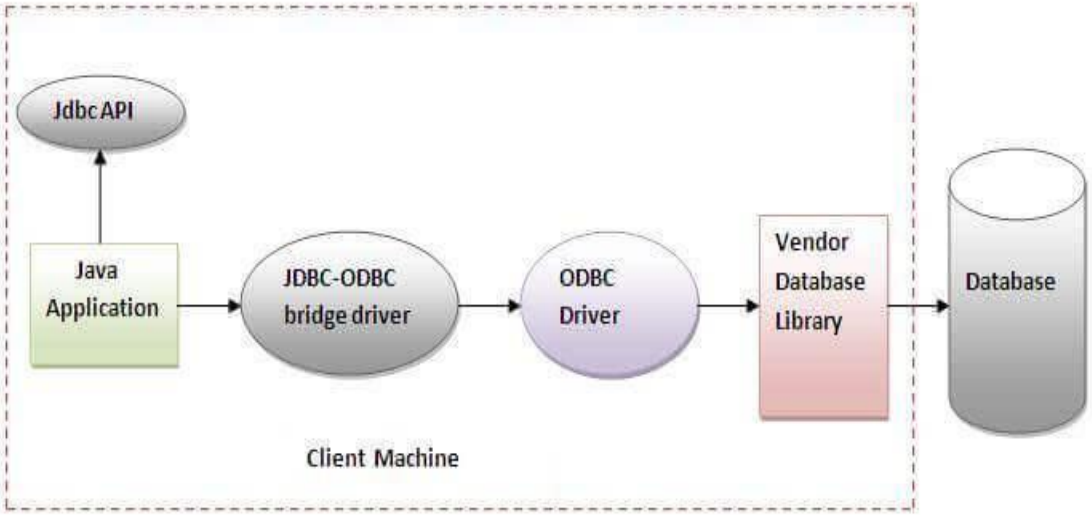


Figure- JDBC-ODBC Bridge Driver

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded क्योंकि JDBC method call ODBC function calls में converted हो जाती है।
- ODBC driver को client machine पर installed किया जाना required होता है।

2) Native-API driver

Native API driver database की client-side libraries का use करता है। Driver JDBC method calls को database API के native calls में convert करता है। यह entirely in java में written नहीं होता है।

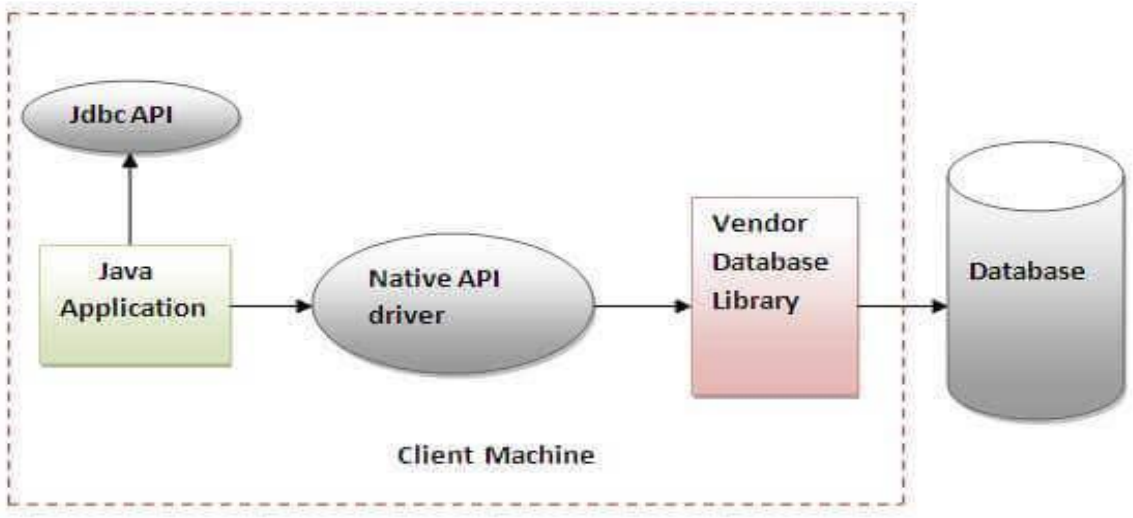


Figure- Native API Driver

Advantage:

- performance JDBC-ODBC bridge driver से upgraded होती है।

Disadvantage:

- Native driver को each client machine पर installed किया जाना required होता है।
- Vendor client library को client machine पर installed किया जाना required होता है।

3) Network Protocol driver

Network Protocol driver middleware (application server) का use करता है जो JDBC calls को directly या indirectly vendor-specific database protocol में convert करता है। यह fully in java में written होता है।

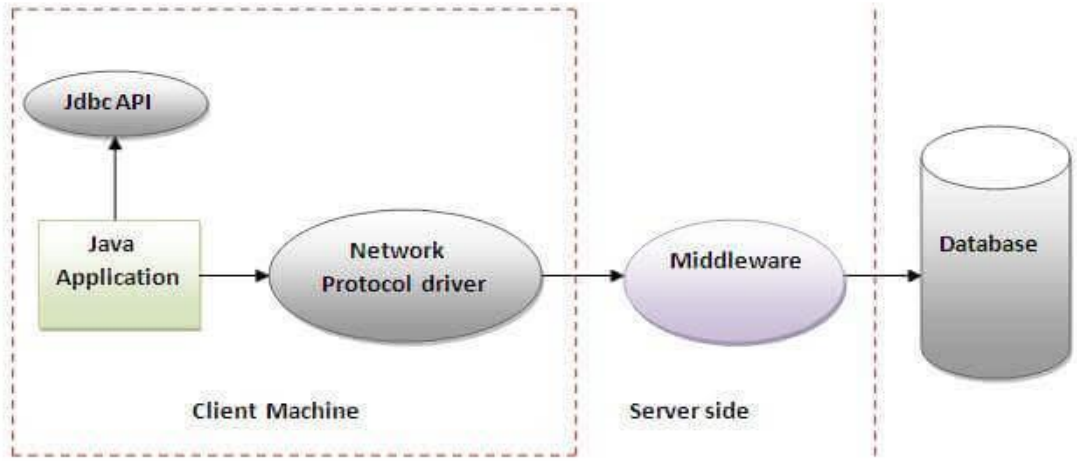


Figure- Network Protocol Driver

Advantage:

- No client side library required होती है क्योंकि application server many tasks perform कर सकता है जैसे auditing, load balancing, logging आदि।

Disadvantages:

- Network support client machine पर required होता है।
- Requires database-specific coding middle tier में done की जानी required होती है।
- Maintenance of Network Protocol driver costly हो जाती है क्योंकि इसमें database-specific coding middle tier में done की जानी required होती है।

4) Thin driver

Thin driver JDBC calls को directly vendor-specific database protocol में convert करता है। इसीलिए इसे thin driver के रूप में जाना जाता है। यह fully Java language में written होता है।

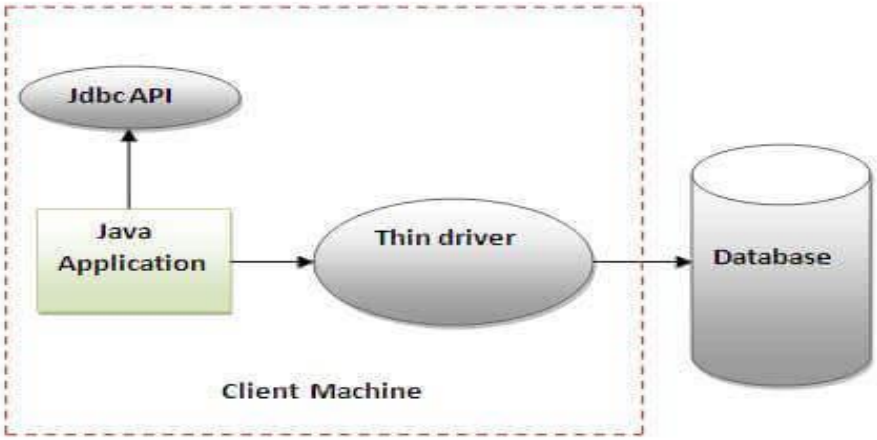


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software required होता है client side या server side।

Disadvantage:

- Drivers Database पर depend करते हैं।