UNIT : 1

Introduction to Java

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by Sun Microsystems in the year 1995. James Gosling is known as the fatherof Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Platform: Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Example:

Simple.java

```
class Simple
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

There are mainly 4 types of applications that can be created using Java programming:

Standalone Application

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

Web Application

An application that runs on the server side and creates a dynamic page is called a web application. Currently, Servlet, JSP, Struts, Spring, Hibernate, JSF, etc. technologies are used for creating web applications in Java.

Enterprise Application

An application that is distributed in nature, such as banking applications, etc. is called an enterprise application. It has advantages like high-level security, load balancing, and clustering. In Java, EJB is used for creating enterprise applications.

Mobile Application

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

Java SE (Java Standard Edition)

It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc. It includes core topics like OOPs, String, Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

Java EE (Java Enterprise Edition)

It is an enterprise platform that is mainly used to develop web and enterprise applications. It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, JPA, etc.

Java ME (Java Micro Edition)

It is a micro platform that is dedicated to mobile applications.

JavaFX

It is used to develop rich internet applications. It uses a lightweight user interface API.

Features of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language.
The features of Java are also known as Java buzzwords.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent
5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun Microsystem, Java language is a simple programming language because:

- Java syntax is based on C++ (so easier for programmers to learn it after C++).
- Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
- There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Object-oriented

Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means the software is organized as a combination of different types of objects that incorporate both data and behavior.
Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism

5. Abstraction
6. Encapsulation

Portable

Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.

Platform Independent

Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language. A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides a software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be executed on multiple platforms, for example, Windows, Linux, Sun Solaris, Mac/OS, etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms, i.e., Write Once and Run Anywhere (WORA).

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- No explicit pointer
- Java Programs run inside a virtual machine sandbox
- Classloader: Classloader in Java is a part of the Java Runtime Environment (JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- Bytecode Verifier: It checks the code fragments for illegal code that can violate access rights to objects.
- Security Manager: It determines what resources a class can access such as reading and writing to the local disk.

Java language provides these securities by default. Some security can also be provided by an application developer explicitly through SSL, JAAS, Cryptography, etc.

Robust

The English meaning of Robust is strong. Java is robust because:

- It uses strong memory management.
- There is a lack of pointers that avoids security problems.
- Java provides automatic garbage collection which runs on the Java Virtual Machine to get

rid of objects which are not being used by a Java application anymore.

- There are exception handling and the type checking mechanism in Java. All these points make Java robust.

Architecture-neutral

Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.

High-performance

Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Distributed

Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.

Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

Dynamic

Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.

Java supports dynamic compilation and automatic memory management (garbage collection).

C++ vs Java

The basic differences between c++ and Java are:

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.
Pointers	C++ supports pointers. You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.

Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Object-oriented	C++ is an object-oriented language. However, in the C language, a single root hierarchy is not possible.	Java is also an object-oriented language. However, everything (except fundamental types) is an object in Java. It is a single root hierarchy as everything gets derived from java.lang.Object.

Note

- Java doesn't support default arguments like C++.
- Java does not support header files like C++. Java uses the import keyword to include different classes and methods.

C++ Program Example

File: main.cpp

```
#include <iostream>
using namespace std;
int main()
{
  cout << "Hello C++ Programming";
  return 0;
}
```

Java Program Example

File: Simple.java

```
class Simple
{
public static void main(String args[])
{
System.out.println("Hello Java");
}
}
```

First Java Program | Hello World Example

```
class Simple
{
   public static void main(String args[]
{
    System.out.println("Hello Java");
 }
}
```

Save the above file as Simple.java

To compile: javac Simple.java

To execute: java Simple

class keyword is used to declare a class in Java.

public keyword is an access modifier that represents visibility. It means it is visible to all.
static is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
void is the return type of the method. It means it doesn't return any value.
main represents the starting point of the program.
String[] args or **String args[]** is used for command line argument
System.out.println() is used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

To write java program,open notepad by start menu -> All Programs -> Accessories -> Notepad and write a simple program.

JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.
JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.

The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVM IS

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Oracle and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

JVM provides definitions for the:

- Memory area
- Class file format
- Register set
- Garbage-collected heap
- Fatal error reporting etc.

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries and other files that JVM uses at runtime.

JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE and development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.

Java Variables

A variable is a name given to the memory location which holds the value while the Java program is executed.
A variable is assigned with a data type.
int data=50;//Here data is variable

There are three types of variables in Java:

- local variable
- instance variable
- static variable

Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

Instance Variable

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.

It is called an instance variable because its value is instance-specific and is not shared among instances.

Static variable

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Data Types

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

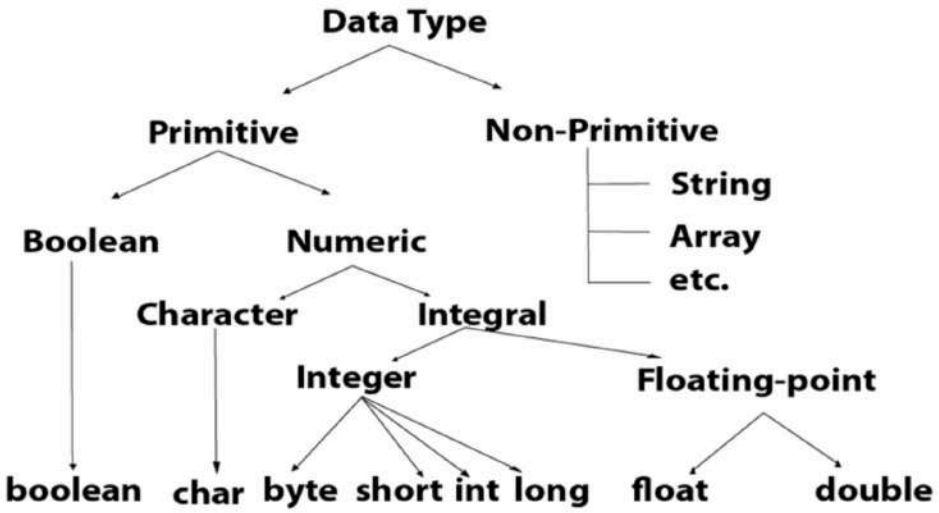
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

Primitive Data Types

Primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.

There are 8 types of primitive data types:

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



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

The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.

The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example: Boolean one = **false**

Byte Data Type

The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.

The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example: byte a = 10, **byte** b = -20

Short Data Type

The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.

Example: short s = 10000, **short** r = -5000

Int Data Type

The int data type is a 32-bit signed two's complement integer. Its value-range lies between -2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is -2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.

The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example: int a = 100000, **int** b = -200000

Long Data Type

The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example: long a = 100000L, **long** b = -200000L

Float Data Type

The float data type is a single-precision 32-bit floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example: float f1 = 234.5f

Double Data Type

The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example: double d1 = 12.3

Char Data Type

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

Example: char letterA = 'A'

Note: Java uses Unicode system not ASCII code system. The \u0000 is the lowest range of Unicode system.

Operators in Java

Operator is a symbol that is used to perform operations.

For example: +, -, *, / etc.

There are many types of operators in Java which are given below:

- Unary Operator,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr++ expr--*
	prefix	*++expr --expr +expr -expr ~* !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=

Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Example1: ++ and --

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

Example2: ~ and !

```
public class OperatorExample
{
```

```
public static void main(String args[])
{
int a=10;

int b=-10;
boolean c=true;
boolean d=false;
System.out.println(~a);//-11 (minus of total positive value which starts from 0)
System.out.println(~b);//9 (positive of total minus, positive starts from 0)
System.out.println(!c);//false (opposite of boolean value)
System.out.println(!d);//true  }}
```

Arithmetic Operators

Arithmetic operators are used to perform addition, subtraction, multiplication, and division. 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
}
}
```

Left Shift Operator

Left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Example:

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

```
System.out.println(10<<2);//10*2^2=10*4=40
System.out.println(10<<3);//10*2^3=10*8=80
System.out.println(20<<2);//20*2^2=20*4=80
System.out.println(15<<4);//15*2^4=15*16=240
}
}
```

Right Shift Operator

Right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Example:

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

Java AND Operator Example: Logical && and Bitwise &

The logical && operator doesn't check the second condition if the first condition is false. It checks the second condition only if the first one is true.

The bitwise & operator always checks both conditions whether first condition is true or false.

Example:

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

```
}
}
```

OR Operator Example: Logical || and Bitwise |

The logical || operator doesn't check the second condition if the first condition is true. It checks the second condition only if the first one is false.

The bitwise | operator always checks both conditions whether first condition is true or false.

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

Ternary Operator

Ternary operator is used as one line replacement for if-then-else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

Example:

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

Assignment Operator

Assignment operator is one of the most common operators. It is used to assign the value on its right to the operand on its left.

Example:

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

```
b-=4;//b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}
}
```

Typecasting Example:

```
class OperatorExample
{
public static void main(String args[])
{
short a=10;
short b=10;
a=(short)(a+b);//20 which is int now converted to short
System.out.println(a);
}
}
```

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

A list of Java keywords or reserved words are given below:

1. **abstract:** Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean:** Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break:** Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. **byte:** Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. **case:** Java case keyword is used with the switch statements to mark blocks of text.
6. **catch:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.

7. **char:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class:** Java class keyword is used to declare a class.
9. **continue:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default:** Java default keyword is used to specify the default block of code in a switch statement.
11. **do:** Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double:** Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. **else:** Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends:** Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final:** Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. **finally:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. **float:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if:** Java if keyword tests the condition. It executes the if block if the condition is true.
21. **implements:** Java implements keyword is used to implement an interface.
22. **import:** Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface:** Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new:** Java new keyword is used to create new objects.

29. **null:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package:** Java package keyword is used to declare a Java package that includes the classes.
31. **private:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. **protected:** Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. **public:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return:** Java return keyword is used to return from a method when its execution is complete.
35. **short:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. **strictfp:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super:** Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. **switch:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this:** Java this keyword can be used to refer the current object in a method or constructor.
42. **throw:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. **throws:** The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. **transient:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void:** Java void keyword is used to specify that a method does not have a return value.
47. **volatile:** Java volatile keyword is used to indicate that a variable may change asynchronously.
48. **while:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

Java Control Statements

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when.

1) If Statement:

The "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax:

```
if(condition)
{
statement 1; //executes when condition is true
}
```

if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

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

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. It is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax :

```
if(condition 1)
{
statement 1; //executes when condition 1 is true
}
else if(condition 2)
{
statement 2; //executes when condition 2 is true
```

```
}
else
{
statement 2; //executes when all the conditions are false
}
```

Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

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 2; //executes when condition 2 is false
}
}
```

Switch Statement:

Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate

- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied. It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

Syntax:

```
switch (expression)
{
  case value1:
 statement1;
 break;
  .
  .
  .
 case valueN:
    statementN;
    break;
 default:
  default statement;
}
```

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.

Looping Statements

Loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly.

1. for loop
2. while loop
3. do-while loop

for loop

It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

Syntax:

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

while loop

The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

Syntax:

```
while(condition)
{
//looping statements
}
```

do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Differences between for, while and do-while

Comparison	for loop	while loop	do-while loop
Introduction	The Java for loop is a control flow statement that iterates a part of the programs multiple times.	The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition.	The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition.
When to use	If the number of iteration is fixed, it is recommended to use for loop.	If the number of iteration is not fixed, it is recommended to use while loop.	If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop.

Syntax	for(init;condition;incr/decr){
	// code to be executed
	}
		while(condition){
		//code to be executed
		}
			do{
			//code to be executed
			}while(condition);

Example	//for loop
	for(int i=1;i<=10;i++){
	System.out.println(i);
	}
		//while loop
		int i=1;
		while(i<=10){
		System.out.println(i);
		i++;
		}
			//do-while loop
			int i=1;
			do{
			System.out.println(i);
			i++;
			}while(i<=10);

Syntax for infinitive loop	for(;;){
	//code to be executed
	}
		while(true){
		//code to be executed
		}
			do{
			//code to be executed
			}while(true);

Naming Convention

Java naming convention is a rule to follow as you decide what to name the identifiers such as class, package, variable, constant, method, etc.

But, it is not forced to follow. So, it is known as convention not rule.

Naming Conventions of the Different Identifiers

The following table shows the popular conventions used for the different identifiers.

Identifiers Type	Naming Rules	Examples

Class	It should start with the uppercaseletter.
	It should be a noun such as Color,Button, System, Thread, etc.
	Use appropriate words, instead ofacronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	It should start with the uppercaseletter.
	It should be an adjective such as Runnable, Remote, ActionListener.Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	It should start with lowercase letter.It should be a verb such as main(), print(), println(). If the name contains multiple words,start it with a lowercase letter followed by an uppercase letter suchas actionPerformed().	class Employee { // meth od void **draw(** **)**

		{
		//code snippet
		}
		}

Variable	It should start with a lowercase lettersuch as id, name.
	It should not start with the specialcharacters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words,start it with the lowercase letter followed by an uppercase letter suchas firstName, lastName.
	Avoid using one-character variablessuch as x, y, z.
		class Employee
		{
		//
		vari
		able
		int
		id;
		//code snippet
		}

Package	It should be a lowercase letter suchas java, lang.
	If the name contains multiple words,it should be separated by dots (.) such as java.util, java.lang.
		//package
		package **com.javatpoint;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters suchas RED, YELLOW.
	If the name contains multiple words,it should be separated by an underscore(_) such as MAX_PRIORITY.
	It may contain digits but not as thefirst letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

Advantage of Naming Conventions in Java

By using standard Java naming conventions, the code becomes easier to read for the programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code doe

CamelCase in Java naming conventions

Java follows camel-case syntax for naming the class, interface, method, and variable.

If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

Objects and Classes

An object in Java is the physical as well as a logical entity, whereas, a class in Java is a logical entity only.

Object

An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
- **Identity:** An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

Example: Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created.

So, an object is the instance(result) of a class.

Object Definitions:

- An object is *a real-world entity*.
- An object is *a runtime entity*.
- The object is *an entity which has state and behavior*.
- The object is *an instance of a class*.

Class

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

- **Fields**
- **Methods**
- **Constructors**
- **Blocks**
- **Nested class and interface**

Declaration of class:

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

Instance variable in Java

A variable which is created inside the class but outside the method is known as an instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when an object or instance is created. That is why it is known as an instance variable.

Method in Java

In Java, a method is like a function which is used to expose the behavior of an object.

Advantage of Method

- Code Reusability
- Code Optimization

new keyword in Java

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Object Initialization

There are 3 ways to initialize object in Java.

1. By reference variable
2. By method
3. By constructor

Creating multiple objects

We can create multiple objects by one type only as we do in case of primitives.

Initialization of primitive variables: **int** a=10, b=20;

Initialization of refernce variables: Rectangle r1=**new** Rectangle(), r2=**new** Rectangle();
(creating two objects of one class type)

Method in Java

In general, a **method** is a way to perform some task. Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code. We can also easily modify code using **methods**.

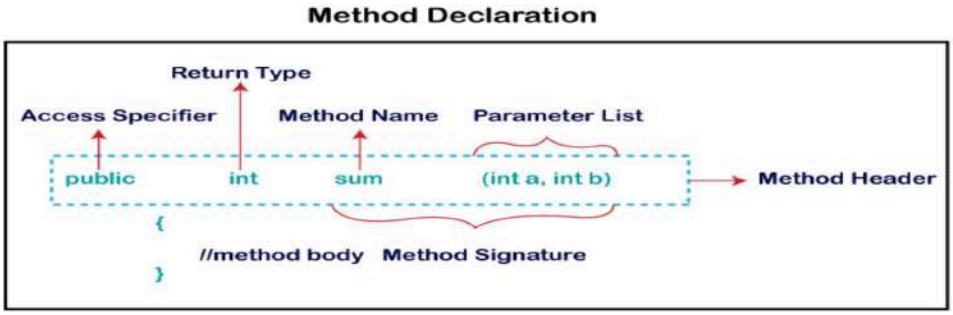
A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation. It is used to achieve the **reusability** of code.

Method can be written once and used many times. We do not require to write code again and again. It also provides the **easy modification** and **readability** of code, just by adding or removing a chunk of code. The method is executed only when we call or invoke it.

The most important method in Java is the **main()** method.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments. It has six components that are known as **method header**, as Shown below.



7.6Mstions on Structures

Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

- **Public:** The method is accessible by all classes when we use public specifier in our application.
- **Private:** When we use a private access specifier, the method is accessible only in the classes in which it is defined.
- **Protected:** When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.
- **Default:** When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc. If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method. Suppose, if we are creating a method for subtraction of two numbers, the method name must be **subtraction().** A method is invoked by its name.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name. If the method has no parameter, left the parentheses blank.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

Naming a Method

While defining a method, remember that the method name must be a **verb** and start with a **lowercase** letter. If the method name has more than two words, the first name must be a verb followed by adjective or noun. In the multi-word method name, the first letter of each word must be in **uppercase** except the first word. For example:

Single-word method name: sum(), area()

Multi-word method name: areaOfCircle(), stringComparision()

It is also possible that a method has the same name as another method name in the same class, it is known as **method overloading**.

Static Method

A method that has static keyword is known as static method. In other words, a method that belongs to a class rather than an instance of a class is known as a static method. We can also create a static method by using the keyword **static** before the method name.

The main advantage of a static method is that we can call it without creating an object. It can access static data members and also change the value of it. It is used to create an instance method. It is invoked by using the class name. The best example of a static method is the **main()** method.

Abstract Method

The method that does not have method body is known as abstract method. In other words, without an implementation is known as abstract method. It always declares in the **abstract class.** It means the class itself must be abstract if it has abstract method. To create an abstract method, we use the keyword **abstract**.

Syntax

```
abstract void method_name();
```

Constructors

Constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Note: It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type not even void.
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Syntax of default constructor:

<class_name>()

{}

Note: If there is no constructor in a class, compiler automatically creates a default constructor.

Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Difference between constructor and method in Java

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name

Java static keyword

The **static keyword** in Java is used for memory management mainly. Static keyword can be applied with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

Java static variable

If any variable declared as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Advantages of static variable

It makes the program **memory efficient** (i.e., it saves memory).

Java static method

If static keyword is applied with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Restrictions for the static method

There are two main restrictions for the static method. They are:

1. The static method can not use non static data member or call non-static method directly.
2. this and super cannot be used in static context.

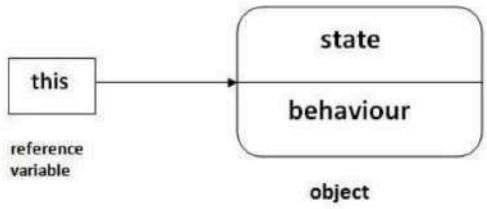
Note: Java main method is static **because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.**

Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



Usage of this keyword

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

Note: If local variables(formal arguments) and instance variables are different, there is no need to use this keyword.

Inheritance in Java

Inheritance is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).

The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.

Advantages

Method Overriding (so runtime polymorphism can be achieved)

Code Reusability.

Terms used in Inheritance

- **Class:** A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class:** Subclass is a class which inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class:** Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability:** As the name specifies, reusability is a mechanism which facilitates you to reuse the fields and methods of the existing class when you create a new class. You can use the same fields and methods already defined in the previous class.

The syntax of Java Inheritance

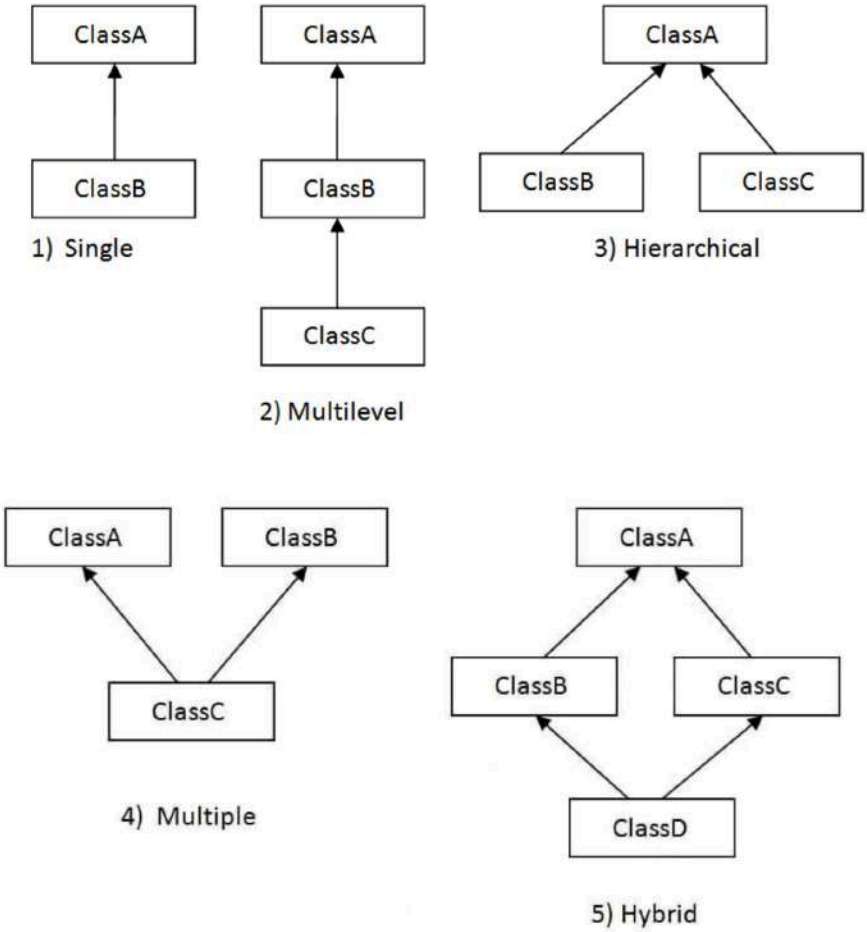
```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

The **extends keyword** indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

1.5M

In the terminology of Java, a class which is inherited is called a parent or superclass, and the new class is called child or subclass.

Types of inheritance in java



.

Single Inheritance

When a class inherits another class, it is known as a *single inheritance*.

Multilevel Inheritance

When there is a chain of inheritance, it is known as *multilevel inheritance*.

Hierarchical Inheritance

When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

Note: To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

Ex: Consider a scenario where A, B, and C are three classes. The C class inherits A and B classes. If A and B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

Since compile-time errors are better than runtime errors, Java renders compile-time error if you inherit 2 classes. So whether you have same method or different, there will be compile time error.

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If only one operation is performed having same name of the methods increases the readability of the program.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

In Java, Method Overloading is not possible by changing the return type of the method only.

java main() method

Java main() method can also be overloaded. Any number of main methods can be created in a class by method overloading. But JVM calls main() method which receives string array as arguments only.

Example:

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

```
System.out.println("main with String[]");
}
public static void main(String args)
{System.out.println("main with String");
}
public static void main()
{System.out.println("main without args");
}
}
```

Output: main with String[].

Method Overriding in Java

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

Note: Static method and main method cannot be overridden.

Super Keyword in Java

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever the instance of subclass is created, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

Super keyword can be used to access the data member or field of parent class. It is used if parent class and child class have same fields.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor.

Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.54.2MC++ vs Java

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

1) Private

The private access modifier is accessible only within the class.

2) Default

If any modifier is not used, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more restrictive than protected, and public.

3) Protected

The **protected access modifier** is accessible within package and outside the package but throughinheritance only.

The protected access modifier can be applied on the data member, method and constructor. Itcan't be applied on the class.

It provides more accessibility than the default modifer.

4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Unit-2

Inheritance and Polymorphism

Inheritance and Polymorphism: Inheritance in java, Super and sub class, Overriding, Object class, Polymorphism, Dynamic binding, Generic programming, Casting objects, Instance of operator,Abstract class, Interface in java, Package in java, UTIL package.

Multithreading in java: Thread life cycle and methods, Runnable interface, Thread synchronization, Exception handling with try catch-finally, Collections in java, Introduction to JavaBeans and Network Programming.

INHERITANCE AND POLYMORPHISM

OBJECT CLASS IN JAVA

Object class is present in java.lang package. The Object class is the parent class of all the classes in java bydefault. In other words, it is the top most class of java. The Object class is beneficial if you want to refer anyobject whose type you don't know. Notice that parent class reference variable can refer thechild class object, known as upcasting.

All the methods of Object class can be used by all the subclasses and arrays. The Object class providesdifferent methods to perform different operations on objects.

Every class in Java is directly or indirectly derived from the Object class. If a class does not extend anyother class then it is a direct child class of Object and if extends another class then it is indirectly derived. Therefore the Object class methods are available to all Java classes. Hence Object class acts as a root of inheritance hierarchy in any Java Program.

Let's take an example, there is getObject() method that returns an object but it can be ofany type like Employee, Student etc, we can use Object class reference to refer that object.

For example:Object obj=getObject();

The Object class provides some common behaviors to all the objects such as object can be compared, objectcan be cloned, object can be notified etc.

Methods of Object class

The Object class provides many methods. They are as follows:

- toString() Method
- hashCode() Method
- equals(Object obj) Method
- getClass() method
- finalize() method
- clone() method
- wait(), notify() notifyAll() Methods

toString() Method:

It's provide string representation or convert object to string form. you can override toString() methodto get your own String representation of objects.

```
public String toString()
{
// Can override and give own definition
}
```

hashCode() Method:

It's generate unique hashcode for each object. The main advantage of saving objects. It's used tooverride for user defined objects for better performance like searching.

equals(Object obj) Method:

It's used to compare the two objects dynamically.

getClass() Method:

It's return runtime class object and used to get metadata information as well.

finalize() method:

This method call required to perform garbage collector.

clone() method:

It used to create the copy or clone of object. **wait(), notify() notifyAll() Methods:** These are used in multithreading.

POLYMORPHISM IN JAVA

The word polymorphism means having many forms. In simple words, we can define polymorphismas the ability of a message to be displayed in more than one form.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
Any Java object that can pass more than one IS-A test is considered to be polymorphic

Types of polymorphism

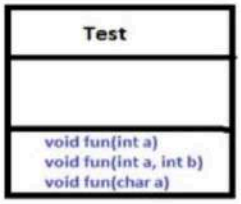
In Java polymorphism is mainly divided into two types:

- Compile-time Polymorphism
- Runtime Polymorphism

Type 1: Compile-time polymorphism

It is also known as static polymorphism. This type of polymorphism is achieved by **functionoverloading (Method Overloading)**.

Method Overloading: When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by change in the number of arguments or/and a change in the type of arguments.



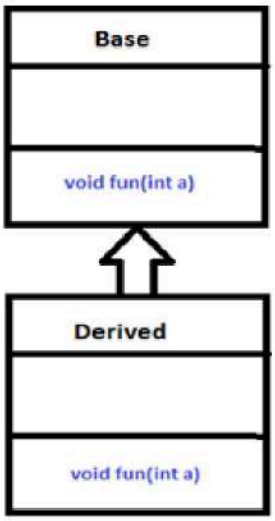
Overloading

Type 2: Runtime polymorphism

Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

In this process, an overridden method is called through the reference variable of a superclass. The determination of the method to be called is based on the object being referred to by the reference variable.

This type of polymorphism is done by method overriding.



Overriding

CASTING OBJECTS

Upcasting and Downcasting in Java

A process of converting one data type to another is knownas Typecasting. Upcasting and Downcasting is the type of object typecasting. In Java, the object can also be typecast like the datatypes. Parent and Child objects are two types of objects. So, there are two types of typecasting possible for an object, i.e., Parent to Child and Child to Parent or can say Upcastingand Downcasting.

Typecasting is used to ensure whether variables are correctly processed by a function or not.In Upcasting and Downcasting, we typecast a child object to a parent object and a parent object to a childobject simultaneously. We can perform Upcasting implicitly or explicitly, but downcasting cannot be implicitly possible.

1) Upcasting

Upcasting is a type of object typecasting in which a **child object** is typecasted to a **parent class object**. By using the Upcasting, we can easily access the variables and methods of the parent class to the child class. Here, we don't access all the variables and the method. We access only some specifiedvariables

and methods of the child class. **Upcasting** is also known as **Generalization** and **Widening**.

UpcastingExample.jav

aclass Parent

```
{
    void PrintData()
{
System.out.println("method of parent class");
}
}

    class Child extends Parent
{
    void PrintData()
{
System.out.println("method of child class");
}
}

    class UpcastingExample
{
    public static void main(String args[])
{
 Parent obj1 = (Parent) new Child();Parent obj2 = (Parent) new Child(); obj1.PrintData();
obj2.PrintData();
}
}
```

2) Downcasting

Upcasting is another type of object typecasting. In Upcasting, we assign a parent class reference objecttothe child class. In Java, we cannot assign a parent class reference object to the child class, but if we perform downcasting, we will not get any compile-time error. However, when we run it, it throwsthe **"ClassCastException"**. Now the point is if downcasting is not possible in Java, then why is it allowedby the compiler? In Java, some scenarios allow us to perform downcasting. Here, the subclassobject is referred by the parent class.

DowncastingExample.java

```
//Parent class
   class Parent { String name;
   void showMessage()
{
System.out.println("Parent method is called");
}
}
     class Child extends Parent
{
     int age;
     void showMessage()
{
System.out.println("Child method is called");
}
}
     public class Downcasting
{
     public static void main(String[] args)
{
     Parent p = new Child();p.name = "Shubham"; Child c = (Child)p; c.age = 18;
    System.out.println(c.name);System.out.println(c.age); c.showMessage();
}
}
```

GENERIC PROGRAMMING

The Java Generic programming is introduced to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time. Before generics, we can store any type of objects in thecollection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

Generic means parameterized types. The idea is to allow type (Integer, String, etc., and user- defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

Why Generics?

The Object is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature. We will discuss that type of safety feature in later examples

Types of Java Generics

Generic Method: Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to performlong, individual type castings.

Generic Classes: A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter,separated by a comma. The classes, which accept one or more parameters, are known as parameterized classes or parameterized types.

Generic Class

Like C++, we use <> to specify parameter types in generic class creation. To create objects of ageneric class, we use the following syntax.

// To create an instance of generic class BaseType

<Type> obj = new BaseType <Type> ()

```
// Java program to show working of user defined
// Generic classes
// We use < > to specify Parameter type
    class Test<T>
{
T obj;  // An object of type T is declaredTest(T
obj)
{
this.obj = obj; }  // constructor

    public T getObject()
{
    return this.obj;
}
}

class Main                  // Driver class to test above
{
    public static void main(String[] args)
{
Test<Integer>       iObj    =    new    Test<Integer>(15);    //    instance    of    Integer    type
System.out.println(iObj.getObject());
```

```
// instance of String type

Test<String> sObj  = new Test<String>("BCA");
System.out.println(sObj.getObject());
}
}
```

Output

```
15
BCA
```

INSTANCE OF OPERATOR

The java instanceof operator is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Simple example of java instanceof

```
class Simple1
{
public static void main(String args[])
{
Simple1 s=new Simple1();
System.out.println(s instanceof Simple1);  //true
}
}
```

Output: true

ABSTRAT CLASSES

If the class acts as base class for many other classes and is not useful on its own, then we canavoid instantiation and only it's declaration is enough. This is achieved by using abstract keyword. Similarly we can have only method declaration and allow derived class to define it by using abstract keyword. Such class is called abstract class and such method is called abstract method. It can have abstract and non-abstract methods (method with the body).

We have seen that by making a method **final** we ensure that the method is not redefined in a sub class. That is, the method can never be sub classed. Java allows us to do something that is exactly opposite to this. That is, we can indicate that a method must always be redefined in a sub class, thus making overriding compulsory. This is done using the modifier keyword **abstract** in the method definition.

```
    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

A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.

When a class contains one or more abstract methods, it should also be declared **Abstract** as shown inthe example above.

While using abstract classes, we must satisfy the following conditions:

- We cannot use abstract classes to instantiate objects directly. Eg: bike b =new bike( ); Is illegal because bike is an abstract class.

- The abstract methods of an abstract class must be defined in its subclass. We cannot declare abstract constructors or abstract static method

INTERFACE IN JAVA

INTERFACES: MULTIPLE INHERITANCE

Java does not support multiple inheritance. That is, classes in Java cannothave more than one superclass. For instance, a definition like:

```
Class A extends B extends C
{
.....
.....
}
```

Is not permitted in Java. A large number of real-life applications require the use ofmultiple inheritancewhere by we inherit methods and properties from several distinct classes.

Java provides an alternate approach known as **interfaces** to support the concept of multiple inheritance.Although a Java class cannot be a subclass of morethan one superclass, it can implement more than one interface, there by enabling us to create classes that build up-on other classes without the problems created bymultiple inheritance.

DEFINING INTERFACES

An interface is basically a kind of **class**. Like classes, interfaces contain methods and variables but with a major difference. The difference is that interfaces define only abstract methods and final fields. This means that interfaces do not specify any code to implement these methods and data fields contain only constants.

The syntax for defining an interface is very similar to that for defining a class.

```
Interface interfacename
{
variable declaration;
methods declaration;
}
```

Here, **interface** is the keyword and **interfacename** is any valid Javavariable(just like class names). Note that all variables are declared as constants.

EXTENDING INTERFACES

Like classes, interfaces can also be extended.That is, an interface can besub interfaced from other interfaces. The new subinterface will inherit all the members of the super interface in the manner similar to subclasses.This is achievedusing the keyword **extends**.

```
Interface name2 extends name1
{
Body of name2
}
```

For example, we can put all the constants in one interface and the methods in the other. This willenable us to use the constants in classes where the methods arenot required. Example

```
Interface ItemConstants
{
Int code=1001 ;
String name="Fan";
}
Interface Item extends ItemConstants
{
Void display();
}
```

IMPLEMENTING INTERFACES

Interfaces are used as "superclasses" whose properties are inherited by classes. Itis therefore necessaryto create a class that inherits the given interface. This is done as follows:

class classname **implements** interfacename

{

body of classname.

}

Here the class classname "implements" the interface interfacename. A more general form of implementation may look like this:

class classname **extends** superclass **implements** interface1, interface2,.....

{

body of classname

}

This shows that a class can extend another class while implementing interfaces.

Program :Implementing interfaces

```
interface Area                    // Interface defined
    {
final static float pi = 3,14F; float compute (float x, float y);
}
    class Rectangle implements Area
```

```
{
return (x*y);
}
public float compute (float x, float y)
}
class Circle implements Area
{
public float compute (float x, float y)
{
return (pi*x*x);
}
}
class InterfaceTest
{
public static void main(String args[])
{
Rectangle rect = new Rectangle();Circle cir = new Circle();Area
area;area =rect;
      System.out.println("Area of Rectangle =+area.compute(2.5,3.5));
      System.out.println ("Area of Circle = "+area.compute(4.5,5.5));
}
}
```

Implementing Multiple Inheritance

Progam:Implementing multiple inheritance

```
class Student
{
int rollNumber;
void getNumber(int n)
{
rollNumber = n;
}
void putNumber ( )
{
System.out.println (" Roll No : " + rollNumber);
}
}
class Test extends Studentfloat part1, part2;void
getMarks (float ml, float m2)
part1 = m1;
part2 = m2;
```

```
}
void putMarks ( )
{
System.out.println("Marks    obtained    ");
System.out.println("Part  1  =  "  +  part  1);
System.out.println("Part 2 = " + part 2);
}
}
      interface Sports
{
float sportWt = 6.OF;void putwt ();
}
      class Results extends Test implements Sports
{
float total;
      public void putWt ( )
{
System.out.println("Sports Wt = " + sportWt);
}
void display ( )
{
total = partl+part2 + sportWt;put
Number();
putMarks();
putWt ();
System.out.println("Total score = " + total);
}
}
class Hybrid
public static void main(String args[])
Results student1 = new Results();
student1.getNumber (1234);
student1.getMarks (27.5F, 33.0F);
student1.display();
```

Output of the Program 10.2:

```
Roll No : 1234
Marks obtained
Part1 = 27.5
Part2 = 33 Sports Wt
= 6
Total score = 66.5
```

JAVA PACKAGES

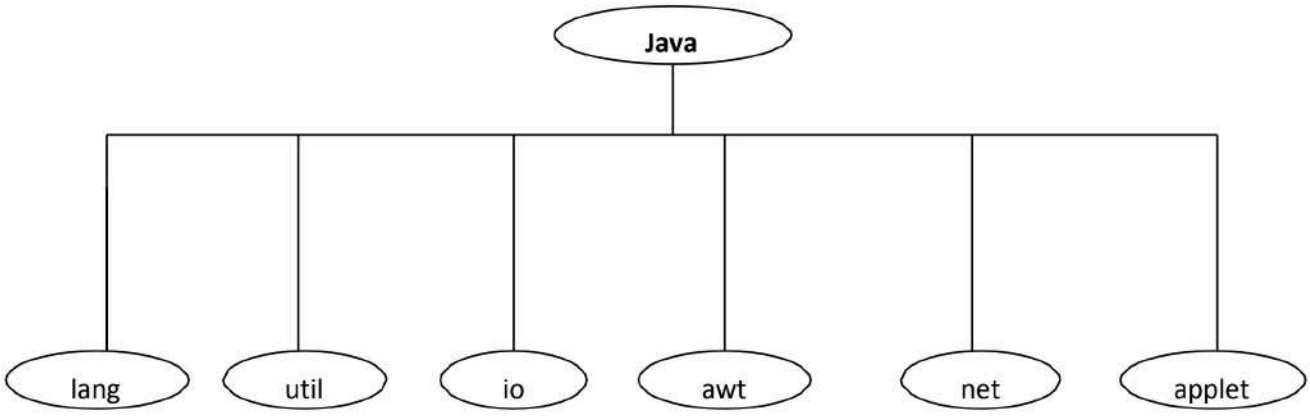
Packages are Java's way of grouping a variety of classes and / or interfaces together. The grouping isusually done according to functionality. In fact, packages act as **containers** for classes.

Benefits:

- The classes contained in the packages of other programs can be easily reused.
- In packages, classes can be unique compared with classes in other packages. That is two classes in two different packages can have the samename. They may be referred by their fully qualified name, comprising thepackage name and the classname.
- Packages provide a way to **"hide"** classes thus preventing other programs or packages from accessing classes that are meant for initernal use only.
- Packages also provide a way for separating **"design"** from **"coding"**. First we can design classes and decide their relationships, and then we can implement the Java code needed for the methods. It is possible to change the implementation of any method without affecting the rest of the design.

JAVA API PACKAGES:

Java API provides a large number of classes grouped into different packages according to functionality.Most of the time we use the packages available with the Java API. Table below shows the classes that belongs to eachpackage.



Frequently used API packages

Package name	Contents
java.lang	**Language support classes**. These are classes that Java compiler itself uses and therefore they are automatically imported.They includeclasses for primitive types, strings, math functions, threads and exceptions.
java.util	**Language utility classes** such as vectors,hash tables,randomnumbers,date, etc.,
java.io	**Input/output support classes.**They provide facilities for the input and output of data.
java.awt	**Set of classes for implementing Graphical User interface(GUI).** They include classes for windows,buttons, lists,menus andsoon.
java.net	**Classes for networking.**They include classes for communicating with local computers as well as with internet servers.
java.applet	**Classes for creating and implementing applets.**

The **package keyword** is used to create a package in java.

Accessing package from another package

There are three ways to access the package from outside the package.

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

Using packagename.*

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to thecurrent package.

Example of package that import the packagename.*

```
//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();
}
}
```

Using Packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example of package by import package.classname

```
//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();
}
}
```

Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now thereis no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Example of package by import fully qualified name

```
//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();
}
```

JAVA.UTIL PACKAGE

It contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

The package **java.util** contains a number of useful classes and interfaces. Although the name ofthe package might imply that these are utility classes, they are really more important than that. In fact, Java depends directly on several of the classes in this package, and many programs will find these classes indispensable. The classes and interfaces in java.util include:

- The **Hashtable** class for implementing hashtables, or associative arrays.
- The **Vector class**, which supports variable-length arrays.
- The **Enumeration** interface for iterating through a collection of elements.
- The **StringTokenizer** class for parsing strings into distinct tokens separated bydelimiter characters.
- The **EventObject** class and the **EventListener** interface, which form the basis of thenew AWT event model in Java 1.1.
- The **Locale** class in Java 1.1, which represents a particular locale forinternationalization purposes.
- The **Calendar** and **TimeZone** classes in Java. These classes interpret the value ofa Date object in the context of a particular calendar system.
- The **ResourceBundle** class and its subclasses, **ListResourceBundle** and **PropertyResourceBundle**, which represent sets

MULTITHREADING IN JAVA

Multithreading is a conceptual programming paradigm where a program (process) is divided into two or more subprograms (processes), which can be implemented at the same time in parallel.

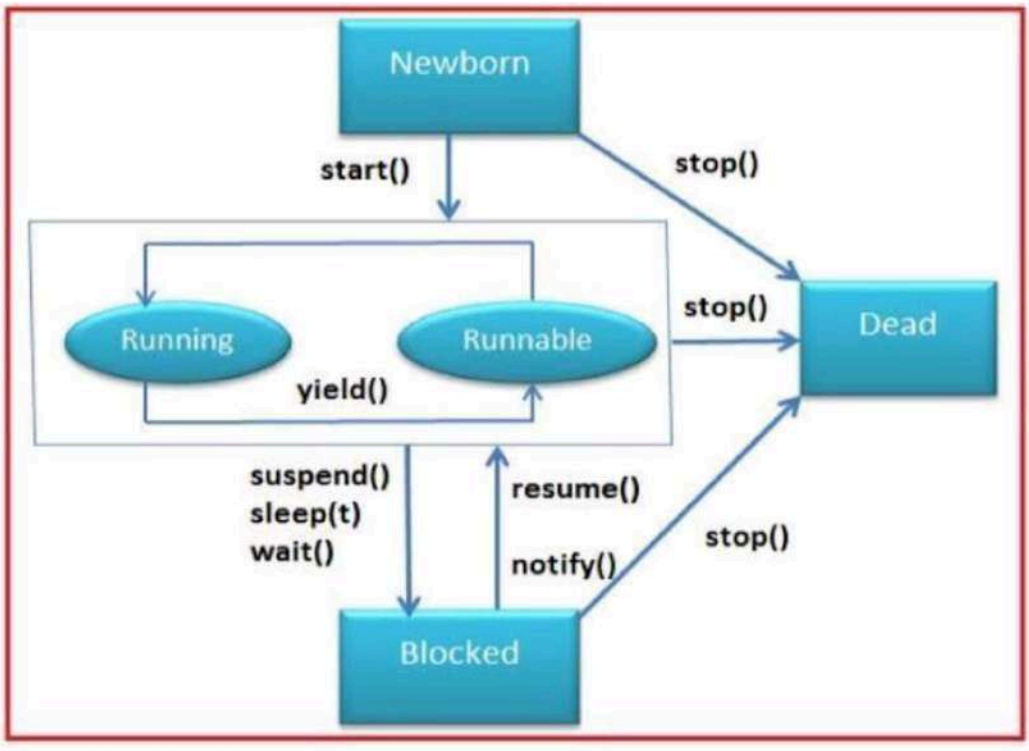
A **thread** is similar to a program that has a single flow of control. It has a beginning, a body, and an end and executes commands sequentially. A unique property of Java is its support for **multithreading**. That is, Java enables us to usemultiple flows of control in developing programs. Each flow of control may be thought of as a separate tiny program (or module) known as a **thread** that runs inparallel. A program that contains multiple flows of controls known as multithreaded program.

Multithreading is useful in a number of ways. It enables programmers to do multiple things at one time. They can divide a long program into threads and execute them in parallel.Threads are extensively used in Java–enabled browsers such as HotJava. These browsers can download a file to the local computer, displaya Web page in the window, output another Web page to a printer and so on.

LIFE CYCLE OF A THREAD

During the lifetime of a thread, there are many states it can enter. They include:

1. Newborn state
2. Runnable state
3. Running state
4. Blocked state
5. Dead state



Newborn State:

When we create a thread object, the thread is born and is said to bein **newborn state**. The thread is notyet scheduled for running. At this state, we can do only one of the following things with it.

- Schedule it for running using **start( )**method.
- Kill it using **stop()** method.

If scheduled, it moves to the running state. If we attempt to use any other method at this stage, an exception will be thrown.

Runnable State:

This **runnable** state means that the thread is ready for execution and iswaiting for the availability of the processor. That is,the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in **round robin** fashion. I.e., first-come, first – serve manner.The thread that relinquishes control joins the queue at the end and again waits for its turn.This process of assigning time to threads is known as t**ime-slicing**.

However, if we want a thread to relinquish control to another thread ofequal priority before its turncomes, we can do so by using the **yield()** method.

Running state:

Running means that the processor has given its time to the thread for its execution. The thread runsuntil it relinquishes control on its own or it is pre-empted by a higher priority thread. A running thread may relinquishits control in one of the following situations.

- It has been suspended using **suspend ( )** method. A suspended threadcan be revived by using the **resume ( )** method. This approach is usefulwhen we want to suspend thread for sometime duet certain reason, butdo not want to kill it.
- It has been made to sleep, we can put a thread to sleep for a specifiedtime period using the method **sleep** (time) where time is in milliseconds. This means that the thread is out of the queue during thistime period. The thread re-enters the runnable state as soon as this timeperiod is elapsed.
- It has been told to wait until some even to occurs.This is done using the **wait()** methods. The thread can be scheduled to run again using the **notify()** method.

Blocked State

A thread is said to be **blocked** when it is prevented from entering into the runnable state and subsequently by running state. This happens when the thread is**suspended, sleeping** or w**aiting** in orderto satisfy certain requirements. A blocked thread is considered **"not runnable** "but not dead and therefore fully qualified torun again.

Dead State

Every thread has life cycle. A running thread ends its life when it has completedexecuting its **run()** method.A thread can be killed as soon it is born, or while it is running or even when it is in **"not runnable"**(blocked) condition.

Thread methods

Use of yield(),stop(),sleep() methods

```
class A extends Thread
        {
public void run()
        {
for (int i =1; i< 5; i++)
        {
if(i==1)
    yield();
System.out.println("\tFrom Thread A : i="+i);
}
System.out.println("exit from A ");
}
}
class B extends Thread
        {
public void run ( )
        {
for (int j=1; j<=5; j++)
{
System.out.println("\tFrom Thread B: j="+j);if
(j==3)
    stop();
System.out.println("Exit from B ");
}
}
class C extends Thread
```

```
{
public void run ( )
{
for (int k=1; k<=5; k++)
{
System.out.println("\tFrom Thread C: k = " +k);if
(k==1)
        try
{
    sleep (1000);
}
    catch(Exception e)
{
{
System.out.println("Exit from C ");
}
}
class ThreadMethods
{
public static void main(String args[])
{
A threadA= new A( );
B threadB= new B( ); C threadC=new C();System.out.println("Start thread A"); threadA.start();
System.out.println("Start thread B"); threadB, start(); System.out.println("Start thread C");
threadC.start();
System.out.println("End of main thread");
}
}
```

IMPLEMENTING THE 'RUNNABLE' INTERFACE

We can create threads in two ways:one by using the extended Thread class and another by implementing the **Runnable** interface.The **Runnable** interface declares the **run()** method that is required for implementing threads inour programs.

Steps:

1. Declare the class as implementing the Runnable interface.
2. Implement the **run()**method.
3. Create a thread by defining an object that is instantiated fromths"runnable"class as the target of the thread.
4. Call the thread's **start ()**method to run the thread.

Program: Using Runnable interface

```
class X implements Runnable
{
public void run ( )
{
for (int i = 1; i<=10; i++)
{
System.out.println("\tThreadX " +i);
}
System.out.println("End of ThreadX");
}
}
class RunnableTest
{
public static void main(String args[])
{
X runnable = new X();
Thread threadX= new Thread ( runnable);Threadx.start();
System.out.println("End of main Thread");
}
}
```

THREAD SYNCHRONIZATION

So far, we have seen threads that use their own data and methods providedinside their run () methods. What happens when they try to use data and methods outside themselves?On such occasions,they may compete for the same resources and may lead to serious problems. For example, one thread may try to read a recordfrom a file while another is still writing to the same file. Depending on the situation, we may get strange results. Java enables overcome this problem using technique known as **synchronization**.

In case of Java, the keyword **synchronized** helps to solve such problems by keeping a watch on such locations. For example, the method that will read information from a file and the method that willupdate the same fil may be declared as **synchronized**.

Example:

```
    synchronized void update()
{
.....
.....                          //code here is synchronized
.....
}
```

When we declare a method synchronized, Java creates a "monitor" and hands it over to the thread that calls the method first time. As long as the thread holds the monitor, no other thread can enter the synchronized section of code. A monitor is like a key and the thread that holds the key can only open the lock.

It is also possible to mark a block of code as **synchronized** as shown below:

```
    synchronized(lock–object)
{
.....
.....                          //code here is synchronized
}
```

Whenever a thread has completed its work of using synchronized method (or block of code),it will hand over the monitor to the next thread that is ready touse the same resource.

An interesting situation may occur when two or more threads are waiting to gain control of a resource. Due to some reason, the condition on which the waiting threads rely onto gain control does not happen.This results in what is known as **deadlock**.

EXCEPTION HANDLING WITH TRY CATCH-FINALLY

EXCEPTIONS

An exception is a condition that is caused by a run – time error in the program. When the Java interpreter encounters an error such as dividing an integer by zero, it creates an exception object and throws it.
The purpose of exception handling mechanism is to provide a mean to **detect** and **report** an **"exceptional circumstances** "so that an appropriate action can be taken. Error handling code that performs the following tasks:

1. Find the problem(**Hit** the exception)
2. Inform that an error has occurred(**Throw** the exception)
3. Receive the error information(**Catch** the exception)
4. Take corrective actions(**Handle** the exception)

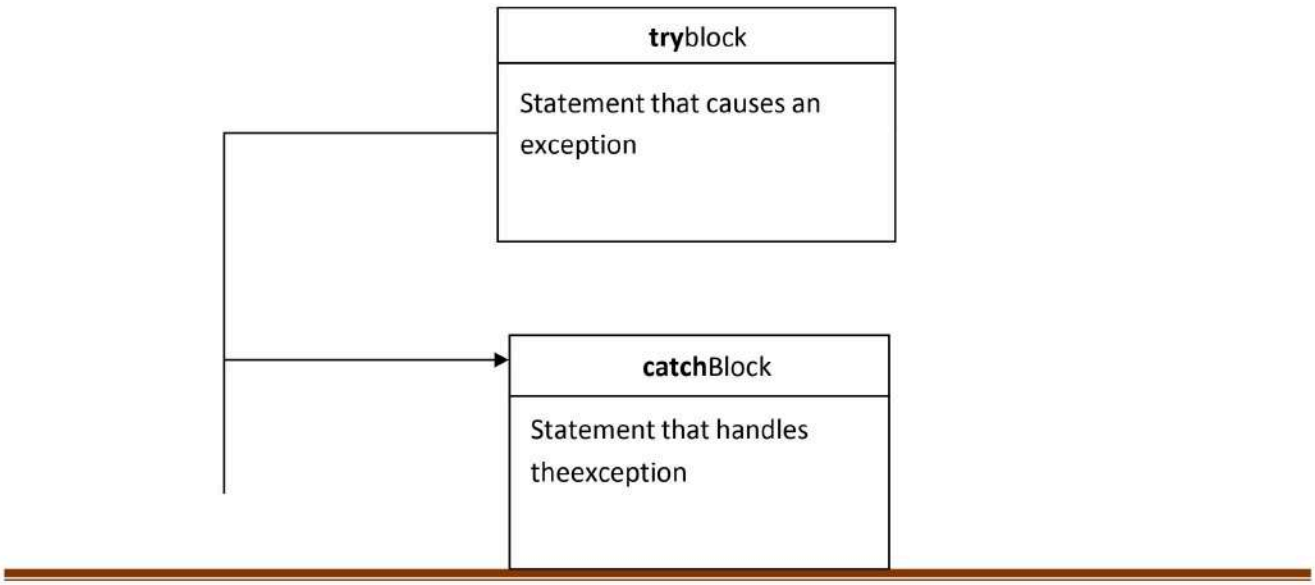
The error handling code basically consists of **two segments**, one to **detect errors** and to **Throw exceptions** and the other to **catch exceptions** and to take appropriate actions.

Exception Type	Cause of Exception
ArithmeticException	Caused by math errors such as division by zero
ArrayIndexOutOfBoundsException	Caused by bad array indexes
ArrayStoreException	Caused when a program tries to store the wrong typeof data in an array

FileNotFoundException	Caused by an attempt to access a nonexistent file
IOException	Caused by general I/O failures ,such as inabilitytoread from a file
NullPointerException	Cause by referencing a null object
NumberFormatException	Caused when a conversion between strings and numberfails
OutOfMemoryException	Caused when there is no not enough memory toallocate a new object
SecurityException	Caused when an applet tries to perform an action not allowed by the browser security setting
StackOverflowException	Caused when the system runs out of stack space
StringIndexOutOfBoundsException	Caused when a program attempts to access anonexistent character position in a string

Try catch block

The basic concepts of exception handling are **throwing** an exception and **catching** it.



Exception handling mechanism

Java uses a keyword **try** to preface a block of code that is likely to cause an error condition and **"throw"** an exception. A catch block defined by the keywordcatch **"catches"** the exception **"thrown"**by the try block and handles it appropriately. The catch block is added immediately after the tryblock.

Eg:

```
.....
.....
try
{
statement;                    //generates an exception
}
catch(Exceptiontype e )
{
statement;                  //processes the exception
.....
.....
```

The **try block** can have one or more statements that could generate an exception. If anyone statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed nextto the tryblock.

The **catch block** too can have one or more statements that are necessary to process the exception. Remember that every **try** statement should be followed by **at least one catch statement**; otherwise compilation error will occur.

Program to illustrate using try and catch for exception handling

Program : Using try and catch for exception handlingclass

```
Error3
{
public static void main(String args[])int a
= 10;
int b = 5;int c
= 5;int x, y ;
try
{
x = a (b-c); // Exception here
}
catch (ArithmeticException e)
System.out.println("Division by zero");
}
```

```
y = a / (b+c);
System.out. println ("y = " + y);
}
}
```

Output

Division by zeroy =1

MULTIPLE CATCH STATEMENTS

It is possible to have more than one catch statement in the catch block.

```
.....
.....
try
{
statement;                    //generates an exception
}
catch(Exceptiontype1e)
{
statement;                    //processesexceptiontype1
}
catch(Exceptiontype2e)
{
statement;                    //processesexceptiontype2
}
.
.
catch(ExceptiontypeN e)
{
statement;                    //processes exception type N
}
```

When an exception in a **try** block is generated, the Java treats the multiple **catch** statements like cases in a **switch** statement. The first statement whose parameter matches with the exception object will be executed, and the remaining statementswill skipped.

Note that Java does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid program abortion.

Eg:

```
catch(Exception e);
```

The catch statement simply ends with a semicolon, which does nothing. Thisstatement will catch an exception and then ignore it.

Program:Using multiple catch blocks

```
class Erro4
{
public static void main(String args[])
{
int a[ ] = {5, 10};
int b = 5;
        try
{
int x = a[2] / b - a [1];
}
    catch (ArithmeticException e)
{
System.out.println (Division by zero");
}
catch (ArraylndexOutOfBounds Exception e)
{
System.out.println("Array index error");
}
    catch (ArrayStoreException e)
{
System.out.println ("Wrong data type");
}
int y = a[1] / a[0];
System.out. println ("y = " + y);
}
}
```

Output:

Array index errory=2

USING finally STATEMENT

Java supports another statement known as finally statement that can be used to handle an exception that is not caught by any of the previous catch statements, finally block can be used to handle any exception generated within a try block. It may be added immediately after the try block or after the lastcatch block When a **finally** block is defined, this is guaranteed to execute, regardless of whether or not an exception isthrown. As a result, we can use it to closing files and releasing system resources.

```
try
{
.....
.....
}
finally
{
.....
}

Ortry
{
....
.....
}
catch(.....)
{
.....
}

catch(.....)
{
.....
}
.
.
.
finally
{
.....
}
```

Try catch finally block

```
class TestFinally
{
public static void main(String args[]
{
try
{
int data=25/0;
```

```
System.out.println(data);
}
catch(NullPointerException e)
{
System.out.println(e);
}
finally
{
System.out.println("Finally block is always executed");
}
System.out.println("rest of the code");
}
}
```

Output:

Finally block is always excecuted

Collections in Java

A Java Collection is a predefined architecture capable of storing a group of elements and behavinglike a single unit such as an object or a group.

The **Collection in Java** is a framework that provides an architecture to store and manipulate the groupof objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

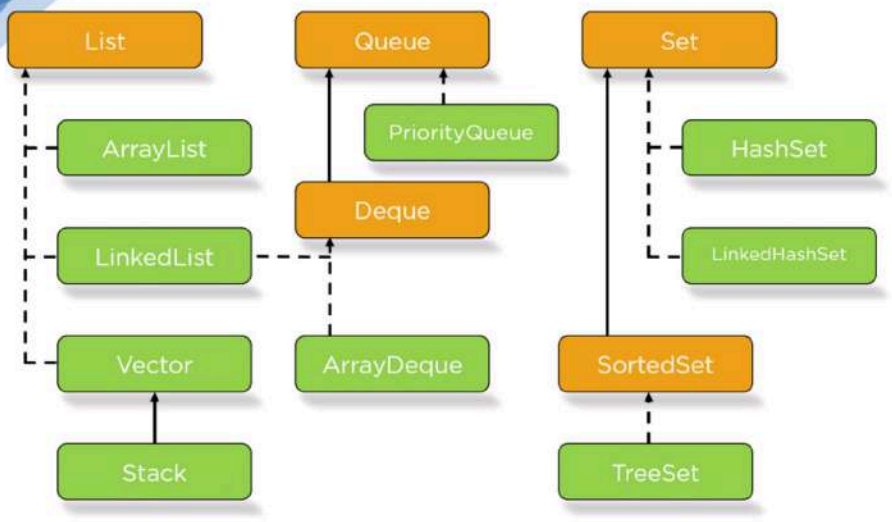
Any group of individual objects which are represented as a single unit)illloiooi9l known as the collection of the objects. In Java, a separate framework named the *"Collection Framework"* has been defined in JDK 1.2 which holds all the collection classes and **i**n it.

The Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the twomain "root" interfaces of Java collection classes.

Java Collection Framework

Java Collection Framework offers the capability to Java Collection to represent a group of elements in classes and Interfaces.

Java Collection Framework enables the user to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements. Followed by the Java Collections Framework, you must learn and understand the Hierarchy of Java collections and various descendants or classes and interfaces involved in the Java Collections.



Need for a Separate Collection Framework

Before the Collection Framework(or before JDK 1.2) was introduced, the standard methods for grouping Java objects (or collections) were **Arrays** or **Vectors**, or **Hashtables**. All of these collections had no common interface. Therefore, though the main aim of all the collections is the same, the implementation of all these collections was defined independently and had no correlationamong them.And also, it is very difficult for the users to remember all the different **methods**, syntax, and **constructors** present in every collection class.

Java Collections aíe the one-stop solutions foí all the data manipulation jobs such as stoíing data, seaíching, soíting, inseítion, deletion, and updating of data. Java collection íesponds as a single object, and a Java Collection Fíamewoík píovides vaíious Inteífacesand Classes.

The following image depicts the Java Collections Hierarchy.

After the Hierarchy of Java collections; you should also get to know the various methods applied tothe Collections in Java to perform the data manipulation operations.

Java Collections Interface Methods

The table below describes the methods available to use against Java Collections for dataManipulationJobs.

Method	Description
add()	Add objects to collection.
isEmpty()	Returns true if is empty

clear()	Removes all elements from the collection
remove()	Remove a selected object
size()	Find the number of elements
stream()	Return Sequential elements
toArray()	Returns elements in array format
hashCode()	Returns Hashcode of the elements
equals(obj X)	Compare an element with the collection
iterator()	Return an iterator over collection
max()	Return max value in the collection
contains()	Returns true is a particular value is present

JAVA BEANS

JavaBeans are reusable software components for Java

- Build re-useable applications or program building blocks called components that can bedeployed in a network on any major operating system platform.
- They are used to encapsulate many objects into a single object (the bean), so that they can bepassed around as a single bean object instead of as multiple individual objects.
- A JavaBeans is a Java Object that is serializable, has a 0-argument constructor, and allows access to properties using getter and setter methods.
- Like Java applets, JavaBeans components (or "Beans") can be used to give World Wide Web pages (or other applications) interactive capabilities such as computing interest rates or varyingpage content ased on user or browser characteristics.

Advantages

- A Bean obtains all of the benefits of Java's "write once, run anywhere" paradigm.
- The properties, events, and methods of a Bean that are exposed to another application canbe controlled.
- Auxiliary software can be provided to help configure a Bean.
- The configuration settings of a Bean can be saved in a persistent storage and can berestored at a later time.
- A Bean may register to receive events from other objects and can generate events that aresent to it.

Disadvantages

- A class with a constructor is subject to being instantiated in an invalid state. If such a class is instantiated manually by a developer (rather than automatically by some kind of
- framework), the developer might not realize that the class has been improperlyinstantiated.
- The compiler can't detect such a problem, and even if it's documented, there's noguarantee that the developer will see the documentation.
- Having to create a getter for every property and a setter for many, most, or all of them, creates an immense amount of boilerplate code.

JavaBeans API

The JavaBeans functionality is provided by a set of classes and interfaces in the java.beans package.

Interface	Description
AppletInitializer	Methods in this interface are used to initialize Beans that are also applets.
BeanInfos	This interface allows the designer to specify information about the events, methods and properties of a Bean.
Customizer	This interface allows the designer to provide a graphical user interface through which a bean may be configured.
DesignMode	Methods in this interface determine if a bean is executing in design mode.
ExceptionListener	A method in this interface is invoked when an exception has occurred.
PropertyChangeListener	A method in this interface is invoked when a bound property is changed.
PropertyEditor	Objects that implement this interface allow the designer to change and display property values.
Visibility	Methods in this interface allow a bean to execute in environments where GUI is not available.

NETWORK PROGRAMMING

Network programming involves writing computer programs that enable processes tocommunicate with each other across a computer network.

Network programming is client–server programming

- The process initiating the communication is a client, and the process waiting for the communication to be initiated is a server. The client and server processes together form a distributed system. In a peer-to-peer communication, the program can act both as a client anda server.

Network programming is socket programming

- The endpoint in an inter process communication is called a socket, or a network socket
- Since most communication between computers is based on the Internet Protocol, an almost equivalent term is Internet socket.
- The data transmission between two sockets is organized by communications protocols, usually implemented in the operating system of the participating computers. Application programs write to and read from these sockets. Therefore, network programming is essentially socket programming.

-Standard API

- Application programs create, control, and use sockets through system calls like socket()bind(), listen(), connect(), send(), recv()
- The term network programming refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.
- The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.
- The java.net package provides support for the two common network protocols:
    - TCP: TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol,which is referred to as TCP/IP.
    - UDP: UDP stands for User Datagram Protocol, a connection-less protocol that allows forpackets of data to be transmitted between applications.

Unit-3

Event and GUI programming

Event Handling in Java

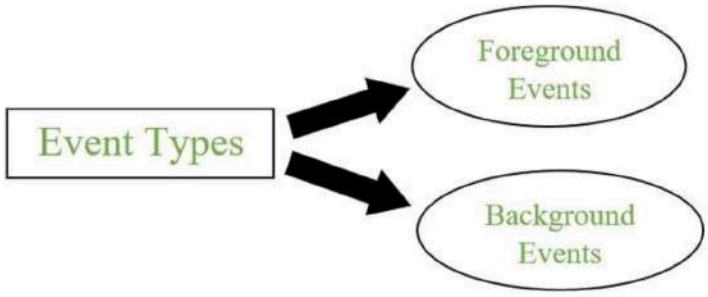
Event

An **event** can be defined as changing the state of an object or behavior by performing actions. Actions can be abutton click, cursor movement, keypress through keyboard or page scrolling, etc.

The **java.awt.event** package can be used to provide various event classes.

Classification of Events

- Foreground Events
- Background Events



Types of Events

1. **Foreground Events**

Foreground events are the events that require user interaction to generate, i.e., foreground events are generateddue to interaction by the user on components in Graphic User Interface (**GUI**). Interactions are nothing but clicking on a button, scrolling the scroll bar, cursor moments, etc.

2. Background Events

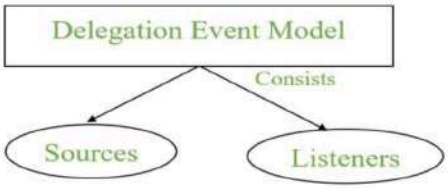
Events that don't require interactions of users to generate are known as background events. Examples of these events are operating system failures/interrupts, operation completion, etc.

Event Handling

It is a mechanism to **control the events** and to **decide what should happen after an event** occurs. To handle theevents, Java follows the ***Delegation Event model.***

Delegation Event model

- It has Sources and Listeners.



Source: Events are generated from the source. There are various sources like buttons, checkboxes, list,menu-item, choice, scrollbar, text components, windows, etc., to generate events.

Listeners: Listeners are used for handling the events generated from the source. Each of these listeners represents interfaces that are responsible for handling events.

Handling Keyboard Events

When a key is pressed, a **KEY_PRESSED** event is generated. This results in a call to the keyPressed ( ) event handler. When the key is released, a **KEY_RELEASED** event is generated and the keyReleased ( ) handler is executed. If a character is generated by the keystroke, then a **KEY_TYPED** event is sent and the keyTyped ( ) handler is invoked. A user interacts with the application by pressing either keys on the keyboard or by using mouse. A programmer should know which key the user has pressed on the keyboard orwhether the mouse is moved, pressed, or released. These are also called 'events'. Knowing these events will enable the programmer to write his code according to the key pressed or mouse event.

KeyListener interface of java.awt.event package helps to know which key is pressed orReleased by the user.

It has 3 methods

1. public **void keyPressed** (KeyEvent ke): This method is called when a key is pressedon the keyboard. This include any key on the keyboard along with special keys like function keys, shift, alter, caps lock, home, end etc.
2. public **void keyTyped**(keyEvent ke) : This method is called when a key is typed onthe keyboard. This is same as keyPressed () method but this method is called when general keys like A to Z or 1 to 9 etc are typed. It cannot work with special keys.
3. public **void keyReleased**(KeyEvent ke): this method is called when a key is release.

KeyEvent class has the following methods to know which key is typed by the user.

1. char **getKeyChar**(): this method returns the key name (or character) related to the keypressed or released.
2. int **getKeyCode**(): this method returns an integer number which is the value of thekey pressed by the user.

EX: **Demonstrate the key event handlers.**

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

/* <applet code="SimpleKey" width=300
height=100>

</applet> */

public class SimpleKey extends Applet implements KeyListener

{

String msg = "";

int X = 10, Y = 20;                    //  output
coordinatespublic void init()
{
addKeyListener(this);
requestFocus();                        // request input focus
```

```
}
public void keyPressed(KeyEvent ke)
{
showStatus("Key Down");
}
public void keyReleased(KeyEvent ke)
{
showStatus("Key Up");
}
public void keyTyped(KeyEvent ke)
{
msg += ke.getKeyChar();
repaint();
}
// Display keystrokes.
public void paint(Graphics g)
{
g.drawString(msg, X, Y); }
}
```

Handling Mouse Events

The user may click, release, drag, or move a mouse while interacting with a the application. If the programmerknows what the user has done, he can write the code according to the mouse events. To trap the mouse events,MouseListener and MouseMotionListener interfaces of java.awt.event package are use.

MouseListener interface has the following methods.

1. void **mouseClicked**(MouseEvent me); this method is invoked when the mouse button hasbeen clicked(pressed and released) on a component.
2. void **mouseEntered**(MouseEvent me): this method is invoked when the mouse enters a component.
3. void **mouseExited**(MouseEvent me): this method is invoked when the mouse exits a component
4. void **mousePressed**(MouseEvent me): this method is invoked when a mouse button has been pressed on acomponent.
5. void **mouseRelease**(MouseEvent me): this method is invoked when a mouse button has been released on acomponent.

MouseMotionListener interface has the following methods.

1. void **mouseDragged**(MouseEvent me): this method is invoked when a mouse button ispressed on a component and then dragged.
2. void **mouseMoved**(MouseEvent me): this method is invoked when a mouse cursor hasbeen moved onto a component and then dragged.

The MouseEvent class has the following methods

1. int **getButton**(): this method returns a value representing a mouse button, when itis clicked it return 1 if left button is clicked, 2 if middle button, and 3 if right button is clicked.
2. int **getX**(); this method returns the horizontal x position of the event relative to the source component.
3. int **getY**(); this method returns the vertical y postion of the event relative to thesource component.

Program to create a text area and display the mouse event when the button on themouse is clicked, when mouse is moved, etc. is done by user.

Handling Mouse Events

To handle mouse events, we must implement the MouseListener and the MouseMotion Listener interfaces.

EX: // Demonstrate the mouse event handlers.

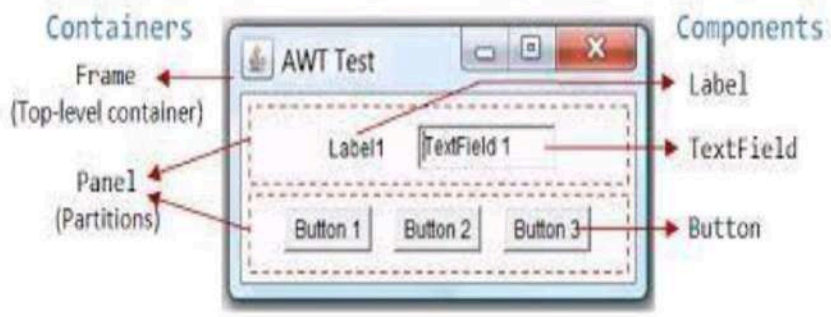
```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="MouseEvents" width=300 height=100>
</applet>
*/
public class MouseEvents extends Applet implements MouseListener, MouseMotionListener
{
String msg = "";
int mouseX = 0, mouseY = 0;                    //    coordinates    of
mousepublic void init()
{
addMouseListener(this); addMouseMotionListener(this);
}
public void mouseClicked(MouseEvent me)
{
mouseX = 0;                //                save
coordinatesmouseY = 10;
msg  =  "Mouse  clicked.";
repaint();
}
public void mouseEntered(MouseEvent me)
{
mouseX = 0;
mouseY = 10;
msg  =  "Mouse  entered.";
repaint();
}
public void mouseExited(MouseEvent me)
{
mouseX= 0;
mouseY = 10;
msg  =  "Mouse  exited.";
repaint();
}
public void mousePressed(MouseEvent me)
{
mouseX   =   me.getX();
mouseY = me.getY();
msg = "Down";repaint();
```

```
}
// Handle button released.
public void mouseReleased(MouseEvent me)
{

mouseX = me.getX();
mouseY = me.getY();msg=
"Up"; repaint();
}
public void mouseDragged(MouseEvent me)

{
mouseX = me.getX();
mouseY = me.getY(); msg
= "*";
showStatus("Dragging mouse at " + mouseX + ", " + mouseY);
repaint();
}
public void mouseMoved(MouseEvent me)
{
showStatus("Moving mouse at " + me.getX() + ", " + me.getY());
}
public void paint(Graphics g)
{
g.drawString(msg, mouseX, mouseY);
}
}
```

Frame and panel

What is the Difference Between Panel and Frame in Java?

The main difference **between Panel and Frame in Java is that the** Panel is an internal region to a frame or another panel that helps to group multiple components together while a Frame is a resizable, movable independent window with a title bar which contains all other components.



What is Panel

Panel is a component that allows placing multiple components on it. It is created using the Panel class. This classinherits the Container class. Refer the below program.

Test.java

```
1  package ex1;
2  import java.awt.*;
3
4  public class PanelTest {
5
6      public static void main(String[] args) {
7
8              Frame f= new Frame("GUI Application");
9              Panel panel=new Panel();
10             panel.setBounds(40,80,200,200);
11             panel.setBackground(Color.LIGHT_GRAY);
12
13             Button b1=new Button("Button 1");
14             b1.setBounds(50,100,80,30);
15             b1.setBackground(Color.blue);
16
17             panel.add(b1);
18             f.add(panel);
19             f.setSize(400,400);
20             f.setLayout(null);
21             f.setVisible(true);
22     }
```

In the above program, f is a Frame object while the panel is a Panel object. The panel object is placed according to the specified location using setBounds method. The color of the panel is Gray. The b1 is a button object that isplaced according to the specified location. The color of the button is blue. Then, b1 button is added to the panel and the panel is added to the Frame f1. Finally, the frame f1 is visible with the components.

What is Frame

Frame is a component that works as the main top-level window of the GUI application. It is created using the Frame class. For any GUI application, the first step is to create a frame. There are two methods to create a frame: by extending the Frame class or by creating an object of Frame class.

According to the above program (Figure 1), f is a Frame object. Other GUI components are added to it. Finally, theframe is displayed. The frame is a resizable and a movable window. It has the title bar. The default visibility of a Frame is hidden. The programmer has to make it visible by using setVisible method and providing the value “true”to it.

Applet Life Cycle in Java

Applet Life Cycle:

Applets are small java programs that are primarily used in internet computing. It can be transported over the internet from one computer to another and run using the Applet Viewer or any web browser that supports Java. An applet is like an application program, it can do many things for us. It can perform **arithmetic operations, display graphics, create animation & games, play sounds** and accept user input, etc. Every Java applet inheritsa set of default behaviours from the Applet class. As a result, when an applet is loaded, it undergoes a series of changes in its states. Applet Life Cycle is defined as how the applet created, started, stopped and destroyed during the entire execution of the application. The methods to execute in the Applet Life Cycle
are **init()**, **start()**, **stop()**, **destroy()**. The Applet Life Cycle Diagram is given as below:

- Born or Initialization State
- Running State
- Idle State
- Dead State

1. Born or Initialization State: Applets enters the initialization state when it is first loaded. It is achieved by **init()** method of Applet class. Then applet is born.

- **init():** The init() method is the first method to run that initializes the applet. It can be invoked only once at the time of initialization. The web browser creates the initialized objects, i.e., the web browser (after checking the security settings) runs the init() method within the applet.

```
public void init()
{
.....
.....
(Action)
}
```

2. Running State: Applet enters the running state when the system calls the **start()** method of Applet class. This occurs automatically after the applet is initialized.

- **start():** The start() method contains the actual code of the applet and starts the applet. It is invoked immediately after the init() method is invoked. Every time the browser is loaded or refreshed, the start() method is invoked. It is also invoked whenever the applet is maximized, restored, or moving from one tabto another in the browser. It is in an inactive state until the init() method is invoked.

```
public void start()
{
.....
.....
(Action)
}
```

3. Display State

Applet moves to the display state whenever it has to perform some output operations onthe screen. This happens immediately after the applet enters into the running state.

The paint () method is called to accomplish this task. Almost every applet will have a paint () method.

```
public void paint(Graphics g)
{
// Any shape's code

}
```

4. Idle State: An applet becomes idle when it is stopped from running. Stopping occurs automatically when weleave the page containing the currently running applet. This is achieved by calling the **stop()** method explicitly.

- **stop():** The stop() method stops the execution of the applet. The stop () method is invoked whenever the applet is stopped, minimized, or moving from one tab to another in the browser, the stop() method is invoked. When we go back to that page, the start() method is invoked again.

```
public void stop()
{
.....
.....
(Action)
}
```

4. Display state:

5. **Dead State:** An applet is said to be dead when it is removed from memory. This occurs automatically byinvoking the **destroy()** method when we want to quit the browser.

```
public void destroy()
{
.....
.....
(Action)
```

```
}
```

- **destroy():** The destroy() method destroys the applet after its work is done. It is invoked when the appl window is closed or when the tab containing the webpage is closed. It removes the applet object fro memory and is executed only once. We cannot start the applet once it is destroyed.

Layout Manager

What is a LayoutManager and types of LayoutManager in Java?

The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.

Types of LayoutManager

- **FlowLayout**: It arranges the components in a container like the words on a page. It fills the top line from **left to right and top to bottom**. The components are arranged in the order as they are added i.e first components appears at top left, if the container is not wide enough to display all thecomponents, it i wrapped around the line. Vertical and horizontal gap between components can b controlled. The components can be **left, center or right aligned.**
- **BorderLayout**: It arranges all the components along the edges or the middle of the container

i.e. **top, bottom, right and left** edges of the area. The components added to the top or bottom getsit preferred height, but its width will be the width of the container and also the components addedto the lef or right gets its preferred width, but its height will be the remaining height of the container. Th components added to the center gets neither its preferred height or width. It covers the remaining area o the container.

- **GridLayout**: It arranges all the components in a grid of **equally sized cells**, adding them from

the **left to right** and **top to bottom**. Only one component can be placed in a cell and each region ofthe gri will have the same size. When the container is resized, all cells are automatically resized. The order o placing the components in a cell is determined as they were added.

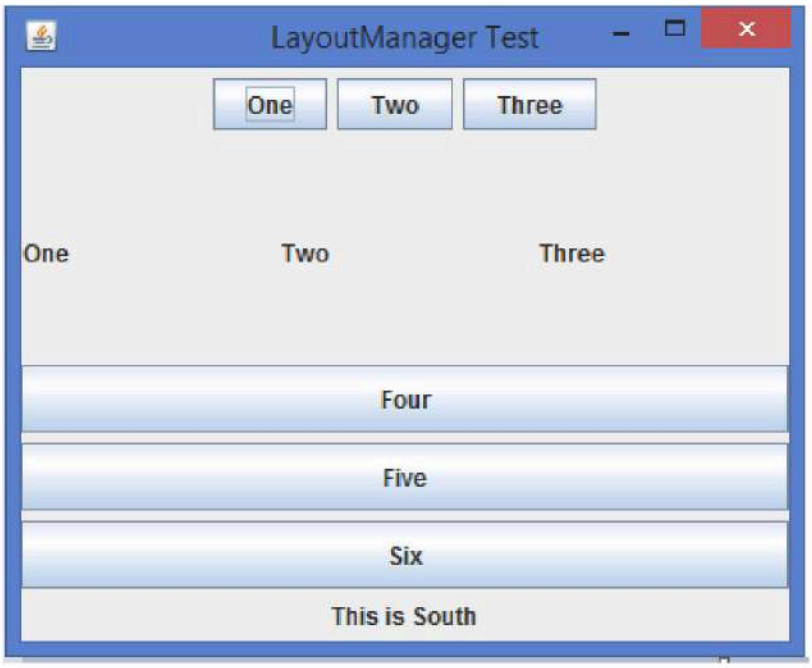
Example

```
import
 java.awt.*;
 import
 javax.swing.*
 ;
public class LayoutManagerTest extends JFrame {
JPanel flowLayoutPanel1, flowLayoutPanel2, gridLayoutPanel1,
 gridLayoutPanel2, gridLayoutPanel3;
JButton one, two, three, four, five,
 six;JLabel bottom, lbl1, lbl2, lbl3;
public LayoutManagerTest() {
setTitle("LayoutManager Test");
```

```
setLayout(new BorderLayout()); // Set BorderLayout for JFrame
flowLayoutPanel1 = new JPanel();
one = new JButton("One"); two = new
JButton("Two");  three  =  new
JButton("Three");
        flowLayoutPanel1.setLayout(new FlowLayout(FlowLayout.CENTER)); // Set FlowLayout
Manager
flowLayoutPanel1.add(one);
flowLayoutPanel1.add(two);
flowLayoutPanel1.add(three);
flowLayoutPanel2 = new JPanel();bottom =
new JLabel("This is South");
        flowLayoutPanel2.setLayout (new FlowLayout(FlowLayout.CENTER)); // Set FlowLayout
Manager
flowLayoutPanel2.add(bottom);
gridLayoutPanel1  =  new  JPanel();
gridLayoutPanel2  =  new  JPanel();
gridLayoutPanel3 = new JPanel();lbl1 = new
JLabel("One");
lbl2 = new JLabel("Two");lbl3 = new
JLabel("Three");  four  =  new
JButton("Four");  five  =  new
JButton("Five");six = new JButton("Six");
gridLayoutPanel2.setLayout(new GridLayout(1, 3, 5, 5)); // Set GridLayout Manager
gridLayoutPanel2.add(lbl1);
gridLayoutPanel2.add(lbl2);
gridLayoutPanel2.add(lbl3);
gridLayoutPanel3.setLayout(new GridLayout(3, 1, 5, 5)); // Set GridLayout Manager
gridLayoutPanel3.add(four);
gridLayoutPanel3.add(five);
gridLayoutPanel3.add(six);
gridLayoutPanel1.setLayout(new GridLayout(2, 1)); // Set GridLayout Manager
gridLayoutPanel1.add(gridLayoutPanel2);
```

```
gridLayoutPanel1.add(gridLayoutPanel3); add(flowLayoutPanel1,
BorderLayout.NORTH);    add(flowLayoutPanel2,    BorderLayout.SOUTH);
add(gridLayoutPanel1,  BorderLayout.CENTER);  setSize(400,  325);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setVisible(true);
}
 public  static  void  main(String  args[])  { new
LayoutManagerTest();
}
}
```

Output



Layout Managers

In Java, Layout Managers is used for arranging the components in order. LayoutMananger is an interface which implements the classes of the layout manager.

Below are some of the class which are used for the representation of layout manager.

1. java.awt.BorderLayout
2. java.awt.FlowLayout

3. java.awt.GridLayout

Border Layout

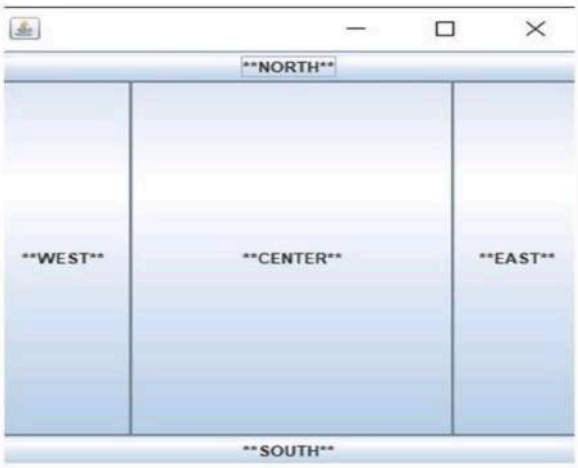
BorderLayout is used, when we want to arrange the components in five regions. The five regions can be north,south, east, west and the centre. There are 5 types of constructor in Border Layout. They are as following:

1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

Example:

```
import java.awt.*;

import javax.swing.*;



public class BorderDemo1

{

JFrame frame;

BorderDemo1()

{

frame=new JFrame();



JButton box1=new JButton("**NORTH**");; JButton box2=new JButton("**SOUTH**");;
```

```
JButton box3=new JButton("**EAST**");; JButton
box4=new JButton("**WEST**");; JButton box5=new
JButton("**CENTER**");;


frame.add(box1,BorderLayout.NORTH);          frame.add(box2,BorderLayout.SOUTH);
frame.add(box3,BorderLayout.EAST);           frame.add(box4,BorderLayout.WEST);
frame.add(box5,BorderLayout.CENTER);


frame.setSize(400,400);
frame.setVisible(true);
}

public static void main(String[] args)

{

new BorderDemo1();

}

}
```



Grid Layout

Grid Layout is used, when we want to arrange the components in a rectangular grid.

There are 3 types of constructor in Grid Layout. They are as following:

1. GridLayout()
2. GridLayout(int rows, int columns)
3. GridLayout(int rows, int columns, inthgap, int vgap)

Example:

```
import java.awt.*;

import javax.swing.*;



public class GridDemo1{

JFrame frame1;

GridDemo1(){

frame1=new JFrame();   JButton box1=new JButton("*1*");
```

```
JButton box2=new
JButton("*2*"); JButton
box3=new JButton("*3*");
JButton box4=new
JButton("*4*"); JButton
box5=new JButton("*5*");
JButton box6=new
JButton("*6*"); JButton
box7=new JButton("*7*");
JButton box8=new
JButton("*8*"); JButton
box9=new JButton("*9*");


frame1.add(box
1);
frame1.add(box
2);
frame1.add(box
3);
frame1.add(box
4);
frame1.add(box
5);
frame1.add(box
6);
frame1.add(box
7);
frame1.add(box
8);
frame1.add(box
9);


frame1.setLayout(new
GridLayout(3,3));
frame1.setSize(500,500);
```

```
frame1.setVisible(true);

}

public static void main(String[] args) {

new GridDemo1();

}

}
```

Flow Layout

Flow Layout is used, when we want to arrange the components in a sequence one after another.

There are 3 types of constructor in the Flow Layout. They are as following:

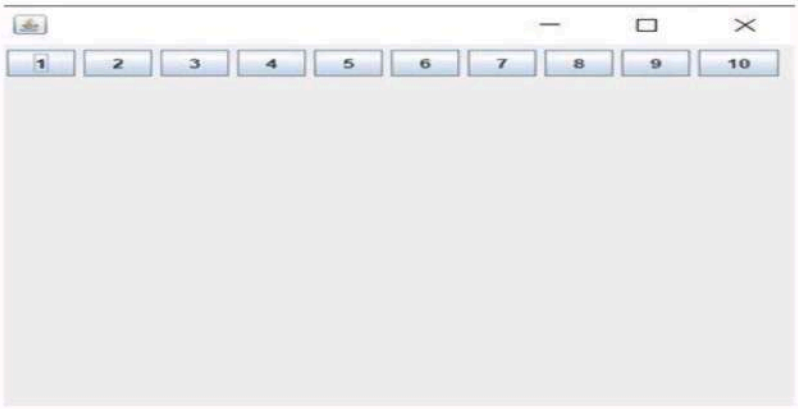
1. FlowLayout()
2. FlowLayout(int align)
3. FlowLayout(int align, inthgap, intvgap)

Example:

JAVA

```
import java.awt.*;

import javax.swing.*;



public class FlowDemo1{
JFrame          frame1;
FlowDemo1(){
frame1=new JFrame();



JButton  box1=new  JButton("1");
JButton  box2=new  JButton("2");
JButton  box3=new  JButton("3");
JButton  box4=new  JButton("4");
JButton  box5=new  JButton("5");
JButton  box6=new  JButton("6");
JButton  box7=new  JButton("7");
JButton  box8=new  JButton("8");
JButton  box9=new  JButton("9");
JButton box10=new JButton("10");



frame1.add(box1);
frame1.add(box2);
frame1.add(box3);
```

```
frame1.add(box4);
frame1.add(box5);
frame1.add(box6);
frame1.add(box7);
frame1.add(box8);
frame1.add(box9);
frame1.add(box10);
frame1.setLayout(new FlowLayout(FlowLayout.LEFT));




frame1.setSize(400,400);
frame1.setVisible(true);
}


ublic static void main(String[] args) {
new FlowDemo1();
}


}
```



Programming with Java GUI components

Java's GUI components include labels, text fields, text areas, buttons, pop-up menus, etc. The SWING Toolkitalso includes containers which can include these components. Containers include frames (windows), canvases(which are used to draw on), and panels (which are used to group components). Panels, buttons, and other components can be placed either directly in frames or in panels inside the frames.

These GUI components are automatically drawn whenever the window they are in is drawn. Thus we will notneed to explicitly put in commands to draw them..

Actions on these GUI components are handled using Java's event model. When a user interacts with a component (clicks on a button, types in a field, chooses from a pop-up menu, etc.), an event is generated by the component that you interact with. For each component of your program, the programmer is required to designate one or more objects to "listen" for events from that component. Thus if your program has a button labeled "start" you must assign one or more objects that will be notified when a user clicks on the button.

1 Buttons:

JButton is a class in package javax.swing that represents buttons on the screen. The most common constructor is:public JButton(String label);
which, when executed, creates a button with "label" printed on it. Generally the button is large enough to display label. There is also a parameter less constructor that creates an unlabeled button. Buttons respond to a variety of messages, but the only one likely to be useful to you is getText(), which returns a String representing the label onthe button.
You add an "**ActionListener**" to a button with the method:
public void addActionListener(ActionListener listener);

Adding buttons to a JFrame or JPanel

We can add a button to a frame (window) or panel of a frame by sending the message:

myFrame.add(startButton);

Normally we include such code in the constructor for the frame or panel. Hence usually we just write this.add(startButton) or simply add(startButton).

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

```
public class ButtonDemo extends JFrame {
protected JButton startButton, stopButton;
// Constructor sets features of the frame, creates buttons, adds them to the frame, and
// assigns an object to listen to them
public ButtonDemo() { super("Button
demo"); // set title bar
setSize(400,200); // sets the size of the window
startButton = new JButton("Start"); // create two new buttons w/labels start and stop
stopButton = new JButton("Stop");
Container contentPane = getContentPane();
contentPane.setLayout(new FlowLayout()); // layout objects from left to right
// until fill up row and then go to next row
contentPane.add(startButton);
contentPane.add(stopButton);
// create an object to listen to both buttons:
ButtonListener myButtonListener = new ButtonListener();
startButton. AddActionListener(myButtonListener);
stopButton.addActionListener(myButtonListener);
}

public static void main(String[] main) {
ButtonDemo app = new ButtonDemo(); // Create an instance of Buttons
app.setVisible(true); // Show it on the screen
}
}
```

Other GUI components

Labels

A JLabel is a very simple component which contains a string. The constructors arepublic
JLabel() // creates label with no text
public JLabel(String text) //create label with
textThe methods available are
public String getText() // return label text
public void setText(String s) // sets the label
text

Text Fields

A JTextField is an area that the user can type one line of text into. It is a good way of getting textinput from the user. The constructors are

public JTextField () // creates text field with no text
public JTextField (int columns)
// create text field with appropriate # of columns
public JTextField (String s) // create text field with s displayed
public JTextField (String s, int columns)
// create text field with s displayed & approp. width

Methods include:

public void setEditable(boolean s)
// if false the TextField is not user editable
public String getText() // return label text

public void setText(String s) // sets the label text

Many other methods are also available for this component (see also the documentation for its superclass,**JTextComponent**).

When the user types into a text field and then hits the return or enter key, it generates an event which can be handled by the same kind of ActionListener used with JButtons. Thus in order to respond to the user's hitting of the return key while typing in a JTextField, myField, we can write: public void actionPerformed(ActionEvent evt) {

String contents = myField.getText();

System.out.println("The field contained: "+contents);

}

Text Areas

JTextArea is a class that provides an area to hold multiple lines of text. It is fairly similar to JTextFieldexcept that no special event is generated by hitting the return key.

The constructors are:

public JTextArea(int rows, int columns)

// create text field with appropriate # rows & columns

public JTextArea(String s, int rows, int columns)

// create text field with rows, columns, & displaying s

Methods:

public void setEditable(boolean s)

// if false the text area is not user editable

public String getText() // return text in the text area

public void setText(String s) // sets the text

public void append(String s) // append the text to text in the text area

JComboBox menus

JComboBox provides a pop-up list of items from which the user can make a selection.The constructor is:

public JComboBox() // create new choice buttonThe most useful methods are:

public void addItem(Object s) // add s to list of choices public int getItemCount() // return # choices in list public Object getItemAt(int index) // return item at indexpublic Object getSelectedItem() // return selected item public int getSelectedIndex() // return index of selected

When an object is added to a JComboBox, the results of sending toString() to the combo box is displayed in the corrresponding menu.

You can also set the combo box so the user can type an element in the combo box rather than being forced to select an item from the list. This is done by sending the combo box the message setEditable(true).

When a user selects an item it generates an ActionEvent which can be handled as usual with an actionPerformed method. One can determine whether the combo box generated the event by sending the event the getSource() method, just as we did with other components

Check Boxes

The JCheckbox class provides support for check box buttons. You can also put check boxes in menus, using

the `JCheckBoxMenuItem` class. Because `JCheckBox` and `JCheckBoxMenuItem` inherit from `AbstractButton`

Check boxes are similar to radio buttons but their selection model is different, by convention. Any number of check boxes in a group — none, some, or all — can be selected. A group of radio buttons, on the other hand, canhave only

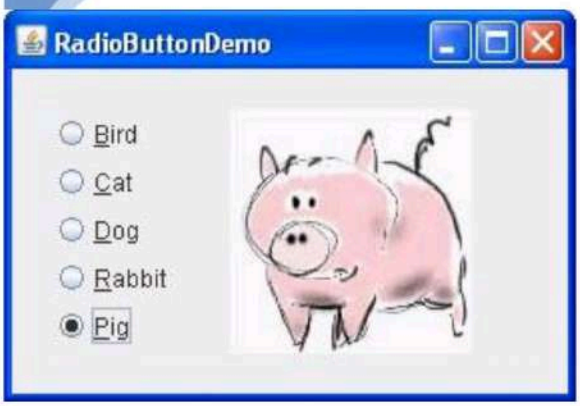
one button selected.

```
chinButton = new JCheckBox ("Chin");

chinButton.setMnemonic          (KeyEvent.VK_C);
chinButton.setSelected (true);

glassesButton   =   new   JCheckBox   ("Glasses");
glassesButton.setMnemonic          (KeyEvent.VK_G);
glassesButton.setSelected (true);

hairButton      =      new      JCheckBox      ("Hair");
hairButton.setMnemonic          (KeyEvent.VK_H);
hairButton.setSelected (true);

teethButton     =     new     JCheckBox     ("Teeth");
teethButton.setMnemonic          (KeyEvent.VK_C);
teethButton.setSelected (true);

        //Register a listener for the check
        boxes.    chinButton.addItemListener
        (this); glassesButton.addItemListener
        (this);      hairButton.addItemListener
        (this);    teethButton.addItemListener
...     (this);
```

Radio buttons

Radio buttons are groups of buttons in which, by convention, only one button at a time can be selected. The Swing release supports radio buttons with the `JRadioButton` and `ButtonGroup` classes. To put a radio button ina menu, use the `JRadioButtonMenuItem` class. Other ways of displaying one-of-many choices are combo boxes and lists. Radio buttons look similar to check boxes, but, by convention, check boxes place no limits on howmany items can be selected at a time.

Because `JRadioButton` inherits from `AbstractButton`, Swing radio buttons have all the usual buttoncharacteristics




```
JRadioButton birdButton = new JRadioButton (birdString);
   birdButton.setMnemonic                    (KeyEvent.VK_B);
   birdButton.setActionCommand                    (birdString);
   birdButton.setSelected (true);

   JRadioButton catButton = new JRadioButton (catString);
   catButton.setMnemonic                     (KeyEvent.VK_C);
   catButton.setActionCommand (catString);


   //Group the radio buttons.
   ButtonGroup group = new ButtonGroup ();
   group.add (birdButton);
   group.add (catButton);

 //Register a listener for the radio buttons.
   birdButton.addActionListener          (this);
   catButton.addActionListener (this);
```

Scrollbar

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allowsus to see invisible number of rows and columns .It can be added to top-level container like Frame or a componentlike Panel. The Scrollbar class extends the **Component** class.

AWT Scrollbar Class Declaration

public class Scrollbar **extends** Component **implements** Adjustable, Accessible

Scrollbar Class Fields

The fields of java.awt. Image class are as follows:

- **static int HORIZONTAL** - It is a constant to indicate a horizontal scroll bar.
- **static int VERTICAL** - It is a constant to indicate a vertical scroll bar.

Menus

A **menu** is a list of buttons each of which have their own corresponding action when selected. Types of menusinclude:

- drop-down (or pull-down) - usually associated with an application's menu bar
- popup - associated with any container component (i.e., often accessed via right button click) o
- cascaded - pops up when another menu item is selected (i.e. a sub menu)

DIALOG BOX

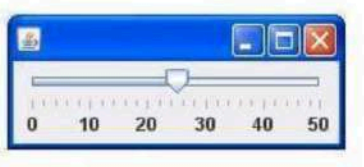
A dialog box is: • a separate window that pops up in response to an event occurring in a window. often used toobtain information from the user (e.g., entering some values such as when filling out a form).

There are various types of commonly used dialog boxes:

1. Message Dialog - displays a message indicating information, errors, warnings etc...
2. Confirmation Dialog - asks a question such as yes/no
3. Input Dialog - asks for some kind of input
4. Option Dialog - asks the user to select some option JAVA has a class called JOptionP

Sliders

The Java JSlider class is used to create the slider. By using JSlider, a user can set value from a specific range.



Introduction to Java Swing

Java Swing is part of Java Foundation Classes. It is used to create window-based applications which makes it suitable for developing lightweight desktop applications. Java Swing is built on top of an abstract windowing toolkit API purely written in Java programming language.

Java Swing provides lightweight and platform-independent components, making it suitable and efficient in designing and developing desktop-based applications (systems).

Features of Swing

The features of the Swing are as follows:**1. Platform Independent:** It is platform-independent; the swing components that are used to build the program are not platform-specific. It can be used on any platform and anywhere.

2. **Lightweight:** Swing components are lightweight, which helps in creating the UI lighter. The swings component allows it to plug into the operating system user interface framework that includes the mappings for screens or devices and other user interactions like keypress and mouse movements.

3. **Plugging:** It has a powerful component that can be extended to provide support for the user interface that helps in a good look and feel to the application. It refers to the highly modular-based architecture that allows it to plug into other customized implementations and frameworks for user interfaces. Its components are imported through a package called java.swing.

4. **Manageable:** It is easy to manage and configure. Its mechanism and composition pattern allows changing the settings at run time as well. The uniform changes can be provided to the user interface without doing any changes to the application code.

5. **MVC:** They mainly follow the concept of MVC that is the Model View Controller. With the help of this, we can do the changes in one component without impacting or touching other components. It is known as loosely coupled architecture as well.

6. **Customizable:** Swing controls can be easily customized. It can be changed, and the visual appearance of the swing component application is independent of its internal representation.

I/O programming

Files

Files can be classified as either text or binary

- Human readable files are text files
- All other files are binary files

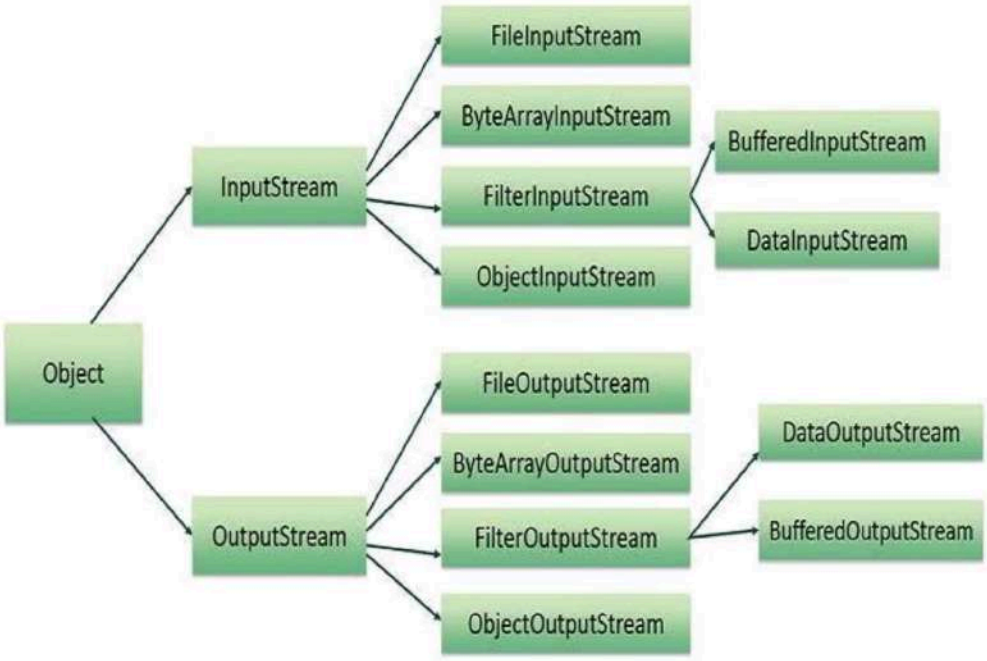
Java provides many classes for performing text I/O and binary I/O

Text I/O

- Use the Scanner class for reading text data from a file
- The JVM converts a file specific encoding when to Unicode when reading a character
- Use the PrintWriter class for writing text data to a file
- The JVM converts Unicode to a file specific encoding when writing a

Binary I/O

- Binary I/O does not involve encoding or decoding and thus is more efficient than text I/O
- Binary files are independent of the encoding scheme on the host machine



- The abstract InputStream is the root class for reading binary data

The abstract OutputStream is the root class for writing binary data

The FileInputStream class

- To construct a FileInputStream object, use the following constructors

public FileInputStream(String filename)

public FileInputStream(File file)

- A java.io. FileNotFoundException will occur if you attempt to create a FileInputStream with a nonexistent file

The FileOutputStream class

- To construct a FileOutputStream object, use the following constructors

public FileOutputStream(String filename)

public FileOutputStream(File file)

public FileOutputStream(String filename, boolean append)

public FileOutputStream(File file, boolean append)

- If the file does not exist, a new file will be created
- If the file already exists, the first two constructors will delete the current contents in the file
- To retain the current content and append new data into the file, use the last two constructors by passing true to the append parameter

Binary file I/O classes

- FileInputStream/FileOutputStream are for reading/writing bytes from/to files
- All the methods in FileInputStream and FileOutputStream are inherited from their superclasses

Binary file I/O using FileInputStream and FileOutputStream

```
public class TestFileStream
{
public static void main(String[] args) throws IOException
{
try
{
```

```
// Create an output stream to the file

FileOutputStream output = new FileOutputStream("temp.dat");

// Output values to the file

for (int i = 1; i <= 10; i++)

output.write(i);

}

try

{

// Create an input stream for the file

FileInputStream input = new FileInputStream("temp.dat");

{

// Read values from the file

int value;

while ((value = input.read()) != -1)

System.out.print(value + " ");

}
}
}
```

Java.io.InputStream

InputStream class is the superclass of all the io classes i.e. representing an input stream of bytes. It represents input stream of bytes. Applications that are defining subclass of InputStream must provide method, returning thenext byte of input.

A reset() method is invoked which re-positions the stream to the recently marked position.

public abstract class InputStream extends Object implements Closeable

Input stream classes

Input stream classes that are used to read 8 – bit bytes include a super class known as **InputStream** and a number of subclasses for supporting various input-related functions.

The super class **InputStream** is an abstract class, and therefore we cannot create instances of this class. Rather, we must use the subclasses that inherit from this class. The **InputStream** class defines **methods** for performing input functions such as

Reading bytes

Closing streams

Marking positions in streams

Skipping ahead in a stream

Finding the number of bytes in a stream

Method	**Description**
read (byte b [ ])	Reads a byte from the input streamReads an array of bytes into b
read (byte b [ ], int n, int m)	Reads m bytes into b starting from nth
byte available ( )	Gives number of bytes available in the input
skip (n)	Skips over n bytes from the input stream
reset ( )	Goes back to the beginning of the stream
close ( )	Closes the input

Note that the class **DataInputStream** extends **FilterInputStream** and implementsthe interface **DataInput**. Therefore, the **DataInputStream** class implements the methods described in **DataInput**in addition to using the methods of **InputStream** class. The **DataInput** interface contains the following methods:

* readShort ( )
* readFloat ( )
* reading ( )
* readUTF ( )
* readLong ( )
* readDouble ( )
* readLine ( )
* readChar ( )
* readBoolean ( )

Output stream classes

Output stream classes are derived from the base class OutputStream. Like InputStream, the OutputStream is an abstract class and therefore we cannot instantiate it. The several subclasses of the **OutputStream** can be used for performing the output operations.

The **OutputStream** includes methods that are designed to perform the following tasks.

Writing bytes

Closing streams

Flushing streams

Summary of OutputStream Methods	
Method	**Description**
write ( )	Writes a byte to the output stream
write (byte [ ] b)	Writes all bytes in the array b to the output stream
write (byte b[ ], int, int m)	Writes m bytes from array b starting from nth byte
close ( )	Closes the output stream
flush ( )	Flushes the output stream

The **DataOutputStream**, a counterpart of **DataInputStream**, implements the interface **DataOutput** and therefore implements the following methods contained in **DataOutput**interfaces.

Object input/output

The ObjectInputStream class of the java.io package can be used to read objects that werepreviously written by ObjectOutputStream . It extends the InputStream abstract class

Working of ObjectInputStream

The **ObjectInputStream** is mainly used to read data written by the **ObjectOutputStream**.

Basically, the **ObjectOutputStream** converts Java objects into corresponding streams. This known as serialization. Those converted streams can be stored in files or transferred through

Now, if we need to read those objects, we will use the**ObjectInputStream** that will convert streams back to corresponding objects. This is known as

Create an ObjectInputStream

In order to create an object input stream, we must import the package, here is how we can create an input stream.

```
java.io.
```

```
ObjectInputStream
```

In the above example, we have created an object input stream *objStrea* that is linked with namedthe file input stream named *fileStream*.

Now, the `objStream` can be used to read objects from the

```
// Creates a file input stream linked with the specified file
FileInputStream fileStream = new FileInputStream(String file);

// Creates an object input stream using the file input stream
ObjectInputStream objStream = new
```

ObjectOutputStream

An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream. The objects can be read (reconstituted) using an ObjectInputStream. Persistent storage of objects canbe accomplished by using a file for the stream.

- Only objects that support the java.io.Serializable interface can be written to streams. The class of each serializable object is encoded including the class name and signature of theclass, the values of the object's fields and arrays, and the closure of any other objects referenced from the initial objects.

- The Java ObjectOutputStream is often used together with a Java ObjectInputStream. The ObjectOutputStream is used to write the Java objects, and the ObjectInputStream is used to read the objects again.

Methods

- **void close() :** Closes the stream.This method must be called to release any resourcesassociated with the stream.

Syntax :public void close()
throws IOException

RandomAccessFile

This class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Declaration :

```
public class RandomAccessFile
  extends Object
    implements DataOutput, DataInput, Closeable
```

Methods of RandomAccessFile Class :

1. **read() : java.io.RandomAccessFile.read()** reads byte of data from file. The byte is returned as aninteger in the range 0-255

2. **read(byte[] b) java.io.RandomAccessFile.read(byte[] b)** reads bytes upto b.length from thebuffer.

Syntax :
public int read(byte[] b)

readChar() : java.io.RandomAccessFile.readChar() reads a character from the file, start readingfrom the File Pointer.