# Java important questions

1. Final, finally and finalize

## final

final is the keyword and access modifier which is used to apply restrictions on a class, method or variable
Final keyword is used with the classes, methods and variables.

Final method is executed only when we call it.
(1) Once declared, final variable becomes constant and cannot be modified.
(2) final method cannot be overridden by sub class.
(3) final class cannot be inherited.

### Finally

finally is the block in Java Exception Handling to execute the important code whether the exception occurs or not.
Finally block is always related to the try and catch block in exception handling.
1) finally block runs the important code even if exception occurs or not.
(2) finally block cleans up all the resources used in try block

Finally block is executed as soon as the try-catch block is executed.
It's execution is not dependant on the exception.

### Finalize

finalize is the method in Java which is used to perform clean up processing just before object is garbage collected.

finalize() method is used with the objects.
finalize method performs the cleaning activities with respect to the object before its destruction.
finalize method is executed just before the object is destroyed.

## 2. Import statement

**Import** statement in Java is helpful to take a class or all classes visible for a program specified under a package, with the help of a single statement. It is pretty beneficial as the programmer do not require to write the entire class definition. Hence, it improves the readability of the program.
**Syntax 1:**
import package1[.package2].(*);

Here,

- **package1:** Top-level package
- **package2:** Subordinate-level package under package1
- ***:** To import all the classes

**Syntax 2:**
import package1[.package2].(myClass);

Here,

- **package1:** Top-level package
- **package2:** Subordinate-level package under the top-level package
- **myClass:** Import only myClass

## 3. throw and throws in java

### throw keyword

Java throw keyword is used throw an exception explicitly in the code, inside the function or the block of code.
The throw keyword is followed by an instance of Exception to be thrown.
We are allowed to throw only one exception at a time i.e. we cannot throw multiple exceptions.
throw is used within the method

```
public static void checkNum(int num) {
    if (num < 1)
     {
        throw new ArithmeticException("\nNumber is negative, cannot calculate square");
     }
    else {
        System.out.println("Square of " + num + " is " + (num*num));
     }
  }
```

### throws

Java throws keyword is used in the method signature to declare an exception which might be thrown by the function while the execution of the code.
The throws keyword is followed by class names of Exceptions to be thrown.
throws is used with the method signature.
We can declare multiple exceptions using throws keyword that can be thrown by the method. For example, main() throws IOException, SQLException.

```
public class TestThrows
{
   public static int divideNum(int m, int n) throws ArithmeticException
  {
      int div = m / n;
      return div;
   }
}
```

**5 methods of Graphics class in Java**

We can divide Graphics class methods into three categories: draw, fill, and miscellaneous. Draw methods are used to draw lines, curves, and outer boundaries of closed curves and images. Fill methods fill the interior area of graphics objects. There are also a few miscellaneous methods that fall in neither categoryfor example, MeasureString and Clear.

Draw Methods
The draw methods of the Graphics class are used to draw lines, curves, and outer boundaries of closed curves and images. Table 3.2 lists the draw methods of the Graphics class.

| Method | Description |
|---|---|
| DrawLine | draws a line beween two points specified by a pair of coordinates. DrawLines draws a se of lines using an array of points. |
| DrawArc | Draws an arc (a portion of an ellipse specified by a pair of coordinates, a width, a height, start and end angles). |
| DrawCurve | Draws a cardinal spline through a specified array of Point structures. |
| DrawEllipse | Draws an ellipse defined by a bounding rectangle specified by a pair of coordinates, a heig and a width. |
| DrawLine | Draws a line connecting two points specified by coordinate pairs. |
| DrawLines | Draws a series of line segments that connect an array of Point structures. |
| DrawPolygo | Draws a polygon defined by an array of Point structures. |
| DrawRectan | Draws a rectangle specified by a coordinate pair, a width, and a height. |
| DrawString | Draws the specified text string at the specified location using specified Brush and Font objects. |

## 6. Wrapper class in Java

The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive*.

The automatic conversion of primitive into an object is known as autoboxing and vice-versa unboxing.

| Primitive Type | Wrapper class |
|---|---|
| boolean | Boolean |
| char | Character |

| | |
|---|---|
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

Example:

```
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
```

## 7. What is Unicode

The Unicode Standard is the universal character representation standard for text in computer processing. Unicode provides a consistent way of encoding multilingual plain text making it easier to exchange text files internationally.

ASCII defines 128 characters, which map to the numbers 0–127. Unicode defines (less than) $2^{21}$ characters, which, similarly, map to numbers **0–$2^{21}$**

The Unicode Standard provides a unique number for every character, no matter what platform, device, application or language. It has been adopted by all modern software providers and now allows data to be transported through many different platforms, devices and applications without corruption.

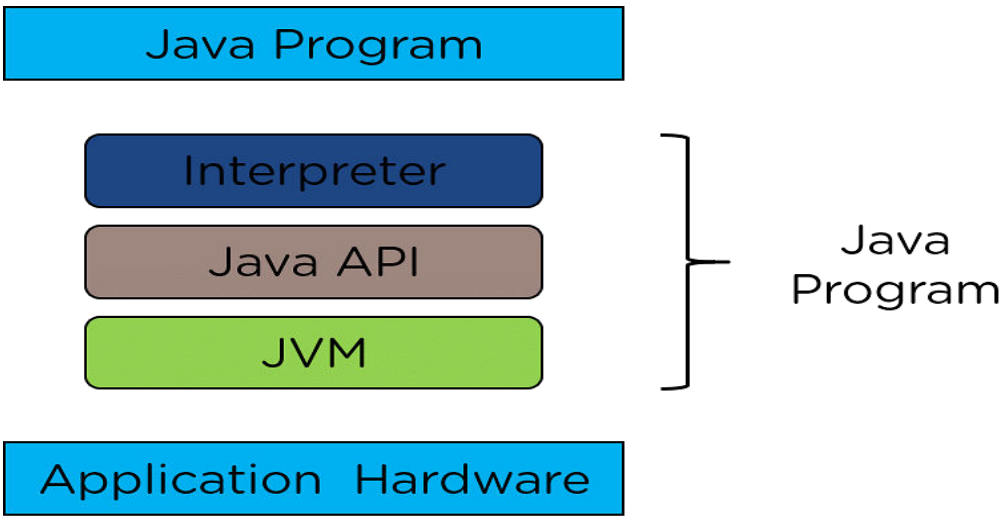
The Unicode Standard defines code points (unique numbers) for characters used in the major languages written today. This includes punctuation marks, diacritics, mathematical symbols, technical symbols, arrows, dingbats, and so on. In all, the Unicode Standard provides codes for over 100,000 characters from the world's alphabets, ideograph sets, and symbol collections, including classical and historical texts of many written languages. The characters can be represented in different encoding forms, such as UTF-8 and UTF-16.

## 8. What are API?

Java application programming interfaces (APIs) **are predefined software tools that easily enable interactivity between multiple applications.**

APIs are important software components bundled with the JDK. APIs in Java include classes, interfaces, and user Interfaces. They enable developers to integrate various applications and websites and offer real-time information.

The following image depicts the fundamental components of the Java API.




## 9. what is an Interface in Java?

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve* abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.

- Interfaces can have abstract methods and variables. It cannot have a method body.
- It cannot be instantiated just like the abstract class.

**Why use Java interface?**

There are mainly three reasons to use interface. They are given below.

- It is used to achieve abstraction.
- By interface, we can support the functionality of multiple inheritance.

```
interface printable
{
   void print();
}
class A6 implements printable
{
      public void print()
     {
```

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

## 10. What is an Abstract class?

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.

Points to Remember

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

## Abstract Method in Java

A method which is declared as abstract and does not have implementation is known as an abstract method.

**Example of abstract method**

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

Example :

```
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();
```

```
 }
}
```

## 11. 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.

| 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 |

## 12. Layout Managers 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.

Actually, layout managers are used to arrange the components in a specific manner. It is an interface that is implemented by all the classes of layout managers.

The Abstract Windowing Toolkit (AWT) has the following five layout managers:

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout

| Sl. No. | LayoutManager & Description |
|---|---|
| 1 | **BorderLayout** - The borderlayout arranges the components to fit in the five regions: east, west, north, south and centre. |
| 2 | **CardLayout** - The CardLayout object treats each component in the container as a card. Only one card is visible at a time. |
| 3 | **FlowLayout** - The FlowLayout is the default layout. It layouts the components in a directional flow. |
| 4 | **GridLayout** - The GridLayout manages the components in form of a rectangular grid. |
| 5 | **GridBagLayout** - This is the most flexible layout manager class. The object of GridBagLayout aligns the component vertically, horizontally or along their baseline without requiring the components of same size. |

## 13. Operators in Java

### Simple Assignment Operator

= Simple assignment operator

### Arithmetic Operators

+ Additive operator (also used for String concatenation)
- Subtraction operator
* Multiplication operator
/ Division operator
% Remainder operator

### Unary Operators

+ Unary plus operator; indicates positive value (numbers are

positive without this, however)
- Unary minus operator; negates an expression
++ Increment operator; increments a value by 1
-- Decrement operator; decrements a value by 1
! Logical complement operator; inverts the value of a boolean

**Equality and Relational Operators**

== Equal to
!= Not equal to
> Greater than
>= Greater than or equal to
< Less than
<= Less than or equal to

**Conditional Operators**

&& Conditional-AND
|| Conditional-OR
?: Ternary (shorthand for if-then-else statement)

**Type Comparison Operator**

instanceof Compares an object to a specified type

**Bitwise and Bit Shift Operators**

~ Unary bitwise complement
<< Signed left shift
>> Signed right shift
>>> Unsigned right shift
& Bitwise AND
^ Bitwise exclusive OR
| Bitwise inclusive OR

## 14. **Control statements in Java**

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java

provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.
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

**Java for-each loop**

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
for(data_type var : array_name/collection_name)
{
//statements
}
```

Consider the following example to understand the functioning of the for-each loop in Java.

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

**15. super keyword in java**

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

Whenever you create the instance of subclass, 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.

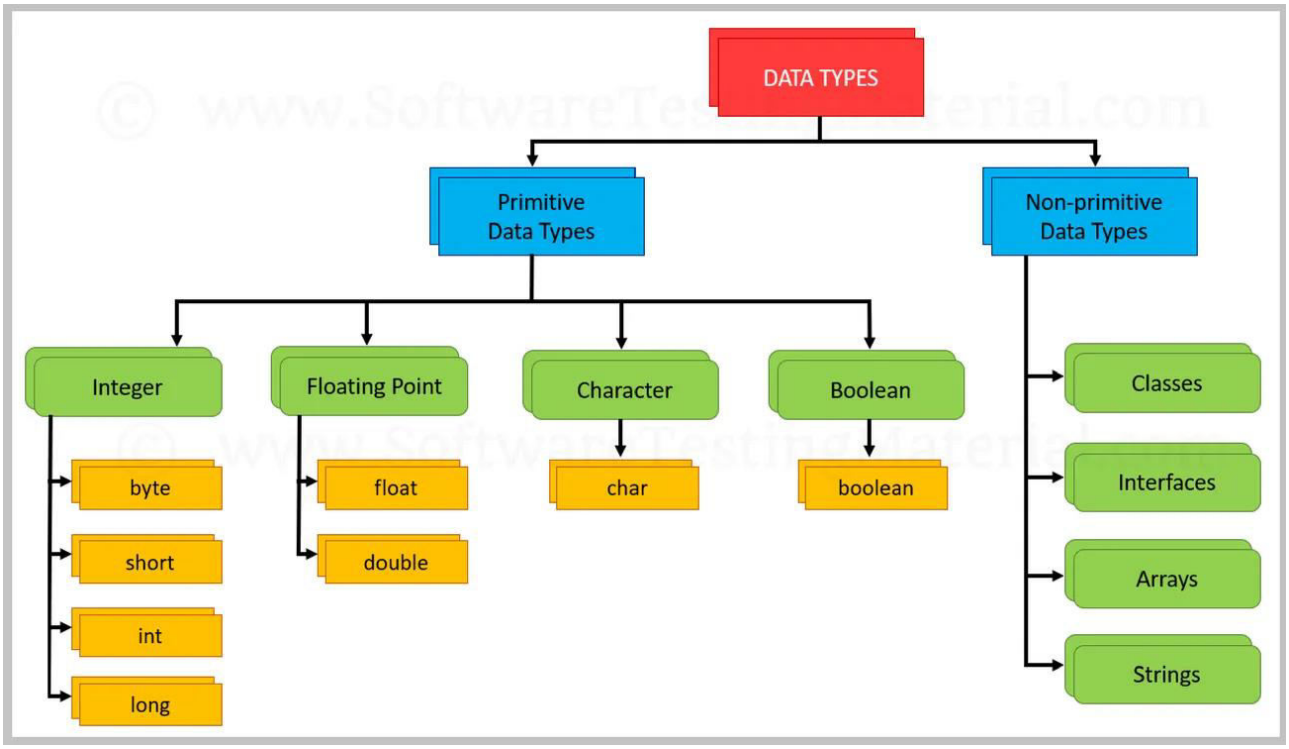
```
class Animal
{
void eat()
{System.out.println("eating...");}
}
class Dog extends Animal
{
  void eat()
  {System.out.println("eating bread...");
   }
void bark()
{System.out.println("barking...");
}
void work()
{
super.eat();
bark();
}
}
class TestSuper2
{
public static void main(String args[])
{
Dog d=new Dog();
d.work();
}
}
```

**16. Static binding and Dynamic binding (Static polymorphism and Dynamic polymorphism)**

| BASIS FOR COMPARISON | STATIC BINDING | DYNAMIC BINDING |
|---|---|---|
| Event Occurrence | Events occur at compile time are "Static Binding". | Events occur at run time are "Dynamic Binding". |
| Information | All information needed to call a function is known at compile time. | All information need to call a function come to know at run time. |
| Advantage | Efficiency. | Flexibility. |
| Time | Fast execution. | Slow execution. |

| BASIS FOR COMPARISON | STATIC BINDING | DYNAMIC BINDING |
|---|---|---|
| Alternate name | Early Binding. | Late Binding. |
| Example | Overloaded function call, overloaded operators. | Virtual function in C++, overridden methods in java. |

**17. Data types in Java**




## Primitive Data Type:

There are 8 primitive data types such as byte, short, int, long, float, double, char, and boolean. Size of these 8 primitive data types wont change from one OS to other.

byte, short, int & long – stores whole numbers
float, double – stores fractional numbers
char – stores characters
boolean – stores true or false

| Data Type | Default size |
|---|---|
| boolean | 1 bit |
| char | 2 byte |

| | |
|---|---|
| byte | 1 byte |
| short | 2 byte |
| int | 4 byte |
| long | 8 byte |
| float | 4 byte |
| double | 8 byte |

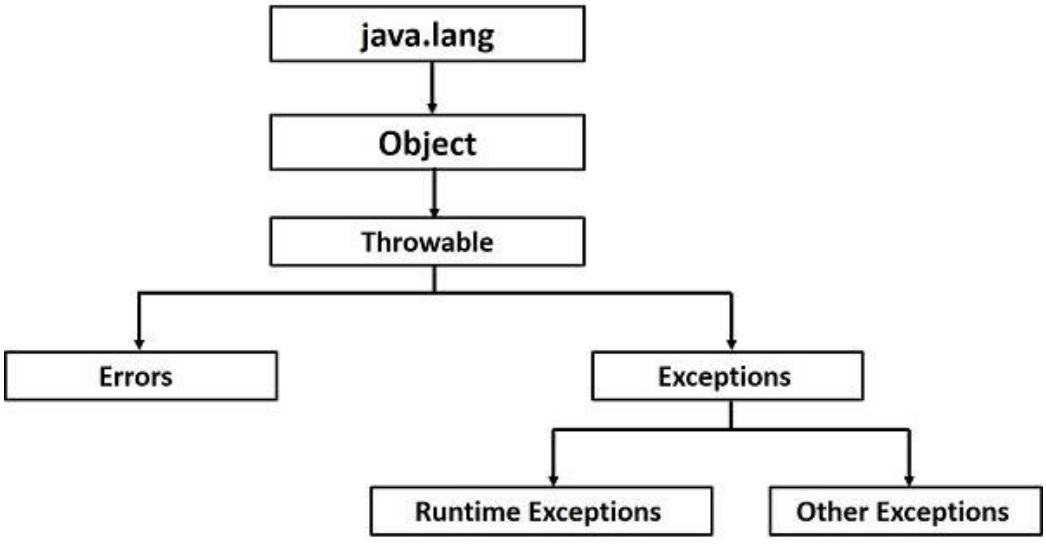
## 18. Exception class in Java

### Exception Hierarchy

All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.

Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.

The Exception class has two main subclasses: IOException class and RuntimeException Class.



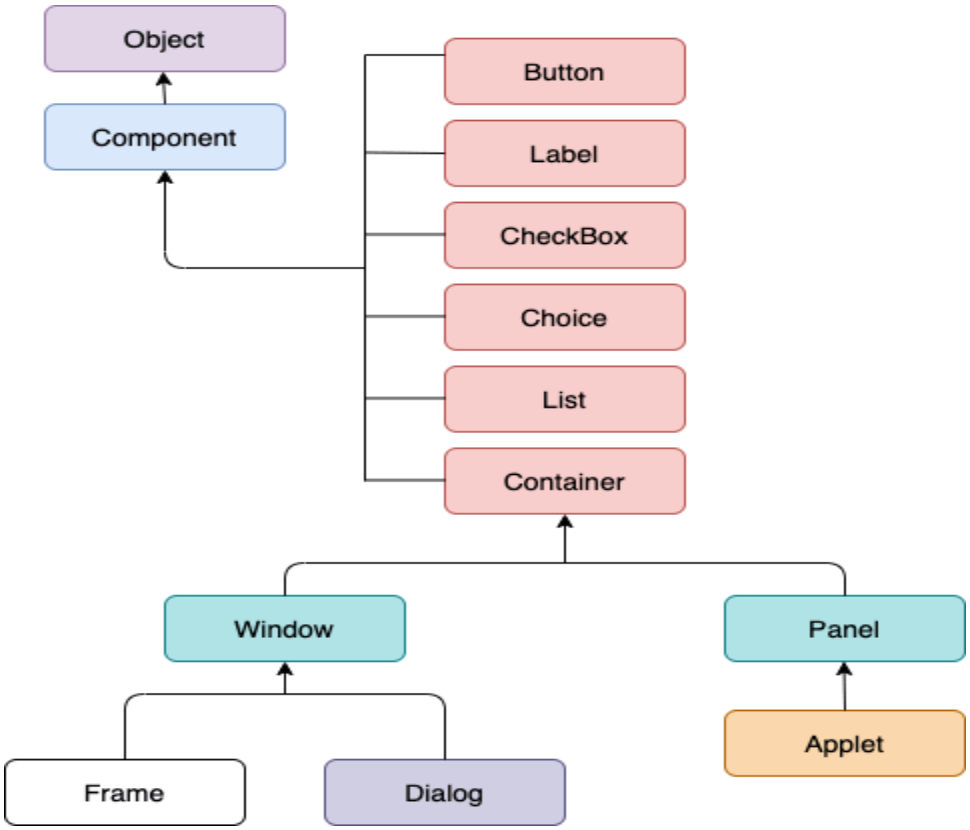

## 19. AWT in Java

### Abstract Window Toolkit

Java AWT (Abstract Window Toolkit) is **an API to develop GUI or window-based applications in java**. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

AWT provides various components like button, label, checkbox, etc. used as objects inside a Java Program. AWT components use the resources of the operating system, i.e., they are platform-dependent, which means, component's view can be changed according to the view of the operating system. The classes for AWT are provided by the Java.awt package for various AWT components.

The following image represents the hierarchy for Java AWT.




**20 . SWING in java**

**Java Swing tutorial** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Swing in java is part of Java foundation class which is lightweight and platform independent. It is **used for creating window based applications**. It includes components like button, scroll bar, text field etc. Putting together all these components makes a graphical user interface.

| No. | Java AWT | Java Swing |
|---|---|---|

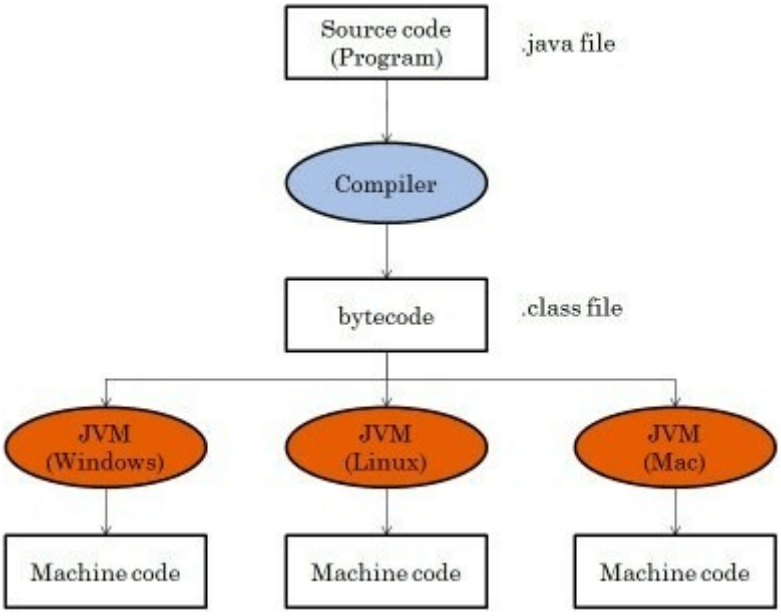
| | | |
|---|---|---|
| 1) | AWT components are **platform-dependent**. | Java swing components are **platform-independent**. |
| 2) | AWT components are **heavyweight**. | Swing components are **lightweight**. |
| 3) | AWT **doesn't support pluggable look and feel**. | Swing **supports pluggable look and feel**. |
| 4) | AWT provides **less components** than Swing. | Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |

### 21 paint and repaint method in Java

The repaint() method is used **to call the update() method internally that calls the paint() method to repaint the component**. The paint() and repaint() both are used to paint a component, but the repaint() method internally calls paint() to paint the component. We cannot override the repaint() method.

### 22 What is bytecode?

Bytecode is **the intermediate representation of a Java program, allowing a JVM to translate a program into machine-level assembly instructions**. When a Java program is compiled, bytecode is generated in the form of a . class file. This . class file contains non-runnable instructions and relies on a JVM to be interpreted.




### 23. Event classes and Event listers in Java

An event listener in Java is designed to process some kind of event — it "listens" for an event, such as a user's mouse click or a key press, and then it responds accordingly. An event listener must be connected to an event object that defines the event.

### Event and Listener (Java Event Handling)

Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling.

**Java Event classes and Listener interfaces**

| Event Classes | Listener Interfaces |
|---|---|
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

**Steps to perform Event Handling**

Following steps are required to perform event handling:

1. Register the component with the Listener

**Registration Methods**

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button -** public void addActionListener(ActionListener a){}
- **MenuItem -** public void addActionListener(ActionListener a){}
- **TextField -** public void addActionListener(ActionListener a){}
  ` public void addTextListener(TextListener a){}
- **TextArea -** public void addTextListener(TextListener a){}
- **Checkbox -** public void addItemListener(ItemListener a){}
- **Choice -** public void addItemListener(ItemListener a){}

**23. Applet in Java**

An applet is **a Java program that can be embedded into a web page**. It runs inside the web browser and works at client side. An applet is embedded in an HTML page using the APPLET or OBJECT tag and hosted on a web server. Applets are used to make the website more dynamic and entertaining.

**A special type of Java program that runs in a Web browser** is referred to as Applet. It has less response time because it works on the client-side. It is much secured executed by the browser under any of the platforms such as Windows, Linux and Mac OS etc.

two types of Applets

A web page can contain **two** types of applets: Local applet. Remote applet.

What is difference between applet and application?

**Applications are just like a Java programs that can be execute independently without using the web browser. Applets are small Java programs that are designed to be included with the HTML web document**. They require a Java-enabled web browser for execution. Application program requires a main function for its execution.

**Methods of Applet**

It provides 4 life cycle methods of applet.

- public void init(): is used to initialized the Applet. It is invoked only once.
- public void start(): is invoked after the init() method or browser is maximized. ...
- public void stop(): is used to stop the Applet. ...
- public void destroy(): is used to destroy the Applet.

**HTML <applet> tag**

HTML <applet> tag was used to embed the Java applet in an HTML document.

Syntax:-

<applet code="URL" height="200" width="100">.....</applet>

```
<!DOCTYPE html>
<html>
<head>
  <title>Applet Tag</title>
 </head>
 <body>
   <p>Example of Applet Tag</p>
    <applet code="Shapes.class" align="right" height="200" width="300">
     <b>Sorry! you need Java to see this</b>
     </applet>
  </body>
</html>
```

**24. What is a constructor in Java**

A constructor in Java is **a special method that is used to initialize objects**. The constructor is called when an object of a 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.

**Rules for creating Java constructor**

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

What are the types of constructor?

There are 3 types of constructors

In Java, constructors can be divided into 2 types:

1. Default Constructor (No-Arg Constructor)
2. Parameterized Constructor

```
public class Hello
 {
    String name;
  Hello() //constructor
  {
      this.name = "BeginnersBook.com";
  }
  public static void main(String[] args)
  {
    Hello obj = new Hello();
    System.out.println(obj.name);
  }
}
```

**25 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.

**26. Dfferentiate abstract classes and interface in java**

**What is an Abstract Class?**

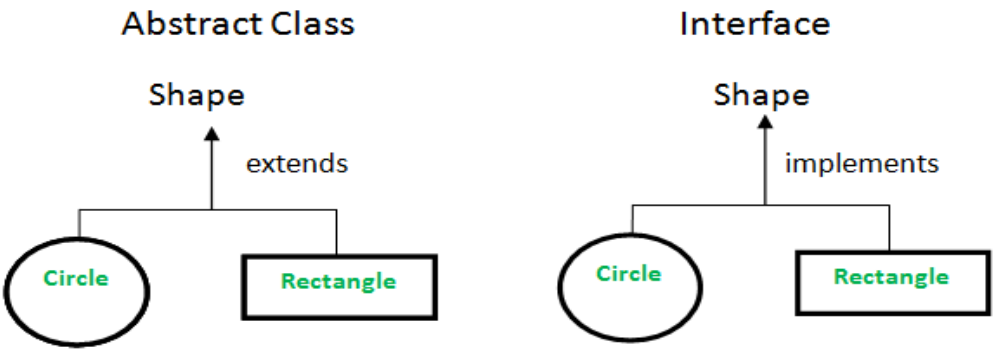
A class that contains an abstract keyword on the declaration is known as an abstract class. It is necessary for an abstract class to have at least one abstract method. It is possible in an abstract class to contain multiple concrete methods.

**What is an Interface?**

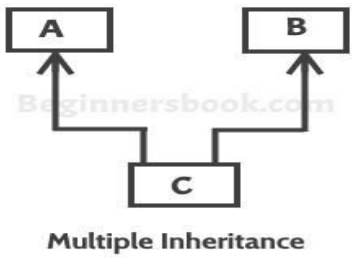
An interface is a sketch that is useful to implement a class. The methods used in the interface are all abstract methods. The interface does not have any concrete method.

Difference between Abstract Class and Interface in Java

| S.No. | Abstract Class | Interface |
|---|---|---|
| 1. | An abstract class can contain both abstract and non-abstract methods. | Interface contains only abstract methods. |
| 2. | An abstract class can have all four; static, non-static and final, non-final variables. | Only final and static variables are used. |
| 3. | To declare abstract class abstract keywords are used. | The interface can be declared with the interface keyword. |
| 4. | It supports multiple inheritance. | It does not support multiple inheritance. |
| 5. | The keyword 'extend' is used to extend an abstract class | The keyword implement is used to implement the interface. |
| 6. | It has class members like private and protected, etc. | It has class members public by default. |

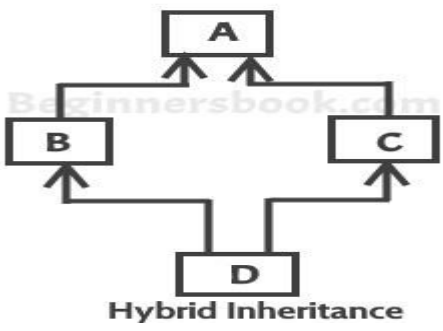



**27. Multiple inheritance in Java**




When **one class extends more than one classes** then this is called multiple inheritance. For example: Class C extends class A and B then this type of inheritance is known as multiple inheritance. Java doesn't allow multiple inheritance.

**Problems with multiple inheritance**



Now we create a new child class D, which extends both class B and class C. Note that we have multiple inheritance (D extends B and C), hierarchical inheritance (B and C extend A), and multilevel inheritance (D extends A, B, and C).

In the **diamond problem**, child classes B and C inherit a method from parent class A. Both B and C override the inherited method. But the new methods in B and C conflict with each other.

Ultimate child class D inherits the two independent and conflicting methods from its multiple parents B and C. It is unclear which method class D should use, so there is ambiguity. Other OOP programming languages implement various methods for addressing the multiple inheritance ambiguity.

**How it is solved?**

The solution to the problem is interfaces. The only way to implement multiple inheritance is to implement multiple interfaces in a class. In java, one class can implements two or more interfaces. This also does not cause any ambiguity because all methods declared in interfaces are implemented in class.\

**28. CLASSPATH in java**

Click on **Advanced System Settings**.-> A dialog box will open. Click on **Environment Variables. ->**

Next step - If the CLASSPATH already exists in System Variables, click on the Edit button then put a semicolon (;) at the end. Paste the Path of MySQL-Connector Java.jar file.

If the CLASSPATH doesn't exist in System Variables, then click on the New button and type Variable name as CLASSPATH and Variable value as *C:\Program Files\Java\jre1.8\MySQL-Connector Java.jar;.;*

CLASSPATH: CLASSPATH is an environment variable which is used by Application ClassLoader to locate and load the **.class** files. The CLASSPATH defines the path, to find third-party and user-defined classes that are not extensions or part of Java platform.

## Difference between PATH and CLASSPATH

| **PATH** | **CLASSPATH** |
|---|---|
| PATH is an environment variable. | CLASSPATH is also an environment variable. |
| It is used by the operating system to find the executable files (.exe). | It is used by Application ClassLoader to locate the .class file. |
| You are required to include the directory which contains .exe | You are required to include all the directories which contain .class and JAR files. |

| | |
|---|---|
| files. | |
| PATH environment variable once set, cannot be overridden. | The CLASSPATH environment variable can be overridden by using the command line option -cp or -CLASSPATH to both javac and java command. |

**29. What is JDBC in Java?**

Java database connectivity (JDBC) is the JavaSoft specification of a standard application programming interface (API) that allows Java programs to access database management systems. The JDBC API consists of a set of interfaces and classes written in the Java programming language.

**JDBC driver**

A JDBC driver is a software component enabling a Java application to interact with a database. JDBC drivers are analogous to ODBC drivers, ADO.NET data providers, and OLE DB providers. To connect with individual databases, JDBC requires drivers for each database.

**What Are the Types of JDBC Drivers?**

- Type 1: JDBC-ODBC bridge.
- Type 2: partial Java driver.
- Type 3: pure Java driver for database middleware.
- Type 4: pure Java driver for direct-to-database.

What is JDBC and ODBC driver?

ODBC is an SQL-based Application Programming Interface (API) created by Microsoft that is used by Windows software applications to access databases via SQL. JDBC is an SQL-based API created by Sun Microsystems to enable Java applications to use SQL for database access.

**30 Stream classes in java**

The Stream class defines objects which accepts a sequence of characters. Streams may also have an output in which case multiple stream objects can be cascaded to build a stream pipe where the output of a stream is directed into the input of the next stream object "down the line".

# Java IO Stream

Java performs I/O through Streams. A Stream is linked to a physical layer by java I/O system to make input and output operation in java. In general, a stream means continuous flow of data. Streams are clean way to deal with input/output without having every part of your code understand the physical.

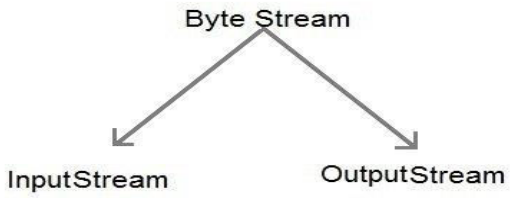
Java encapsulates Stream under java.io package. Java defines two types of streams. They are,

**Byte Stream** : It provides a convenient means for handling input and output of byte.

**Character Stream** : It provides a convenient means for handling input and output of characters. Character stream uses Unicode and therefore can be internationalized.

**Java Byte Stream Classes**

Byte stream is defined by using two abstract class at the top of hierarchy, they are InputStream and OutputStream.




These two abstract classes have several concrete classes that handle various devices such as disk files, network connection etc.

1. ***Some important Byte stream classes.***

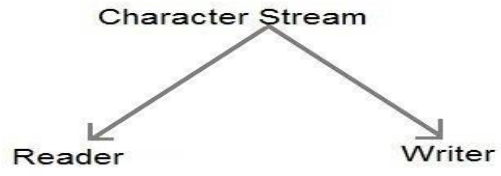
| Stream class | Description |
|---|---|
| **BufferedInputStream** | Used for Buffered Input Stream. |
| **BufferedOutputStream** | Used for Buffered Output Stream. |
| **DataInputStream** | Contains method for reading java standard datatype |
| **DataOutputStream** | An output stream that contain method for writing java standard data type |
| **FileInputStream** | Input stream that reads from a file |
| **FileOutputStream** | Output stream that write to a file. |
| **InputStream** | Abstract class that describe stream input. |
| **OutputStream** | Abstract class that describe stream output. |
| **PrintStream** | Output Stream that contain print() and println() method |

These classes define several key methods. Two most important are

1. read() : reads byte of data.
2. write() : Writes byte of data.

**Java Character Stream Classes**

Character stream is also defined by using two abstract class at the top of hierarchy, they are Reader and Writer.



These two abstract classes have several concrete classes that handle unicode character.

Some important Charcter stream classes

| Stream class | Description |
|---|---|
| **BufferedReader** | Handles buffered input stream. |
| **BufferedWriter** | Handles buffered output stream. |
| **FileReader** | Input stream that reads from file. |
| **FileWriter** | Output stream that writes to file. |
| **InputStreamReader** | Input stream that translate byte to character |
| **OutputStreamReader** | Output stream that translate character to byte. |
| **PrintWriter** | Output Stream that contain print() and println() method. |
| **Reader** | Abstract class that define character stream input |
| **Writer** | Abstract class that define character stream output |

## 31. Features of OOPS

The main aim of object-oriented programming is to implement real-world entities, for example, object, classes, abstraction, inheritance, polymorphism, etc.

**Object** means a real-world entity such as a pen, chair, table, computer, watch, etc. **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation


### Object

Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of

response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

**Class**

**Collection of objects** is called class. It is a logical entity.

A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

**Inheritance**

*When one object acquires all the properties and behaviours of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

**Polymorphism**


If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.
Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

**Abstraction**

**Hiding internal details and showing functionality** is known as abstraction. For example phone call, we don't know the internal processing.
In Java, we use abstract class and interface to achieve abstraction.

**Encapsulation**


*Binding (or wrapping) code and data together into a single unit are known as encapsulation*. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

**32. Overriding and overloading in Java**

**Overloading occurs when two or more methods in one class have the same method name but different parameters.**

There are two ways to overload the method in java

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

```
class Adder
{
  static int add(int a,int b)
  {
    return a+b;
  }
  static int add(int a,int b,int c)
  {
     return a+b+c;
   }
}
class TestOverloading1
{
 public static void main(String[] args)
 {
  System.out.println(Adder.add(11,11));
  System.out.println(Adder.add(11,11,11));
  }
}
```

**Overriding occurs when two methods have the same method name and parameters**. One of the methods is in the parent class, and the other is in the child class.

**Rules for Java Method Overriding**

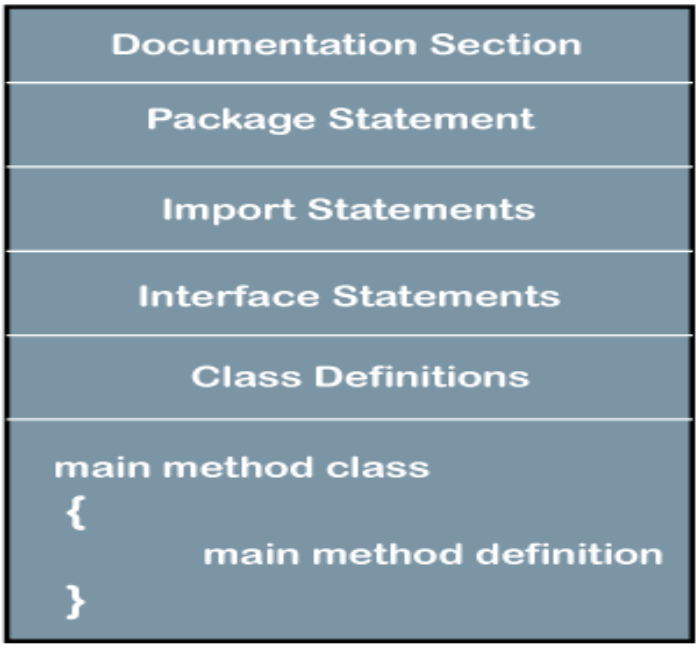
1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.

```
class Vehicle
{
 //defining a method
 void run()
    {
      System.out.println("Vehicle is running");
     }
}
//Creating a child class
class Bike2 extends Vehicle
 {
     //defining the same method as in the parent class
     void run()
      {
       System.out.println("Bike is running safely");
       }
```

```
 public static void main(String args[])
{
   Bike2 obj = new Bike2();//creating object
   obj.run();//calling method
 }
}
```

33. Structure of Java program

Java is an object-oriented programming, **platform-independent,** and **secure** programming language that makes it popular. Using the Java programming language, we can develop a wide variety of applications. So, before diving in depth, it is necessary to understand the **basic structure of Java program** in detail. In this section, we have discussed the basic **structure of a Java program**. At the end of this section, you will able to develop the Hello world Java program, easily.




Structure of Java Program

34. Exception handling in Java

## Exception handling in java with examples

Exception handling is one of the most important feature of java programming that allows us to handle the runtime errors caused by exceptions.

**What is an exception?**

An Exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs program execution gets terminated. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled in Java. By handling

the exceptions we can provide a meaningful message to the user about the issue rather than a system generated message, which may not be understandable to a user.

### Why an exception occurs?

There can be several reasons that can cause a program to throw exception. For example: Opening a non-existing file in your program, Network connection problem, bad input data provided by user etc.

### Exception Handling

If an exception occurs, which has not been handled by programmer then program execution gets terminated and a system generated error message is shown to the user. For example look at the system generated exception below:

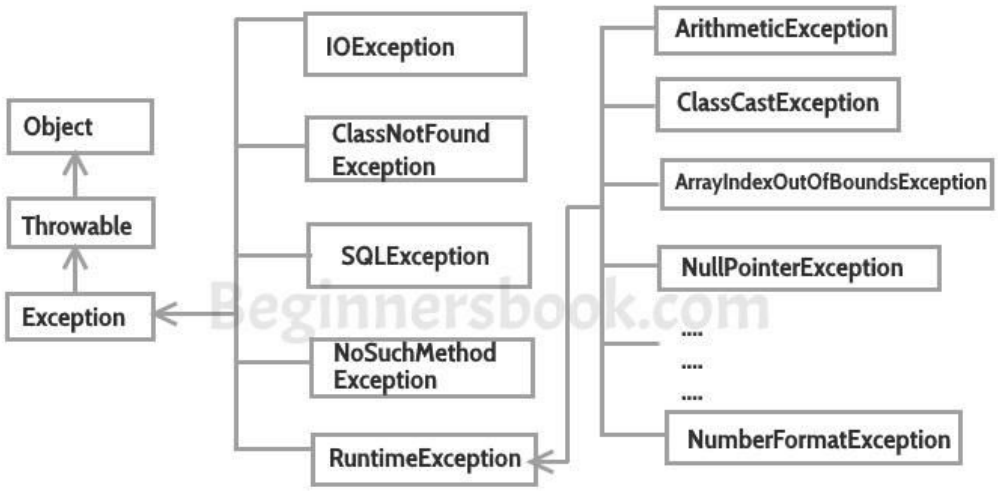
## Advantage of exception handling

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly. By handling we make sure that all the statements execute and the flow of program doesn't break.

### Difference between error and exception

**Errors** indicate that something severe enough has gone wrong, the application should crash rather than try to handle the error.
**Exceptions** are events that occurs in the code. A programmer can handle such conditions and take necessary corrective actions. Few examples:

- **NullPointerException** – When you try to use a reference that points to null.
- **ArithmeticException** – When bad data is provided by user, for example, when you try to divide a number by zero this exception occurs because dividing a number by zero is undefined.
- **ArrayIndexOutOfBoundsException** – When you try to access the elements of an array out of its bounds, for example array size is 5 (which means it has five elements) and you are trying to access the 10th element.




**35. Life cycle of Thread**
**36. Life cycle of Applet**

**37. Arrays in Java**
An array is a collection of similar types of data. We can create an array of the string type that can store 100 names.
String[] array = new String[100];

**There are two types of arrays in Java they are –**

**Single dimensional array** − A single dimensional array of Java is a normal array where, the array contains sequential elements (of same type) −

int[] myArray = {10, 20, 30, 40}

**Printing Array**

```
int myArray[]={33,3,4,5};
for(int i:myArray)
    System.out.println(i);
```

**Multi-dimensional array** − A multi-dimensional array in Java is an array of arrays. A two dimensional array is an array of one dimensional arrays and a three dimensional array is an array of two-dimensional arrays.

Int [][] mArray={{1,2},{2,3},{3,4}};

**Printing Array**

```
for(int i=0;i<3;i++)
{
 for(int j=0;j<3;j++)
   {
     System.out.print(mArray[i][j]+" ");
    }
}
```

## 38. JDK, JRE & JVM

**1. JDK** (Java Development Kit) is a Kit that provides the environment to **develop and execute(run)** the Java program. JDK is a kit(or package) that includes two things

- Development Tools(to provide an environment to develop your java programs)
- JRE (to execute your java program).

**2. JRE** (Java Runtime Environment) is an installation package that provides an environment to **only run(not develop)** the java program(or application)onto your machine. JRE is only used by those who only want to run Java programs that are end-users of your system.

**3. JVM (Java Virtual Machine)** is a very important part of both JDK and JRE because it is contained or inbuilt in both. Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program line by line, hence it is also known as an ***interpreter*.**