INDEX

INTRODUCTION

Java is a high-level programming language originally developed by Sun Microsystems and released in 1995. Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. This tutorial gives a complete understanding of Java.Java is an object oriented multithreaded programming language. It is designed to be small, simple and portable across different platforms as well as OS.

WHAT IS JAVA?

It is an object-oriented language similar to C++, but with advanced and simplified features. Java is **free to access** and can **run** on **all platforms**.

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

WHERE IT IS USED?

According to Sun, 3 billion devices run 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:

1) **Standalone Application**:

It is also known as Desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

2) **Web Application**:

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

3) **Enterprise Application:**

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

4) **Mobile Application:**

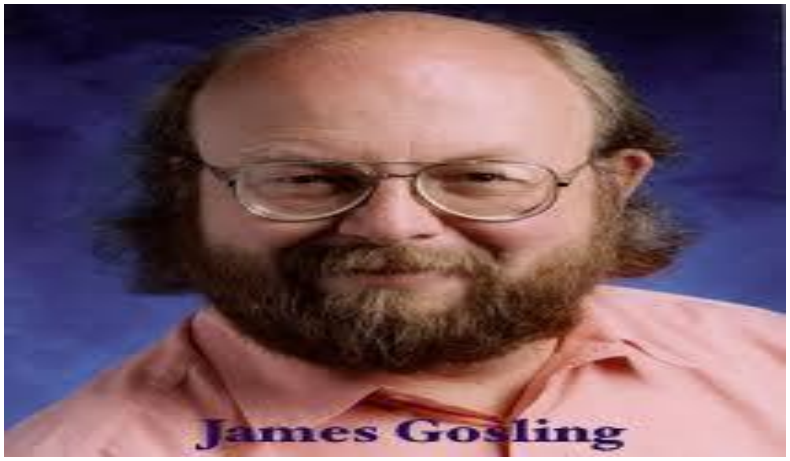
An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

HISTORY OF JAVA

Java history is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major point that describes the history of java.



1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

WHY OAK NAME FOR JAVA LANGUAGE?

5) **Why Oak?** Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java.

8) Java is an island of Indonesia where first coffee was produced (called java coffee).

9) Notice that Java is just a name not an acronym.

10) Originally developed by James Gosling at Sun Microsystems (which is now a subsidiary of Oracle Corporation) and released in 1995.

11) In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.

12) JDK 1.0 released in (January 23, 1996).

JAVA VERSION HISTORY:

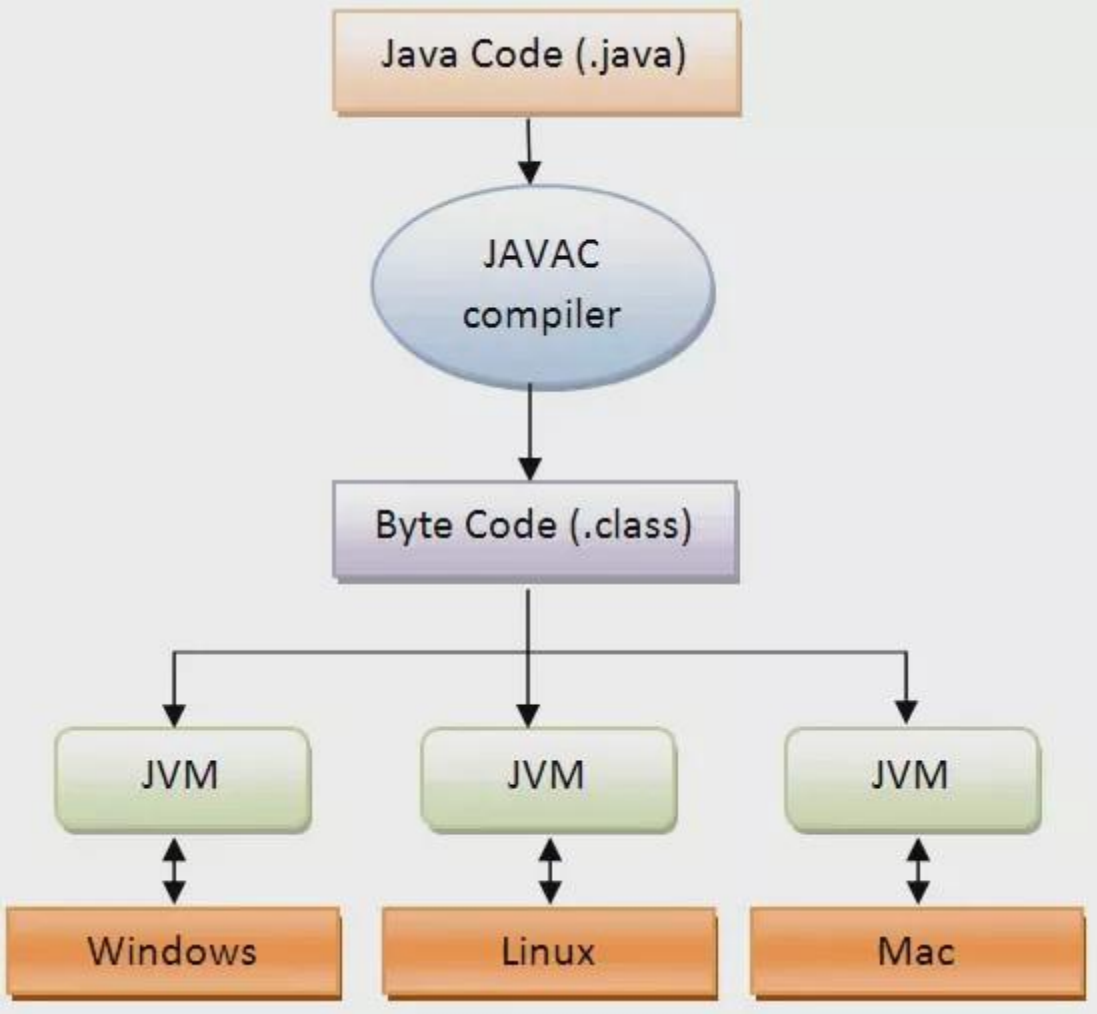
There a re many java versions that has been released. Current stable release of Java is Java SE 8.

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan, 1996)
- JDK 1.1 (19th Feb, 1997)
- J2SE 1.2 (8th Dec, 1998)
- J2SE 1.3 (8th May, 2000)
- J2SE 1.4 (6th Feb, 2002)
- J2SE 5.0 (30th Sep, 2004)
- Java SE 6 (11th Dec, 2006)
- Java SE 7 (28th July, 2011)
- Java SE 8 (18th March, 2014)

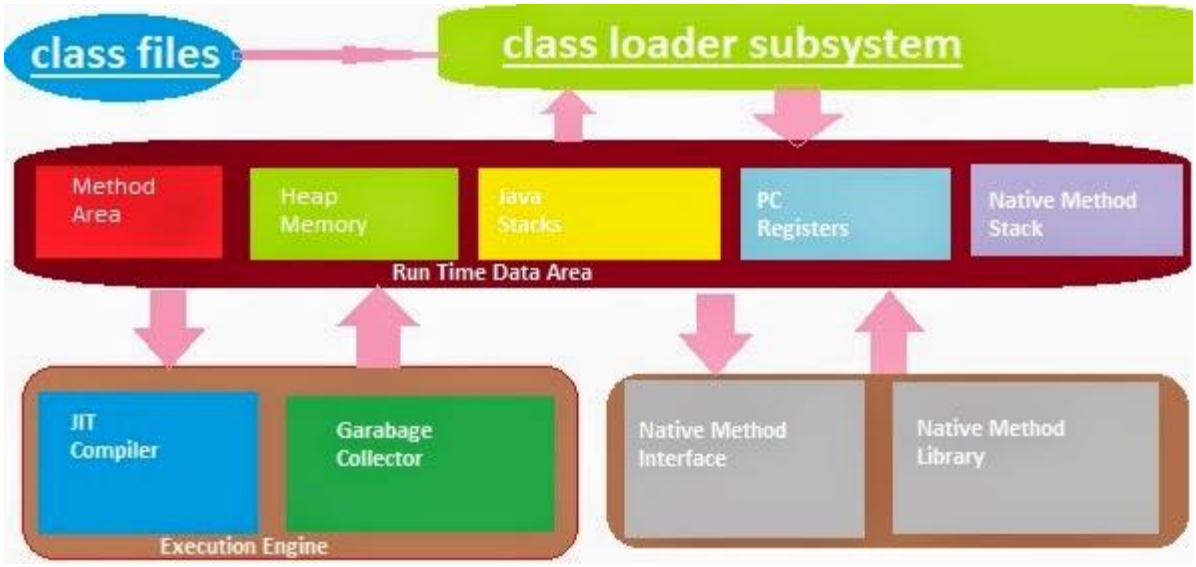
WHAT IS JVM:

Java Virtual Machine (JVM) is a virtual machine that resides in the real machine (your computer) and the **machine language for JVM is byte code**. This makes it easier for compiler as it has to generate byte code for JVM rather than different machine code for each type of machine. JVM executes the byte code generated by compiler and produce output. **JVM is the one that makes java platform independent**.

JVMs are available for many hardware and software platforms. The use of the same bytecode for all JVMs on all platforms allows Java to be described as a write once, run anywhere programming language, versus write once, compile anywhere, which describes cross-platform compiled languages.



JVM ARCHITECTURE:



Lets see how JVM works:

Class Loader: The class loader reads the .class file and save the byte code in the **method area**.

Method Area: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each .class file.

Heap: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each .class file.

Stack: Stcnm,ack is a also a part of JVM memory but unlike Heap, it is used for storing temporary variables.

PC Registers: This keeps the track of which instruction has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.

Native Method stack: A native method can access the runtime data areas of the virtual machine.

Native Method interface: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.

Garbage collection: A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.

Simple Java Program:

```
public class FirstJavaProgram {

  public static void main(String[] args){

    System.out.println("This is my first program in java");

  }//End of main

}//End of FirstJavaProgram Class
```

Output: This is my first program in java

Description:

Public: This makes the main method public that means that we can call the method from outside the class.

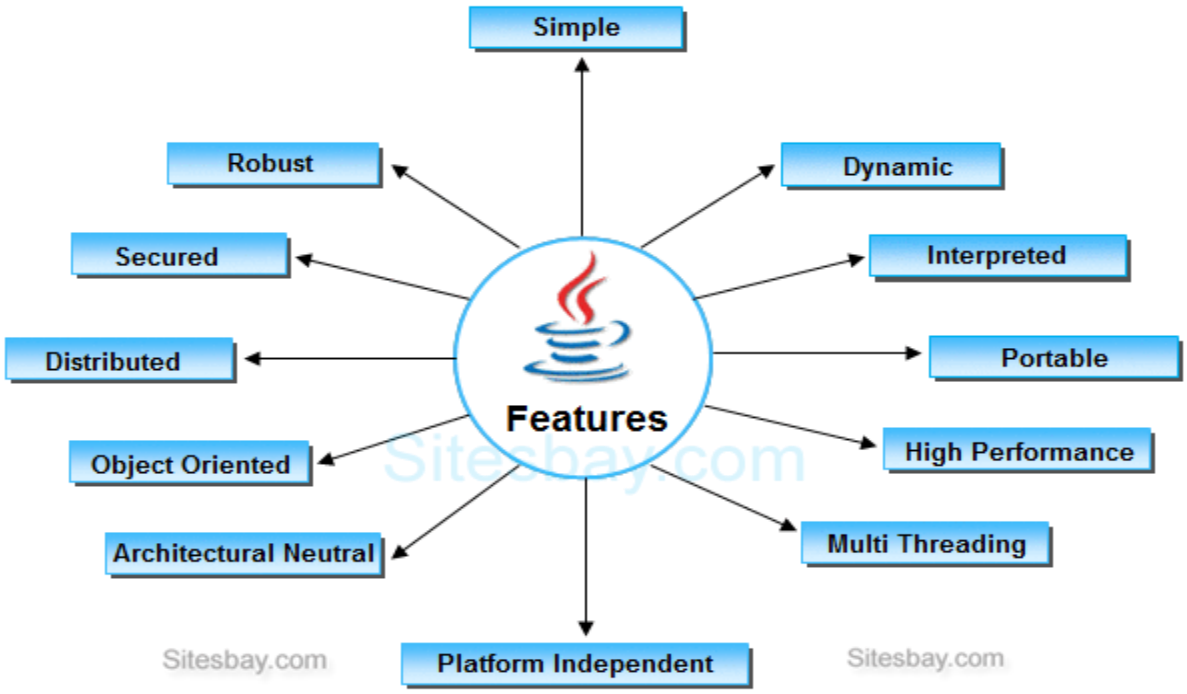
Static: We do not need to create object for static methods to run. They can run itself.

Void: It does not return anything.

Main: It is the method name. This is the entry point method from which the JVM can run your program.

(String[] args): Used for command line arguments that are passed as strings. We will cover that in a separate post.

FEATURES OF JAVA



There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

- Simple
- Object-Oriented
- Platform independent
- Secured
- Robust
- Architecture neutral
- Portable
- Dynamic
- Interpreted
- High Performance
- Multithreaded
- Distributed

SIMPLE:

According to Sun, Java language is simple because: syntax is based on C++ (so easier for programmers to learn it after C++). Removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. No need to remove unreferenced objects because there is Automatic Garbage Collection in java.

OBJECT-ORIENTED:

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.

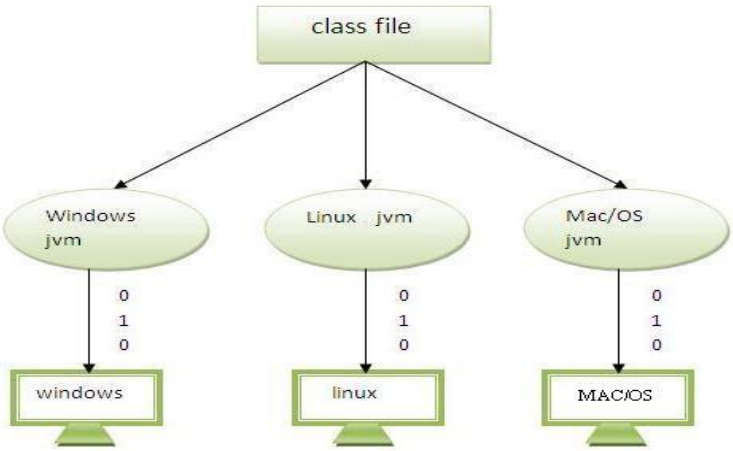
Basic concepts of OOPs are:

1. Object
2. Class
3. Inheritance
4. Polymorphism
5. Abstraction
6. Encapsulation

PLATFORM INDEPENDENT:

A platform is the hardware or software environment in which a program runs. There aretwo types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs from most other platforms in the sense that it's 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)

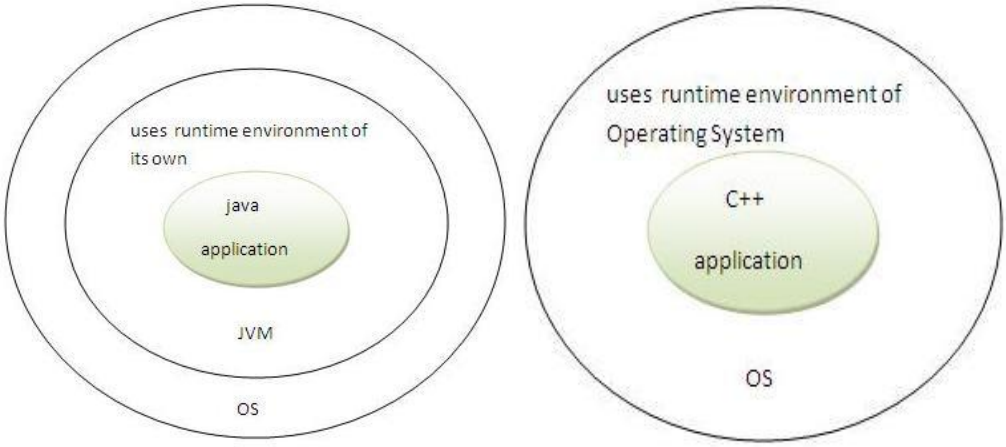


Java code can be run on multiple platforms e.g.Windows,Linux,Sun Solaris,Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere (WORA).

SECURED:

Java is secured because:

- No explicit pointer
- Programs run inside virtual machine sandbox



- **Classloader:** adds security by separating the package for the classes of the local file system from those that are imported from network sources
- **Bytecode Verifier:** checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk

These security are provided by java language. Some security can also be provided by application developer through SSL,JAAS,cryptography etc

ROBUST:

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust

ARCHITECTURE-NEUTRAL:

Thereis no implementation dependent feature e.g. size of primitive types is set

PORTABLE:

We may carry the java bytecode to any platform

High Performance:

Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)

DISTRIBUTED:

We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Java development environment has 2 Part:

1. **Compiler:**
   When program is compiled, it is translated into machine code i.e. specific to processor but java compiler generates byte code instead of machine code. This byte code id independent on machine.
2. **Interpreter:**
   It executes java0020program. When java program is executed, it also check byte code.

Java is an Object-Oriented Language. As a language that has the Object Oriented feature, Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects

- Instance
- Method

DATA TYPES AND VARIABLE IN JAVA

WHAT IS VARIABLE:

A variable is a name which is associated with a value that can be changed. For example when I write int i=10; here variable name is **i** which is associated with value 10, int is a data type that represents that this variable can hold integer values.

To declare a variable follow this syntax:

data_type variable_name = value;

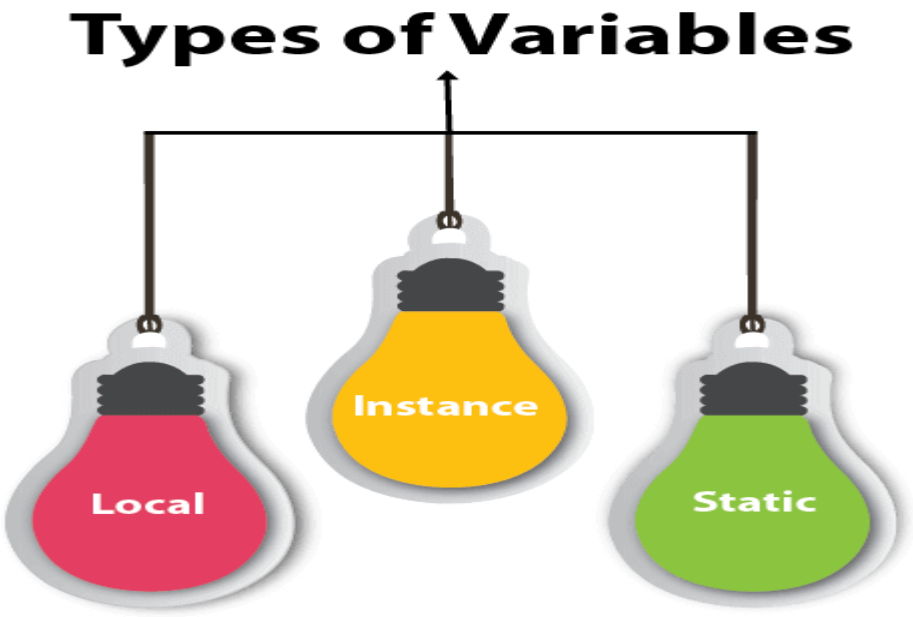
There are two types of datatypes in java, primitive and non-primitive.When datatypes are required as true objects java provides classes for these primitive datatypes, known as wrapper classes.

Variable: Variable is name of reserved area allocated in memory.

1) **int** data=50;//Here data is variable

TYPES OF VARIABLE:-

There are three types of variables in java



- **Local Variable:-** A variable that is declared inside the method is called local variable.

- **Instance Variable:-** A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static.
- **Static variable:-**A variable that is declared as static is called static variable. It cannot be local.

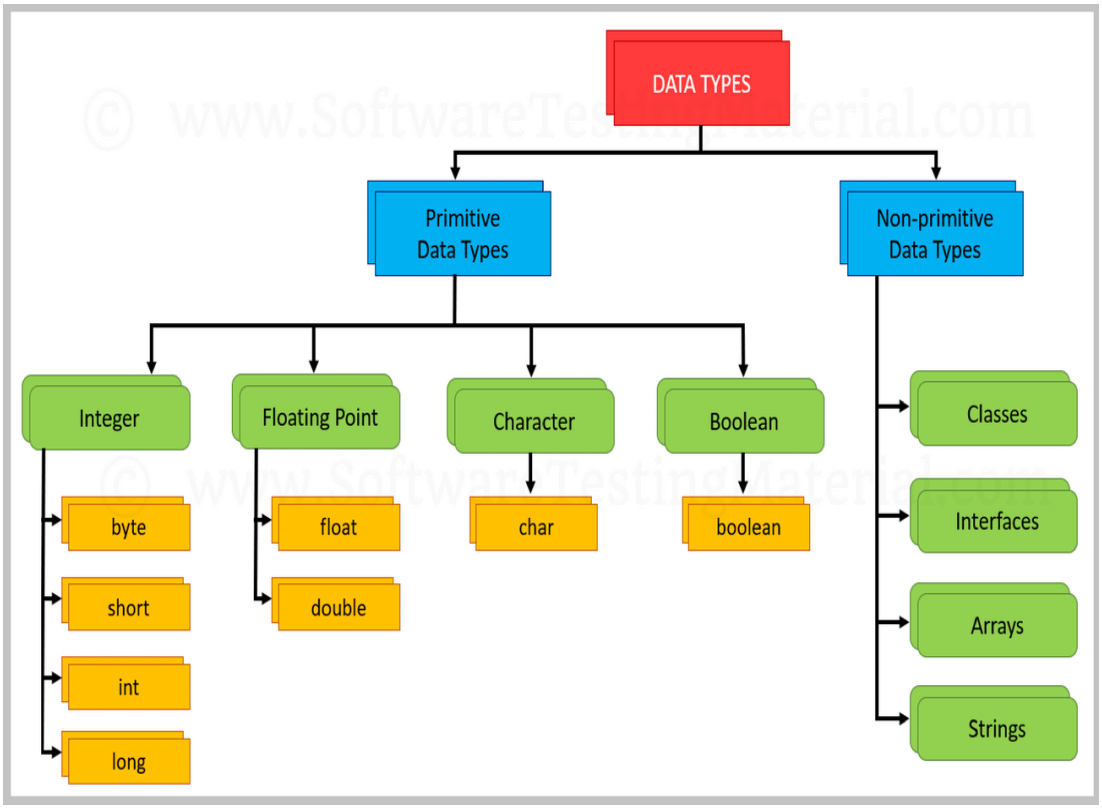
Example to understand the types of variables

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

DATA TYPES IN JAVA:

In java, there are two types of data types

- **primitive data types**
- **Non-primitive data types**

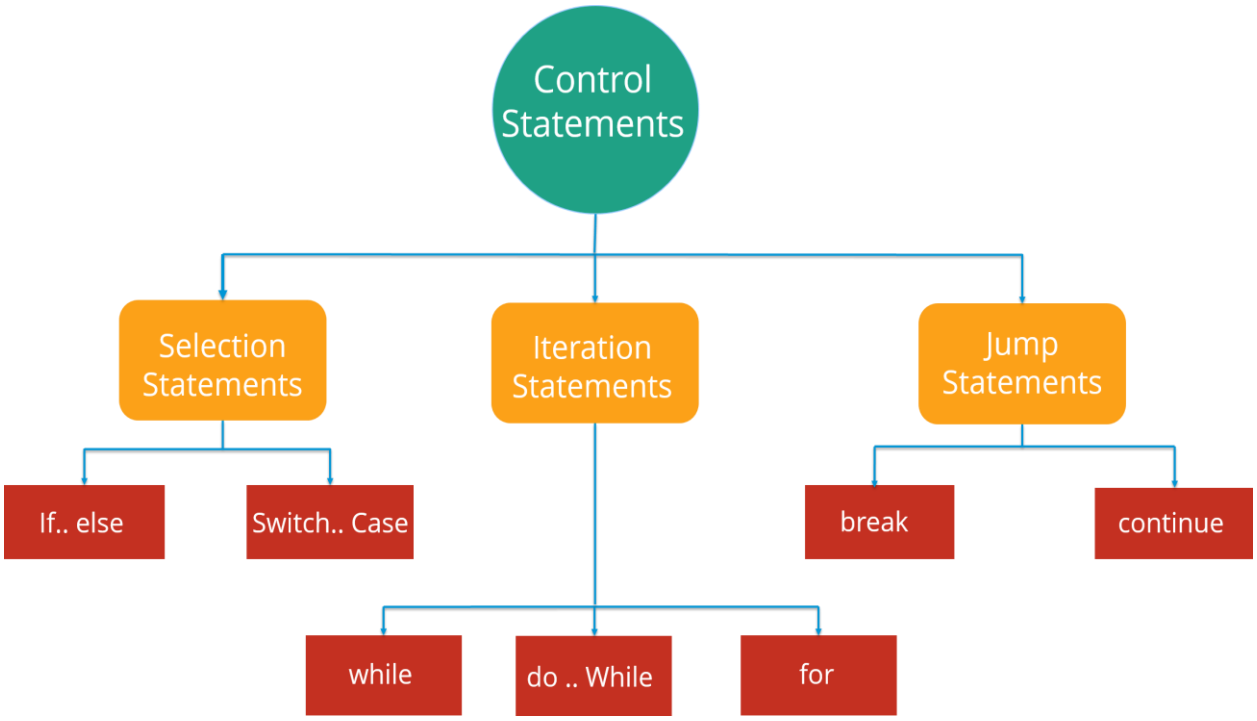


OPERATORS:

Operator in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

Operators	**Precedence**
Postfix	*expr*++ *expr*--
Unary	++*expr* --*expr* +*expr* -*expr* ~ !
Multiplicative	* / %
Additive	+ -
Shift	<<>>>>>
Relational	<><= >= instanceof
Equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
Ternary	? :
Assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

STATEMENTS IN JAVA



Java If else Statement:

The Java *if statement* is used to test the condition. It returns *true* or *false*. There are various types of if statement in java.

IF…ELSE LOOP:

Syntax:

```
If (condition 1)
        Statement 1;
Else
        Statement 2;
```

```
public class IfExample {
        public static void main(String[] args) {
        int age=20;
        if(age>18){
```

```
            System.out.print("Age is greater than 18");
            }
        }
        }
```

If else example:

```
        public class IfElseExample {
            public static void main(String[] args) {
                int number=13;
                   if(number%2==0){
                        System.out.println("even number");
                }else{
                System.out.println("odd number");
                }
            }
            }
```

NESTED IF…ELSE LOOP:

Syntax:

```
If (condition 1)
        Statement 1;
Else if(condition 2)
        Statement 2;
Else if(condition 3)
        Statement 3;
.
.
.
Else
        Statement;
```

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

```
int marks=65;


if(marks<50){
        System.out.println("fail");
        }
else if(marks>=50 && marks<60){
        System.out.println("D grade");
        }
        else if(marks>=60 && marks<70){
        System.out.println("C grade");
        }
        else if(marks>=70 && marks<80){
                System.out.println("B grade");
        }
        else if(marks>=80 && marks<90){
        System.out.println("A grade");
        }else if(marks>=90 && marks<100){
        System.out.println("A+ grade");
                }else{
        System.out.println("Invalid!");
         }
        }
        }
```

FOR LOOP:

The Java FOR LOOP is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

- Simple For Loop
- For-each or Enhanced For Loop

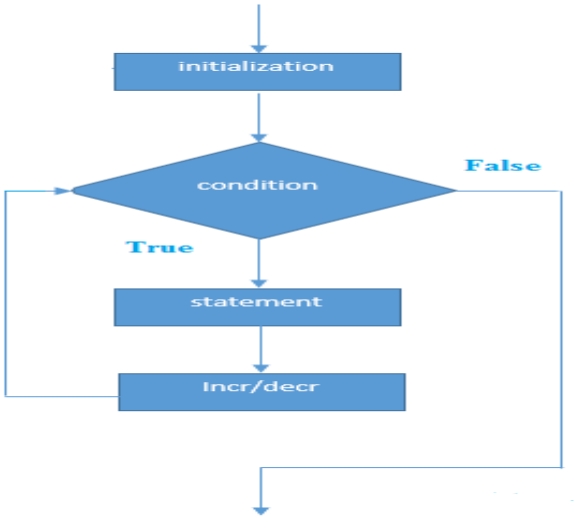
- Labeled For Loop

JAVA SIMPLE FOR LOOP

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

Syntax:

```
for(initialization;condition;incr/decr){

//code to be executed

}
```



Example:

```
public class ForExample {

public static void main(String[] args) {

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

  System.out.println(i);

}

}

}
```

JAVA FOR-EACH LOOP

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:

```
for(Type var:array){
//code to be executed
}
```

Example:

```
public class ForEachExample {
public static void main(String[] args) {
int arr[]={12,23,44,56,78};
for(int i:arr){
        System.out.println(i);
}
}
}
```

JAVA LABELED FOR LOOP

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.Normally, break and continue keywords breaks/continues the inner most for loop only.

Syntax:

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

Example:

```
public class LabeledForExample {
```

```
public static void main(String[] args) {
        aa:
                for(int i=1;i<=3;i++){
                        bb:
                                for(int j=1;j<=3;j++){
                                        if(i==2&&j==2){
                        break aa;
                                }
                                System.out.println(i+" "+j);
                                }
                }
        } }
```

If you use **break bb;**, it will break inner loop only which is the default behavior of any loop.

```
public class LabeledForExample {
public static void main(String[] args) {
aa:
        for(int i=1;i<=3;i++){
           bb:
                for(int j=1;j<=3;j++){
                        if(i==2&&j==2){
                for(int bb;
                        }
            System.out.println(i+" "+j);
                        }       }       }       }
```

JAVA INFINITIVE FOR LOOP:

If you use two semicolons ;; in the for loop, it will be infinitive for loop.

Syntax:

```
for(;;){
```

```
//code to be executed
}
```

Example:

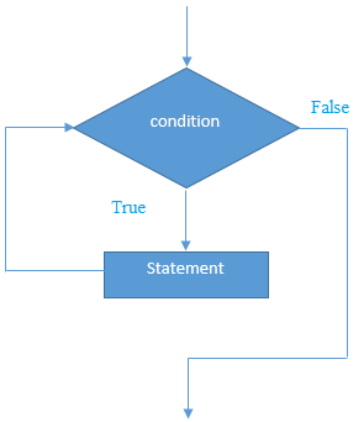
```
public class ForExample {
    public static void main(String[] args) {
    for(;;){
    System.out.println("infinitive loop");
    }
    }
    }
```

While loop

The Java WHILE LOOP is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:

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



Example:

```
public class WhileExample {
```

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

{
   int i=1;
   while(i<=10){
      System.out.println(i);
   i++;
   } }
}
```

JAVA INFINITIVE WHILE LOOP

If you pass **true** in the while loop, it will be infinitive while loop.

Syntax:

```
while(true){

//code to be executed

}
```

Example:

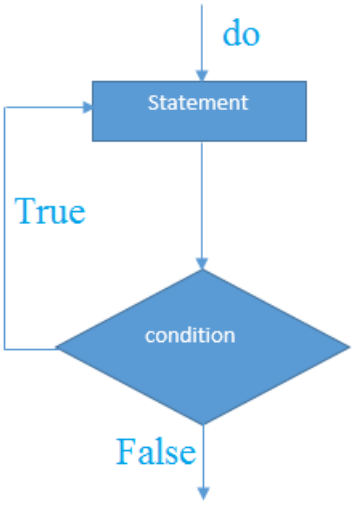
```
public class WhileExample2 {

public static void main(String[] args) {

while(true){

    System.out.println("infinitive while loop");

        } } }
```

DO….WHILE LOOP

The Java DO-WHILE LOOP is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use while loop.It is executed at least once because condition is checked after loop body.

Syntax:

```
do{

//code to be executed

}while(condition);
```



Example:

```
public class DoWhileExample {
        public static void main(String[] args) {
            int i=1;
            do{
                System.out.println(i);
                i++;
            }while(i<=10);
            }
            }
```

JAVA INFINITIVE DO-WHILE LOOP

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

Syntax:

```
while(true){
//code to be executed
}
```

Example:

```
public class DoWhileExample2 {
```

```
        public static void main(String[] args) {
        do{
                System.out.println("infinitive do while loop");
        }while(true);
        }
    }
```

SWITCH LOOP:

Syntax:

```
Switch ( expression)
{
Case 0:
        Statement 1;
        Break;
Case 1:
        Statement 2;
        Break;
.
.
Case n:
        Statement n;
        Break;
Default:
        Statement;
}
```

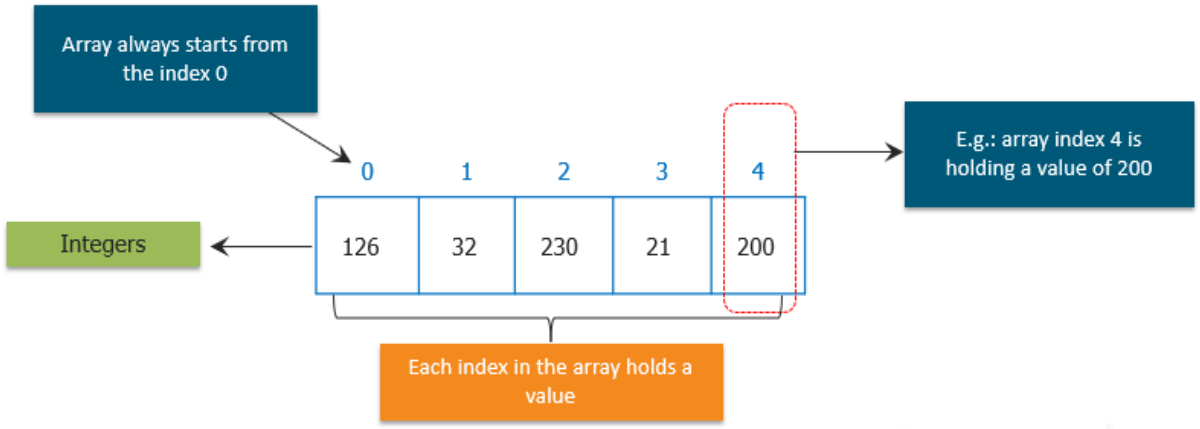
Example:

```
class SwitchDemo{
    public static void main(String args[]){
      int a,b,choice;
      System.out.println("1.Addition\n2.Substraction\n3.Multiplication\n4.Division");
      choice=1;
```

```
switch(choice)
{
        case 1:
                System.out.println("Addition="+(a+b));
                break;
        case 2:
                System.out.println("Substraction="+(a-b));
                break;
        case 3:
                System.out.println("Multiplication="+(a*b));
                break;
        case 4:
                System.out.println("Division="+(a/b));
                break;
        default :
                System.out.println("Invalid Condition!");
}
}
}
```

ARRAYS:

Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information. It is zero based index, the highest index of the array element is always(n-1)th index.



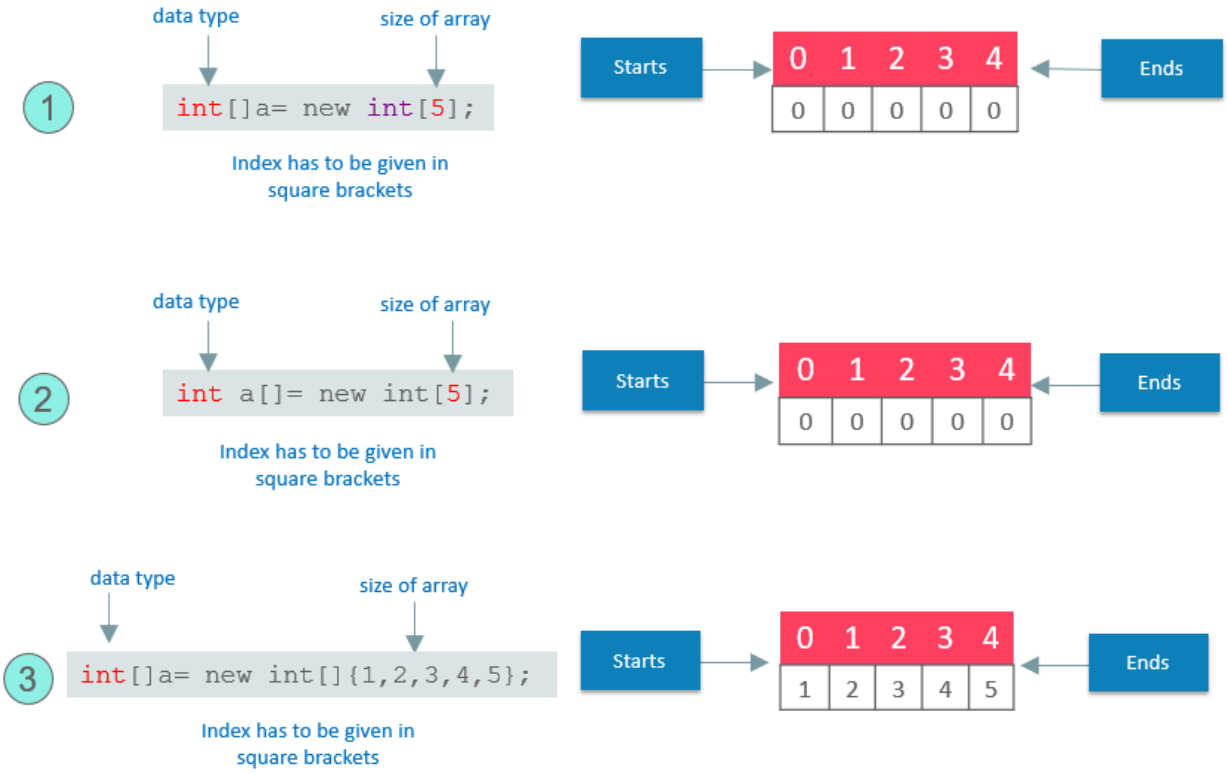
There are two types of array:

- **single dimensional array**
- **multidimensional array**

syntax :

enosis learning

Creating Software Professionals



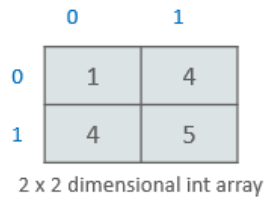
Example:

```
class Arraydemo
{
        public static void main(String args[])
        {
                int num[]= new int[5];
                for(int i=0;i<num.length;i++)
                {
                        num[i]=i*2;
                }
                for(int i=0;i<num.length;i++)
                {
                System.out.println("num["+i+"]"+"="+num[i]);
                }
        }
}
```

syntax to declare multidimensional array:

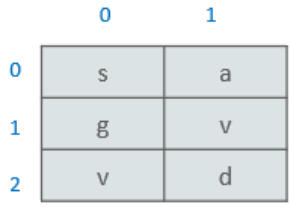
datatype [][]arrayName= new datatype [row size][col size];

```
int [][]a= new int [2][2];
```



2 x 2 dimensional int array

```
char [][]a= new char[3][2];
```



CLASSES AND OBJECTS

CLASS

A class is a blue print from which individual objects are created, it's a logical entity.

Syntax:

class className{

data members;

member functions(){

//block of code

}

}

A sample of a class is given below:

```
Public class Car
{
        Int speed;
        String color;
        Void cal_speed()
        {
        }
        Public static void main(String args[])
```

```
        {
        Car object =new  Car();
        Object.cal_speed();
        }
    }
```

Objects

Object is instance of a class, it is real time entity which exits in this world. It has state and behavior state describes the properties and behavior describes the actions.

They have **states** and **behaviors**.

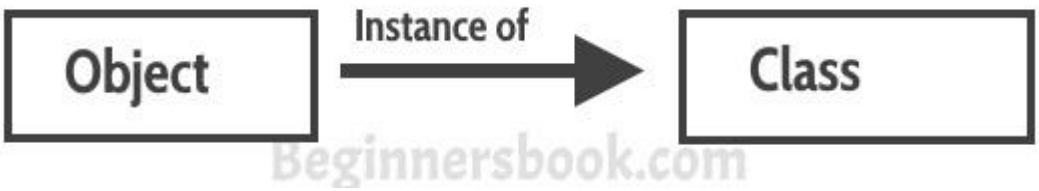
Examples of states and behaviors

Example 1:

Object: House

State: Address, Color, Area

Behavior: Open door, close door




```
class Demo
    {
       Public static void main(String args[])
    {
                Demo d=new Demo();              //d is the object of class Demo
    }
    }
```

Constructor

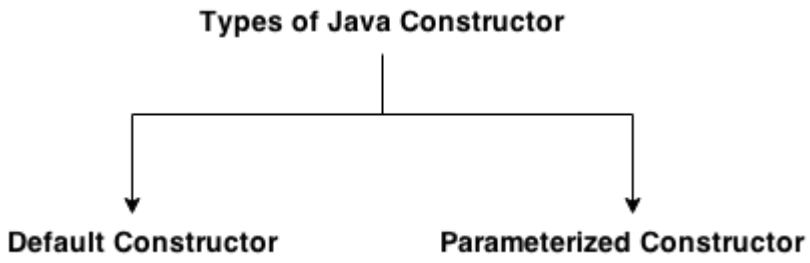
Constructor in java is a *special type of method* that is used to initialize the object.Java constructor is *invoked at the time of object creation*. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for creating java constructor

1. Constructor name must be same as its class name
2. Constructor must have no explicit return type

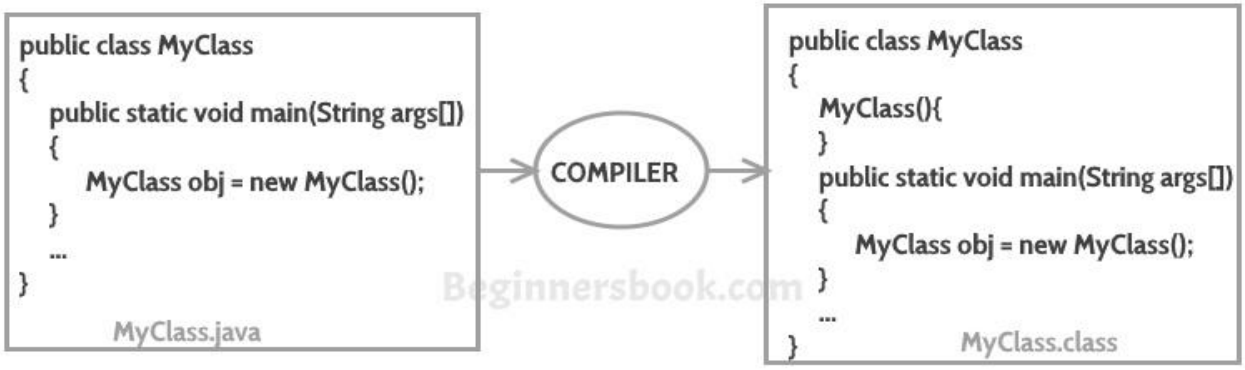
Types of java constructors

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



DEFAULT CONSTRUCTOR :

A constructor that has no parameter is known as default constructor.



Example:

```
Class Bike
        {
            Bike()
            {
                    System.out.println("Bike constructor get invoked");
            }
            Public static void main(String ar[])
```

```
        {
                Bike b=new Bike();
        }
    }
```

PARAMETERIZED CONSTRUCTOR :

Constructors that have parameters is known as parameterized constructors.

Why use parameterized constructor?

Parameterized constructor is used to provide different values to the distinct objects.

```
Class Bike
        {
        Bike(int i)
            {
                Int speed;
                Speed=i;
                System.out.println("Bike constructor get invoked having speed"
+speed);
            }
            Public static void main(String ar[])
            {
                Bike b=new Bike(100);
            }
    }
```

OVERLOADING METHODS:

If class has multiple methods by same name but different parameters, it is known as Method Overloading. Method overloading increases the readability of the program.

Example:

```
    Class Calculate
    {
        void add()
        {
            int a=10, b=30,c;
```

```
                c=a+b;
                System.out.println("Addition is : "+c);
        }
        void add(int a,int  b)
        {
                int c;
                c=a+b;
                System.out.println("Addition is : "+c);
        }
        public static void main(String ar[])
        {
                Calculate cal=new Calculate();
                Cal.add();
                Cal.add(50,60);
        }
}
```

GARBAGE COLLECTION :

In java, garbage means unreferenced objects. Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

ADVANTAGE OF GARBAGE COLLECTION :

- It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.
- It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

FINALIZE() METHOD :

The finalize() method is invoked each time before the object is garbage collected. This method can be used to perform cleanup processing. This method is defined in Object class as:

```
protected void finalize(){    }
```

GC() METHOD :

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public class TestGarbage1

{
```

```
public void finalize()
{
        System.out.println("object is garbage collected");
}
public static void main(String args[])
{
        TestGarbage1 s1=new TestGarbage1();
        TestGarbage1 s2=new TestGarbage1();
        s1=null;
         s2=null;
        System.gc();
} }
```

NESTED CLASSES:

In Java, just like methods, variables of a class too can have another class as its member. Writing a class within another is allowed in Java. The class written within is called the **nested class**, and the class that holds the inner class is called the **outer class**.

Example :

```
classOuter_Demo
{
        classNested_Demo
        {
        }
}
```

INHERITANCE

Inheritance :

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

extends Keyword :

extends is the keyword used to inherit the properties of a class. Below given is the syntax of extends keyword.

```
classSuper
        {
        .....
        .....
        }
classSubextendsSuper
        {
        .....
        .....
        }
```

```
class Teacher
{
        String designation = "Teacher";
         String collegeName = "Beginnersbook";
         void does()
        {
          System.out.println("Teaching");
          }
}
public class PhysicsTeacher extends Teacher
{
          String mainSubject = "Physics";
```

```
    public static void main(String args[])
  {
    PhysicsTeacher obj = new PhysicsTeacher();
    System.out.println(obj.collegeName);
    System.out.println(obj.designation);
    System.out.println(obj.mainSubject);
    obj.does();
    }
  }
```

Types of Inheritance

1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance

OVERRIDINGMETHOD:

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 subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.

Rules :

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

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

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

It doesn't call super/base class method. So to avoid this super keyword is used."

SUPER KEYWORD:

The **super** keyword is similar to **this** keyword following are the scenarios where the super keyword is used.

- It is used to **differentiate the members** of superclass from the members of subclass, if they have same names.
- It is used to **invoke the superclass** constructor from subclass.

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

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

POLYMORPHISM

Polymorphism in java is a concept by which we can perform a SINGLE ACTION BY DIFFERENT WAYS. Polymorphism is derived from **two greek** words: **poly** and **morphs**. The word "**poly**" means **many** and "**morphs**" means **forms.** So **polymorphism** means **many forms**.

- There are two types of polymorphism in java: compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.
- If you overload static method in java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.
- Polymorphism also **represents HAS-A relationship**.

RUN TIME 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.

Let's first understand the upcasting before Runtime Polymorphism.

Example :

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

```
        {
                System.out.println("eating fruits");
        }
}
 class BabyDog extends Dog
{
        void eat()
        {
                System.out.println("drinking milk");
        }
        public static void main(String args[])
        {
                Animal a1,a2,a3;
                a1=new Animal();
                a2=new Dog();
                a3=new BabyDog();
                a1.eat();
                a2.eat();
                a3.eat();
        } }
```

FINAL KEYWORD :

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context.

Final can be used with the :

1. **Variable**
2. **Method**
3. **Class**

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only.

FINALVARIABLE:

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

Example:

```
class Bike9
{
                final int speedlimit=90;
                void run()
                {
                  speedlimit=400;
                }
                public static void main(String args[])
                {
                Bike9 obj=new  Bike9();
                obj.run();
                }
        }
```

Output:

Compile time error.

FINAL METHOD :

If we declare any method as final,we cannot override this method into the derived class.

```
class Bike
{
        final void run()
```

```
        {
                System.out.println("running");
        }
}
class Honda extends Bike
{
    void run()
    {
        System.out.println("running safely with 100kmph");
    }
    public static void main(String args[])
    {
        Honda honda= new Honda();
        honda.run();
    }
}
```

Output:

Compile time error.

FINAL CLASS:

if any class declared as final class it will not be extended. If we don't want to allow to extend the class by any other class we will mark it as **final class.**

Example :

```
final class Bike{    }


class Honda1 extends Bike
{
    void run()
    {
        System.out.println("running safely with 100kmph");
```

```
        }
        public static void main(String args[])
        {
                Honda1 honda= new Honda();
                honda.run();
        }
    }
```

Output:

Compile time error.

ABSTRACT KEYWORD

Abstract class:

- A class that is declared with abstract keyword, is known as abstract class in java. It can have abstract and non-abstract methods (method with body).
- It needs to be extended and its method implemented. It cannot be instantiated.
- Abstract class can have abstract as well as concrete methods

Example :

abstract class Car() { }

Abstract Method:

- A method that is declared as abstract and does not have implementation is known as abstract method.
- the class which is going to extends the abstract class should give the definition of the abstract method other wise the extending class has to be declared as a abstract.

Example :

abstract void printStatus();

```
abstract class Bike
{
    abstract void run();
```

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

If you are extending any abstract class that have abstract method, you must either provide the implementation of the method or make this class abstract.

INTERFACE:

- An **interface in java** is a blueprint of a class. It has static constants and abstract methods only.
- The interface in java is **a mechanism to achieve fully abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve fully abstraction and multiple inheritance in Java.
- Java Interface also **represents IS-A relationship**.
- It cannot be instantiated just like abstract class.

- Java does not support multiple inheritance so interfaces are implemented.

Example :

```
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();
        }
}
```

The java compiler adds public and abstract keywords before the interface method.

More, it adds public, static and final keywords before data members.

EXCEPTION HANDLING

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. The core advantage of exception handling is **to maintain the normal flow of the application**.

An exception (or exceptional event) is a problem that arises during the **execution** of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore these exceptions are to be handled.

An exception can occur for many different reasons, below given are some scenarios where exception occurs.

- A user has entered invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

TYPES OF EXCEPTION

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

CHECKED EXCEPTION

- The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

UNCHECKED EXCEPTION

- The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

ERROR

- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc

Example:

```
publicclassUnchecked_Demo
{
        publicstaticvoid main(Stringargs[])
        {
                intnum[]={1,2,3,4};
        System.out.println(num[5]);     }       }
```

```
package calc;l1=new Label("First Value");
                    l1.setBounds(50, 80, 80, 20);
                    l2=new Label("Second Value");
                    l2.setBounds(50, 110, 80, 20);
                    l3=new Label("Result Value");
                    l3.setBounds(50, 140, 80, 20);
                    tf1=new TextField();
                    tf1.setBounds(150, 80, 100, 20);
                    tf2=new TextField();
                    tf2.setBounds(150, 110, 100, 20);
                    tf3=new TextField();
                    tf3.setBounds(150, 140, 100, 20);
                    b1=new Button("+");
                    b1.setBounds(50, 180, 50, 30);
                    b1.addActionListener(this);
                    b2=new Button("-");
                    b2.setBounds(110, 180, 50, 30);
                    b2.addActionListener(this);
                    b3=new Button("*");
                    b3.setBounds(170, 180, 50, 30);
                    b3.addActionListener(this);
                    b4=new Button("/");
                    b4.setBounds(230, 180, 50, 30);
                    b4.addActionListener(this);
                    add(l1);
                    add(l2);
                    add(l3);
```

```
                add(tf1);
                add(tf2);
                add(tf3);
                add(b1);
                add(b2);
                add(b3);
                add(b4);
        }
        @Override
        public void actionPerformed(ActionEvent e) {
                String s1=tf1.getText();
                String s2=tf2.getText();

                int v1=Integer.parseInt(s1);
                int v2=Integer.parseInt(s2);
                int v3=0;
                if(e.getSource()==b1) {
                        v3=v1+v2;
                }
                if(e.getSource()==b2) {
                        v3=v1-v2;
                }
                if(e.getSource()==b3) {
                        v3=v1*v2;
                }
```

- **Try.....catch.....finally**

- TRY BLOCK : The code which has chances to generate an exception i.e. write into try block. When exception occurtey block throw that exception and stop execution.
- CATCH BLOCK : The exception thrown by try block is catched by catch block. It matches type of exception. When exception is catched bye catch block, it is executed.
- FINALLY BLOCK : Finally block will be executed after try or catch block i.e finally bock executed whether exception occur or not.
- Try block has catch block and finally block. One try block has multiple catch blocks.

Example :

```
Class Div
{
Public static void main(String args[])
{
        inta,b,c;
        try
        {
        DataInputStream din=new DataInputStream(System.in);
        System.out.println("Enter 1st No");
        a=Integer.parseInt(din.readLine());
        System.out.println("Enter 2nd No");
        b=Integer.parseInt(din.readLine());
        c=a/b;
        System.out.println("Division is : ",c);
        }
        Catch(IOException e)
        {
                System.out.println(e);
        }
```

```
Catch(ArithamaticException e1)
{
        System.out.println(e1);
}
Finally
{
        System.out.println("I am always execute");
} }
```

THROWSKEYWORD

Instead of try…catch block we can use throws keyword to handle exception.

Example:

```
Class Div
{
        Public static void main(String args[]) throws Exception
        {
                inta,b,c;
        DataInputStream din=new DataInputStream(System.in);
        System.out.println("Enter 1st No");
        a=Integer.parseInt(din.readLine());
        System.out.println("Enter 2nd No");
        b=Integer.parseInt(din.readLine());
        c=a/b;
        System.out.println("Division is : ",c);
        }  }
```

THROWKEYWORD:

The java runtime system throw exception bydefault. It is possible to throw an exception explicitly using throw keyword.

Syntax:

Throw throwableinstance(object of exception class)

E**xample :**

```
classMyexp extends Exception
{
Public Myexp()
{
 Public static String toString()
{
Return("Plz Enter Correct value");
} }
}
classexp
{
        int  a;
        public static void main(String args[])throws Exception
        {
        DataInputStream din=new DataInputStream(System.in);
        System.out.println("Enter No");
        a=Integer.parseInt(din.readLine());
        if(a>10)
        throw new Myexp();
        System.out.println("Value of A is : " ,a);
        } }
```

UserDefined Exception**:**

- java custom exceptions are used to handle exceptions as per user requirementsBy the help of custom exception, you can have your own exception and message

```
class InvalidAgeException extends Exception
{
        InvalidAgeException(String s)
        {
        System.out.print(s+":-");
        }}

public class ExceptionDemo {

         static void validate(int age)
        {
        try
        {
                if(age<0 || age>100)
                {
                        throw new InvalidAgeException("You are not valide user");
                }
                else
                {
                        System.out.println("you can vote");
                }
        }
        catch(Exception e)
{
                System.out.println(e);
 }
        }
        public static void main(String args[])
        {
                validate(110);
        }
}
```

1. At a time only one Exception is occured and at a time only one catch block is executed.
2. All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .
3. If you don't handle exception, before terminating program, JVM executes finally block
4. For each try block there can be zero or more catch blocks, but only one finally block.
5. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort

WRAPPER CLASSES

Wrapper class in java provides the mechanism *TO CONVERT PRIMITIVE INTO OBJECT AND OBJECT INTO PRIMITIVE*. One of the eight classes of JAVA.LANG package are known as wrapper class in java. The lists of eight wrapper classes are given below:

Primitive Type	**Wrapper class**
Boolean	Boolean
Char	Character
Byte	Byte
Short	Short
Int	Integer
Long	Long
Float	Float
Double	Double

Example:

```
public class WrapperExample1
{
public static void main(String args[])
        {
                int a=20;
                Integer i=Integer.valueOf(a);
                Integer j=a;
                System.out.println(a+" "+i+" "+j); }
}
```

Packages

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management.

Some of the existing packages in Java are:

- **java.lang** - bundles the fundamental classes
- **java.io** - classes for input , output functions are bundled in this package

Programmers can define their own packages to bundle group of classes/interfaces, etc. It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, annotations are related. to define package we use **package** keyword.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. Using packages, it is easier to provide access control and it is also easier to locate the related classes.

Example 1.

```
package mypackage1;

public class Simple{

public static void main(String args[]){

System.out.println ("Package Created");

}

}
```

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

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

There can be only one public class in a java source file and it must be saved by the public class name.

I/O PACKAGES

FileInputStream and FileOutputStream (File Handling):

In Java, FileInputStream and FileOutputStream classes are used to read and write data in file. In another words, they are used for file handling in java.

Java FileOutputStream Class:

Java FileOutputStream is an output stream for writing data to a file.If you have to write primitive values then use FileOutputStream.Instead, for character-oriented data, prefer FileWriter.But you can write byte-oriented as well as character-oriented data.

Example of Java FileOutputStream class

```
import java.io.*;
class Test{
    public static void main(String args[]){
    try{
            FileOutputstream fout=new FileOutputStream("abc.txt");
            String s="Sachin Tendulkar is my favourite player";
            byte b[]=s.getBytes();//converting string into byte array
            fout.write(b);
            fout.close();
            System.out.println("success...");
      }catch(Exception e){system.out.println(e);}
      }
```

Java FileInputStream class

Java FileInputStream class obtains input bytes from a file.It is used for reading streams of raw bytes such as image data. For reading streams of characters, consider using FileReader. It should be used to read byte-oriented data for example to read image, audio, video etc.

```
import java.io.*;
    class SimpleRead{
    public static void main(String args[]){
    try{
        FileInputStream fin=new FileInputStream("abc.txt");
```

```
int i=0;
while((i=fin.read())!=-1){
        System.out.println((char)i);
    }
    fin.close();
    }catch(Exception e){system.out.println(e);}
}   }
```

Example of Reading the data of current java file and writing it into another file:

We can read the data of any file using the FileInputStream class whether it is java file, image file, video file etc. In this example, we are reading the data of C.java file and writing it into another file M.java.

```
import java.io.*;
class C{
        public static void main(String args[])throws Exception{
        FileInputStream fin=new FileInputStream("C.java");
        FileOutputStream fout=new FileOutputStream("M.java");
        int i=0;
        while((i=fin.read())!=-1){
            fout.write((byte)i);
        }
        fin.close();
        }   }
```

Java FileWriter and FileReader:

Java FileWriter and FileReader classes are used to write and read data from text files. These are character-oriented classes, used for file handling in java. Java has suggested not to use the FileInputStream and FileOutputStream classes if you have to read and write the textual information.

Constructor	Description
FileWriter(String file)	creates a new file. It gets file name in string.
FileWriter(File file)	creates a new file. It gets file name in File object.

Methods of FileWriter class

Method	Description
1) public void write(String text)	writes the string into FileWriter.
2) public void write(char c)	writes the char into FileWriter.
3) public void write(char[] c)	writes char array into FileWriter.
4) public void flush()	flushes the data of FileWriter.
5) public void close()	closes FileWriter.

Java FileReader Class:

Java FileReader class is used to read data from the file. It returns data in byte format like FileInputStream class.

Constructors of FileWriter class

Constructor	Description
FileReader(String file)	It gets filename in string. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.
FileReader(File file)	It gets filename in file instance. It opens the given file in read mode. If file doesn't exist, it throws FileNotFoundException.

Methods of FileReader class

Method	Description
1) public int read()	returns a character in ASCII form. It returns -1 at the end of file.
2) public void close()	closes FileReader.

Java FileReader Example :In this example, we are reading the data from the file abc.txt file.

```
import java.io.*;
    class Simple{
        public static void main(String args[])throws Exception{
        FileReader fr=new FileReader("abc.txt");
        int i;
        while((i=fr.read())!=-1)
```

```
                System.out.println((char)i);
                fr.close();
                }
        }
```

READING DATA FROM KEYBOARD

There are many ways to read data from the keyboard. For example:

- InputStreamReader
- Console
- Scanner
- DataInputStream etc.

InputStreamReader Class:

InputStreamReader class can be used to read data from keyboard. It performs two tasks:

- connects to input stream of keyboard
- converts the byte-oriented stream into character-oriented stream

BufferedReaderClass:

BufferedReader class can be used to read data line by line by readLine() method.

```
import java.io.*;
        class G5{
        public static void main(String args[])throws Exception{
        InputStreamReader r=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(r);
        System.out.println("Enter your name");
        String name=br.readLine();
        System.out.println("Welcome "+name);
                }
                }
```

Console:

```
String text=System.console().readLine();

System.out.println("Text is: "+text);
```

Methods of **Console class**

Let's see the commonly used methods of Console class.

Method	Description
1) public String readLine()	is used to read a single line of text from the console.
2) public String readLine(String fmt,Object... args)	it provides a formatted prompt then reads the single line of text from the console.
3) public char[] readPassword()	is used to read password that is not being displayed on the console.
4) public char[] readPassword(String fmt,Object... args)	it provides a formatted prompt then reads the password that is not being displayed on the console.

How to get the object of Console:

System class provides a static method console() that returns the unique instance of Console class.

Java Console Example

```
import java.io.*;

class ReadStringTest{

public static void main(String args[]){

                Console c=System.console();

                System.out.println("Enter your name: ");

                String n=c.readLine();

                System.out.println("Welcome "+n);

                }

        }
```

Java Scanner Class:

There are various ways to read input from the keyboard, the java.util.Scanner class is one of them. The **Java Scanner** class breaks the input into tokens using a delimiter that is whitespace bydefault. It provides many methods to read and parse various primitive values.

Java Scanner class is widely used to parse text for string and primitive types using regular expression. Java Scanner class extends Object class and implements Iterator and Closeable interfaces.

Commonly used methods of Scanner class

There is a list of commonly used Scanner class methods:

Method	Description
public String next()	it returns the next token from the scanner.
public String nextLine()	it moves the scanner position to the next line and returns the value as a string.
public byte nextByte()	it scans the next token as a byte.
public short nextShort()	it scans the next token as a short value.
public int nextInt()	it scans the next token as an int value.
public long nextLong()	it scans the next token as a long value.
public float nextFloat()	it scans the next token as a float value.
public double nextDouble()	it scans the next token as a double value.

Java Scanner Example to get input from console:

Let's see the simple example of the Java Scanner class which reads the int, string and double value as an input:

```
import java.util.Scanner;

    class ScannerTest{

    public static void main(String args[]){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter your rollno");

        int rollno=sc.nextInt();

        System.out.println("Enter your name");

        String name=sc.next();

        System.out.println("Enter your fee");

        double fee=sc.nextDouble();

        System.out.println("Rollno:"+rollno+" name:"+name+" fee:"+fee);
```

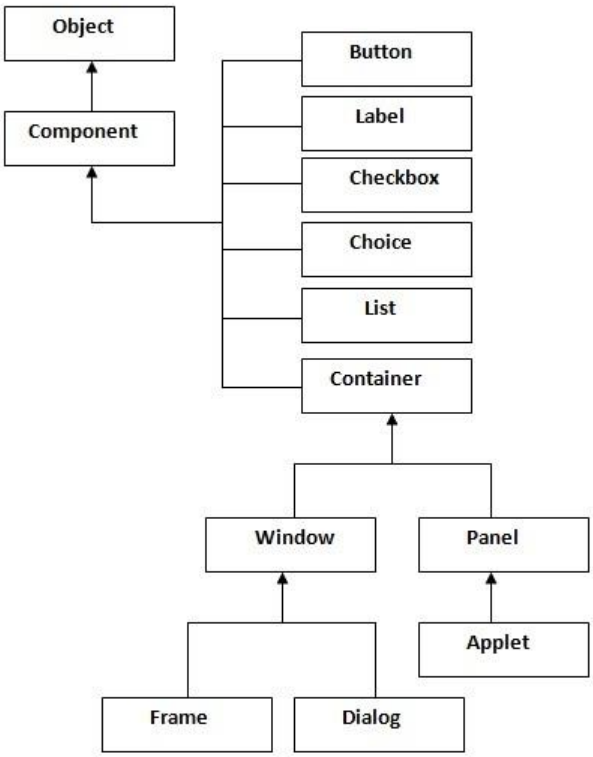
```
            sc.close();

    }}
```

JAVA AWT

Java AWT (Abstract Windowing Toolkit) is *an API to develop GUI or window-based application 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 uses the resources of system.The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy:

The hierarchy of Java AWT classes are given below.



Container: The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window: The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel: The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame: The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

- **By extending Frame class (inheritance)**
- **By creating the object of Frame class (association)**

Simple example of AWT by inheritance

```
import java.awt.*;

class First extends Frame{

First(){

Button b=new Button("click me");

b.setBounds(30,100,80,30);// setting button position

add(b);//adding button into frame

setSize(300,300);//frame size 300 width and 300 height

setLayout(null);//no layout manager

setVisible(true);//now frame will be visible, by default not visible

}

public static void main(String args[]){

First f=new First();

}}
```

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

Simple example of AWT by association

```
import java.awt.*;
class First2{
First2(){
Frame f=new Frame();

Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[]){
First2 f=new First2();  }}
```

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.

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. **Implement the Listener interface and overrides its methods**
2. **Register the component with the Listener**

EventHandling Codes:

We can put the event handling code into one of the following places:

1. **Same class**
2. **Other class**
3. **Annonymous class**

GRAPHICAL USER INTERFACE

Graphical User Interface (GUI) offers user interaction via some graphical components. For example our underlying Operating System also offers GUI via window,frame,Panel, Button, Textfield, TextArea, Listbox, Combobox, Label, Checkbox etc. These all are known as components. Using these components we can create an interactive user interface for an application.

GUI provides result to end user in response to raised events.GUI is entirely based events. For example clicking over a button, closing a window, opening a window, typing something in a textarea etc. These activities are known as events.GUI makes it easier for the end user to use an application. It also makes them interesting.

Examples of GUI based Applications

Following are some of the examples for GUI based applications.

- Automated Teller Machine (ATM)
- Airline Ticketing System
- Information Kiosks at railway stations
- Mobile Applications
- Navigation Systems

ADVANTAGES OF GUI OVER CUI

- GUI provides graphical icons to interact while the CUI (Character User Interface) offers the simple text-based interfaces.
- GUI makes the application more entertaining and interesting on the other hand CUI does not.
- GUI offers click and execute environment while in CUI every time we have to enter the command for a task.
- New user can easily interact with graphical user interface by the visual indicators but it is difficult in Character user interface.
- GUI offers a lot of controls of file system and the operating system while in CUI you have to use commands which is difficult to remember.
- Windows concept in GUI allow the user to view, manipulate and control the multiple applications at once while in CUI user can control one task at a time.
- GUI provides multitasking environment so as the CUI also does but CUI does not provide same ease as the GUI do.Using GUI it is easier to control and navigate the operating system which becomes very slow in command user interface. GUI can be easily customized.

AWT LABEL CLASS

Introduction

Label is a passive control because it does not create any event when accessed by the user. The label control is an object of Label. A label displays a single line of read-only text. However the text can be changed by the application programmer but cannot be changed by the end user in any way.

Field

Following are the fields for **java.awt.Component** class:

- **static int CENTER** -- Indicates that the label should be centered.
- **static int LEFT** -- Indicates that the label should be left justified.
- **static int RIGHT** -- Indicates that the label should be right justified.

Class methods

S.N.	Method & Description
1	void addNotify() Creates the peer for this label.
2	AccessibleContext getAccessibleContext() Gets the AccessibleContext associated with this Label.
3	int getAlignment() Gets the current alignment of this label.
4	String getText() Gets the text of this label.
5	protected String paramString() Returns a string representing the state of this Label.
6	void setAlignment(int alignment) Sets the alignment for this label to the specified alignment.
7	void setText(String text) Sets the text for this label to the specified text.

Methods inherited:-This class inherits methods from the following classes:

- java.awt.Component
- java.lang.Object

```
import java.awt.*;

class LabelExample{

public static void main(String args[]){

        Frame f= new Frame("Label Demo");
```

```
        Label l1,l2;
        l1=new Label("First .");
        l1.setBounds(50,100, 100,30);
        l2=new Label("Second.");
        l2.setBounds(50,150, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }    }
```

AWT CHECKBOX CLASS

Introduction

Checkbox control is used to turn an option on(true) or off(false). There is label for each checkbox representing what the checkbox does.The state of a checkbox can be changed by clicking on it.

Class methods

S.N.	Method & Description
1	void addItemListener(ItemListener l) Adds the specified item listener to receive item events from this check box.
2	void addNotify() Creates the peer of the Checkbox.
3	AccessibleContext getAccessibleContext() Gets the AccessibleContext associated with this Checkbox.
4	CheckboxGroup getCheckboxGroup() Determines this check box's group.
5	ItemListener[] getItemListeners() Returns an array of all the item listeners registered on this checkbox.
6	String getLabel()

	Gets the label of this check box.
7	<T extends EventListener>T[] getListeners(Class<T> listenerType) Returns an array of all the objects currently registered as FooListeners upon this Checkbox.
8	Object[] getSelectedObjects() Returns an array (length 1) containing the checkbox label or null if the checkbox is not selected.
9	boolean getState() Determines whether this check box is in the on or off state.
10	protected String paramString() Returns a string representing the state of this Checkbox.
11	protected void processEvent(AWTEvent e) Processes events on this check box.
12	protected void processItemEvent(ItemEvent e) Processes item events occurring on this check box by dispatching them to any registered ItemListener objects.
13	void removeItemListener(ItemListener l) Removes the specified item listener so that the item listener no longer receives item events from this check box.
14	void setCheckboxGroup(CheckboxGroup g) Sets this check box's group to the specified check box group.
15	void setLabel(String label) Sets this check box's label to be the string argument.

```
import java.awt.*;

public class CheckboxExample

{

 CheckboxExample(){

 Frame f= new Frame("Checkbox Demo");

  Checkbox checkbox1 = new Checkbox("Selenium");

  checkbox1.setBounds(100,100, 50,50);

  Checkbox checkbox2 = new Checkbox("SSRS", true);
```

```
checkbox2.setBounds(100,150, 50,50);
        f.add(checkbox1);
        f.add(checkbox2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
            }
        public static void main(String args[])
        {  new CheckboxExample();
        }    }
```

AWT LIST CLASS

Introduction

The List represents a list of text items. The list can be configured to that user can choose either one item or multiple items.

public class List **extends** Component **implements** ItemSelectable, Accessible

```
    import java.awt.*;
    public class ListExample  {
            ListExample(){
        Frame f= new Frame();
        List l1=new List(5);
        l1.setBounds(100,100, 75,75);
        l1.add("Pune");
         l1.add("Mumbai");
        l1.add("Nashik");
```

```
l1.add("Nagpur");

l1.add("Thane");

f.add(l1);

f.setSize(400,400);

f.setLayout(null);

f.setVisible(true);  }

public static void main(String args[])  {

new ListExample();

}  }
```

AWT DIALOG CLASS

Introduction

Dialog control represents a top-level window with a title and a border used to take some form of input from the user.

Class constructors

S.N.	Constructor & Description
1	Dialog(Dialog owner) Constructs an initially invisible, modeless Dialog with the specified owner Dialog and an empty title.
2	Dialog(Dialog owner, String title) Constructs an initially invisible, modeless Dialog with the specified owner Dialog and title.

Class methods

S.N.	Method & Description
1	void addNotify() Makes this Dialog displayable by connecting it to a native screen resource.
2	AccessibleContext getAccessibleContext() Gets the AccessibleContext associated with this Dialog.
3	Dialog.ModalityType getModalityType()

	Returns the modality type of this dialog.
4	String getTitle() Gets the title of the dialog.
5	void hide() Deprecated. As of JDK version 1.5, replaced by setVisible(boolean).
6	boolean isModal() Indicates whether the dialog is modal.
7	void setResizable(boolean resizable) Sets whether this dialog is resizable by the user.
8	void setTitle(String title) Sets the title of the Dialog.

```
import java.awt.*;

import java.awt.event.*;

   public class DialogExample {

                    private static Dialog d;

                    DialogExample() {

                       Frame f= new Frame();

                       d = new Dialog(f , "Dialog Box", true);

                       d.setLayout( new FlowLayout() );

                       Button b = new Button ("OK");

                       b.addActionListener ( new ActionListener()

                       {    public void actionPerformed( ActionEvent e )  {

                              DialogExample.d.setVisible(false);      }

                       });

                       d.add( new Label ("Click Ok to Continue"));

                       d.add(b);
```

```
        d.setSize(300,300);
        d.setVisible(true);
    }
    public static void main(String args[])
    {
        new DialogExample();
    }
}
```

AWT MENU AND MENUITEM

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
        Frame f= new Frame(" MenuItem Example");
        MenuBar mb=new MenuBar();
        Menu menu=new Menu("File");
        Menu submenu=new Menu("Sub Menu");
        MenuItem i1=new MenuItem("Open");
        MenuItem i2=new MenuItem("Save");
        MenuItem i3=new MenuItem("Save As");
```

```
        MenuItem i4=new MenuItem("Delete");
        MenuItem i5=new MenuItem("Print");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public static void main(String args[])
    {
    new MenuExample();
    } }
```

TOOLKIT CLASS

Toolkit class is the abstract superclass of every implementation in the Abstract Window Toolkit. Subclasses of Toolkit are used to bind various components. It inherits Object class.

```
        import java.awt.*;
        public class ToolkitExample {
```

```
    public static void main(String[] args) {
        Toolkit t = Toolkit.getDefaultToolkit();
        System.out.println("Screen resolution = " + t.getScreenResolution
());
        Dimension d = t.getScreenSize();
        System.out.println("Screen width = " + d.width);
        System.out.println("Screen height = " + d.height);
    }
}
```

JAVA SWING

Java Swing 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.

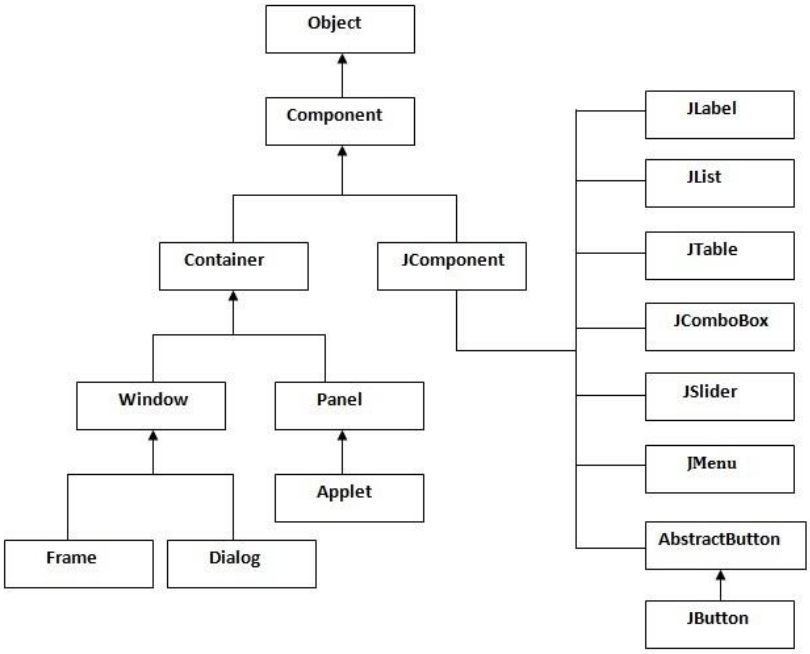
DIFFERENCE BETWEEN AWT AND SWING

There are many differences between java awt and swing that are given below.

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 componentssuch as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

WHAT IS JFC:

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.



COMMONLY USED METHODS OF COMPONENT CLASS

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

- **By creating the object of Frame class (association)**
- **By extending Frame class (inheritance)**

File: FirstSwingExample.java

```
import javax.swing.*;

    public class FirstSwingExample {
```

```
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame


JButton b=new JButton("click");//creating instance of JButton
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
f.add(b);//adding button in JFrame
f.setSize(400,500);//400 width and 500 height
f.setLayout(null);//using no layout managers
f.setVisible(true);//making the frame visible
}}
```

Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

File: Simple.java

```
import javax.swing.*;
public class Simple {
    JFrame f;
    Simple(){
        f=new JFrame();//creating instance of JFrame
        JButton b=new JButton("click");//creating instance of JButton
        b.setBounds(130,100,100, 40);
        f.add(b);//adding button in JFrame
        f.setSize(400,500);//400 width and 500 height
        f.setLayout(null);//using no layout managers
        f.setVisible(true);//making the frame visible
    }
public static void main(String[] args) {
```

```
new Simple();
                }
        }
```

The setBounds(int xaxis, int yaxis, int width, int height)is used in the above example that sets the position of the button.

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
Import javax.swing.*;

public class Simple2 extends JFrame{//inheriting JFrame

JFrame f;

Simple2(){
                JButton b=new JButton("click");//create button
                b.setBounds(130,100,100, 40);
                add(b);//adding button on frame
                setSize(400,500);
                setLayout(null);
                setVisible(true);
                }
                public static void main(String[] args) {
                new Simple2();
                }  }
```

JBUTTON

Commonly used Constructors:

- **JButton():** creates a button with no text and icon.

- **JButton(String s):** creates a button with the specified text.
- **JButton(Icon i):** creates a button with the specified icon object.

```
import java.awt.event.*;
import javax.swing.*;
public class ImageButton{
        ImageButton(){
                JFrame f=new JFrame();
                JButton b=new JButton(new ImageIcon("b.jpg"));
                b.setBounds(130,100,100, 40);
                f.add(b);
                f.setSize(300,400);
                f.setLayout(null);
                f.setVisible(true);
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
public static void main(String[] args) {
new ImageButton();
                }}
```

JRADIOBUTTON CLASS

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.It should be added in ButtonGroup to select one radio button only.

Commonly used Constructors of JRadioButton class:

- **JRadioButton():** creates an unselected radio button with no text.
- **JRadioButton(String s):** creates an unselected radio button with specified text.
- **JRadioButton(String s, boolean selected):** creates a radio button with the specified text and selected status.

Example of JRadioButton class:

```
import javax.swing.*;
        public class Radio {
JFrame f;
Radio(){
f=new JFrame();
JRadioButton r1=new JRadioButton("A) Male");
JRadioButton r2=new JRadioButton("B) FeMale");
r1.setBounds(50,100,70,30);
r2.setBounds(50,150,70,30);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String[] args) {
 new Radio();
} }
```

BUTTONGROUP CLASS:

The ButtonGroup class can be used to group multiple buttons so that at a time only one button can be selected.

JRadioButton example with event handling

```
import javax.swing.*;
import java.awt.event.*;
class RadioExample extends JFrame implements ActionListener{
JRadioButton rb1,rb2;
```

```
JButton b;
RadioExample(){
rb1=new JRadioButton("Male");
rb1.setBounds(100,50,100,30);
rb2=new JRadioButton("Female");
rb2.setBounds(100,100,100,30);
ButtonGroup bg=new ButtonGroup();
bg.add(rb1);bg.add(rb2);
b=new JButton("click");
b.setBounds(100,150,80,30);
b.addActionListener(this);
add(rb1);add(rb2);add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public void actionPerformed(ActionEvent e){
if(rb1.isSelected()){
JOptionPane.showMessageDialog(this,"You are male");
}
if(rb2.isSelected()){
JOptionPane.showMessageDialog(this,"You are female");
}
}
public static void main(String args[]){
new RadioExample();
}}
```

JCOMBOBOX CLASS:

The JComboBox class is used to create the combobox (drop-down list). At a time only one item can be selected from the item list.

Commonly used methods of JComboBox class:

1) **public void addItem(Object anObject):** is used to add an item to the item list.
2) **public void removeItem(Object anObject):** is used to delete an item to the item list.
3) **public void removeAllItems():** is used to remove all the items from the list.
4) **public void setEditable(boolean b):** is used to determine whether the JComboBox is editable.
5) **public void addActionListener(ActionListener a):** is used to add the ActionListener
6) **public void addItemListener(ItemListener i):** is used to add the ItemListener

Example of JComboBox class:

```
import javax.swing.*;
public class Combo {
JFrame f;
Combo(){
   f=new JFrame("Combo ex");
   String country[]={"India","Aus","U.S.A","England","Newzeland"};
   JComboBox cb=new JComboBox(country);
   cb.setBounds(50, 50,90,20);
   f.add(cb);
   f.setLayout(null);
   f.setSize(400,500);
   f.setVisible(true);

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

}
}
```

JTABLE CLASS

The JTable class is used to display the data on two dimensional tables of cells

Commonly used Constructors of JTable class:

- **JTable():** creates a table with empty cells.
- **JTable(Object[][] rows, Object[] columns):** creates a table with the specified data.

Example of JTable class:

```
import javax.swing.*;
public class MyTable {
   JFrame f;
MyTable(){
   f=new JFrame();
  String data[][]={ {"101","Amit","670000"},
            {"102","Jai","780000"},
                  {"101","Sachin","700000"}};
   String column[]={"ID","NAME","SALARY"};
   JTable jt=new JTable(data,column);
   jt.setBounds(30,40,200,300);
   JScrollPane sp=new JScrollPane(jt);
   f.add(sp);
   f.setSize(300,400);
//  f.setLayout(null);
   f.setVisible(true);
}
public static void main(String[] args) {
   new MyTable();  }}
```

JPROGRESSBAR CLASS:

The JProgressBar class is used to display the progress of the task.

Commonly used Constructors of JProgressBar class:

- **JProgressBar():** is used to create a horizontal progress bar but no string text.
- **JProgressBar(int min, int max):** is used to create a horizontal progress bar with the specified minimum and maximum value.
- **JProgressBar(int orient):** is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
- **JProgressBar(int orient, int min, int max):** is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used methods of JProgressBar class:

1) **public void setStringPainted(boolean b):** is used to determine whether string should be displayed
2) **public void setString(String s):** is used to set value to the progress string.
3) **public void setOrientation(int orientation):** is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants..
4) **public void setValue(int value):** is used to set the current value on the progress bar.

LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout

9. javax.swing.SpringLayout etc.

BORDERLAYOUT:

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

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**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

GRIDLAYOUT:

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class:

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

FLOWLAYOUT:

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class:

1. **public static final int LEFT**
2. **public static final int RIGHT**

3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class:

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

CARDLAYOUT CLASS:

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class:

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class:

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

BOXLAYOUT CLASS:

The BoxLayout is used to arrange the components either vertically or horizontally. Forthis purpose, BoxLayout provides four constants. They are as follows:

Note: BoxLayout class is found in javax.swing package.

1. **public static final int X_AXIS**
2. **public static final int Y_AXIS**
3. **public static final int LINE_AXIS**
4. **public static final int PAGE_AXIS**

Constructor of BoxLayout class:

BoxLayout(Container c, int axis): creates a box layout that arranges the components with the given axis.

Example of BorderLayout class:

```
import java.awt.*;

import javax.swing.*;

  public class Border {

JFrame f;

Border(){

f=new JFrame();

   JButton b1=new JButton("NORTH");;

   JButton b2=new JButton("SOUTH");;

   JButton b3=new JButton("EAST");;

   JButton b4=new JButton("WEST");;

   JButton b5=new JButton("CENTER");;



   f.add(b1,BorderLayout.NORTH);

   f.add(b2,BorderLayout.SOUTH);

   f.add(b3,BorderLayout.EAST);
```

```
        f.add(b4,BorderLayout.WEST);
        f.add(b5,BorderLayout.CENTER);
        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new Border();
    }
```

Example of GridLayout class:

```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout{
JFrame f;
MyGridLayout(){
   f=new JFrame();
   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
      JButton b6=new JButton("6");
      JButton b7=new JButton("7");
   JButton b8=new JButton("8");
      JButton b9=new JButton("9");
   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);  f.add(b6);f.add(b7);f.add(b8);f.add(b9);
     f.setLayout(new GridLayout(3,3));
   //setting grid layout of 3 rows and 3 columns
   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new MyGridLayout();
}
}
```

Example of FlowLayout class:

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

public class MyFlowLayout{
    JFrame f;
    MyFlowLayout(){
        f=new JFrame();

        JButton b1=new JButton("1");
        JButton b2=new JButton("2");
        JButton b3=new JButton("3");
        JButton b4=new JButton("4");
        JButton b5=new JButton("5");
        f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

        f.setLayout(new FlowLayout(FlowLayout.RIGHT));
        //setting flow layout of right alignment

        f.setSize(300,300);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new MyFlowLayout();
    }
}
```

Example of BoxLayout

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

public class BoxLayoutExample1 extends Frame {
 Button buttons[];
 public BoxLayoutExample1 () {
  buttons = new Button [5];
  for (int i = 0;i<5;i++) {
```

```
        buttons[i] = new Button ("Button "
    + (i + 1));

        add (buttons[i]);

      }


    setLayout (new BoxLayout (this, BoxL
    ayout.Y_AXIS));

    setSize(400,400);

    setVisible(true);
}

public static void main(String args[])
{

BoxLayoutExample1 b=new BoxLayout
Example1();

}

}
```

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

ADVANTAGE OF APPLET

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

DRAWBACK OF APPLET

- Plugin is required at client browser to execute applet.

LIFECYCLE OF JAVA APPLET

1. Applet is initialized.
2. Applet is started.
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.

Lifecycle methods for Applet:The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

JAVA.AWT.COMPONENT CLASS

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

HOW TO RUN AN APPLET?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java

import java.applet.Applet;

import java.awt.Graphics;

public class First extends Applet{

public void paint(Graphics g){

g.drawString("welcome",150,150);

}
```

```
}
```

myapplet.html

```
<html>

<body>

<applet code="First.class" width="300" height="300">

</applet>

</body>

</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet{

public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}
}
/*
<applet code="First.class" width="300" height
="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java

c:\>appletviewer First.java
```

DISPLAYING GRAPHICS IN APPLET

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.

4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
  public class GraphicsDemo extends Applet{

public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
```

```
myapplet.html
<html>
<body>
<applet code="GraphicsDemo.class" width="3
00" height="300">
</applet>
</body>
</html>
```

Animation in Applet:

Applet is mostly used in games and animation. For this purpose image is required to be moved.

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet {

  Image picture;

  public void init() {
    picture =getImage(getDocumentBase(),"bike_1.gif");
  }
```

```
 public void paint(Graphics g) {
  for(int i=0;i<500;i++){
   g.drawImage(picture, i,30, this);

   try{Thread.sleep(100);}catch(Exception e){}
  }
 }
}
```

In the above example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

ASSIGNMENTS

1. Write a Java program that takes a number as input and prints its multiplication table.
2. Write a Java program to compute the specified expressions and print the output.
3. Write a Java program to print the area and perimeter of a circle
4. Write a Java program that takes three numbers as input to calculate and print the average of the numbers.
5. Write a Java program to swap two variables.
6. Write a Java program to print a face.
7. Write a Java program that reads a number in inches, converts it to meters.
8. Write a Java program to takes the user for a distance (in meters) and the time was taken (as three numbers: hours, minutes, seconds), and display the speed, in meters per second, kilometers per hour and miles per hour (hint: 1 mile = 1609 meters).
9. Write a Java program that accepts two integers from the user and then prints the sum, the difference, the product, the average, the distance (the difference

between integer), the maximum (the larger of the two integers), the minimum (smaller of the two integers).

10 Write a Java program to calculate the average value of array elements.

11 Write a Java program to remove a specific element from an array.

12 Write a Java program to find the maximum and minimum value of an array

13 Write a Java program to find the second largest element in an array.

14 Write a Java program to find the second smallest element in an array.

15 Write a Java program to add two matrices of the same size.

16 Write a Java program to convert all the characters in a string to lowercase

17 Write a Java program to concatenate a given string to the end of another string

18 Write a Java program to check whether a given string ends with the contents of string

19 Write a java program to print current date and time in the specified format.

20 Write a Java program to get the last index of a string within a string

21 Write a java program to get the length of a given string

22 Write a Java program to replace all the 'd' characters with 'f' characters.

23 Write a Java program to get a substring of a given string between two specified positions

24 Write a Java program to trim any leading or trailing whitespace from a given string.

25 Write Java program to create user defined exception to check accepted mobile number is valid or not?

PROJECT

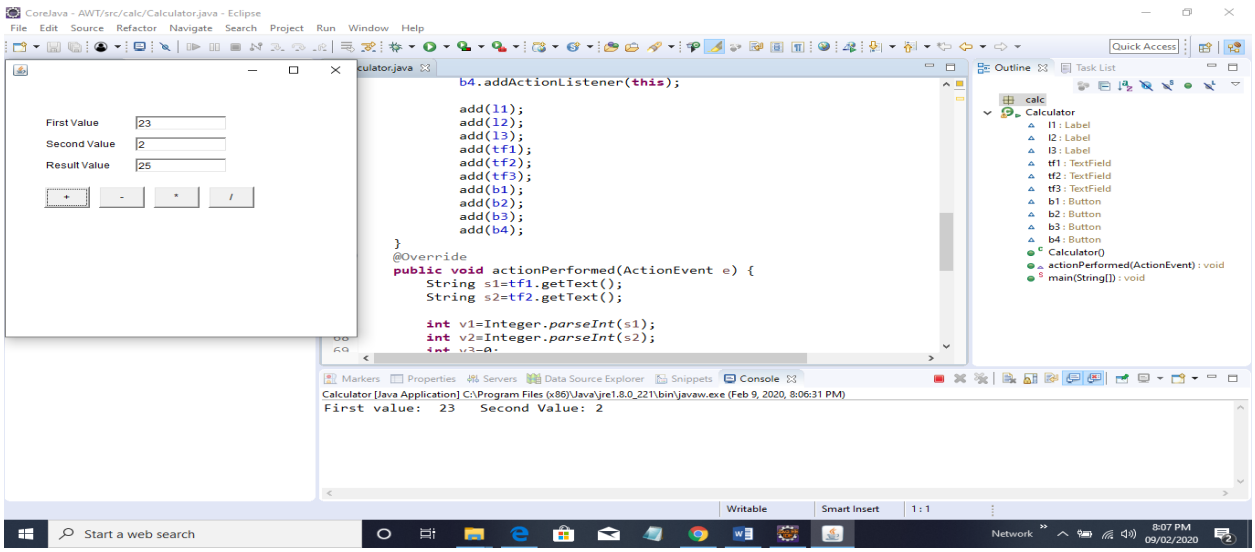
Calculator:

```
package calc;
import java.awt.Button;
import java.awt.Frame;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class Calculator extends Frame implements ActionListener {
	Label l1,l2,l3;
	TextField tf1,tf2,tf3;
	Button b1,b2,b3,b4;
	public Calculator() {
			setSize(400,400);
			setLayout(null);
			setVisible(true);

			l1=new Label("First Value");
			l1.setBounds(50, 80, 80, 20);
			l2=new Label("Second Value");
			l2.setBounds(50, 110, 80, 20);
			l3=new Label("Result Value");
			l3.setBounds(50, 140, 80, 20);
			tf1=new TextField();
			tf1.setBounds(150, 80, 100, 20);
			tf2=new TextField();
```

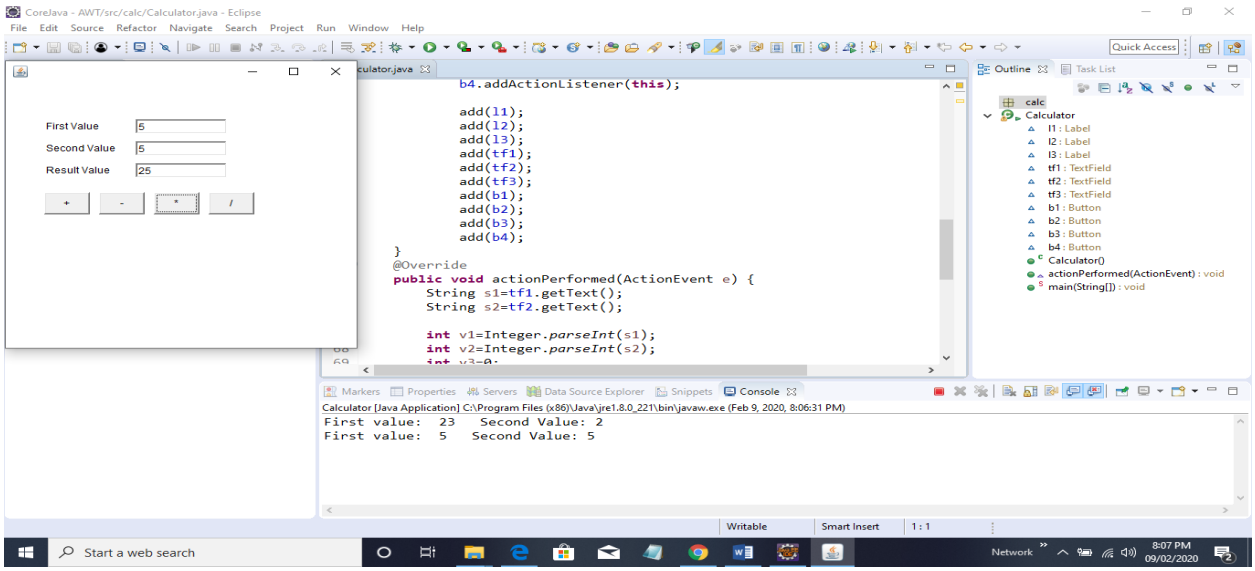
```
                tf2.setBounds(150, 110, 100, 20);
                tf3=new TextField();
                tf3.setBounds(150, 140, 100, 20);
                b1=new Button("+");
                b1.setBounds(50, 180, 50, 30);
                b1.addActionListener(this);
                b2=new Button("-");
                b2.setBounds(110, 180, 50, 30);
                b2.addActionListener(this);
                b3=new Button("*");
                b3.setBounds(170, 180, 50, 30);
                b3.addActionListener(this);
                b4=new Button("/");
                b4.setBounds(230, 180, 50, 30);
                b4.addActionListener(this);
                add(l1);
                add(l2);
                add(l3);
                add(tf1);
                add(tf2);
                add(tf3);
                add(b1);
                add(b2);
                add(b3);
                add(b4);
        }
        @Override
        public void actionPerformed(ActionEvent e) {
```

```
            String s1=tf1.getText();
            String s2=tf2.getText();

            int v1=Integer.parseInt(s1);
            int v2=Integer.parseInt(s2);
            int v3=0;
            if(e.getSource()==b1) {
                  v3=v1+v2;
            }
            if(e.getSource()==b2) {
                  v3=v1-v2;
            }
            if(e.getSource()==b3) {
                  v3=v1*v2;
            }
            if(e.getSource()==b4) {
                  v3=v1/v2;
            }
            tf3.setText(String.valueOf(v3)); //converts different types of values into string
            System.out.println("First value:  "+s1+"   Second Value: "+s2);
      }
      public static void main(String[] args) {

            new Calculator();
      }


}
```

Result 1: Addition



Result 2:Multipliction



Project 2 :

Registration form

```
package Test;
import java.awt.Button;
import java.awt.Checkbox;
import java.awt.Frame;
```

```
import java.awt.Label;
import java.awt.TextArea;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JRadioButton;
import javax.swing.plaf.basic.BasicBorders.RadioButtonBorder;
public  class RegisterationForm extends Frame implements ActionListener {
        Label l1,l2,l3,l4,l5,l6;
        Button b1;
        TextField tf1,tf2,tf3;
        TextArea ta1;
        Checkbox c1,c2,c3;
        JRadioButton r1,r2,r3;
         RegisterationForm()
         {
                setSize(600,700);
                setVisible(true);
                setLayout(null);

                l1=new Label("First name");
                l1.setBounds(50, 50, 100, 30);
                l2=new Label("Last name");
                l2.setBounds(50, 100, 100, 30);
                l3=new Label("Email_id");
                l3.setBounds(50, 150, 100, 30);
                l4=new Label("Address");
                l4.setBounds(50, 200, 100, 30);
```

```
l5=new Label("Gender");
l5.setBounds(50, 300, 100, 30);
l6=new Label("Hobbies");
l6.setBounds(50, 400, 100, 30);
tf1=new TextField();
tf1.setBounds(150, 50, 100, 30);
tf2=new TextField();
tf2.setBounds(150, 100, 100, 30);
tf3=new TextField();
tf3.setBounds(150, 150, 100, 30);
ta1=new TextArea();
ta1.setBounds(150, 200, 200, 70);
r1=new JRadioButton("male");
r1.setBounds(150, 300, 100, 50);
r2=new JRadioButton("female");
r2.setBounds(150, 330, 100, 50);
r3=new JRadioButton("other");
r3.setBounds(150, 360, 100, 50);
c1=new Checkbox("writing");
c1.setBounds(150, 400, 100, 50);
c2=new Checkbox("dancing");
c2.setBounds(150, 430, 100, 50);
c3=new Checkbox("reading");
c3.setBounds(150, 460, 100, 50);
b1=new Button("submit");
b1.setBounds(50, 600, 100, 30);
b1.addActionListener(this);
add(b1);
```

```
        add(l1);
        add(l2);
        add(l3);
        add(l4);
        add(l5);
        add(l6);
        add(tf1);
        add(tf2);
        add(tf3);
        add(ta1);
        add(c1);
        add(c2);
        add(c3);
        add(r1);
        add(r2);
        add(r3);

    }
    @Override
        public void actionPerformed(ActionEvent e) {
        String s1=tf1.getText();
        if(e.getSource()==b1)
        {

         System.out.println("Hello"+" "+s1+" Have a nice Day :)");
    }
    }
```

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

}

}
```

Result:

Fill the entire form and when you click on submit button , on console it will print the great message for the person who is registered..