What do you know about Java?

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. Some features include Object Oriented, Platform Independent, Robust, and Interpreted, Multi-threaded

IDE: Netbeans, Eclipse, etc.

An integrated development environment (IDE) is a programming environment that has been packaged as an application program, typically consisting of a code editor, a [compiler](http://whatis.techtarget.com/definition/compiler), a debugger, and a graphical user interface ([GUI](http://searchwindevelopment.techtarget.com/definition/GUI)) builder

Compiler:

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's [processor](http://searchcio-midmarket.techtarget.com/definition/processor) uses.

# Object and Class in Java

* **Object –** Object is an instance of a class ,Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors -wagging, barking, eating. An object is an instance of a class.
* **Class -** A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.

Let us now look deep into what are objects. If we consider the real-world we can find many objects around us, Cars, Dogs, Humans, etc. All these objects have a state and behavior.

If we consider a dog, then its state is - name, breed, color, and the behavior is - barking, wagging, running

If you compare the software object with a real world object, they have very similar .characteristics.

Software objects also have a state and behavior. **A software object's state is stored in fields and behavior is shown via methods.**

### Class in Java

A class is a group of objects that has common properties. It is a template or blueprint from which objects are created.

A class in java can contain:

* **data member**
* **method**
* **constructor**
* **block**
* **class and interface**

### Syntax to declare a class:

1. class <class\_name>{
2. data member;
3. method;
4. }

### Simple Example of Object and Class

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the objects value.

1. class Student1{
2. int id;//data member (also instance variable)
3. String name;//data member(also instance variable)
5. public static void main(String args[]){
6. Student1 s1=new Student1();//creating an object of Student
7. System.out.println(s1.id);
8. System.out.println(s1.name);
9. }
10. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student1)

Output:0 null

**What are the different ways to create an object in Java?**

|  |
| --- |
| There are many ways to create an object in java. They are:   * By new keyword * By newInstance() method * By clone() method * By factory method etc.   **1) new keyword**  This approach we commonly use in Java to create an object.  Example :   1. ArrayList<String> list = new ArrayList<String>();   **2) Class.forName() / Reflection**  This approach is also known as way of creating an object using reflection API. This simply returns the Class object associated with the class given. Commonly used in JDBC connection.  Example:   1. SampleClass sampleObj = Class.forName("SampleClass")   **3) clone()**  This approach is used to create a copy of an object in Java.  Example:   1. ArrayList<String> list = new ArrayList<String>(); 2. .. 3. .. 4. ArrayList<String> newList= null; 5. newList= list.clone();   **4) deserialization()**  This is an approach where we get object from its serialized form and convert/deserialize  Example:   1. ObjectInputStream ois = new ObjectInputStream("<some inputstream obj>"); 2. Object obj = ois.readObject();   **5) newInstance() / with ClassLoader way**  This approach is some what related to the 2nd one. Here,  Example:   1. MyAppClass appObj = (MyAppClass)sampleObj.newInstance(); |

**Annonymous object**

|  |
| --- |
| Annonymous simply means nameless.An object that have no reference is known as annonymous object. |
| If you have to use an object only once, annonymous object is a good approach. |

1. class Calculation{
3. void fact(int  n){
4. int fact=1;
5. for(int i=1;i<=n;i++){
6. fact=fact\*i;
7. }
8. System.out.println("factorial is "+fact);
9. }
11. public static void main(String args[]){
12. new Calculation().fact(5);//calling method with annonymous object
13. }
14. }

Output:Factorial is 120

Bhag milka bhag

# Polymorphism in Java

Created functions or reference variable will behaves differently is a concept of Polymorphism

We **can create functions or reference variables which behaves differently in different programmatic context**.

In java language, polymorphism is essentially considered into two versions.

1. Compile time polymorphism (static binding or method overloading)
2. Runtime polymorphism (dynamic binding or method overriding)

Method overloading

If a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behaviour of the method because its name differs. So, we perform method overloading to figure out the program quickly.

## Advantage of method overloading?

Method overloading **increases the readability of the program**.

### Different ways to overload the method

|  |
| --- |
| There are two ways to overload the method in java |

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

#### In java, Methood Overloading is not possible by changing the return type of the method.

### 1)Example of Method Overloading by changing the no. of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

1. class Calculation{
2. void sum(int a,int b){System.out.println(a+b);}
3. void sum(int a,int b,int c){System.out.println(a+b+c);}
5. public static void main(String args[]){
6. Calculation obj=new Calculation();
7. obj.sum(10,10,10);
8. obj.sum(20,20);
10. }
11. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Calculation)

Output:30

40

### 2)Example of Method Overloading by changing data type of argument

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two double arguments.

1. class Calculation2{
2. void sum(int a,int b){System.out.println(a+b);}
3. void sum(double a,double b){System.out.println(a+b);}
5. public static void main(String args[]){
6. Calculation2 obj=new Calculation2();
7. obj.sum(10.5,10.5);
8. obj.sum(20,20);
10. }
11. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Calculation2)

Output:21.0

40

### Que) Why Method Overloaing is not possible by changing the return type of method?

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

1. class Calculation3{
2. int sum(int a,int b){System.out.println(a+b);}
3. double sum(int a,int b){System.out.println(a+b);}
5. public static void main(String args[]){
6. Calculation3 obj=new Calculation3();
7. int result=obj.sum(20,20); //Compile Time Error
9. }
10. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Calculation3)

int result=obj.sum(20,20); //Here how can java determine which sum() method should be called

### Can we overload main() method?

Yes, by method overloading. You can have any number of main methods in a class by method overloading. Let's see the simple example:

1. class Overloading1{
2. public static void main(int a){
3. System.out.println(a);
4. }
6. public static void main(String args[]){
7. System.out.println("main() method invoked");
8. main(10);
9. }
10. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Overloading1)

Output:main() method invoked

10

# Method Overriding in Java

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

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

### Usage of Java Method Overriding

* Method overriding is used to provide specific implementation of a method that is already provided by its super class.
* Method overriding is used for runtime polymorphism

#### Rules for Java Method Overriding

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).

### Understanding the problem without method overriding

Let's understand the problem that we may face in the program if we don't use method overriding.

1. class Vehicle{
2. void run(){System.out.println("Vehicle is running");}
3. }
4. class Bike extends Vehicle{
6. public static void main(String args[]){
7. Bike obj = new Bike();
8. obj.run();
9. }
10. }

Output:Vehicle is running

Problem is that I have to provide a specific implementation of run() method in subclass that is why we use method overriding.

### Example of method overriding

In this example, we have defined the run method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

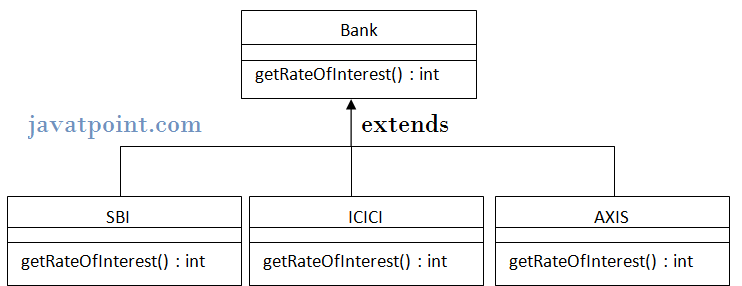
1. class Vehicle{
2. void run(){System.out.println("Vehicle is running");}
3. }
4. class Bike2 extends Vehicle{
5. void run(){System.out.println("Bike is running safely");}
7. public static void main(String args[]){
8. Bike2 obj = new Bike2();
9. obj.run();
10. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Bike2)

Output:Bike is running safely

### Real example of Java Method Overriding

Consider a scenario, Bank is a class that provides functionality to get rate of interest. But, rate of interest varies according to banks. For example, SBI, ICICI and AXIS banks could provide 8%, 7% and 9% rate of interest.



class Bank{

int getRateOfInterest(){return 0;}

}

  class SBI extends Bank{

int getRateOfInterest(){return 8;}

}

  class ICICI extends Bank{

int getRateOfInterest(){return 7;}

}

class AXIS extends Bank{

int getRateOfInterest(){return 9;}

}

 class Test2{

public static void main(String args[]){

SBI s=new SBI();

ICICI i=new ICICI();

AXIS a=new AXIS();

System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());

System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());

System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());

}

}

### Why we cannot override static method?

Because static method is bound with class whereas instance method is bound with object. Static belongs to class area and instance belongs to heap area.

### Can we override java main method?

No, because main is a static method.

# Encapsulation in Java

**Encapsulation in java** is a *process of wrapping code and data together into a single unit*, for example capsule i.e. mixed of several medicines.

To achieve encapsulation in Java

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values.

## Benefits of Encapsulation:

* The fields of a class can be made read-only or write-only.
* A class can have total control over what is stored in its fields.
* The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code.

1:Create private data members

2:right click on inside of the class and select “Source” 🡪Generate getters and setters option

3:we can see the datamember method names and create another class and set the name and get the name correspondingly.

EX: **public** **class** Encapsulation {

**public** String getAddress() {

**return** Address;

}

**public** **void** setAddress(String address) {

Address = address;

}

**public** **int** getEmpid() {

**return** empid;

}

**public** **void** setEmpid(**int** empid) {

**this**.empid = empid;

}

**private** String mohan;

**private** String Address;

**private** **int** empid;

**public** String getMohan() {

**return** mohan;

}

**public** **void** setMohan(String mohan) {

**this**.mohan = mohan;

}

**public** **class** EncapsulationGetter {

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

// **TODO** Auto-generated method stub

// EncapsulationGetter get = new EncapsulationGetter();

Encapsulation get = **new** Encapsulation();

get.setMohan("hello");

get.setMohan("Rohan");

System.*out*.println(get.getMohan());

get.setMohan("Ranjan");

System.*out*.println(get.getMohan());

get.setAddress("Malavalli");

System.*out*.println(get.getAddress());

get.setEmpid(21);

System.*out*.println(get.getEmpid());

}

# Constructor in Java

**Constructor in java** is a *special type of method* that is used to initialize the object.

Java constructor is *called 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**

There are basically two rules defined for the constructor.

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

**Types of java constructors**

There are two types of constructors:

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

### Q) Does constructor return any value?

**Ans:**yes, that is current class instance (You cannot use return type yet it returns a value).

### Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling method etc. You can perform any operation in the constructor as you perform in the method.

## Java Default Constructor

|  |
| --- |
| A constructor that have no parameter is known as default constructor. |

### Syntax of default constructor:

1. <class\_name>(){}

## Example of default constructor

|  |
| --- |
| In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation. |

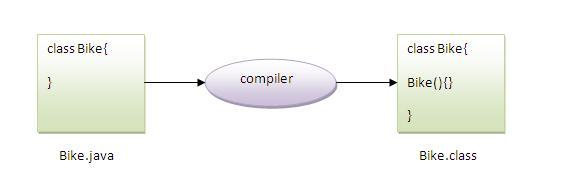
1. class Bike1{
2. Bike1(){System.out.println("Bike is created");}
3. public static void main(String args[]){
4. Bike1 b=new Bike1();
5. }
6. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Bike1)

Output:

Bike is created

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



### Q) What is the purpose of default constructor?

Default constructor provides the default values to the object like 0, null etc. depending on the type.

### Example of default constructor that displays the default values

1. class Student3{
2. int id;
3. String name;
5. void display(){System.out.println(id+" "+name);}
7. public static void main(String args[]){
8. Student3 s1=new Student3();
9. Student3 s2=new Student3();
10. s1.display();
11. s2.display();
12. }
13. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student3)

Output:

0 null

0 null

**Explanation:**In the above class,you are not creating any constructor so compiler provides you a default constructor.Here 0 and null values are provided by default constructor.

### Java parameterized constructor

|  |
| --- |
| A constructor that have parameters is known as parameterized constructor. |

### Why use parameterized constructor?

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

### Example of parameterized constructor

|  |
| --- |
| In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor. |

1. class Student4{
2. int id;
3. String name;
5. Student4(int i,String n){
6. id = i;
7. name = n;
8. }
9. void display(){System.out.println(id+" "+name);}
11. public static void main(String args[]){
12. Student4 s1 = new Student4(111,"Karan");
13. Student4 s2 = new Student4(222,"Aryan");
14. s1.display();
15. s2.display();
16. }
17. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student4)

Output:

111 Karan

222 Aryan

## Constructor Overloading in Java

|  |
| --- |
| Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking into account the number of parameters in the list and their type. |

### Example of Constructor Overloading

1. class Student5{
2. int id;
3. String name;
4. int age;
5. Student5(int i,String n){
6. id = i;
7. name = n;
8. }
9. Student5(int i,String n,int a){
10. id = i;
11. name = n;
12. age=a;
13. }
14. void display(){System.out.println(id+" "+name+" "+age);}
16. public static void main(String args[]){
17. Student5 s1 = new Student5(111,"Karan");
18. Student5 s2 = new Student5(222,"Aryan",25);
19. s1.display();
20. s2.display();
21. }
22. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Student5)

Output:

111 Karan 0

222 Aryan 25

## Difference between constructor and method in java

There are many differences between constructors and methods. They are given below.

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

**Access Modifiers:**

Access modifiers are keywords that we add to those definitions to change their meanings.

Access modifiers can be used to define Access control for classes, methods and variables.

A class cannot be private or protected.

Private Access modifiers: can only be accessed with In a class

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

 class A{

 private int data=40;

 private void msg(){System.out.println("Hello java");}

 }



 public class Simple{

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

    A obj=new A();

    System.out.println(obj.data);//Compile Time Error

    obj.msg();//Compile Time Error

    }

**Default Access modifier:**

If we will not declare any modifier, java will be treated as Default Modifiers

This can be accessible with the package.

1. //save by A.java
2. package pack;
3. class A{
4. void msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. package mypack;
8. import pack.\*;
9. class B{
10. public static void main(String args[]){
11. A obj = new A();//Compile Time Error
12. obj.msg();//Compile Time Error
13. }
14. }

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

**Protected Modifier:**

Which can be accicable with in package and also outside of the package through inheritance

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

1. //save by A.java
2. package pack;
3. public class A{
4. protected void msg(){System.out.println("Hello");}
5. }
6. //save by B.java
7. package mypack;
8. import pack.\*;
10. class B extends A{
11. public static void main(String args[]){
12. B obj = new B();
13. obj.msg();
14. }
15. }

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | | The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers. |  Example of public access modifier  1. //save by A.java 3. package pack; 4. public class A{ 5. public void msg(){System.out.println("Hello");} 6. } 7. //save by B.java 9. package mypack; 10. import pack.\*; 12. class B{ 13. public static void main(String args[]){ 14. A obj = new A(); 15. obj.msg(); 16. } 17. }   Output:Hello Understanding all java access modifiers Let's understand the access modifiers by a simple table.   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** | | **Private** | Y | N | N | N | | **Default** | Y | Y | N | N | | **Protected** | Y | Y | Y | N | | **Public** | Y | Y | Y | Y |  Java access modifiers with method overriding If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.   1. class A{ 2. protected void msg(){System.out.println("Hello java");} 3. } 5. public class Simple extends A{ 6. void msg(){System.out.println("Hello java");}//C.T.Error 7. public static void main(String args[]){ 8. Simple obj=new Simple(); 9. obj.msg(); 10. } 11. }  |  | | --- | | The default modifier is more restrictive than protected. That is why there is compile time error. |  Java static keyword The **static keyword** in java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than instance of the class.  The static can be:   1. variable (also known as class variable) 2. method (also known as class method) 3. block 4. nested class   **Java static variable**  If you declare any variable as static, it is known static variable.   * The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees,college name of students etc. * The static variable gets memory only once in class area at the time of class loading.   It makes your program **memory efficient** (i.e it saves memory).  Suppose there are 500 students in my college, now all instance data members will get memory each time when object is created.All student have its unique rollno and name so instance data member is good.Here, college refers to the common property of all objects.If we make it static,this field will get memory only once. Java static property is shared to all objects.Example of static variable  1. //Program of static variable 3. class Student8{ 4. int rollno; 5. String name; 6. static String college ="ITS"; 8. Student8(int r,String n){ 9. rollno = r; 10. name = n; 11. } 12. void display (){System.out.println(rollno+" "+name+" "+college);} 14. public static void main(String args[]){ 15. Student8 s1 = new Student8(111,"Karan"); 16. Student8 s2 = new Student8(222,"Aryan"); 18. s1.display(); 19. s2.display(); 20. } 21. }   Output:111 Karan ITS  222 Aryan ITS  Static variable can be accessed through out the program , we can access static variable in 3 ways  EX:  Class A{  Static int A=20;  Public void main(String[] args){  Syso(A);--------Directly  Syso(A.A);---- using Class  A dna = new A();  Dna.A;-using Object  }  Static Variable stores its memory in the **Method area** when JVM starts.outside of the class we can call the static variable using 2 ways  1:using Class  ClassA.a  2: using Reference.  ClassA db = new ClassA();  Db.a;  **2) Java static method**  If you apply static keyword with any method, it is known as static method.   * A static method belongs to the class rather than object of a class. * A static method can be invoked without the need for creating an instance of a class. * static method can access static data member and can change the value of it.  Example of static method  1. //Program of changing the common property of all objects(static field). 3. class Student9{ 4. int rollno; 5. String name; 6. static String college = "ITS"; 8. static void change(){ 9. college = "BBDIT"; 10. } 12. Student9(int r, String n){ 13. rollno = r; 14. name = n; 15. } 17. void display (){System.out.println(rollno+" "+name+" "+college);} 19. public static void main(String args[]){ 20. Student9.change(); 22. Student9 s1 = new Student9 (111,"Karan"); 23. Student9 s2 = new Student9 (222,"Aryan"); 24. Student9 s3 = new Student9 (333,"Sonoo"); 26. s1.display(); 27. s2.display(); 28. s3.display(); 29. } 30. }   Output:111 Karan BBDIT  222 Aryan BBDIT  333 Sonoo BBDIT Final Keyword In Java The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:   1. variable 2. method 3. class  1) Java final variable If you make any variable as final, you cannot change the value of final variable(It will be constant). Example of final variable 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.   1. class Bike9{ 2. final int speedlimit=90;//final variable 3. void run(){ 4. **speedlimit=400;** 5. } 6. public static void main(String args[]){ 7. Bike9 obj=new  Bike9(); 8. obj.run(); 9. } 10. }//end of class  \   **2) Java final method**  If you make any method as final, you cannot override it.  **Example of final method**   1. class Bike{ 2. final void run(){System.out.println("running");} 3. } 5. class Honda extends Bike{ 6. **void run(){System.out.println("running safely with 100kmph");}** 8. public static void main(String args[]){ 9. Honda honda= new Honda(); 10. honda.run(); 11. } 12. }   Compile Error. 3) Java final class If you make any class as final, you cannot extend it. Example of final class  1. final class Bike{} 3. **class Honda1 extends Bike{** 4. void run(){System.out.println("running safely with 100kmph");} 6. public static void main(String args[]){ 7. Honda1 honda= new Honda(); 8. honda.run(); 9. } 10. }   [Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Honda1)  Output:Compile Time Error What is Data Type? A type of data that a variable can hold such as *integer*, *floating-point* numbers, *character, boolean* or an *alphanumeric string.*  Byte: 1 byte = 8 bits=2^8 =256= -128 to – 1 and 0 to 127  Ex:byte I = 2, 3 Wrong ex= 128 ,300, -136 etc..   |  | | --- | | In java, there are two types of data types   * primitive data types * non-primitive data types |   datatype in java   |  |  |  | | --- | --- | --- | | **Data Type** | **Default Value** | **Default size** | | boolean | False | 1 bit | | char | '\u0000' | 2 byte | | byte | 0 | 1 byte | | short | 0 | 2 byte | | int | 0 | 4 byte | | long | 0L | 8 byte | | float | 0.0f | 4 byte | | double | 0.0d | 8 byte |   **Why char uses 2 byte in java and what is \u0000 ?**  because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.To get detail about Unicode see below. |
|  |

**Variables:**

Variables are the names which can be used to store some values.

int a, b, c; // Declares three ints, a, b, and c.

int a = 10, b = 10; // Example of initialization

byte B = 22; // initializes a byte type variable B.

double pi = 3.14159; // declares and assigns a value of PI.

char a = 'a'; // the char variable a iis initialized with value 'a'

## Local variables:

* Local variables are declared in methods, constructors, or blocks.
* Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.
* Acces s modifiers cannot be used for local variables.
* Local variables are visible only within the declared method, constructor or block.
* Local variables are implemented at stack level internally.
* There is no default value for local variables so local variables should be declared and an initial value should be assigned before the first use.

## Example:

Here, *age* is a local variable. This is defined inside *pupAge()* method and its scope is limited to this method only.

public class Test{

public void pupAge(){

int age;

age = age + 7;

System.out.println("Puppy age is : " + age);

}

public static void main(String args[]){

Test test = new Test();

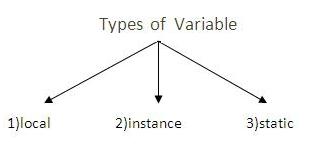
test.pupAge();

}

}

### Types of Variable

|  |
| --- |
| There are three types of variables in java   * local variable * instance variable * static variable |



#### 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

1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class

**Identifiers in Java**

All Java components require names. Name used for classes, methods, interfaces and variables are called **Identifier**. Identifier must follow some rules. Here are the rules:

* All identifiers must start with either a letter( a to z or A to Z ) or currency character($) or an underscore.
* After the first character, an identifier can have any combination of characters.
* A Java **keyword** cannot be used as an identifier.
* Identifiers in Java are case sensitive, foo and Foo are two different identifiers.

### Type Casting

Assigning a value of one type to a variable of another type is known as **Type Casting**.

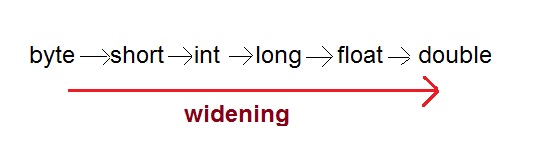
**Example :**

int x = 10;

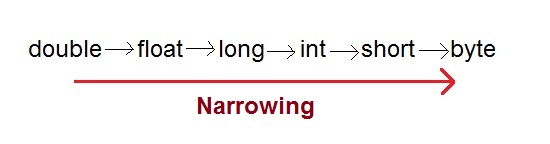
byte y = (byte)x;

In Java, type casting is classified into two types,

* Widening Casting(Implicit)



* Narrowing Casting(Explicitly done)



#### Widening or Automatic type converion

Automatic Type casting take place when,

* the two types are compatible
* the target type is larger than the source type

**Example :**

public class Test

{

public static void main(String[] args)

{

int i = 100;

long l = i; **//no explicit type casting required**

float f = l; **//no explicit type casting required**

System.out.println("Int value "+i);

System.out.println("Long value "+l);

System.out.println("Float value "+f);

}

}

**Output :**

Int value 100

Long value 100

Float value 100.0

#### Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.

**Example :**

public class Test

{

public static void main(String[] args)

{

double d = 100.04;

long l = (long)d; **//explicit type casting required**

int i = (int)l; **//explicit type casting required**

System.out.println("Double value "+d);

System.out.println("Long value "+l);

System.out.println("Int value "+i);

}

}

**Output :**

Double value 100.04

Long value 100

Int value 100

### Java Operators

Java provides a rich set of operators enviroment. Java operators can be devided into following categories

* Arithmetic operators
* Relation operators
* Logical operators
* Bitwise operators
* Assignment operators
* Conditional operators
* Misc operators

#### Arithmetic operators

Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

|  |  |
| --- | --- |
| **Operator** | **description** |
| + | adds two operands |
| - | subtract second operands from first |
| \* | multiply two operand |
| / | divide numerator by denumerator |
| % | remainder of division |
| ++ | Increment operator increases integer value by one |
| -- | Decrement operator decreases integer value by one |

#### Relation operators

The following table shows all relation operators supported by Java.

|  |  |
| --- | --- |
| **Operator** | **description** |
| == | Check if two operand are equal |
| != | Check if two operand are not equal. |
| > | Check if operand on the left is greater than operand on the right |
| < | Check operand on the left is smaller than right operand |
| >= | check left operand is greater than or equal to right operand |
| <= | Check if operand on left is smaller than or equal to right operand |

#### Logical operators

Java supports following 3 logical operator. Suppose a=1 and b=0;

|  |  |  |
| --- | --- | --- |
| **Operator** | **description** | **example** |
| && | Logical AND | (a && b) is false |
| || | Logical OR | (a || b) is true |
| ! | Logical NOT | (!a) is false |

#### Bitwise operators

Java defines several bitwise operators that can be applied to the integer types long, int, short, char and byte

|  |  |
| --- | --- |
| **operator** | **description** |
| & | Bitwise AND |
| | | Bitwise OR |
| ^ | Bitwise exclusive OR |
| << | left shift |
| >> | right shift |

Now lets see truth table for bitwise &, | and ^

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **b** | **a & b** | **a | b** | **a ^ b** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

The bitwise shift operators shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted. Both operands have the same precedence.**Example**

a = 0001000

b= 2

a << b= 0100000

a >> b= 0000010

#### Assignment Operators

Assignment operator supported by Java are as follows

|  |  |  |
| --- | --- | --- |
| **operator** | **description** | **example** |
| = | assigns values from right side operands to left side operand | a=b |
| += | adds right operand to the left operand and assign the result to left | a+=b is same as a=a+b |
| -= | subtracts right operand from the left operand and assign the result to left operand | a-=b is same as a=a-b |
| \*= | multiply left operand with the right operand and assign the result to left operand | a\*=b is same as a=a\*b |
| /= | divides left operand with the right operand and assign the result to left operand | a/=b is same as a=a/b |
| %= | calculate modulus using two operands and assign the result to left operand | a%=b is same as a=a%b |

#### Misc operator

There are few other operator supported by java language.

#### Conditional operator

It is also known as ternary operator and used to evaluate Boolean expression

epr1 ? expr2 : expr3

If **epr1**Condition is true? Then value **expr2** : Otherwise value **expr3**

#### instanceOf operator

This operator is used for object reference variables. The operator checks whether the object is of particular type (class type or interface type)

# **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.

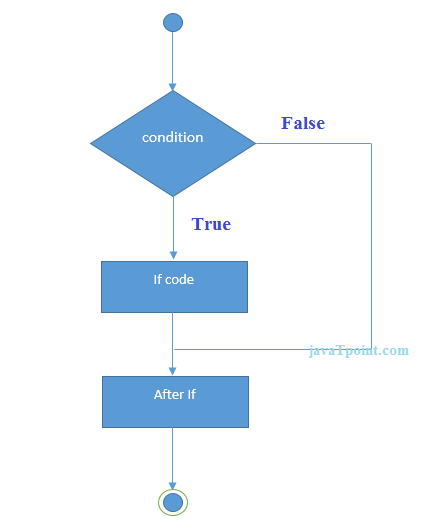
* if statement
* if-else statement
* nested if statement
* if-else-if ladder

## Java IF Statement

The if statement tests the condition. It executes the if statement if condition is true.

**Syntax:**

1. **if**(condition){
2. //code to be executed
3. }



**Example:**

1. **public** **class** IfExample {
2. **public** **static** **void** main(String[] args) {
3. **int** age=20;
4. **if**(age>18){
5. System.out.print("Age is greater than 18");
6. }
7. }
8. }

Output:

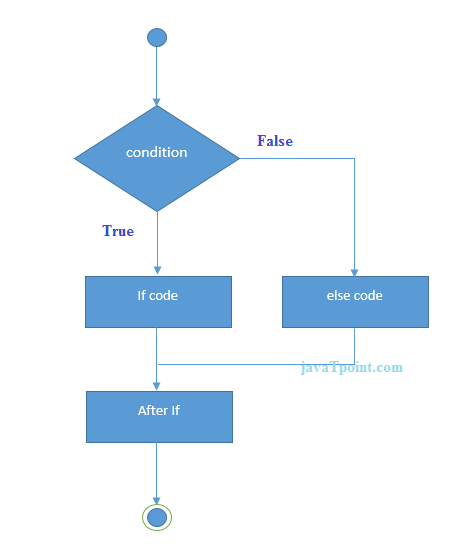
Age is greater than 18

## Java IF-else Statement

The if-else statement also tests the condition. It executes the if block if condition is true otherwise else block.

**Syntax:**

1. **if**(condition){
2. //code if condition is true
3. }**else**{
4. //code if condition is false
5. }



**Example:**

1. **public** **class** IfElseExample {
2. **public** **static** **void** main(String[] args) {
3. **int** number=13;
4. **if**(number%2==0){
5. System.out.println("even number");
6. }**else**{
7. System.out.println("odd number");
8. }
9. }
10. }

Output:

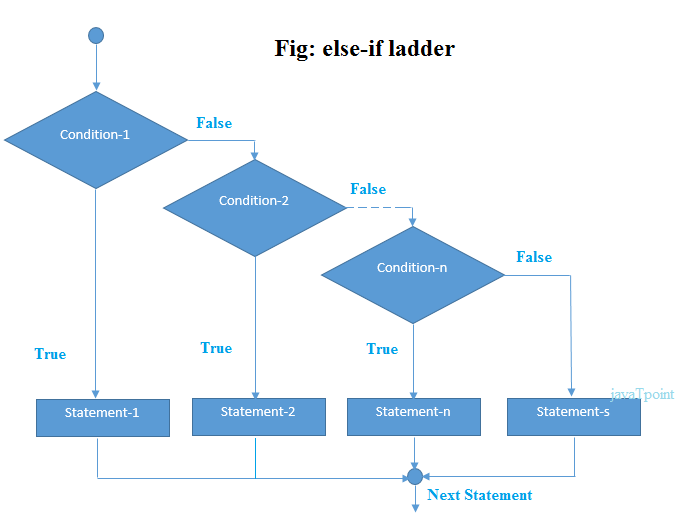
odd number

## Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

**Syntax:**

1. **if**(condition1){
2. //code to be executed if condition1 is true
3. }**else** **if**(condition2){
4. //code to be executed if condition2 is true
5. }
6. **else** **if**(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. **else**{
11. //code to be executed if all the conditions are false
12. }



**Example:**

1. **public** **class** IfElseIfExample {
2. **public** **static** **void** main(String[] args) {
3. **int** marks=65;
5. **if**(marks<50){
6. System.out.println("fail");
7. }
8. **else** **if**(marks>=50 && marks<60){
9. System.out.println("D grade");
10. }
11. **else** **if**(marks>=60 && marks<70){
12. System.out.println("C grade");
13. }
14. **else** **if**(marks>=70 && marks<80){
15. System.out.println("B grade");
16. }
17. **else** **if**(marks>=80 && marks<90){
18. System.out.println("A grade");
19. }**else** **if**(marks>=90 && marks<100){
20. System.out.println("A+ grade");
21. }**else**{
22. System.out.println("Invalid!");
23. }
24. }
25. }

Output:

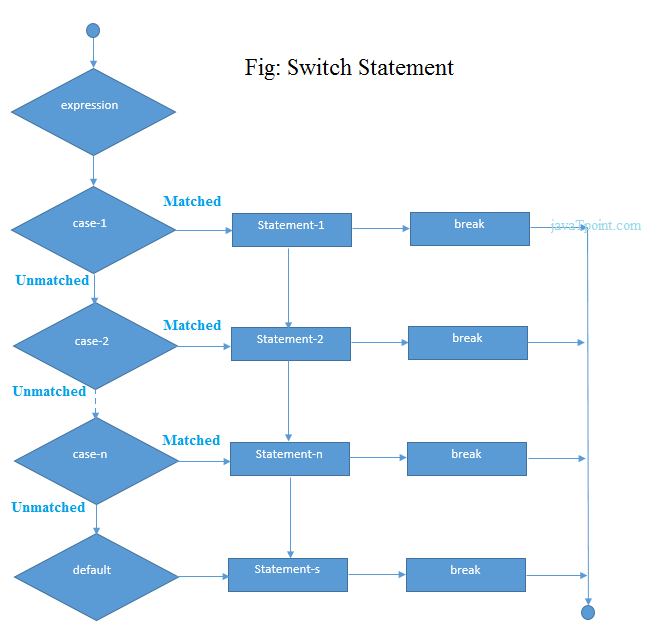
C grade

# **Java Switch Statement**

The Java switch statement is executes one statement from multiple conditions. It is like if-else-if ladder statement.

**Syntax:**

1. **switch**(expression){
2. **case** value1:
3. //code to be executed;
4. **break**;  //optional
5. **case** value2:
6. //code to be executed;
7. **break**;  //optional
8. ......
10. **default**:
11. code to be executed **if** all cases are not matched;
12. }



**Example:**

1. **public** **class** SwitchExample {
2. **public** **static** **void** main(String[] args) {
3. **int** number=20;
4. **switch**(number){
5. **case** 10: System.out.println("10");**break**;
6. **case** 20: System.out.println("20");**break**;
7. **case** 30: System.out.println("30");**break**;
8. **default**:System.out.println("Not in 10, 20 or 30");
9. }
10. }
11. }

Output:

20

## Java Switch Statement if fall-through

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

**Example:**

1. **public** **class** SwitchExample2 {
2. **public** **static** **void** main(String[] args) {
3. **int** number=20;
4. **switch**(number){
5. **case** 10: System.out.println("10");
6. **case** 20: System.out.println("20");
7. **case** 30: System.out.println("30");
8. **default**:System.out.println("Not in 10, 20 or 30");
9. }
10. }
11. }

Output:

20

30

Not in 10, 20 or 30

1. EX: **for**(**int** i=0; i <= 3 ; i++)
2. {
3. **switch**(i)
4. {
5. **case** 0:
6. System.out.println("i is 0");
7. **break**;
9. **case** 1:
10. System.out.println("i is 1");
11. **break**;
13. **case** 2:
14. System.out.println("i is 2");
15. **break**;
17. **default**:
18. System.out.println("i is grater than 2");
20. }

public class SwitchDemo {

public static void main(String[] args) {

int month = 8;

String monthString;

switch (month) {

case 1: monthString = "January";

break;

case 2: monthString = "February";

break;

case 3: monthString = "March";

break;

case 4: monthString = "April";

break;

case 5: monthString = "May";

break;

case 6: monthString = "June";

break;

case 7: monthString = "July";

break;

case 8: monthString = "August";

break;

case 9: monthString = "September";

break;

case 10: monthString = "October";

break;

case 11: monthString = "November";

break;

case 12: monthString = "December";

break;

default: monthString = "Invalid month";

break;

}

System.out.println(monthString);

}

}

**import** java.util.Scanner;

**public** **class** NumberOfDays {

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

            Scanner in = **new** Scanner(System.*in*);

            System.*out*.print("Enter month's number: ");

**int** monthNumber;

            monthNumber = in.nextInt();

**switch** (monthNumber) {

**case** 1:

**case** 3:

**case** 5:

**case** 7:

**case** 8:

**case** 10:

**case** 12:

                  System.*out*.println("Number of days: 31");

**break**;

**case** 4:

**case** 6:

**case** 9:

**case** 11:

                  System.*out*.println("Number of days: 30");

**break**;

**case** 2:

                  System.*out*.println("Number of days: 28");

**break**;

**default**:

                  System.*out*.println("Invalid month.");

**break**;

            }

      }

}

Enter month's number: 5

Number of days: 31

**package** javaStart;

**import** java.util.Scanner;

// This program is a simple calculator designed in java using switch case statement to perform some basic arithmetic operation on two integers entered by user.

//It enables the user to enter a choice that corresponds to one of the four basic arithmetic operation, addition, subtraction, multiplication and division.

**public** **class** SwitchCalculator{

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

Scanner sc= **new** Scanner(System.*in*);

System.*out*.println("Following options ");

System.*out*.println("1: Addition");

System.*out*.println("2: Subtraction");

System.*out*.println("3: Multiplication");

System.*out*.println("4: Division");

System.*out*.println("Please select the option");

**int** i = sc.nextInt();

System.*out*.println("Enter first number");

**int** a = sc.nextInt();

System.*out*.println("Enter second number");

**int** b = sc.nextInt();

**double** result=0;

**switch** (i) {

**case** 1:

result = a+b;

**break**;

**case** 2:

result = a-b;

**break**;

**case** 3:

result = a \*b;

**break**;

**case** 4:

**if** (b==0)

{

System.*out*.println("B value should not be zeo and its not divisible");

// break;

}

**else**

result = a/b;

**break**;

**default**:

System.*out*.println("you have entered a wrong choice");

}

System.*o ut*.println("The result of 2 number is :" +result);

}

}

# While loop in Java with examples

**What is while loop?**  
Like for loop, It also executes a block of statements repeatedly until the condition(Boolean expression) returns false.

### Syntax of while loop

while (Boolean expression) {

statement(s) //block of statements

}

The logic in while loop is simple it executes the block of statements when the Boolean expression returns true. It gets terminated when the Boolean expression returns false.

### While loop example

class WhileLoopExample {

public static void main(String args[]){

int i=10

while(i>1){

System.out.println(i);

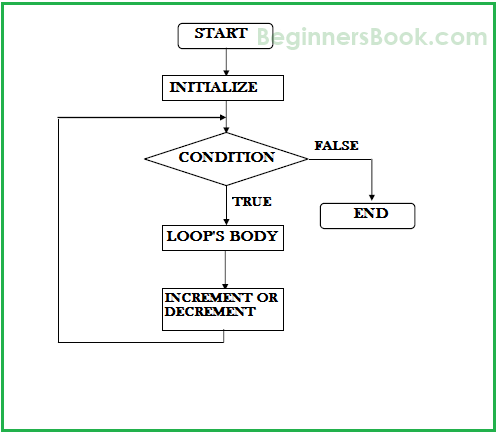
i--;

}

}

}

### While loop flow diagram



### While loop example to iterate an array

Here we are iterating and displaying array elements using while loop.

class WhileLoopExample3 {

public static void main(String args[]){

int arr[4]={2,11,45,9};

//i starts with 0 as array index starts with 0 too

int i=0;

while(i<4){

System.out.println(arr[i]);

i++;

}

}

}

**Output:**

2

11

45

9

# do-while loop in Java with example

do-while loop is similar to while loop, however there is a single difference between these two. Unlike while loop, do-while guarantees at-least one execution of block of statements. This happens because the do-while loop evaluates the boolean expression at the end of the loop’s body. Therefore the set of statements gets executed at-least once before the check of boolean expression.

**Syntax of while loop:**

do{

statement(s) //block of statements

}while (Boolean expression);

Here we are iterating and displaying array elements using do-while loop.

class DoWhileLoopExample2 {

public static void main(String args[]){

int arr[4]={2,11,45,9};

//i starts with 0 as array index starts with 0 too

int i=0;

do{

System.out.println(arr[i]);

i++;

}while(i<4);

}

}

Output:  
2  
11  
45  
9

# For loop in Java with example

It executes a block of statements repeatedly until the specified condition returns false.Syntax of for loop:

for (initialization; condition; increment/decrement) {

statement(s) //block of statements

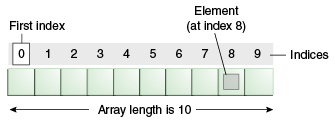
}

# **Java Array**

Normally, array is a collection of similar type of elements that have contiguous memory location.

**Java array** is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



### Advantage of Java Array

* **Code Optimization:** It makes the code optimized, we can retrieve or sort the data easily.
* **Random access:** We can get any data located at any index position.

### Disadvantage of Java Array

* **Size Limit:** We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

### Types of Array in java

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

### Single Dimensional Array in java

### Syntax to Declare an Array in java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

### Instantiation of an Array in java

1. arrayRefVar=**new** datatype[size];

### Example of single dimensional java array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

1. **class** Testarray{
2. **public** **static** **void** main(String args[]){
4. **int** a[]=**new** **int**[5];//declaration and instantiation
5. a[0]=10;//initialization
6. a[1]=20;
7. a[2]=70;
8. a[3]=40;
9. a[4]=50;
11. //printing array
12. **for**(**int** i=0;i<a.length;i++)//length is the property of array
13. System.out.println(a[i]);
15. }}

# **Java String**

In java, string is basically an object that represents sequence of char values

An array of characters works same as java string. For example:

**char**[] ch={'j','a','v','a','t','p','o','i','n','t'};

String s=**new** String(ch);

is same as:

String s="javatpoint";

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.

**public** **class** StringEx {

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

String myString = "Hello World";

**char**[] ch ={'h','e','l','l','o'};

String s = **new** String(ch);

System.*out*.println(s);

String myStr="Hello" + "World";

System.*out*.println(myString);

System.*out*.println(myString.length());

System.*out*.println(myString.toLowerCase());

System.*out*.println(myString.toUpperCase());

System*out*.println(myString.contains("n"));

System.*out*.println(myString.contentEquals("Hello"));

System.*out*.println(myStr);

System.*out*.println(myString.replace('o', 'R'));

System.*out*.println(myString.indexOf('o'));

System.*out*.println(myString.charAt(6));

}

l}

String[] str= new String[10];

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Java String class methods The java.lang.String class provides many useful methods to perform operations on sequence of char values.   |  |  |  | | --- | --- | --- | | **No.** | **Method** | **Description** | | 1 | [char charAt(int index)](http://www.javatpoint.com/java-string-charat) | returns char value for the particular index | | 2 | [int length()](http://www.javatpoint.com/java-string-length) | returns string length | | 3 | [static String format(String format, Object... args)](http://www.javatpoint.com/java-string-format) | returns formatted string | | 4 | [static String format(Locale l, String format, Object... args)](http://www.javatpoint.com/java-string-format) | returns formatted string with given locale | | 5 | [String substring(int beginIndex)](http://www.javatpoint.com/java-string-substring) | returns substring for given begin index | | 6 | [String substring(int beginIndex, int endIndex)](http://www.javatpoint.com/java-string-substring) | returns substring for given begin index and end index | | 7 | [boolean contains(CharSequence s)](http://www.javatpoint.com/java-string-contains) | returns true or false after matching the sequence of char value | | 8 | [static String join(CharSequence delimiter, CharSequence... elements)](http://www.javatpoint.com/java-string-join) | returns a joined string | | 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](http://www.javatpoint.com/java-string-join) | returns a joined string | | 10 | [boolean equals(Object another)](http://www.javatpoint.com/java-string-equals) | checks the equality of string with object | | 11 | [boolean isEmpty()](http://www.javatpoint.com/java-string-isempty) | checks if string is empty | | 12 | [String concat(String str)](http://www.javatpoint.com/java-string-concat) | concatinates specified string | | 13 | [String replace(char old, char new)](http://www.javatpoint.com/java-string-replace) | replaces all occurrences of specified char value | | 14 | [String replace(CharSequence old, CharSequence new)](http://www.javatpoint.com/java-string-replace) | replaces all occurrences of specified CharSequence | | 15 | [String trim()](http://www.javatpoint.com/java-string-trim) | returns trimmed string omitting leading and trailing spaces | | 16 | [String split(String regex)](http://www.javatpoint.com/java-string-split) | returns splitted string matching regex | | 17 | [String split(String regex, int limit)](http://www.javatpoint.com/java-string-split) | returns splitted string matching regex and limit | | 18 | [String intern()](http://www.javatpoint.com/java-string-intern) | returns interned string | | 19 | [int indexOf(int ch)](http://www.javatpoint.com/java-string-indexof) | returns specified char value index | | 20 | [int indexOf(int ch, int fromIndex)](http://www.javatpoint.com/java-string-indexof) | returns specified char value index starting with given index | | 21 | [int indexOf(String substring)](http://www.javatpoint.com/java-string-indexof) | returns specified substring index | | 22 | [int indexOf(String substring, int fromIndex)](http://www.javatpoint.com/java-string-indexof) | returns specified substring index starting with given index | | 23 | [String toLowerCase()](http://www.javatpoint.com/java-string-tolowercase) | returns string in lowercase. | | 24 | [String toLowerCase(Locale l)](http://www.javatpoint.com/java-string-tolowercase) | returns string in lowercase using specified locale. | | 25 | [String toUpperCase()](http://www.javatpoint.com/java-string-touppercase) | returns string in uppercase. | | 26 | [String toUpperCase(Locale l)](http://www.javatpoint.com/java-string-touppercase) | returns string in uppercase using specified locale. |  Java – Exceptions 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.   Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.  Based on these we have three categories of Exceptions you need to understand them to know how exception handling works in Java,   * **Checked exceptions:** A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the Programmer should take care of (handle) these exceptions.   Checked exceptions are those exceptions which are known to JVM.Whenever the program interacts out side the JVM, JVM knows what kind of exceptions can occur.  For example, if you use **FileReader** class in your program to read data from a file, if the file specified in its constructor doesn't exist, then an *FileNotFoundException* occurs, and compiler prompts the programmer to handle the exception.  import java.io.File;  import java.io.FileReader;  public class FilenotFound\_Demo {  public static void main(String args[]){  File file=new File("E://file.txt");  FileReader fr = new FileReader(file);  }    }  If you try to compile the above program you will get exceptions as shown below.    C:\>javac FilenotFound\_Demo.java  FilenotFound\_Demo.java:8: error: unreported exception FileNotFoundException; must be caught or declared to be thrown  FileReader fr = new FileReader(file);  ^  1 error Example of checked Exception in Java API Following are some Examples of Checked Exception in Java library: IOException  [SQLException](http://javarevisited.blogspot.com/2012/01/javasqlsqlexception-invalid-column.html)  DataAccessException  [ClassNotFoundException](http://javarevisited.blogspot.com/2011/08/classnotfoundexception-in-java-example.html)  InvocationTargetException  **Note:** Since the methods **read()** and **close()** of FileReader class throws IOException, you can observe that compiler notifies to handle IOException, along with FileNotFoundException.   * **Unchecked exceptions:** An Unchecked exception is an exception that occurs at the time of execution, these are also called as Runtime Exceptions, these include programming bugs, such as logic errors or improper use of an API. runtime exceptions are ignored at the time of compilation.   Those exceptions which are unknown to JVM are known as Unchecked Exceptions. They are the defects or we can say bugs in program code.  For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an *ArrayIndexOutOfBoundsExceptionexception* occurs.  public class Unchecked\_Demo {    public static void main(String args[]){  int num[]={1,2,3,4};  System.out.println(num[5]);  }  }  If you compile and execute the above program you will get exception as shown below.  Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5  at Exceptions.Unchecked\_Demo.main(Unchecked\_Demo.java:8)  NullPointerException  ArrayIndexOutOfBound  IllegalArgumentException  IllegalStateException   * **Errors:** These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation.   Errors are thrown by the Java runtime system and indicate some irrecoverable conditions that occur during the prog execution.  Once thrown by the system,the application cannot recover from it and will come on halt  Error need not be handled(catched)  Java.lang.OutOfMemoryError-There is no extra space to create a new object inside the memory,that means the total memory in the RAM occupied by the existing object and we don’t have any extra space to add the new object and may not be run the prog.  Java.lang.StackoverflowError-Invoke a method in cyclic order.example: Exception Hierarchy: All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class.  Errors are abnormal conditions that happen in case of severe failures, these are not handled by the java programs. Errors are generated to indicate errors generated by the runtime environment. Example : JVM is out of Memory. Normally programs cannot recover from errors.  The Exception class has two main subclasses: IOException class and RuntimeException Class.  Java Exceptions  Here is a list of most common checked and unchecked [Java's Built-in Exceptions](http://www.tutorialspoint.com/java/java_builtin_exceptions.htm).  6.1 Exception Handling  In Java we have three methods to handle the exceptions that may arrive in our program. These  three methods are:  Try-Catch-Finally  Throw  Throws  6.2 Try-Catch-Finally Block  In try block we write the legal statements which main lead to an error/exception at run time. The  structure of try block is give by  try{  //Code which may raise exceptions  }  In catch block we write the code which must execute when an error/exception arise from the try block. The structure of catch block is given1  by  catch( <Exception-Type> ) {  // handling exception logic is defined here  }  We can have |

Threads:

MultiTasking:

Executing several task simultaneously is a concept of **Multitasking/Multithreading**

Weather it is process based or thread based the main objective of Multitasking is ,To Increase Performance and decrease response time of the system .

It has 2 types of Multitasking

1: Process based Multitasking and

2: Thread based Multitasking

**Process**: Executing the several task simultaneously where each task is separate independent Program.(Process) There is no dependent between the Programs.

Ex:

Am writing a program in an editor

2:at the same time am listening songs from the MP3 from the same system

3:at the same time am downloading some movie from the internet

Here there are 3 programs

This is called Process based multitasking because it will not dependent on each other.

This example is best applicable for OS(Operating System) level and for programmatically level we should go for Thread based multitasking

Thread Based: Executing several task simultaneously where each task is a separate independent part of the same program.

Ex:I wrote a program which has 10000 line of code and it takes 10 hours to execute ,but first 5000 line will not dependent on next 5000 line. If not dependent we can run into 2 parts and execute parallel

**Each independent part is called Thread**

|  |
| --- |
| 2000 |
| 2000 |
| 2000 |
| 2000 |

Thread based Multitasking is best suitable yet Programmatic Level.

The main Important application areas of Multithreading are:

1:To develop multimedia graphics

2:To develop animations

3:To develop videogames

4:for developing web servers and application servers etc…

EX:Search a string in the system ..

When compared with OLD languages developing Multithreaded applications in java is very easy because java provides inbuilt support for multithreading with rich API.(Thread ,runnable,Threadgroup..)

**Definig a thread.**

We can define a thread in two ways

1:By Extending thread class

2:By implementing Runnable interface

**By Extending thread class:**

**Class myThread extends Thread**

**{**

**Public void run(){**

**For(int -=0li<10;i++**

**{---------------------------------------🡪Exicuted by child thread**

**Syso(“child thread”)**

**} JOB**

**Class ThreadDemo**

**{**

**Public static void main(String[] args){**

**Mythread a = new MyThread()-----------🡪Thread Instantiation**

**a.start();---------Starting of thread**

**For(int -=0li<10;i++**

**{**

**Syso(“child thread”)**

**} -🡪Executing of main thread**

**Case1: Thread Sheduler:**It is the part of JVM ,it is responsible to schedule threads.i.e if multiple threads are waiting to get the chance of execution then in which order threads will be executed is decided by thread Scheduler.

We cannot except exact algorithm followed by thread scheduler It is varied from JVM to JVM.hence we cannot except thread execute order and exact output.

Hence whenever situation comes to multithreading there is no guarantee for exact output.but we can provide several possible outputs.

The following are various possible for the above program.

Case1:

|  |  |  |
| --- | --- | --- |
| Child thread  Child thread  Child thread  Main thread  Main thread  Main thread | Main thread  Main thread  Main thread  Child thread  Child thread  Child thread | Child thread  Child thread  Child thread  Child thread  Child thread  Child thread |

**Case 2:Difference between t.start and t.run**

**In the case of t.start a new thread will be created which is the responsible for the execution of run method but**

**In the case of t.run a new thread wont be created and the run method will be executed just like a normal call by main thread.**

**Hence in the above program if we replace t.strat with t.run then the output is** Child thread

Child thread

Child thread

Main thread

Main thread

Main thread

**This total o/p produced by only main thread**

**Case 3:Importance of Thread class Start Method:**

**Thread class start method is responsible to Register the thread with thread scheduler and all other mandatory activities.hence with out executing thread class start method there is no chance of starting a new thread in java.**

**Due to this thread class start method is considred as heart of multithreading.**

**Start(){**

**1: Register with Thread scheduler**

**2: Perform all mandatary activity**

**3: Invoke run method**

**}**

**Case 4: Overloaded of run method :**

**: Overloaded of run method is always possible but thread.start method can invoke no argument run method. The other overloaded method we have to call explicitly like a normal method call**

**public** **class** ThreadOverloading **extends** Thread {

**public** **void** run(){

System.*out*.println("Child thread");

}

**public** **void** run(**int** i){

System.*out*.println("int thread");

}

}

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

// **TODO** Auto-generated method stub

ThreadOverloading e = **new** ThreadOverloading();

e.start();

OP---------- Child thread

Case 5:if we are not overriding run method , then thread class run method will be executed which has empty implementation hence we wont get any output

**public** **class** ThreadOverloading **extends** Thread {

}

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

// **TODO** Auto-generated method stub

ThreadOverloading e = **new** ThreadOverloading();

e.start();

}

OP🡪No Output

Note:it is highly recommended to over rite run method otherwise don’t go for multithreading concept.

Case6: Overriding of Start Method.

If we override start method then our start method will be exictued just like a normal method call and new thread wont be created..

Note:it is not recommended to override start method otherwise there don’t go for multithreading concept

**public** **class** ThreadOverloading **extends** Thread {

**public** **void** start(){

System.*out*.println("Child thread");

}

**public** **void** run{

System.*out*.println("int thread");

}

}

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

// **TODO** Auto-generated method stub

ThreadOverloading e = **new** ThreadOverloading();

e.start();

syop(“hello”)

op🡪Child thread

hello

**public** **class** ThreadOverloading **extends** Thread {

**public** **void** start(){

**super.start();**

System.*out*.println("Start thread");

}

**public** **void** run{

System.*out*.println("run thread");

}

}

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

// **TODO** Auto-generated method stub

ThreadOverloading e = **new** ThreadOverloading();

e.start();

syop(“hello”)

op-🡪run thread

start thread or any of these 3 method

hello

Case8:

Programmer Sheduler

Thread

10.2 States of Thread

Newly Started Thread: When ever we create an object of thread, it is known as newly created

thread. The new created thread doesn’t perform any task until it is started.

Runnable State: When a thread is read to run, that state is known as Runnable State. Here also

the thread is not started yet.

Running State: When we call the start() method of thread the thread comes into running state

and the block of code (i.e. thread task) starts executing.

Blocked State: Either by suspending a thread, pause a thread or a thread is waiting for resource

which is busy, can send the running thread into blocked state. Some the block state may lead to dead locks. We must intelligently configure our thread execution and we must monitor each

and every thread.

Dead State: When a thread finished its job, it releases all the system resources held by thread

and changes its state to dead.

Case 9:After starting a thread if we are trying to restart the same thread we will get runtime exception saying Illegal thread exception.

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

// **TODO** Auto-generated method stub

ThreadOverloading e = **new** ThreadOverloading();

e.start();

Syso(“hello”)

e.start

op🡪illegalthread exception

**Defining a thread by Implementing Runnable Interface**

We can define a thread by implementing runnable interface

Mythread--------🡪Thread--------Runnable

My Runnable

Runnable interface present in java.lang package and it contain only one method Run method

Public void run();

**public** **class** ThreadRunnable **implements** Runnable {

**public** **void** run(){

**for(int i=0;i<3;i++){**

**System.*out*.println("Child thread");------🡪Job of the thread and executed by child thread**

**}**

}

}

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

// **TODO** Auto-generated method stub

ThreadRunnable r = **new** ThreadRunnable();

Thread th = **new** Thread(**r**);🡪r refers to target runnable

th.start();

**for**(**int** i=0;i<3;i++){

System.*out*.println("Main thread");-🡪Exiuted by main thread

}

We will get Mixed output and we cannot tell exact output

**Case Study:**

ThreadRunnable r = **new** ThreadRunnable();

Thread t1= **new** Thread();

Thread t2 = **new** Thread(**r**);

Case1:t1.start();

A thread will be created and which is responsible for the execution of thread class run method,which has empty implementation

Case 1:t1.run();

No new thread will be created and thread class will be exictued just like a normal method call.

Case 3:t2.start()

A new thread will be created which is responsible for the execution of myrunnable class run method

Case 4:t2.run();

A new thread wont be created and my runnable run method will be eicuted jist like a normal method call

Case 5:r.start();

We will get compile time error saying “my runnable class doesnot have start capability CE:cannot find symbol method start() location class myrunnable

Case 6:r.run();

No new thread will be created and my runnable run method will be executed just like a normal method call.

Which Approach is best to define a thread:

Among 2 ways of defining a thread Implementes runnable approach is recommended

In the first approach our class always extends thread class , there is no chance of extending any other class hence we are missing inheritance benfit

But in the second approach while implementing runnable interface we can extend any other classh hence we wont miss any inheritance benefit

**Thread class constructors**

Thread t = new thread();

Thread t = new thread(runnable r)

Thread t = new thread(string name)

Thread t = new thread(runnable r, string name)

Thread t = new thread(threadGroup g , string name)

Thread t = new thread(threadGroup g , runnable r)

Thread t = new thread(threadGroup g , runnable r, string name)

Thread t = new thread(threadGroup g , runnable r, string name , long stacksize)

Durgas approach to define a thread (not recommended to use)

**public** **class** ThreadHybrid **extends** Thread {

**public** **void** run(){

System.*out*.println("Child Thread");

}

**public** **class** ThreadHybrid1 {

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

// **TODO** Auto-generated method stub

ThreadHybrid t = **new** ThreadHybrid();

Thread t2 = **new** Thread(t);

t2.start();

System.*out*.println("Main thread");

}

OP🡪 Main thread

Child Thread

Or

Child Thread

Main thread

**Getting and setting name of a thread**

**Every thread in java has some name, it may be default name generated by JVM or custamized name provided by programmer.**

**We can get and set name of a thread using**

1. **Public final String getname()**
2. **Public final void setname(String name)**

**package** javaStart;

**public** **class** Threadgetname **extends** Thread{

}

**public** **class** Threadgetname1 {

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

// **TODO** Auto-generated method stub

System.*out*.println(Thread.*currentThread*().getName());

Threadgetname tr = **new** Threadgetname();

System.*out*.println(tr.getName());

Thread.*currentThread*().setName("mohan");

System.*out*.println(Thread.*currentThread*().getName());

Result: main

Thread-0

Mohan

**Thread Priorities:**

Every thread in java has some priority, it may be default priority generated by the JVM or customized priority provided by programmer.

The valid range of thread priority is 1 to 10; where 1 is the minimum priority and 10 is maximum priority.Thread class defines the following constants to represent some standard priority.

Thread.MIN.Priority-🡪1

Thread.norm.priority🡪5

Thread.max.priority🡪10

Thread scheduler will use priorities while allocating processor,the thread which is having highest priority will get chance first.if 2 threads having same priority then we cannot expect exact execution order .it depends on thread scheduler.

Thread class defines the following methods to get and set Priority of a thread.

Public final int getPriority();

Public final void setPriority(int p); allowed values range 1 to 10 ,otherwise runtime exception illegal argument exception.

Ex:t.setPriority(7) -🡪Valid

t.setPriority(17)🡪illegal exception.

**Default Priority**

**The default priority only for the main thread is 5,but all remaining threads default priority will be inherited from parent to child i.e what ever priority parent thread has the same priotiy will be there for child thread.**

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

// **TODO** Auto-generated method stub

System.*out*.println(Thread.*currentThread*().getPriority());

Thread.*currentThread*().setPriority(7);

System.*out*.println(Thread.*currentThread*().getPriority());

Threadgetname n = **new** Threadgetname();

System.*out*.println(n.getPriority());

OP🡪 5

7

7

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

// **TODO** Auto-generated method stub

Threadgetname tn = **new** Threadgetname();

tn.setPriority(6);

tn.start();

System.*out*.println("Main Thread");

}

If we are commenting priority ,both main and child threads have the same priority 5.and hence we cannot expect execution order and exact output.

If we are not commenting priority,then main thread has a priority 5 and child thread has the priority 10.hence child thread will get the chance first followed by main thread.in this case output is childthread and mainthread

**public** **class** ThreadPriorityex {

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

// **TODO** Auto-generated method stub

Threadgetname tn = **new** Threadgetname();

tn.setPriority(10);

tn.start();

System.*out*.println("Main Thread");

}

Note:Some platforms will not support ploperly for thread priorities.

**We can prevent a thread execution by using**

**1:Yield();**

**2:Jion();**

**3:sleep();**

**Yield method:Yield method causes to pause current execution thread ,to give the chance for the waiting threads of same priority.if there is no witing thread or all waiting threads have low priority then same thread can continue its execution.**

**If multiple threads are waiting with same priority then which waiting thread will get the chance ,we cant expect it depends on thread shedular.**

**The thread which is yielded when it will get chance once again it depends on thread shedular and we cannot expect exactly.**

**public** **class** Threadyeild **extends** Thread {

**public** **void** run(){

**for**(**int** i=0;i<10;i++){

System.*out*.println("Child thread");

Thread.*yield*();

}

}

**public** **class** ThreadYeildMain {

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

// **TODO** Auto-generated method stub

Threadyeild tn = **new** Threadyeild();

tn.start();

**for**(**int** i=0;i<10;i++){

System.*out*.println("Main thread");

}

}

In the above program if we are commenting thread.yeild then both threads willb eexicuted simultaneously and we cannot expect which thread can complete first.

If we are not commenting thread.yeild then child thread always calls yield method because of that main thread will get chance more number of times and the chance of completing main thread first is high.

Some platforms will not properly support.

JION();

If a thread wants to wait until completing some other thread then we should go for JOIN menthod.

Ex:If a thread t1 wants to wait until completing t2 then t1 has to call **t2.join**;

If T1 exicutes t2.join() , then immediately t1 will be entered into waiting state until t2 completes.

Once t2 completes then t1 can continue its execution.

Venue fixing wedding card printing wedding card destribution

**t.join t2.join**

**Waiting until another thread completes**

**Public final void join(); throws interrupted exception**

**Public final void join(long millisec); throws interrupted exception**

**Public final void join(long millisec,int nanosec); throws interrupted exception**

**Note:every join method thorws interrupted exception which is checked exception.hence complusary we should handle this exception either by using try catch or thorws keyword otherwise we will get compile time error**

**t.start if TS allocates processor ifrun()**

**Newthread------🡪Ready-----------------🡪running--------🡪Dead**

**If t2 comletes t2.join**

**If time expires t2.join(1000),t2.join(1000,100)**

**If waiting thread get interrupted waiting time**

**public** **class** ThreadJoin **extends** Thread {

**public** **void** run(){

**for**(**int** i=0;i<10;i++){

System.*out*.println("Hello Child");

**try**{

Thread.*sleep*(1000);

}**catch** (InterruptedException e)

{

}

}

}

**public** **class** ThreadJoin1 {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

ThreadJoin jn = **new** ThreadJoin();

jn.start();

jn.join();

jn.join(1000);

**for**(**int** i=0;i<10;i++){

System.*out*.println("Main thread");

}

}

Case 2: **waiting child thread until completing main thread.**

**public** **class** ThreadJoinchildwait **extends** Thread {

**static** Thread *mt*;

**public** **void** run(){

**try**{

*mt*.join();

}**catch** (InterruptedException e){

}

**for**(**int** i=0;i<10;i++){

System.*out*.println("Child thread");

}

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

ThreadJoinchildwait.*mt* = Thread.*currentThread*();

ThreadJoinchildwait tc = **new** ThreadJoinchildwait();

tc.start();

**for**(**int** i=0;i<10;i++){

System.*out*.println("Main thread");

Thread.*sleep*(2000);

}

}

**In the above ex, child thread calls join method on main thread object hence child thread has to wait until completing main thread.in these case output is main thread ……………… child thread**

**Case 3: if main thread calls join method on child object and child thread calls join method on main thread object then both thread will wait forever result deadlock.**

**Case 4: if a thread calls join method on the same thread itself result deadlock**

**Psvm(String){**

**Thread.currentThread().join();**

**}**

**Sleep():if a thread don’t want to perform any operation for a particular amount of time then we should go for sleep method.**

**Public static native void sleep(long millisec) throws interrupted exception**

**Public static native void sleep(long millisec, int Milliseconds) throws IE**

**Every sleep method throws IE , which is checked execption hence whenever we are using sleep method complusery we should handle IE either by try cathch or throws keyword or else will get CE.**

**t.start if ts allocate processor if run() completes**

**New/bornstate-------🡪ready--------🡪running --------------🡪dead state**

**Thread.sleep(1000)**

**If time expires Thread.sleep(1000,100)**

If sleeping interupted Slleping state

**public** **class** Threadsleep {

**public** **static** **void** main(String[] args) **throws** InterruptedException {

// **TODO** Auto-generated method stub

**for**(**int** i=0;i<10;i++){

System.*out*.println("Main thread :" +i);

Thread.*sleep*(2000);

}

}

Notify and nofiy ON

**How a thread can interrupted another thread:**

**A thread can interrupt a sleeping thread or waiting thread by using interrupt() of the thread class.**

**Public void interrupt();**

**public** **class** ThreadInterrupt **extends** Thread {

**public** **void** run(){

**try**{

**for**(**int** i=0;i<10;i++){

System.*out*.println("Main thread :" +i);

Thread.*sleep*(2000);

}

}**catch**(InterruptedException e){

System.*out*.println("Thread got interupted");

}

}

}

**package** javaStart;

**public** **class** ThreadInterrupt1 {

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

// **TODO** Auto-generated method stub

ThreadInterrupt t = **new** ThreadInterrupt();

t.start();

//t.interrupt();

System.*out*.println("End of main thread");

}

}

OP🡪

End of main thread

Main thread :0

Thread got interrupted

**Note:when ever we are calling interrupt method if the target thread not in sleeping state or waiting state then there is no impact of interrupt call immediately.interrupt call will be waited until target thread entered into sleep or watitng state.**

**If the target thread entered into sleeping or waiting state then immediately interrupt call will interrupt the target thread.**

**If the target thread never entered into sleep or waitng state in its life time, interrupt call will get wasted.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Property** | **Yeild** | **join** | **sleep** |
| **Purpose** | **If a thread wants to pause its execution to give the chance for remaining threads of same priority** | **If a thread wants to wait until completing some other thread** | **If a thread don’t want to perform any operation for a particular amount of time** |
| **Is it overloaded** | **No** | **Yes** | **yes** |
| **Is it final** | **No** | **Yes** | **no** |
| **Is it thows IE** | **No** | **Yes** | **yes** |
| **Is it native** | **Yes** | **no** | **Sleep(long ms)Sleep(long ms) sleep(long wait,int ms)-non native** |
| **static** | **Yes** | **no** | **yes** |

### IO Stream

Java performs I/O through **Streams**.

A stream can be defined as a sequence of data. there are two kinds of Streams

* **InPutStream:** The InputStream is used to read data from a source.
* **OutPutStream:** the OutputStream is used for writing data to a destination.

In file class there are 3 constructor

1:File f = new file(String)

2:File f = new file(Subdirname ,String)

3:File f = new file(File sibdirname , stringname)

Important methods present in file class.:

Boolean f.exist(): to check physical file is available or not

f.createnewFile()—boolean🡪

f.mkdir()—Boolean🡪

f.isFile()🡪

f.isDirectory()🡪weather points to directory or file

String[] s = f1.list()🡪Will list all the available file under the specified directory

**int** count =0;

File f = **new** File("D:\\MAVEN\_Workspace\\JavaLearning");

String[] s=f.list();

**for**(String s1 : s){

count++;

System.*out*.println(s1);

}

System.*out*.println("The total dir and files :"+count);

}

f.length🡪number of character present inside the file.

f.delete()🡪to delete the file or dir.

Why we will use 2 back slash in file directory,because if there is only one backslash the compiler treats it as escape character

FileWriter:to write data to the file.

Constructor:

1:FileWriter fw = new Filewriter(String name);

2:fileWriter fw = new FileWriter(File f)

These file constructors overriding of existing data

3:FileWriter fw = new Filewriter(String name , boolean append);

2:fileWriter fw = new FileWriter(File F , Boolean Append)

These file constructors appending to existing data

If the file already not available ,these constructor will crate file.

**Various methods of file writer:**

**1: write(int ch) write single char to the file .**

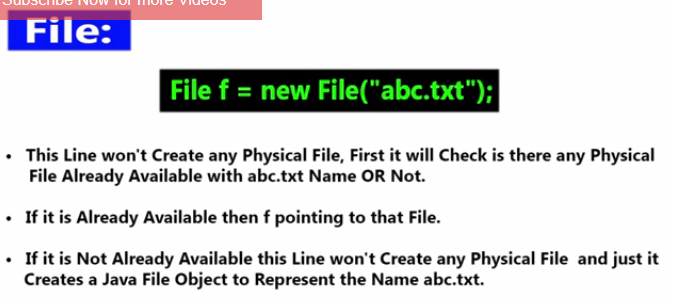
**2: write(char[] ch)---write multiple char**

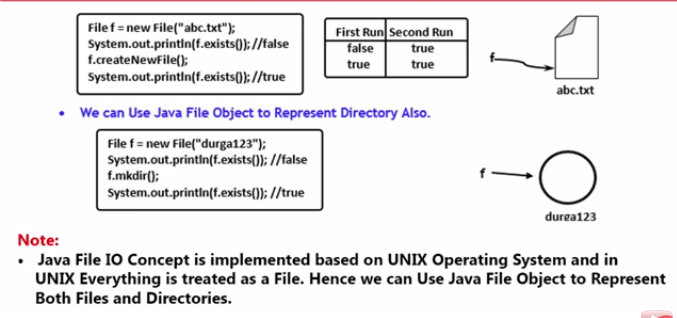
**3:write(string s)🡪to write string**

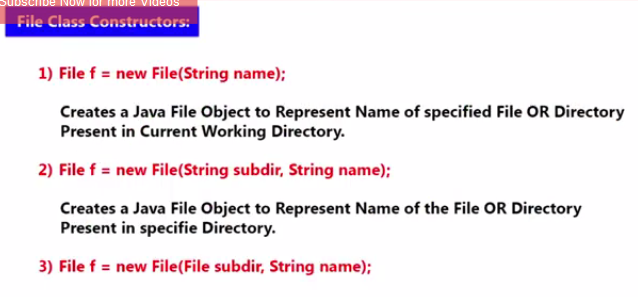
**4:flush()—to give the guaranteed that all the given data entered should be written into the file we use flush method**

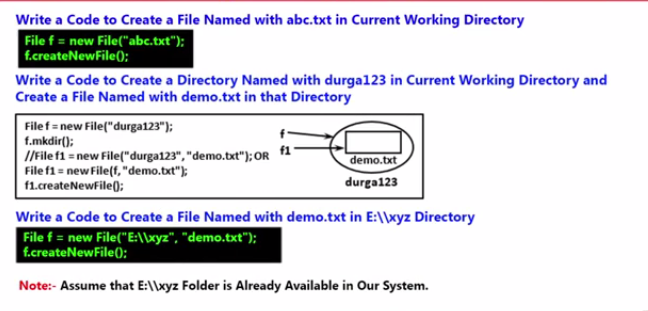
**5:close() 🡪we have to close the writer method or reader**

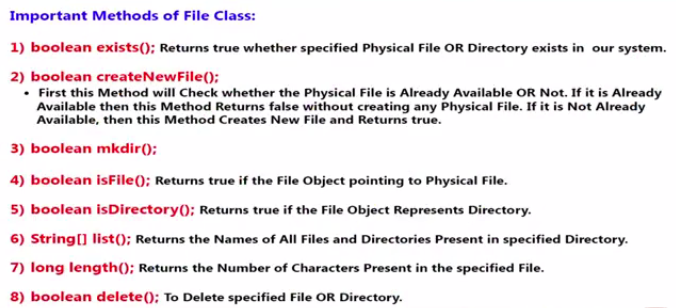
****

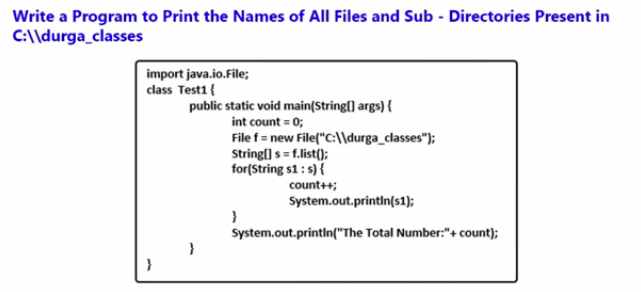


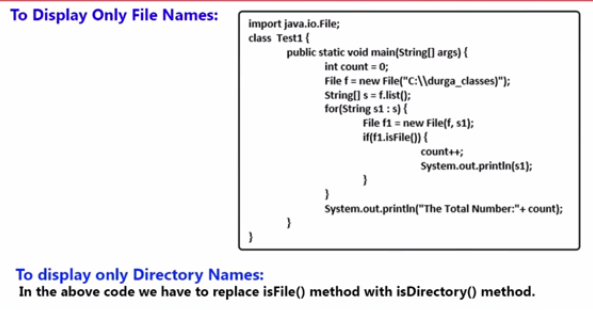


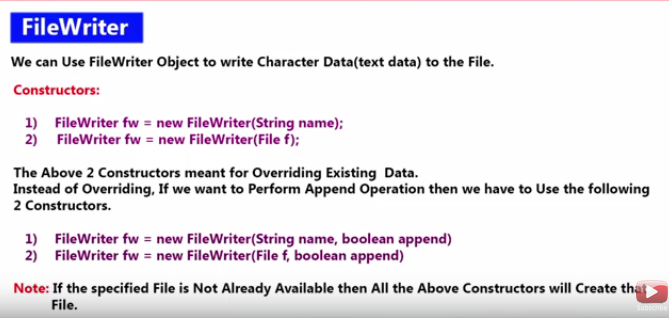


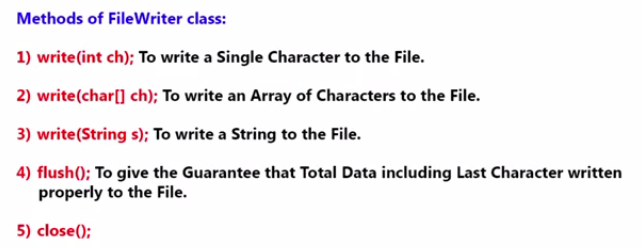


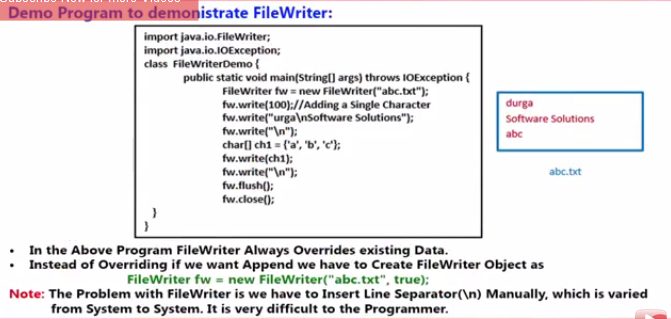


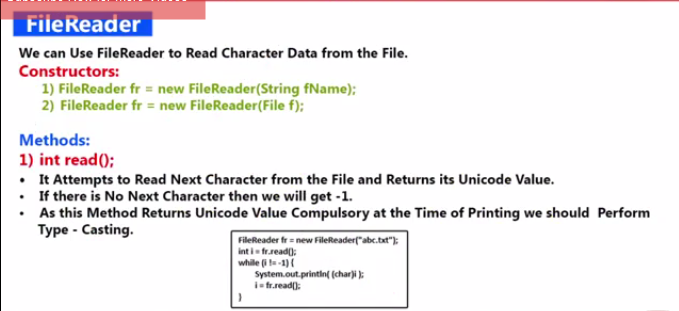


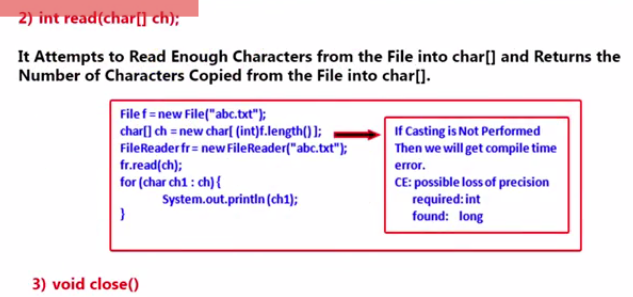


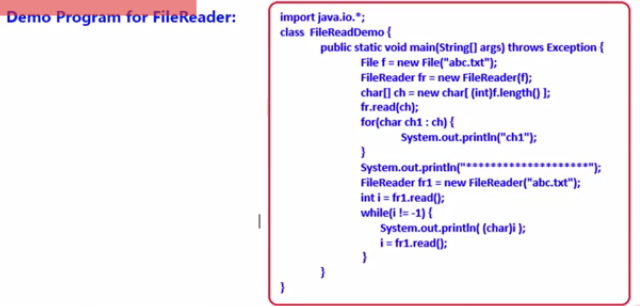


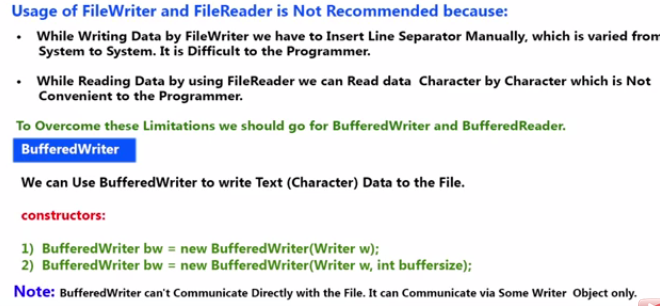


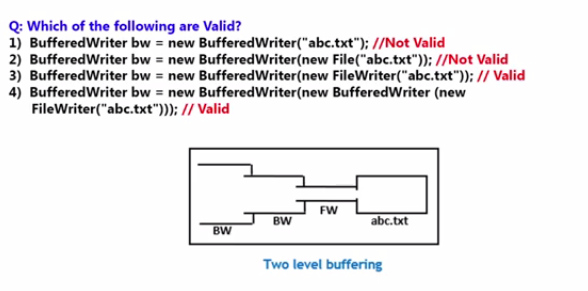


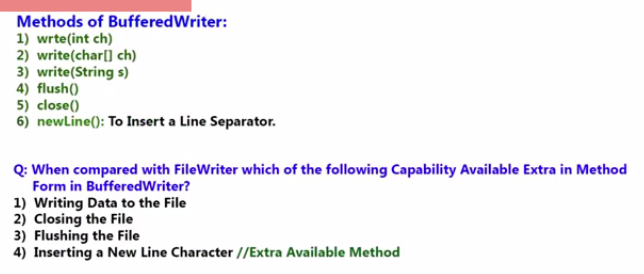


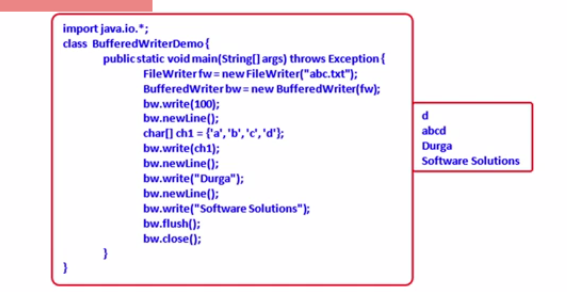


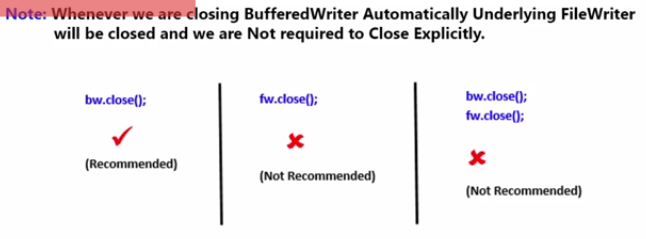




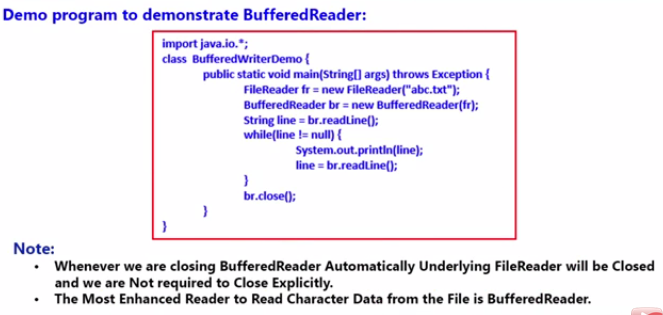


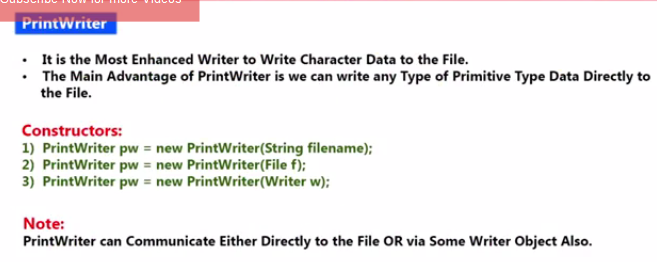


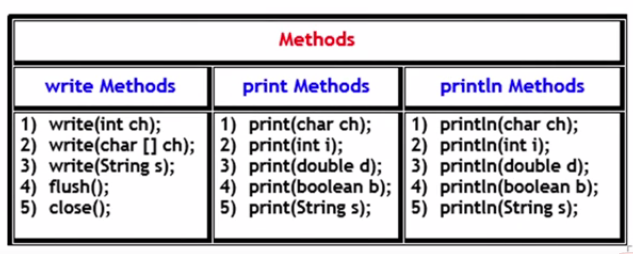


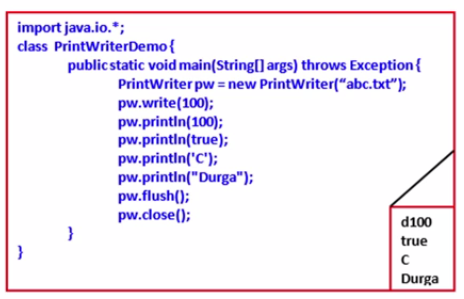


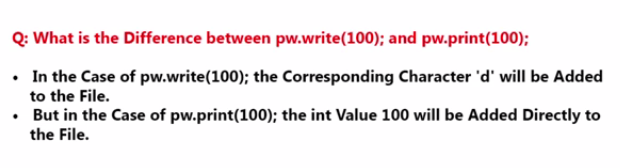


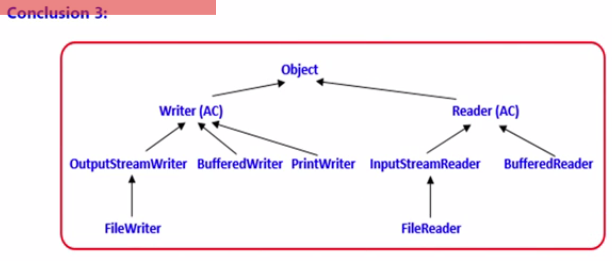
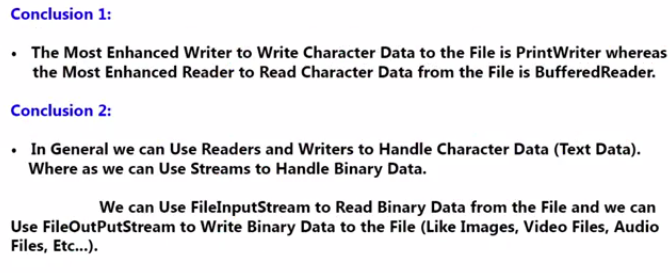


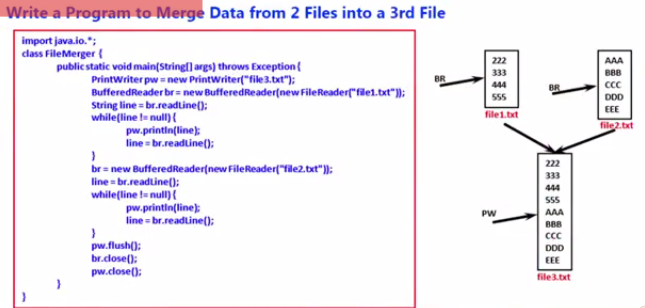


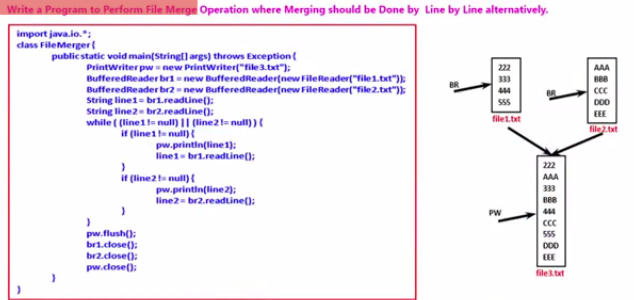


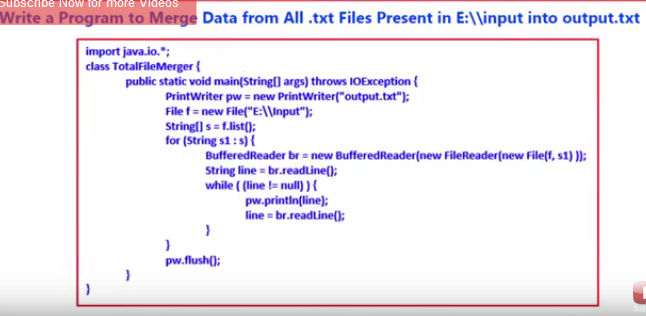


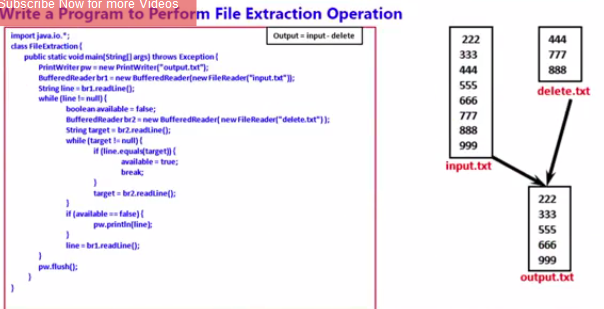


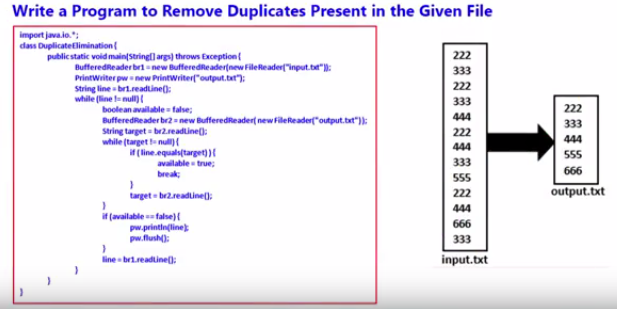












Program: All file operations.

public class MyFileOperations {

    public static void main(String[] a){

        try{

            File file = new File("fileName");

            //Tests whether the application can read the file

            System.out.println(file.canRead());

            //Tests whether the application can modify the file

            System.out.println(file.canWrite());

            //Tests whether the application can modify the file

            System.out.println(file.createNewFile());

            //Deletes the file or directory

            System.out.println(file.delete());

            //Tests whether the file or directory exists.

            System.out.println(file.exists());

            //Returns the absolute pathname string.

            System.out.println(file.getAbsolutePath());

            //Tests whether the file is a directory or not.

            System.out.println(file.isDirectory());

            //Tests whether the file is a hidden file or not.

            System.out.println(file.isHidden());

            //Returns an array of strings naming the

            //files and directories in the directory.

            System.out.println(file.list());

        } catch(Exception ex){

        }

# Java - 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).

Using extends keyword the My\_Calculation inherits the methods addition () and Subtraction () of Calculation class.

Copy and paste the program given below in a file with name My\_Calculation.java

class Calculation{

int z;

public void addition(int x, int y){

z = x+y;

System.out.println("The sum of the given numbers:"+z);

}

public void Subtraction(int x,int y){

z = x-y;

System.out.println("The difference between the given numbers:"+z);

}

}

public class My\_Calculation extends Calculation{

public void multiplication(int x, int y){

z = x\*y;

System.out.println("The product of the given numbers:"+z);

}

public static void main(String args[]){

int a = 20, b = 10;

My\_Calculation demo = new My\_Calculation();

demo.addition(a, b);

demo.Subtraction(a, b);

demo.multiplication(a, b);

}

}

From the above example, created an object My\_Calculation and it has taken all the properties from Calculation.

## The 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.

### Differentiating the members

If a class is inheriting the properties of another class. And if the members of the superclass have the names same as the sub class, to differentiate these variables we use super keyword as shown below.

class Super\_class{

int num = 20;

//display method of superclass

public void display(){

System.out.println("This is the display method of superclass");

}

}

public class Sub\_class extends Super\_class {

int num = 10;

//display method of sub class

public void display(){

System.out.println("This is the display method of subclass");

}

public void my\_method(){

//Instantiating subclass

Sub\_class sub = new Sub\_class();

//Invoking the display() method of sub class

sub.display();

//Invoking the display() method of superclass

super.display();

//printing the value of variable num of subclass

System.out.println("value of the variable named num in sub class:"+ sub.num);

//printing the value of variable num of superclass

System.out.println("value of the variable named num in super class:"+ super.num);

}

public static void main(String args[]){

Sub\_class obj = new Sub\_class();

obj.my\_method();

}

}

## IS-A Relationship

IS-A is a way of saying: This object is a type of that object. Let us see how the **extends** keyword is used to achieve inheritance.

public class Animal{

}

public class Mammal extends Animal{

}

public class Reptile extends Animal{

}

public class Dog extends Mammal{

}

Now, based on the above example, In Object Oriented terms, the following are true −

* Animal is the superclass of Mammal class.
* Animal is the superclass of Reptile class.
* Mammal and Reptile are subclasses of Animal class.
* Dog is the subclass of both Mammal and Animal classes.

Now, if we consider the IS-A relationship, we can say −

* Mammal IS-A Animal
* Reptile IS-A Animal
* Dog IS-A Mammal
* Hence : Dog IS-A Animal as well

With use of the extends keyword the subclasses will be able to inherit all the properties of the superclass except for the private properties of the superclass.

We can assure that Mammal is actually an Animal with the use of the instance operator.

## Example

class Animal{

}

class Mammal extends Animal{

}

class Reptile extends Animal{

}

public class Dog extends Mammal{

public static void main(String args[]){

Animal a = new Animal();

Mammal m = new Mammal();

Dog d = new Dog();

System.out.println(m instanceof Animal);

System.out.println(d instanceof Mammal);

System.out.println(d instanceof Animal);

}

}

This would produce the following result −

true

true

true

## HAS-A relationship

These relationships are mainly based on the usage. This determines whether a certain class **HAS-A** certain thing. This relationship helps to reduce duplication of code as well as bugs.

Lets us look into an example −

public class Vehicle{}

public class Speed{}

public class Van extends Vehicle{

private Speed sp;

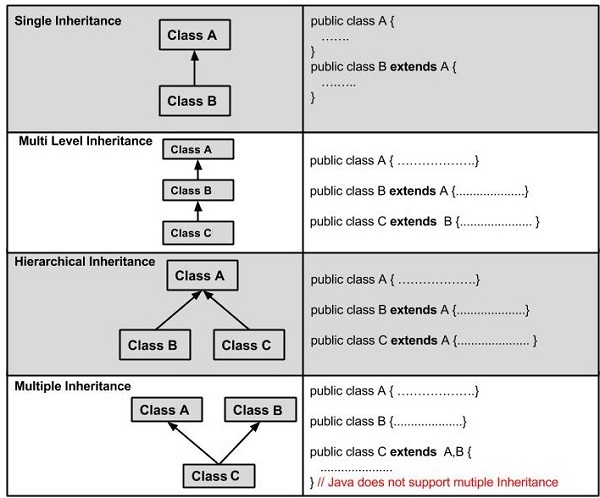
}

This shows that class Van HAS-A Speed. By having a separate class for Speed, we do not have to put the entire code that belongs to speed inside the Van class., which makes it possible to reuse the Speed class in multiple applications.

In Object-Oriented feature, the users do not need to bother about which object is doing the real work. To achieve this, the Van class hides the implementation details from the users of the Van class. So basically what happens is the users would ask the Van class to do a certain action and the Van class will either do the work by itself or ask another class to perform the action.

## Types of inheritance

There are various types of inheritance as demonstrated below.



A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class. Therefore following is illegal −

public class extends Animal, Mammal{}

However, a class can implement one or more interfaces. This has made Java get rid of the impossibility of multiple inheritance.

# Interface in Java

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.

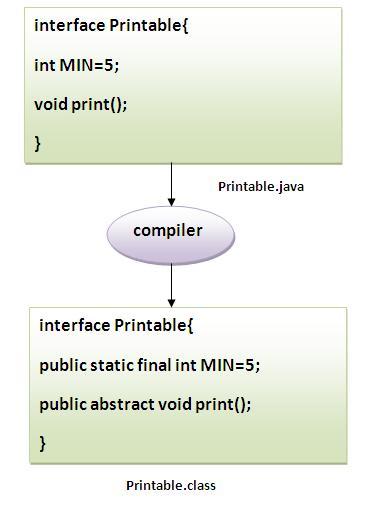
**Why use Java interface?**

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

* It is used to achieve fully abstraction.
* By interface, we can support the functionality of multiple inheritance.
* It can be used to achieve loose coupling.

#### The java compiler adds public and abstract keywords before the interface method and public, static and final keywords before data members.

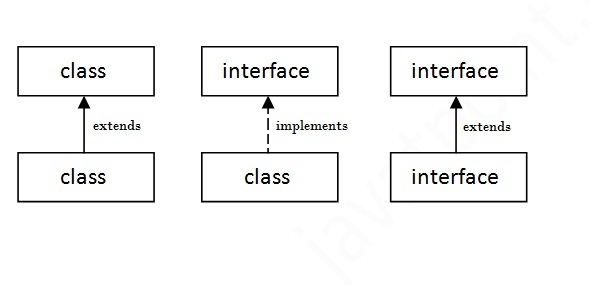
Interface fields are public, static and final bydefault, and methods are public and abstract.



s

#### Understanding relationship between classes and interfaces

As shown in the figure given below, a class extends another class, an interface extends another interface but a **class implements an interface**.



## Simple example of Java interface

|  |
| --- |
| In this example, Printable interface have only one method, its implementation is provided in the A class. |

1. interface printable{
2. void print();
3. }
5. class A6 implements printable{
6. public void print(){System.out.println("Hello");}
8. public static void main(String args[]){
9. A6 obj = new A6();
10. obj.print();

|  |
| --- |
| }  } |
|  |
| Multiple inheritance in Java by interface If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.  multiple inheritance in java | |

 interface Printable{

 void print();

 }



 interface Showable{

 void show();

 }



 class A7 implements Printable,Showable{



 public void print(){System.out.println("Hello");}

 public void show(){System.out.println("Welcome");}



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

 A7 obj = new A7();

 obj.print();

 obj.show();

--------------------------------------------------------------------------------------------------------------------------------------

### Abstract class

If a class contain any abstract method then the class is declared as abstract class. An abstract class is never instantiated. It is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method.

**Syntax :**

abstract class class\_name { }

#### Abstract method

Method that are declared without any body within an abstract class is known as abstract method. The method body will be defined by its subclass. Abstract method can never be final and static. Any class that extends an abstract class must implement all the abstract methods declared by the super class.

**Syntax :**

abstract return\_type function\_name (); // No definition

**public** **abstract** **class** Abstract1 {

**abstract** **void** mohan();

**abstract** **void** chandan();

**public** **void** shashikant(){

System.*out*.println("Hell dude");

}

**public** **class** Abstract2 **extends** Abstract1{

**void** mohan() {

System.*out*.println("Hello how are u");

}

**void** chandan() {

System.*out*.println("Hello how are u chandan");

}

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

Abstract1 a1 = **new** Abstract2();

a1.chandan();

a1.mohan();

a1.shashikant();

}

#### Abstract class with concrete(normal) method.

Abstract classes can also have normal methods with definitions, along with abstract methods.

abstract class A

{

abstract void callme();

public void normal()

{

System.out.println("this is concrete method");

}

}

class B extends A

{

void callme()

{

System.out.println("this is callme.");

}

public static void main(String[] args)

{

B b=new B();

b.callme();

b.normal();

}

}

**Output :**

this is callme.

this is concrete method.

#### Points to Remember

1. Abstract classes are not Interfaces. They are different, we will study this when we will study Interfaces.
2. An abstract class must have an abstract method.
3. Abstract classes can have Constructors, Member variables and Normal methods.
4. Abstract classes are never instantiated.
5. When you extend Abstract class with abstract method, you must define the abstract method in the child class, or make the child class abstract.

#### Abstraction using abstract class

Abstraction is an important feature of OOPS. It means hiding complexity. Abstract class is used to provide abstraction. Although it does not provide 100% abstraction because it can also have concrete method. Lets see how abstract class is used to provide abstraction.

abstract class Vehicle

{

public abstract void engine();

}

public class Car extends Vehicle {

public void engine()

{

System.out.println("Car engine");

//car engine implementation

}

public static void main(String[] args)

{

Vehicle v = new Car();

v.engine();

}

}

**Output :**

Car engine

Here by casting instance of **Car** type to **Vehicle** reference, we are hiding the complexity of **Car** type under **Vechicle**. Now the **Vehicle** reference can be used to provide the implementation but it will hide the actual implementation process.

# Difference between abstract class and interface

Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

But there are many differences between abstract class and interface that are given below.

|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can have static methods, main method and constructor**. | Interface **can't have static methods, main method or constructor**. |
| 5) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 6) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 7) **Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |

Simply, abstract class achieves partial abstraction (0 to 100%) whereas interface achieves fully abstraction (100%).

### Example of abstract class and interface in Java

Let's see a simple example where we are using interface and abstract class both.

1. //Creating interface that has 4 methods
2. interface A{
3. void a();//bydefault, public and abstract
4. void b();
5. void c();
6. void d();
7. }
9. //Creating abstract class that provides the implementation of one method of A interface
10. abstract class B implements A{
11. public void c(){System.out.println("I am C");}
12. }
14. //Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
15. class M extends B{
16. public void a(){System.out.println("I am a");}
17. public void b(){System.out.println("I am b");}
18. public void d(){System.out.println("I am d");}
19. }
21. //Creating a test class that calls the methods of A interface
22. class Test5{
23. public static void main(String args[]){
24. A a=new M();
25. a.a();
26. a.b();
27. a.c();
28. a.d();
29. }}

Reading Properties File in Java

First create an package called config and then create a file with the extension Employee.properties.

Enter the employee properties like name=mohan,age=22

Then create a new class called Read\_Properties and the code as below

**Properties is the class used to read the properties file so need to create a reference of the properties**

**FileInputStream is used to read the file**

**public** **class** ReadingProperties\_employee {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

System.*out*.println(System.*getProperty*("user.dir"));

Properties pow = **new** Properties();

FileInputStream file = **new** FileInputStream("D:\\MAVEN\_Workspace\\JavaLearning\\src\\config\\Employee.properties");

pow.load(file);

System.*out*.println(pow.getProperty("name"));

System.*out*.println(pow.getProperty("DOB"));

}

}

Output🡪mohan , 23/12/1984

For example if mohan changed its city,then instead of changing in class file ,good practice is to change it in properties file..

Instead of hardcoding the path we can use as below

System.getProperty(“user.dir”)->which tells the user current directory

We can change the code as

FileInputStream file = **new** FileInputStream(System.*getProperty*("user.dir")+"\\src\\config\\Employee.properties");

4330

2185

1247

# Break, Continue and return statements Example in Selenium webdriver

Branching statements will transfer the control from one statement to another statement. the following are branching statements in java programming  
*1. break*  
*2. continue*  
*3. return*

### 1. break:

break statement is used to stop the execution and comes out of loop. Mostly break statement is used in switch in order to stop fall through and in loops also we use break.

**Example**

package com.seleniumeasy.controlstatements;

**public** **class** BreakDemo {

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

**for**(**int** i=1;i<=20;i++) {

System.**out**.println(i);

**if**(i==7)

**break**;

}

        }

}

**Let us checkout selenium example for this break statement: -**

boolean bValueExists = **false**;

*// For each option in the list, first verify if it's the one that you want and then click on it*

**for** (WebElement option : listOptions) {

**if** (option.getText().contains(sOptionToSelect)) {

option.click();

System.**out**.println("Selected option from the list")

bValueExists = **true**;

**break**;

}

}

In the above example, If we don't use Break statement, it will try to check all the list values and then comes out of the for loop. If say there are list of 20 values, and the One which we are looking is at 2nd position. Now when the condition is satisfied, we should come out of the loop instead of verifying all the remaining 18 values from the list.

You can check here for more detailed examples with selenium [Working with Ajax / JQuery fields](http://seleniumeasy.com/selenium-tutorials/working-with-ajax-or-jquery-auto-complete-text-box-using-webdriver), [Working with checkboxes](http://seleniumeasy.com/selenium-tutorials/working-with-checkbox-using-resuable-methods) etc.

### 2. Continue:

continue statement is also same as break statement the only difference is when break statement executes it comes out of loop where as continue comes out of loop and jumps to the conditional statement of loop. continue is used only in loops to jump from present iteration and executes for next iteration.

**Example:**

package com.seleniumeasy.controlstatements;

**public** **class** ContinueDemo {

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

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

**if** (i % 2 == 0) {

**continue**;

            }

            System.**out**.println(i);

        }

    }

}

### 3. return:

return statements are used in methods which returns a value or statement from current to calling method. return statement must be always last statement in the method

**Example:**

package com.seleniumeasy.controlstatements;

**public** **class** ReturnDemo {

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

**int** c;

ReturnDemo rd =**new** ReturnDemo();

c=rd.add(10,20);

System.**out**.println(c);

}

**public** **int** **add**(**int** a, **int** b){

**int** c=a+b;

**return** c;

    }

}

**Let us look into the below simple selenium example of 'return' :-**

/\*\*

\* This method is used to get the current url

\* **@return**, returns current url

\*/

**public** String **getCurrentURL**()

{

String strURL= **null**;

**try** {

strURL= driver.getCurrentUrl();

}

**catch**(Exception ex) {

System.out.println("Exception occured while getting the current url : "+ex.getStackTrace());

Assert.fail("Exception occured");

}

**return** strURL;

}

The above method is to get current URL of the application. 'driver.getCurrentUrl();' will return URL as string and the same string will be returned to the calling method.

When ever working in a framework, for most of the methods, you will find return statements which can be re-used for multiple times. Each time any method calls, it will try to return the value. You can check here for more examples on return statements. Where we have applied for [Working with Select examples](http://seleniumeasy.com/selenium-tutorials/working-with-select-using-resuable-methods), where some times we may return Boolean value like true/false [when working with Window popups](http://seleniumeasy.com/selenium-tutorials/how-to-handle-windows-popups-using-selenium-webdriver) etc.

# Switch Case example in Java Selenium

It is also one type of Conditional Statements, which is basically used to write for menu driven programs. **When ever there are more number of options to select then we will go for Switch statement** i.e., single expression with many possible options. The same thing can be done using if..else statements but it can get very confusing and if..else statement is difficult to change when it becomes highly nested.

**Syntax:**

**switch**(expression) {

**Case** value1: statement1;

**Break**;

**Case** value2: statement2;

**Break**;

**Case** value3:statement3;

**Break**;

……..

**default**: statement;

}

This expression is compared with each case value until the match found. If no case value is matched then default case will get executed. The expression value type may be of type byte, short, int, char. From 1.5v Wrapper classes and enum are allowed to take as conditional expression and from 1.7v Strings are also allowed.

Integer values are checked for equality using == operator and String value invoke equals() for checking equality.

**Here is the simple example for switch :-**

**public** **class** SwitchDemo {

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

**int** x=2;

**switch** (x) {

**case** 1:

System.**out**.println("one");

**break**;

**case** 2:

System.**out**.println("two");

**break**;

**case** 3:

System.**out**.println("three");

**break**;

**default**:

System.**out**.println("default");

}

}

}

**Output: two**

*Break is optional in switch case. If we don’t mention break for the statement after the case, once the match case value found, the statements after next case also executes irrespective of value matches or not.*

**Let us look this Example:**

**public** **class** SwitchDemo {

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

**int** x=2;

**switch** (x) {

**case** 1:

System.**out**.println("one");

**break**;

**case** 2:

System.**out**.println("two");

**case** 3:

System.**out**.println("three");

**break**;

**default**:

System.**out**.println("default");

}

}

}

Output:  
two  
three  
**This is called fall through in switch. So it is necessary to use break for every case.**

**Now let us see the example in Selenium using Switch.**

**public** **void** **openBrowser**(String browserType) {

**switch** (browserType) {

**case** "firefox":

driver = **new** FirefoxDriver();

**break**;

**case** "chrome":

driver = **new** ChromeDriver();

**break**;

**case** "IE":

driver = **new** InternetExplorerDriver();

**break**;

**default**:

System.**out**.println("browser : " + browserType + " is invalid, Launching Firefox as browser of choice..");

driver = **new** FirefoxDriver();

}

}

Now, based on the browser type value we pass, the driver will be initiated. We can also use the above script to run [selenium tests in multiple browsers](http://seleniumeasy.com/selenium-tutorials/testing-in-multiple-browsers) using the same switch. We can define it as [Parameterized tests](http://seleniumeasy.com/testng-tutorials/parameterization-in-testng) that execute based on each parameter we defined for each test in testng.xml file.

ANT:

Here comes the importance of a build tool like Ant. It stores, executes and automates all process in a sequential order mentioned in Ant's configuration file (usually build.xml).

Relating to ANT demo,Please see AntProject under C:\Users\mohan.m.gowda\AppiumproSeptember\AntProject

1:Create a testing.xml and include 2 classes testA and TestB

2:

-ant is a build and compile tool

-ant helps us to run programs on unix or linux based systems

-ant always looks for build.xml where it will have

properties and targets defined

-different ant commands that we can run using command prompt is

ant clean

ant compile

ant run

ant makexsltreports

- there will be targets set for clean , compile , run ,

makexsltreports in build.xml

-ant compile command will compile the entire java code

and creates build folder after code compile

- ant run command will execute the entire tests

as per the class files mentioned in build.xml to execute

- ant clean command will delete the build folder that gets created after code compile

- ant makexsltreports command will create the xslt reports for the tests executed as mentioned in testng.xml

when batch file is created in notepad and saved in the extension .bat file with below commands in it

s

ant clean compile run

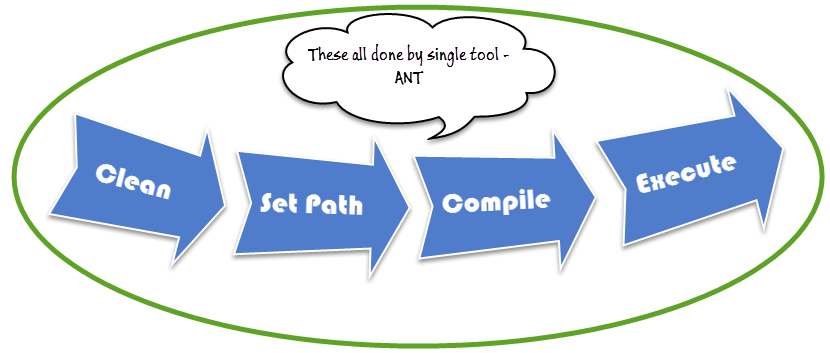
while will clean , compile and run the code of tests as per in testng.xml

-while configuring ant , we have to create a new system variable called ant\_home and give the path of the extracted zip folder of ant downloadsand also put the path of the ant folder/bin in the path variable and tools.jar file has to be copied from jdk folder to jre folder if not found

- there will different properties in build.xml like

ws.home which holds the directory of ant

ws.jars which holds the path of project jars



**Benefit of Ant build**

1. Ant creates the application life cycle i.e. clean, compile, set dependency, execute, report, etc.
2. Third party API dependency can be set by Ant i.e. other Jar file's class path is set by Ant build file.
3. A complete application is created for End to End delivery and deployment.
4. It is a simple build tool where all configurations can be done using XML file and which can be executed from the command line.
5. It makes your code clean as configuration is separate from actual application logic.

**Understanding Build.xml**

Build.xml is the most important component of Ant build tool. For a Java project, all cleaning, setup, compilation and deployment related task are mentioned in this file in XML format. When we execute this XML file using command line or any IDE plugin, all instructions written into this file will get executed in sequential manner.

Let's understand the code within a sample build.XML

* Project tag is used to mention a project name and basedir attribute. The basedir is the root directory of an application
* <project name="YTMonetize" basedir=".">
* Property tags are used as variables in build.XML file to be used in further steps

<property name="build.dir" value="${basedir}/build"/>

<property name="external.jars" value=".\resources"/>

<property name="ytoperation.dir" value="${external.jars}/YTOperation"/>

<property name="src.dir"value="${basedir}/src"/>

* Target tags used as steps that will execute in sequential order. Name attribute is the name of the target. You can have multiple targets in a single build.xml

<target name="setClassPath">

* path tag is used to bundle all files logically which are in the common location

<path id="classpath\_jars">

* pathelement tag will set the path to the root of common location where all files are stored

<pathelement path="${basedir}/"/>

* pathconvert tag used to convert paths of all common file inside path tag to system's classpath format

<pathconvert pathsep=";" property="test.classpath" refid="classpath\_jars"/>

* fileset tag used to set classpath for different third party jar in our project

<fileset dir="${ytoperation.dir}" includes="\*.jar"/>

* Echo tag is used to print text on the console

<echo message="deleting existing build directory"/>

* Delete tag will clean data from given folder

<delete dir="${build.dir}"/>

* mkdir tag will create a new directory

<mkdir dir="${build.dir}"/>

* javac tag used to compile java source code and move .class files to a new folder

<javac destdir="${build.dir}" srcdir="${src.dir}">

<classpath refid="classpath\_jars"/>

</javac>

* jar tag will create jar file from .class files

<jar destfile="${ytoperation.dir}/YTOperation.jar" basedir="${build.dir}">

* manifest tag will set your main class for execution

<manifest>

<attribute name="Main-Class" value="test.Main"/>

</manifest>

* 'depends' attribute used to make one target to depend on another target

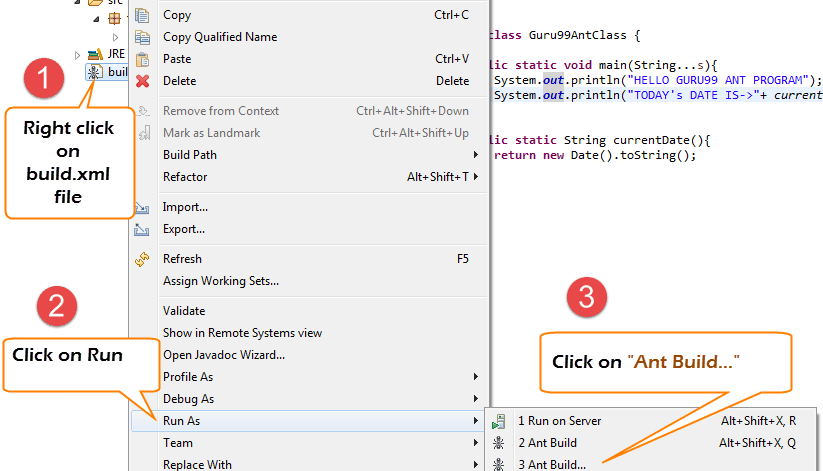
<target name="run" depends="compile">

* java tag will execute main function from the jar created in compile target section

<java jar="${ytoperation.dir}/YTOperation.jar" fork="true"/>

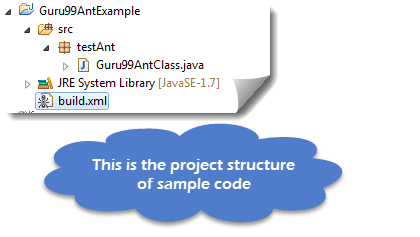
**Run Ant using Eclipse plugin**

To run Ant from eclipse go to build.xml file -> right click on file -> Run as… -> click Build file

[](http://cdn.guru99.com/images/1-2015/031015_1057_UsingApache9.png)

**Example:**

We will take a small sample program that will explain Ant functionality very clearly. Our project structure will look like –

[](http://cdn.guru99.com/images/1-2015/031015_1057_UsingApache10.png)

Here in this example we have 4 targets

1. Set class path for external jars,
2. Clean previously complied code
3. Compile existing java code
4. Run the code

Guru99AntClass.class

package testAnt;

import java.util.Date;

public class Guru99AntClass {

public static void main(String...s){

System.out.println("HELLO GURU99 ANT PROGRAM");

System.out.println("TODAY's DATE IS->"+ currentDate() );

}

public static String currentDate(){

return new Date().toString();

}

}

Build.xml

<?xml version="1.0" encoding="UTF-8" standalone="no"?>

<!--Project tag used to mention the project name, and basedir attribute will be the root directory of the application-->

<project name="YTMonetize" basedir=".">

<!--Property tags will be used as variables in build.xml file to use in further steps-->

<property name="build.dir" value="${basedir}/build"/>

<property name="external.jars" value=".\resources"/>

<property name="ytoperation.dir" value="${external.jars}/YTOperation"/>

<property name="src.dir"value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. name attribute will be the name of the target and 'depends' attribute used to make one target to depend on another target-->

<target name="setClassPath">

<path id="classpath\_jars">

<pathelement path="${basedir}/"/>

</path>

<pathconvert pathsep=";"property="test.classpath" refid="classpath\_jars"/>

</target>

<target name="clean">

<!--echo tag will use to print text on console-->

<echo message="deleting existing build directory"/>

<!--delete tag will clean data from given folder-->

<delete dir="${build.dir}"/>

</target>

<target name="compile" depends="clean,setClassPath">

<echo message="classpath:${test.classpath}"/>

<echo message="compiling.........."/>

<!--mkdir tag will create new director-->

<mkdir dir="${build.dir}"/>

<echo message="classpath:${test.classpath}"/>

<echo message="compiling.........."/>

<!--javac tag used to compile java source code and move .class files to a new folder-->

<javac destdir="${build.dir}" srcdir="${src.dir}">

<classpath refid="classpath\_jars"/>

</javac>

<!--jar tag will create jar file from .class files-->

<jar destfile="${ytoperation.dir}/YTOperation.jar"basedir="${build.dir}">

<!--manifest tag will set your main class for execution-->

<manifest>

<attribute name="Main-Class" value="testAnt.Guru99AntClass"/>

</manifest>

</jar>

</target>

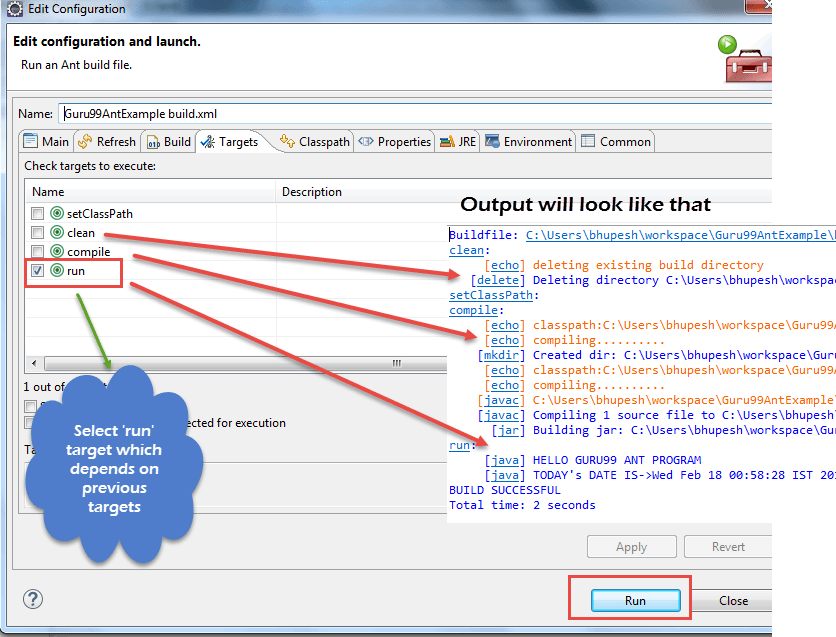
<target name="run" depends="compile">

<!--java tag will execute main function from the jar created in compile target section-->

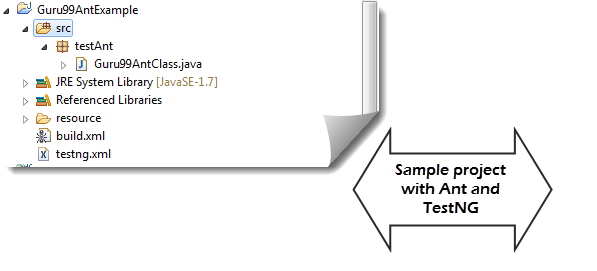
<java jar="${ytoperation.dir}/YTOperation.jar"fork="true"/>

</target>

</project>

[](http://cdn.guru99.com/images/1-2015/031015_1057_UsingApache11.png)

### How to Execute TestNG code using Ant

[](http://cdn.guru99.com/images/1-2015/031015_1057_UsingApache12.png)

Here we will create a class with testng methods and set class path for testing in build.xml.

Now to execute testng method we will create another testng.xml file and call this file from build.xml file.

**Step 1)** We create a **"Guru99AntClass.class"** in package **testAnt**

Guru99AntClass.class

package testAnt;

import java.util.Date;

import org.testng.annotations.Test;

public class Guru99AntClass {

@Test

public void Guru99AntTestNGMethod(){

System.out.println("HELLO GURU99 ANT PROGRAM");

System.out.println("TODAY's DATE IS->"+ currentDate() );

}

public static String currentDate(){

return new Date().toString();

}

}

**Step 2) Create a target to load this class in Build.xml**

<!-- Load testNG and add to the class path of application -->

<target name="loadTestNG" depends="setClassPath">

<!—using taskdef tag we can add a task to run on the current project. In below line, we are adding testing task in this project. Using testing task here now we can run testing code using the ant script -->

<taskdef resource="testngtasks" classpath="${test.classpath}"/>

</target>

**Step 3)** Create testng.xml

testng.xml

<?xml version="1.0"encoding="UTF-8"?>

<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">

<suite name="YT"thread-count="1">

<test name="GURU99TestNGAnt">

<classes>

<class name="testAnt.Guru99AntClass">

</class>

</classes>

</test>

</suite>

**Step 4)** Create Target in Build.xml to run this TestNG code

<target name="runGuru99TestNGAnt" depends="compile">

<!-- testng tag will be used to execute testng code using corresponding testng.xml file. Here classpath attribute is setting classpath for testng's jar to the project-->

<testng classpath="${test.classpath};${build.dir}">

<!—xmlfileset tag is used here to run testng's code using testing.xml file. Using includes tag we are mentioning path to testing.xml file-->

<xmlfileset dir="${basedir}" includes="testng.xml"/>

</testng>

Step 5) The complete Build.xml

<?xml version="1.0"encoding="UTF-8"standalone="no"?>

<!--Project tag used to mention the project name, and basedir attribute will be the root directory of the application-->

<project name="YTMonetize" basedir=".">

<!--Property tags will be used as variables in build.xml file to use in further steps-->

<property name="build.dir"value="${basedir}/build"/>

<!-- put testng related jar in the resource folder -->

<property name="external.jars" value=".\resource"/>

<property name="src.dir" value="${basedir}/src"/>

<!--Target tags used as steps that will execute in sequential order. name attribute will be the name

of the target and 'depends' attribute used to make one target to depend on another target-->

<!-- Load testNG and add to the class path of application -->

<target name="loadTestNG"depends="setClassPath">

<taskdef resource="testngtasks"classpath="${test.classpath}"/>

</target>

<target name="setClassPath">

<path id="classpath\_jars">

<pathelement path="${basedir}/"/>

<fileset dir="${external.jars}" includes="\*.jar"/>

</path>

<pathconvert pathsep=";"property="test.classpath"refid="classpath\_jars"/>

</target>

<target name="clean">

<!--echo tag will use to print text on console-->

<echo message="deleting existing build directory"/>

<!--delete tag will clean data from given folder-->

<delete dir="${build.dir}"/>

</target>

<target name="compile"depends="clean,setClassPath,loadTestNG">

<echo message="classpath:${test.classpath}"/>

<echo message="compiling.........."/>

<!--mkdir tag will create new director-->

<mkdir dir="${build.dir}"/>

<echo message="classpath:${test.classpath}"/>

<echo message="compiling.........."/>

<!--javac tag used to compile java source code and move .class files to a new folder-->

<javac destdir="${build.dir}"srcdir="${src.dir}">

<classpath refid="classpath\_jars"/>

</javac>

</target>

<target name="runGuru99TestNGAnt"depends="compile">

<!-- testng tag will be used to execute testng code using corresponding testng.xml file -->

<testng classpath="${test.classpath};${build.dir}">

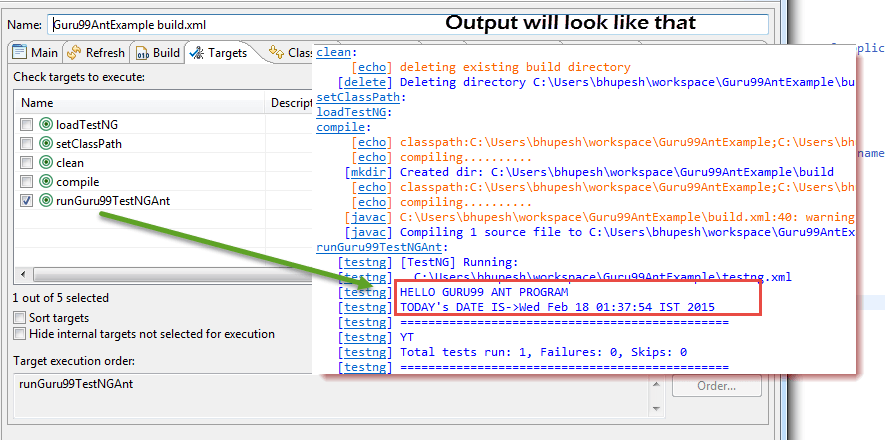
<xmlfileset dir="${basedir}"includes="testng.xml"/>

</testng>

</target>

</project>

**Step 6)** Output

[](http://cdn.guru99.com/images/1-2015/031015_1057_UsingApache13.png)

How to Run the batch file

1: First go the current project path and set the same in the Command Prompt

2: ant clean compile run makexsltreports

3:save as “MY\_TEST\_SUITE.bat”

@echo off

C:

cd D:\MAVEN\_Workspace\Log4j

D:

ant clean compile run makexsltreports

**Listeners**

where ever we need to catch the error but continue the execution we need to add Listeners

1: using the below link, C:\Users\mohan.m.gowda\Desktop\Downloads\Selenium\_Pack\_7\_July\Selenium\_Pack\_7\_July\code\Module 20\CORE\_WEBDRIVER\Core\_Framework\_TestNG\_Webdriver\src\com\qtpselenium\listener

2: add “implements IInvokedMethodListener” to the class

3: add the unimplemented code from TestsListenerAdapter and copy ErrorUtil class to the package

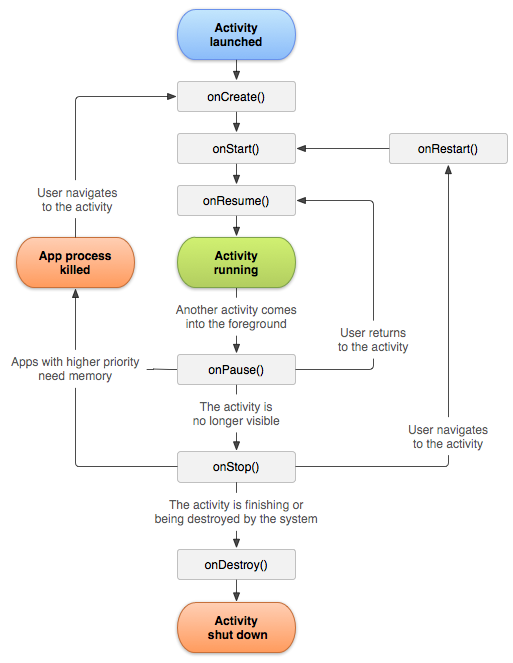
4: add this line under LoginTest

ErrorUtil.*addVerificationFailure*(t);

AVD:Android Vertiual device which can be used to create an virtual device as per our settings.

Activity\_main.xml will be found under Appname🡪App🡪res🡪 Activity\_main.xml which contains design and text of an HTML

MainActivity.java will be present under Appname🡪App🡪 Java 🡪Package name , which can be used to write source code of an activity



The above image shows various functions like 'onCreate()', 'onStart()' which gets called at various points of an Activity Lifecycle. Let's look at each of the methods/functions in detail.

### onCreate

onCreate() is called when your Activity is getting created for the first time. It is called only once during the entire Activity Lifecycle. One of the important things you are supposed to do is to set the Activity Layout through setContentView function.

Also, you can use onCreate to initialize your variables. In any Android application, whenever you create an Activity, the minimum method which you need to override is onCreate.

class MainActivity extends Activity

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

end

If you notice, OnCreate function is getting passed a variable of class Bundle. Bundle is typically used to store the state of your Activity. Take the example of screen rotation, during which your Activity gets killed and OnCreate is called again. You can determine if the Activity was already there using Bundle so that you do not have to create the Activity again.

Why is this necessary, imagine you have a form and user has already filled some of the fields. Suddenly the user rotates his screen. Using Bundle, Android retains the values of these fields and re-populates the data after rotation automatically. The value of Bundle will always be null when Activity is getting created for the first time.

### onStart:

onStart gets called just before the Activity becomes visible to the user. If you notice, onStart is called from two places - after onRestart and OnCreate. onStart is always followed by OnResume or OnStop. You can use onStart to reset Activity data, reinitialize variables etc.

### onResume:

onResume gets called when your Activity comes into the foreground, and it becomes visible to the user. At this point, the Activity is on top of the Activity stack, and the user can start interacting with the Activity. onResume is typically used to register Listeners, bind to Services etc.

onResume is a good place to refresh your UI with any new changes which might have occurred during the period in which the Activity was not visible. For example, if you are polling a Service in the background (like checking for new tweets), onResume is a good place to update your screen with new results.

### onPause:

onPause is called when another android activity comes on top of your Activity. Typically anything that steals your user away from your Activity will result in onPause.

In OnPause, we either release the resources, or save the application data, or stop background threads etc.

It is always guaranteed that whenever your Activity is becoming invisible or partially invisible, onPause will be called. But once onPause is called, Android reserves the right to kill your Activity at any point. Hence you should not be relying on receiving any further events.

### onStop:

onStop is called when your Activity is no longer visible to the user, it is similar to onPause but here you will not see your android activity entirely. You can use this method as well to store the state of your application and shut down time intensive or CPU intensive operations. This method is guaranteed to be called as of API level 11.

So what is the difference between onPause and OnStop ? If an Activity comes into the foreground and fills the screen such that your current activity is not at all visible, your current android activity will be called with both onPause and onStop . If, however, an Activity that comes to foreground does not fill the screen and your current Activity is partially visible, your current Activity will be called with only onPause.

Typically whenever you see a dialog box which requires your attention like battery low, network connection your current android activity becomes partially visible and popup box comes on the top. This is the point where only onPause will be called.

### onRestart:

It is similar to onCreate, but onRestart gets called only after onStop. This is the method which you can use to know if your application is starting afresh or getting restarted.

In onRestart, you will get your application to save the state and reinitialize all the variables. onStart gets called after this.

### OnDestroy:

This is the method which will be called when your Activity is getting killed. This is the final call the Activity will receive in its Lifecycle.

When the user press back button on any Activity the foreground activity gets destroyed and control will return to the previous Activity.

But remember the fact, there is no guaranty that onDestroy will be called. Only when the system is low on resources or user press the back button or if you use finish() explicitly in your code, onDestroy gets called.

Even though you should always use onPause and onStop to clean up resources, release connections etc; onDestory is there to let your app have the final chance to clean things up before the Activity cease to exist.

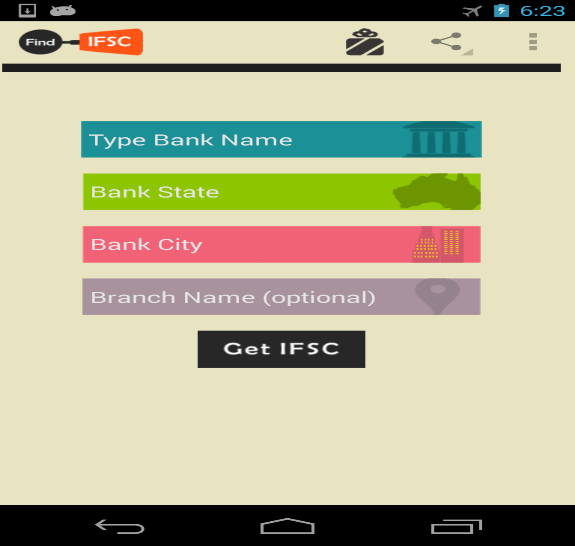
So we have seen the complete Activity Lifecycle functions. Lets see what are the different states of an Activity.

## Activity States

The Android OS uses a priority queue to assist in managing activities running on the device. Based on the state in which an Android Activity is, it will be assigned a priority within the OS. This system helps Android identify Activities that are no longer in use, allowing Android to reclaim memory and resources. Following are the states an Activity can go through during its lifetime:

### Active or Running

Activities are considered active or running if they are in the foreground. This state denotes the top of the Activity stack. The Activity gets assigned the highest priority and will only be killed by Android in extreme situations, such as if the Activity tries to use more memory more than what is available on the device. It can cause the Activity to become unresponsive.



### Paused

An Android Activity is in the Paused state if the device goes to sleep or if it is covered with another Activity partially or completely. Paused activities are very much alive, that is, they maintain all the states and member information. They remain attached to the window manager too. This is considered to be the second highest priority in the Android Activity stack. Paused Activites will only get killed by Android to keep the Active/Running Activity stable and responsive.



### Stopped

Android Activities that are completely obscured by another activity are considered stopped or in the background. Stopped activities still try to retain their state and member information for as long as possible, but stopped activities are considered to be the lowest priority of the three states.

## Android Activity Lifecycle Example App

We have seen various states of Android Activity and seen all the Lifecycle methods. Let's walk through a code example and see these Lifecycle methods in action.

Before proceeding, you should import **CodelearnActivityLifeCycle** project from [Codelearn Example apps on github](https://github.com/pranayairan/Code-Learn-Android-Example). If you are new to github, [download the zip](https://github.com/pranayairan/Code-Learn-Android-Example/archive/master.zip), unzip & import CodelearnActivityLifeCycle project in Eclipse.

* Import the project into workspace
* Once imported, deploy the app & follow the steps below.

### First Run

When your application runs for the first time, you will see onCreate, OnStart and onResume method gets called. Notice the messages which gets displayed.

### Stopping

Press the home button and exit the app. You will notice that when you press the home button, onPause is called first followed by onStop

### Restarting

Now open the application again by clicking on the application icon. Notice that you will see onRestart getting called followed by OnStart and OnResume

### Destroy

Now once you saw onRestart, just press the back button, this will exit your application but notice that onDestory is called when you exit. And before onDestory, onPause and onStop is called.

Summarizing the chapter, you understood why an Activity Lifecycle exists, what is its importance and what are the different Lifecycle states which an activity goes through.

This Keyword:

This can be used inside any method to refer the current object

Android operating system is a stack of software components which is roughly divided into five sections and four main layers as shown below in the architecture diagram.



**Linux kernel**

At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

**Libraries**

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

**Android Libraries**

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

* **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
* **android.content** − Facilitates content access, publishing and messaging between applications and application components.
* **android.database** − Used to access data published by content providers and includes SQLite database management classes.
* **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
* **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
* **android.text** − Used to render and manipulate text on a device display.
* **android.view** − The fundamental building blocks of application user interfaces.
* **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
* **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

**Android Runtime**

This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.

The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

The Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

**Application Framework**

The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

The Android framework includes the following key services −

* **Activity Manager** − Controls all aspects of the application lifecycle and activity stack.
* **Content Providers** − Allows applications to publish and share data with other applications.
* **Resource Manager** − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
* **Notifications Manager** − Allows applications to display alerts and notifications to the user.
* **View System** − An extensible set of views used to create application user interfaces.

**Applications**

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.