**Hybrid Inheritance:**

It is a mix of two or more types of inheritance.

Example:-

**public** **class** Parent {

**void** display() {

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

}

}

**public** **interface** Mother {

**void** display();

}

**public** **class** Child **extends** Parent **implements** Parent2{

**void** display() {

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

}

**public** **void** disp() {

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

}

}

**public** **class** GrandChild {

**void** display() {

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

}

}

**class** Daughter **extends** Parent {

**void** display() {

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

}

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

Daughter daughter =**new** Daughter();

daughter.display();

}

}

**Multiple Inheritance:**

Multiple Inheritance is a feature of an object-oriented concept, where a class can inherit properties of more than one parent class. The problem occurs when there exist methods with the same signature in both the superclasses and subclass. On calling the method, the compiler cannot determine which class method to be called and even on calling which class method gets the priority.

Java does not support multiple inheritance. This means that a class cannot extend more than one class.

However, a class can implement one or more interfaces, which has helped Java get rid of the impossibility of multiple inheritance. The extends keyword is used once, and the parent interfaces are declared in a comma-separated list.

Example:

**public** **interface** Imouse {

**public** **void** click();

}

**public** **class** Computer {

}

**public** **class** CpuImpl **extends** Computer **implements** Imouse{

@Override

**public** **void** click() {

System.***out***.println("resource gets selected");

}

}

**public** **class** MultipleInheritanceExp {

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

CpuImpl cpuImpl=**new** CpuImpl();

cpuImpl.click();

cpuImpl.doubleClick();

cpuImpl.rightClick();

}

}

**length**

* An array is an object that holds a fixed number of values of the same type.
* The length variable in an array returns the length of an array i.e. a number of elements stored in an array.
* Once arrays are initialized, its length cannot be changed, so the length variable can directly be used to get the length of an array.
* The length variable is used only for an array.

Example

**public** **class** ArrrayLength {

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

**int** array[] = {1, 2, 3, 4, 5, 6, 7};

System.***out***.println("Length of an array is: " + array.length);

}

}

**length()**

* The length() method is a static method of String class.
* The length() returns the length of a string object i.e. the number of characters stored in an object.
* String class uses this method because the length of a string can be modified using the various operations on an object.
* The String class internally uses a char[] array that it does not expose to the outside world.

Example

**public** **class** StringLength {

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

String str = "Welcome to Tutorials Point";

System.***out***.println("Length of String is: " + str.length());

}

}

//sorting array and for each loop

**public** **class** ArrayExp2 {

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

**int**[] a= {20,30,40,10,50,5,45,11};

**for**(**int** i=0;i<a.length-1;i++) {

**for**(**int** j=i+1;j<a.length;j++) {

**if**(a[i]>a[j]) {

**int** temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

}

// for(int i=0;i<a.length;i++) {

// System.out.print(a[i]+" ");

//}

**for**(**int** x : a) {

System.***out***.print(x+" ");

}

}

}