ARRAYS AND COLLECTIONS SORT

There are two in-built methods to sort in Java.

Arrays.Sort() works for arrays which can be of primitive data type.

```
package sample;
// A sample Java program to demonstrate working of
// Arrays.sort().
// It by default sorts in ascending order.
import java.util.Arrays;
public class ArraysSort {
    public static void main(String[] args)
    {
        int[] arr = { 13, 7, 6, 45, 21, 9, 101, 102 };
        Arrays.sort(arr);
        System.out.printf("Modified arr[] : %s",
                Arrays.toString(arr));
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Modified arr[] : [6, 7, 9, 13, 21, 45, 101, 102]
Process finished with exit code 0
```

Collections.sort() works for objects Collections like ArrayList and LinkedList.

```
package sample;
// Java program to demonstrate working of Collections.sort()
import java.util.*;

public class CollectionsSort {
    public static void main(String[] args)
    {
        // Create a list of strings
        ArrayList<String> al = new ArrayList<String>();
        al.add("Geeks For Geeks");
        al.add("Friends");
        al.add("Dear");
        al.add("Is");
        al.add("Superb");

      /* Collections.sort method is sorting the
      elements of ArrayList in ascending order. */
        Collections.sort(al);

        // Let us print the sorted list
        System.out.println("List after the use of"
                + " Collection.sort() :\n" + al);
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
List after the use of Collection.sort() :
[Dear, Friends, Geeks For Geeks, Is, Superb]
```

CLASSES AND OBJECTS

Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. MODIFIERS : A class can be public or has default access
2. CLASS NAME: The name should begin with an initial letter (capitalized by convention).

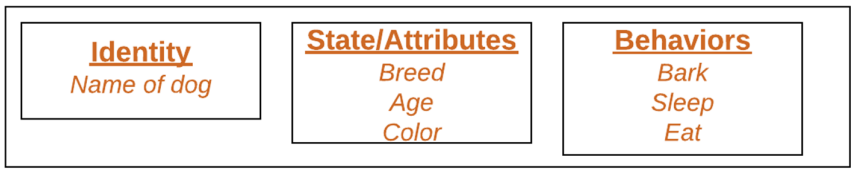
3. SUPERCLASS (IF ANY): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. INTERFACES (IF ANY): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. BODY: The class body surrounded by braces, { }.

Object

It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects.

1. STATE: It is represented by attributes of an object. It also reflects the properties of an object.
2. BEHAVIOR: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. IDENTITY: It gives a unique name to an object and enables one object to interact with other objects.

Example of an object: dog




```java
package sample;
// Class Declaration
public class Dog {
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;

    // Constructor Declaration of Class
    public Dog(String name, String breed,
               int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }

    // method 1
    public String getName()
    {
        return name;
    }

    // method 2
    public String getBreed()
    {
        return breed;
    }

    // method 3
    public int getAge()
    {
        return age;
    }

    // method 4
    public String getColor()
    {
        return color;
    }
```

```
    @Override
    public String toString()
    {
        return("Hi my name is "+ this.getName()+
                ".\nMy breed,age and color are " +
                this.getBreed()+"," + this.getAge()+
                ","+ this.getColor());
    }

    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        System.out.println(tuffy.toString());
    }

}
```

WAYS TO CREATE OBJECT OF A CLASS

Using new keyword: It is the most common and general way to create object in java.

Example:

```
// creating object of class Test
Test t = new Test();
```

Using Class.forName(String className) method : There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name.

```
// creating object of public class Test
// consider class Test present in com.p1 package
    Test obj = (Test)Class.forName("com.p1.Test").newInstance();
```

Using clone() method: clone() method is present in Object class. It creates and returns a copy of the object.

```
// creating object of class Test
Test t1 = new Test();
// creating clone of above object
Test t2 = (Test)t1.clone();
```

Deserialization: De-serialization is technique of reading an object from the saved state in a file.

```
FileInputStream file = new FileInputStream(filename);
ObjectInputStream in = new ObjectInputStream(file);
Object obj = in.readObject();
```

INHERTANCE

Inheritance is an important pillar of OOP (Object Oriented Programming)

Important terminology:

- **Super Class:** The class whose features are inherited is known as super class (or a base class or a parent class).
- **Sub Class:** The class that inherits the other class is known as sub class (or a derived class, extended class, or child class).

How to use inheritance in Java

*The keyword used for inheritance is **extends**.*

Syntax:

```
class derived-class extends base-class
{
        //methods and fields
        }
```

Example: In below example of inheritance, class Bicycle is a base class, class MountainBike is a derived class which extends Bicycle class and class Test is a driver class to run program.

```java
package sample;
//Java program to illustrate the
// concept of inheritance
// base class
public class Bicycle {
    // the Bicycle class has two fields
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    // the Bicycle class has three methods
    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

    public void speedUp(int increment)
    {
        speed += increment;
    }

    // toString() method to print info of Bicycle
    public String toString()
    {
        return("No of gears are "+gear
                +"\n"
                + "speed of bicycle is "+speed);
    }
}
```

```java
package sample;
// derived class
public class MountainBike extends Bicycle
{

    // the MountainBike subclass adds one more field
    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear,int speed,
                        int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    // the MountainBike subclass adds one more method
    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method
    // of Bicycle to print more info
    @Override
    public String toString()
    {
        return (super.toString()+
                "\nseat height is "+seatHeight);
    }
}
```

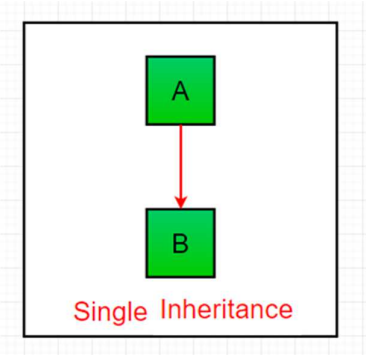
```java
package sample;
// driver class
public class TestDriverClass {
    public static void main(String args[])
    {
        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
No of gears are 3
speed of bicycle is 100
seat height is 25

Process finished with exit code 0
```

Types of Inheritance in Java

SINGLE INHERITANCE: In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.




```java
package inheritanceInJava;
//Java program to illustrate the
// concept of single inheritance
import java.lang.*;
public class one {
    public void print()
    {
        System.out.println("Have a great day");
    }
}
```

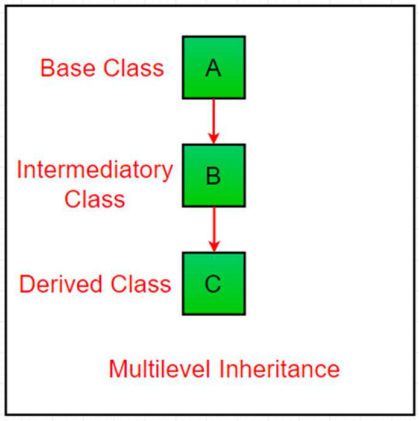
```java
package inheritanceInJava;

public class two extends one
{
    public void print_for()
    {
        System.out.println(" Thanks");
    }
}
```

```java
package inheritanceInJava;
// Driver class
public class Main {
    public static void main(String[] args)
    {
        two g = new two();
        g.print();
        g.print_for();
        g.print();
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Have a great day
 Thanks
Have a great day

Process finished with exit code 0
```

MULTILEVEL INHERITANCE: In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.




```java
package multilevelInheritance;
// Java program to illustrate the
// concept of Multilevel inheritance
import java.lang.*;
public class one {
    public void printHide()
    {
        System.out.println("Hide");
    }
}
```

```java
package multilevelInheritance;
public class two extends one
{
    public void printAnd()
    {
        System.out.println("and");
    }
}
```

```java
package multilevelInheritance;
public class three extends two
{
    public void printSeek()
    {
        System.out.println("Seek");
    }
}
```

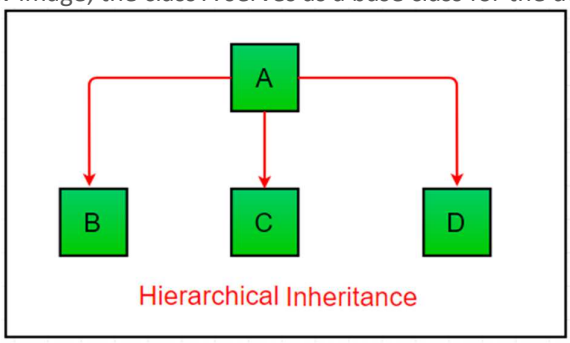
```java
package multilevelInheritance;
// Drived class
public class Main {
    public static void main(String[] args)
    {
       three three = new three();
       three.printHide(); //calling grand parent class method
       three.printAnd(); //calling parent class method
       three.printSeek(); //calling local method
```

```
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Hide
and
Seek

Process finished with exit code 0
```

HIERARCHICAL INHERITANCE: In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class. In below image, the class A serves as a base class for the derived class B, C and D.




```
package hierarchicalInheritance;
// Java program to illustrate the
// concept of Hierarchical inheritance

public class One {
    public void printHot()
    {
        System.out.println("Hot");
    }
}
```

```
package hierarchicalInheritance;

public class Two extends One {
    public void printAnd()
    {
        System.out.println("And");
    }
}
```

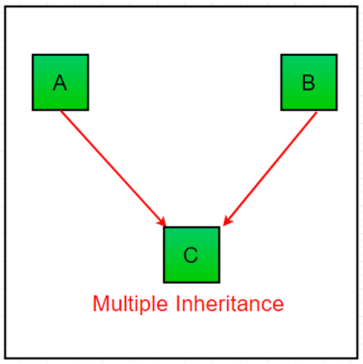
```
package hierarchicalInheritance;

public class Three extends One {
   public void printCold (){
       System.out.printf("Cold");
   }
}
```

```
package hierarchicalInheritance;
// Drived class
public class Main {
    public static void main (String arg[]){
        Two two = new Two();
        Three three = new Three();
        two.printHot(); //calling parent class method
        two.printAnd(); //calling local class method
        three.printHot(); //calling parent class method
        three.printCold(); //calling local class method
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Hot
And
Hot
Cold
Process finished with exit code 0
```

MULTIPLE INHERITANCE (THROUGH INTERFACES): In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Please note that Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces. In image below, Class C is derived from interface A and B.




```java
package multipleIinheritance;

public interface One {
    public void printDay();
}
```

```java
package multipleIinheritance;

public interface Two {
    public void printNight();
}
```

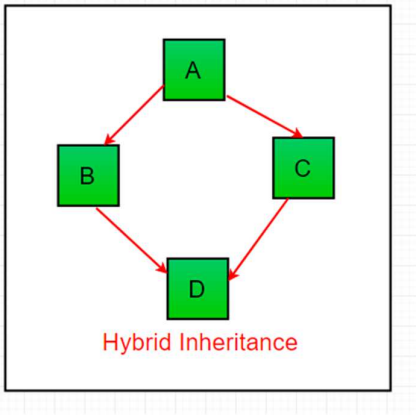
```java
package multipleIinheritance;

public interface Three extends One, Two {
    public void printAwesome ();
}
```

```java
package multipleIinheritance;
// Java program to illustrate the
// concept of Multiple inheritance

public class Main implements Three {
    public void printAwesome() {
        System.out.printf("Awesome ");
    }

    public void printDay() {
        System.out.printf("Day ");
    }

    public void printNight() {
        System.out.printf("Night ");
    }
    public static void main (String arg[]){
        Main main = new Main();
        main.printAwesome(); // Calling Parent Interface Method
        main.printDay(); // Calling local Interface Method
        main.printAwesome(); // Calling Parent Interface Method
        main.printNight(); // Calling local Interface Method
    }
}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
Awesome Day Awesome Night
Process finished with exit code 0
```

HYBRID INHERITANCE (THROUGH INTERFACES): It is a mix of two or more of the above types of inheritance. Since java doesn't support multiple inheritance with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.



ENCAPSULATION

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

- declare class variables/attributes as `private` (only accessible within the same class)
- provide public **setter** and **getter** methods to access and update the value of a `private` variable

Get and Set

`private` variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **getter** and **setter** methods.
The `get` method returns the variable value, and the `set` method sets the value.

```
package encapsulate;
// Java program to demonstrate encapsulation
public class Encapsulate {
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String userName;
    private int userRoll;
    private int userAge;

    // get method for age to access
    // private variable userAge
    public int getAge()
    {
        return userAge;
    }

    // get method for name to access
    // private variable userName
    public String getName()
    {
        return userName;
    }

    // get method for roll to access
    // private variable userRoll
    public int getRoll()
    {
        return userRoll;
    }
```

```
    // set method for age to access
    // private variable userage
    public void setAge( int newAge)
    {
        userAge = newAge;
    }

    // set method for name to access
    // private variable userName
    public void setName(String newName)
    {
        userName = newName;
    }

    // set method for roll to access
    // private variable userRoll
    public void setRoll( int newRoll)
    {
        userRoll = newRoll;
    }

}
```

The program to access variables of the class EncapsulateDemo is shown below:

```
package encapsulate;

public class TestEncapsulation {
    public static void main (String[] args)
    {
        Encapsulate obj = new Encapsulate();

        // setting values of the variables
        obj.setName("Jai");
        obj.setAge(19);
        obj.setRoll(66);

        // Displaying values of the variables
        System.out.println("User's name: " + obj.getName());
        System.out.println("User's age: " + obj.getAge());
        System.out.println("User's roll: " + obj.getRoll());

        // Direct access of userRoll is not possible
        // due to encapsulation
        // System.out.println("User's roll: " + obj.userName);
    }

}
```

Output

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java.exe" ...
User's name: Jai
User's age: 19
User's roll: 66

Process finished with exit code 0
```

ABSTRACTION

Data **abstraction** is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next chapter).

The `abstract` keyword is a non-access modifier, used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects
- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

An abstract class can have both abstract and regular methods: