Object & Classes

Java is an Object-Oriented Language. It supports

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

- **Object - 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.**

```
public class Puppy{
public Puppy(){
}

public Puppy(String name){
// This constructor has one parameter, name.
}
}
```

- **Class - A class can be defined as a template/blue print that describes the behaviors/states that object of its type support.**

```
public class Dog{
String breed;
int age;
String color;

void barking(){
}

void hungry(){
}

void sleeping(){
}
}
```

A class can contain any of the following variable types.

- **Local variables: Variables defined inside methods, constructors or blocks are called local variables. The variable will be declared and initialized within the method and the variable will be destroyed when the method has completed.**
- **Instance variables: Instance variables are variables within a class but outside any method. These variables are instantiated when the class is loaded. Instance variables can be accessed from inside any method, constructor or blocks of that particular class.**
- **Class variables: Class variables are variables declared within a class, outside any method, with the static keyword.**

Constructors:

- **Every class has a constructor. If we do not explicitly write a constructor for a class the Java compiler builds a default constructor for that class.**
- **Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor**

Creating an object:

- **Declaration:** A variable declaration with a variable name with an object type.
- **Instantiation:** The 'new' keyword is used to create the object.
- **Initialization:** The 'new' keyword is followed by a call to a constructor. This call initializes the new object.
-

```
public class Puppy{

public Puppy(String name){
// This constructor has one parameter,
name. System.out.println("Passed Name
is :"+ name );
}
 public static void main(String[]args){
// Following statement would create an
object myPuppy
Puppy myPuppy =new Puppy("tommy");
}
}
```

Accessing Instance variables and Methods:

Instance variables and methods are accessed via created objects.

```
public class Puppy{

int puppyAge;

public Puppy(String name){
// This constructor has one parameter, name.
System.out.println("Passed Name is :"+ name );
}
public void setAge(int age ){
puppyAge = age;
}

public int getAge(){
System.out.println("Puppy's age is :"+ puppyAge );
return puppyAge;
}
 public static void main(String[]args){
/* Object creation */
Puppy myPuppy =newPuppy("tommy");

/* Call class method to set puppy's age */
myPuppy.setAge(2);

/* Call another class method to get puppy's age */
  myPuppy.getAge();

/* You can access instance variable as follows as well */
System.out.println("Variable Value :"+ myPuppy.puppyAge );
}
}
```

Points to Remember:

1. Every class has a constructor whether it's a normal class or a abstract class.
2. Constructors are not methods and they don't have any return type.
3. Constructor name should match with class name .
4. Constructor can use any access specifier, they can be declared as private also. Private constructors are possible in java but there scope is within the class only.
5. **Like constructors method can also have name same as class name, but still they have return type, though which we can identify them that they are methods not constructors.**
6. If you don't implement any constructor within the class, compiler will do it for.
7. **this() and super() should be the first statement in the constructor code.** If you don't mention them, compiler does it for you accordingly.
8. Constructor overloading is possible but overriding is not possible. Which means we can have overloaded constructor in our class but we can't override a constructor.
9. Constructors can not be inherited.
10. If Super class doesn't have a no-arg(default) constructor then compiler would not insert a default constructor in child class as it does in normal scenario.
11. Interfaces **do not have constructors**.
12. Abstract class can have constructor and it gets invoked when a class, which implements interface, is instantiated. (i.e. object creation of concrete class).
13. A constructor can also invoke another constructor of the same class – By using this(). If you want to invoke a parameterized constructor then do it like this: **this(parameter list)**.

Difference between Constructor and Method

1. The purpose of constructor is to initialize the object of a class while the purpose of a method is to perform a task by executing java code.
2. Constructors cannot be abstract, final, static and synchronised while methods can be.
3. Constructors do not have return types while methods do.

Singleton Classes:

The Singleton's purpose is to control object creation, limiting the number of objects to one only. Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields. Singletons often control access to resources such as database connections or sockets. For example, if you have a license for only one connection for your database or your JDBC driver has trouble with multithreading, the Singleton makes sure that only one connection is made or that only one thread can access the connection at a time

```
// File Name: Singleton.java
public class Singleton{

private static Singleton singleton =new Singleton();

/* A private Constructor prevents any other
  * class from instantiating.
  */
private Singleton(){}

/* Static 'instance' method */
public static Singleton getInstance(){
return singleton;
}
/* Other methods protected by singleton-ness */
protected static void demoMethod(){
System.out.println("demoMethod for singleton");
}
}

// File Name: SingletonDemo.java
public class SingletonDemo{
public static void main(String[] args){
Singleton tmp =Singleton.getInstance();

   tmp.demoMethod();
}
}
```

Inheritance

The most commonly used keyword would be extends and implements. These words would determine whether one object IS-A type of another.

In Object Oriented terms the following are true:

- **Animal is the superclass of Mammal class.**
- **Animal is the superclass of Reptile class.**
- **Mammal IS-A Animal**
- **Reptile IS-A 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);
}
}
```

IS-A (Inheritance) :

IS-A relationship denotes “one object is type of another”. IS-A relation denotes Inheritance methodology.

Has-A (Association) :

In Object orientation design, We can say “class one is in Has-A relationship with class B if class A holds reference of Class B”.

```
class Laptop {
   // Code for Laptop class goes here.
}
interface Formatable {
   // Members of Interface.
}
class Dell extends Laptop implements Formatable {
   // More code related to Dell goes here.
   // Dell class will inherit all accessible members of Laptop class.
   // Dell IS-A Laptop.
   // And Dells class also implements all method of Formatable interface, since
   // Dell is not an abstract class.
   // so Dell IS-A Formatable.
}
public class TestRelationship {
   public static void main(String[] args) {
      Dell oDell = new Dell();
      if (oDell instanceof Laptop) {
         System.out.println("Dell IS-A Laptop.");
      }
      if (oDell instanceof Formatable) {
         System.out.println("Dell IS-A Formatable.");
```

```
class HardDisk {
    public void writeData(String data) {
        System.out.println("Data is being written : " + data);
    }
}

class UseDell {
    // segate is referece of HardDisk class in UseDell class.
    // So, UseDell Has-A HardDisk
    HardDisk seagate = new HardDisk();
    public void save (String data) {
        seagate.writeData(data);
    }
}
```

Overridding:

The benefit of overriding is: ability to define a behavior that's specific to the subclass type which means a subclass can implement a parent class method based on its requirement.

```
class Animal{

public void move(){
System.out.println("Animals can move");
}
}
class Dog extendsAnimal{
public void move(){
System.out.println("Dogs can walk and run");
}
public void bark(){
System.out.println("Dogs can bark");
}
}

public class TestDog{

public static void main(String args[]){
Animal a =new Animal();// Animal reference and object
Animal b =new Dog();// Animal reference but Dog object

  a.move();// runs the method in Animal class
  b.move();//Runs the method in Dog class
  b.bark();
}
}
```

super Keyword:

When invoking a superclass version of an overridden method the super keyword is used.

```
class Animal{

public void move(){
System.out.println("Animals can move");
}
}
class Dog extends Animal{
public void move(){
super.move();// invokes the super class method
System.out.println("Dogs can walk and run");
}
}

public class TestDog{

public static void main(String args[]){

Animal b =new Dog();// Animal reference but Dog
 object
b.move();//Runs the method in Dog class
}
}
```

this is a **keyword in Java.** It can be used inside the method or constructor of a class. It(this) works as a reference to the current object, whose method or constructor is being invoked.

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
rollno=rollno;
name=name;
fee=fee;
}
void display(){
System.out.println(rollno+" "+name+" “+fee);
}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:
0 null 0.0
0 null 0.0

```
class Student{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee){
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
```

```
class TestThis2{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000
112 sumit 6000
```

```
//this keyword to invoke a method
class A{
void m(){System.out.println("hello m");}
void n(){
System.out.println("hello n");
//m();//same as this.m()
this.m();
}
}
class TestThis4{
public static void main(String args[]){
A a=new A();
a.n();
}}
```

```
//this keyword to invoke current constructor
class A{
A(){System.out.println("hello a");}
A(int x){
this();
System.out.println(x);
}
}
class TestThis5{
public static void main(String args[]){
A a=new A(10);
}}
```