Understanding Classes and Objects in Java

The term *Object-Oriented* explains the concept of organizing the software as a combination of different types of objects that incorporates both data and behavior. Hence, Object_oriented programming(OOPs) is a programming model, that simplifies software development and maintenance by providing some rules. Programs are organised around objects rather than action and logic. It increases the flexibility and maintainability of the program. Understanding the working of the program becomes easier, as OOPs brings data and its behavior(methods) into a single(objects) location.

Basic concepts of OOPs are:

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

This article deals with **Objects** and **Classes** in Java.

Requirements of Classes and Objects in Object Oriented Programming

Classes: 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. Classes are required in OOPs because:

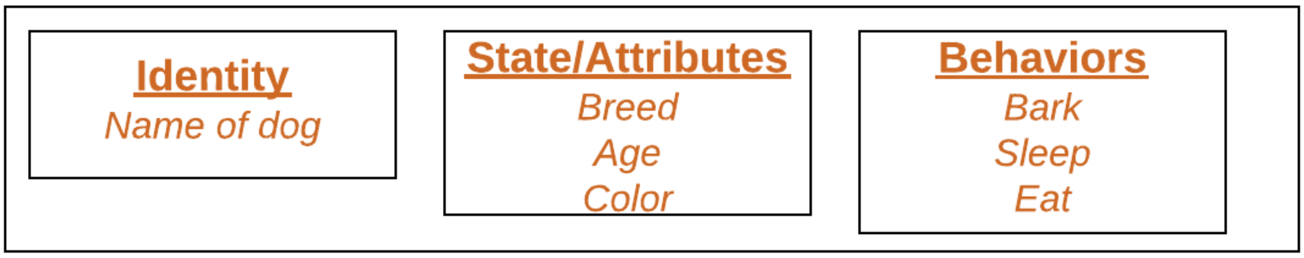
- It provides template for creating objects, which can bind code into data.
- It has definitions of methods and data.
- It supports inheritance property of Object Oriented Programming and hence can maintain class hierarchy.
- It helps in maintaining the access specifications of member variables.

Objects: It is the basic unit of Object Oriented Programming and it represents the real life entities.
Real-life entities share two characteristics : they all have attributes and behavior.
An object consists of:

- **State:** It is represented by *attributes* of an object. It also shows properties of an object.
- **Behavior:** It is represented by *methods* of an object. It shows response of an object with other objects.
- **Identity:** It gives a unique name to an object. It also grants permission to one object to interact with other objects.

Objects are required in OOPs because they can be created to call a non-static function which are not present inside the Main Method but present inside the Class and also provide the name to the space which is being used to store the data.

Example : For addition of two numbers, it is required to store the two numbers separately from each other, so that they can be picked and the desired operations can be performed on them. Hence creating two different objects to store the two numbers will be an ideal solution for this scenario.



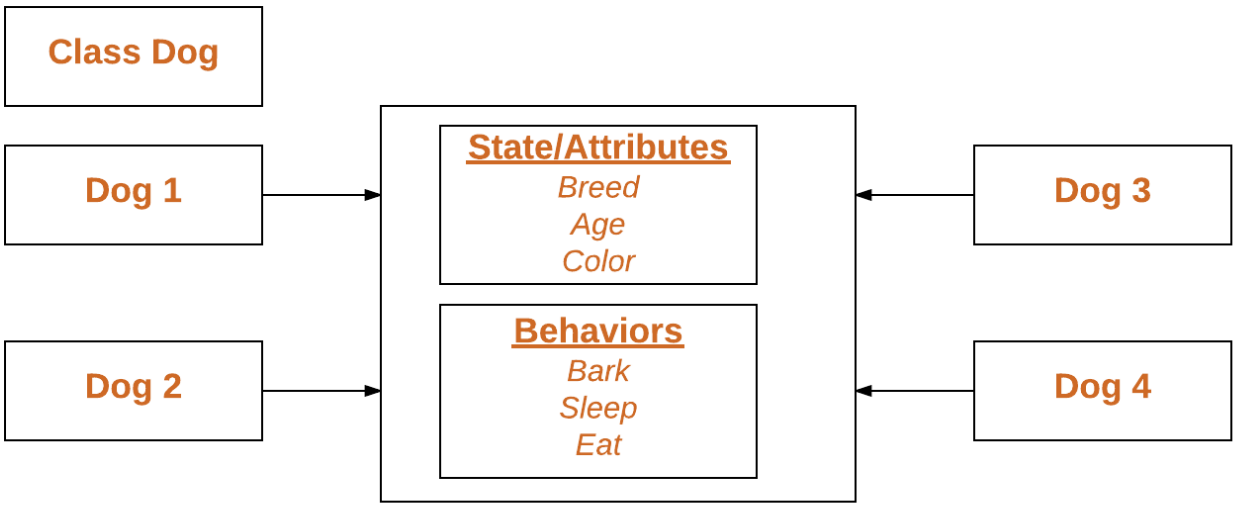
Example to demonstrate the use of Objects and classes in OOPs

Objects relate to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example :



As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general, we **can’t** create objects of an abstract class or an interface.

```
Dog tuffy;
```

If we declare reference variable(tuffy) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object using new

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

```
// Java program to illustrate the concept
// of classes and objects

// Class Declaration
```

```java
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());
    }
}
```

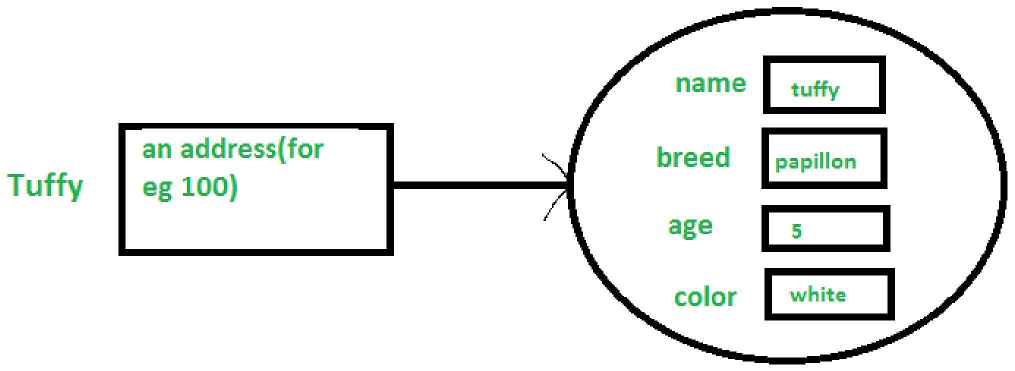
Output:

```
Hi my name is tuffy.
My breed, age and color are papillon, 5, white
```

- This class contains a single constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type. The Java compiler differentiates the constructors based on the number and the type of the arguments. The constructor in the *Dog* class takes four arguments. The following statement provides “tuffy”, “papillon”, 5, “white” as values for those arguments:

  ```
  Dog tuffy = new Dog("tuffy", "papillon", 5, "white");
  ```

The result of executing this statement can be illustrated as :



Note : All classes have at least **one** constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor. This default constructor calls the class parent's no-argument constructor (as it contain only one statement i.e super();), or the *Object* class constructor if the class has no other parent (as Object class is parent of all classes either directly or indirectly).

Different ways to create Objects

1. ***Using new keyword:*** It is the simplest way to create object. By using this method, the desired constructor can be called.
   Syntax:

```
ClassName ReferenceVariable = new ClassName();
```

```
// Java program to illustrate the
// creating and accessing objects
// using new keyword

// base class
class Dog {

    // the class Dog has two fields
    String dogName;
    int dogAge;

    // the class Dog has one constructor
    Dog(String name, int age)
    {
        this.dogName = name;
        this.dogAge = age;
    }
}

// driver class
public class Test {
    public static void main(String[] args)
    {
        // creating objects of the class Dog
        Dog ob1 = new Dog("Bravo", 4);
        Dog ob2 = new Dog("Oliver", 5);

        // accessing the object data through reference
        System.out.println(ob1.dogName + ", " + ob1.dogAge);
        System.out.println(ob2.dogName + ", " + ob2.dogAge);
    }
}
```

Output:

```
Bravo, 4
Oliver, 5
```

2. ***Using Class.newInstance() method:*** It is used to create new class dynamically. It can invoke any no-argument constructor. This method return class **Class** object on which newInstance() method is called, which will return the object of that class which is being passed as command line argument.
   Reason for different exceptions raised:-

   ClassNotFoundException will occur, if the passed class doesn't exist.
   InstantiationException will occur, if the passed class doesn't contain default constructor as newInstance() method internally calls the default constructor of that particular class.
   IllegalAccessException will occur, if the driving class doesn't has the access to the definition of specified class definition.

   Syntax:

```
ClassName ReferenceVariable =
                   (ClassName) Class.forName("PackageName.ClassName").newInstance();
```

```
// Java program to demostrate
// object creation using newInstance() method

// Base class
class Example {
    void message()
    {
        System.out.println("Hello Geeks !!");
    }
}

// Driver class
class Test {
    public static void main(String args[])
    {
        try {
            Class c = Class.forName("Example");
            Example s = (Example)c.newInstance();
            s.message();
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

   Output:

```
Hello Geeks !!
```

3. ***Using newInstance() method for Constructor class:*** It is a reflective way to create object. By using it one can call parametrized and private constructor. It wraps the thrown exception with an InvocationTargetException. It is used by different frameworks- Spring, Hibernate, Struts etc. Constructor.newInstance() method is preffered over Class.newInstance() method.
   Syntax:

```
Constructor constructor = ClassName.class.getConstructor();
 ClassName ReferenceVariable = constructor.newInstance();
```

   Example:

```
// java program to demostrate
// creation of object
// using Constructor.newInstance() method

import java.lang.reflect.Constructor;
import java.lang.reflect.InvocationTargetException;

public class ConstructorExample {

    // different exception is thrown
    public static void main(String[] args)
        throws NoSuchMethodException,
               SecurityException,
               InstantiationException,
               IllegalAccessException,
               IllegalArgumentException,
               InvocationTargetException
    {
        Constructor constructor = ExampleClass.class
                                      .getConstructor(String.class);
        ExampleClass exampleObject = (ExampleClass)constructor
                                         .newInstance("GeeksForGeeks");
        System.out.println(exampleObject.getemp_name());
    }
}

class ExampleClass {

    // private variable declared
    private String emp_name;

    public ExampleClass(String emp_name)
    {
        this.emp_name = emp_name;
    }

    // get method for emp_named to access
    // private variable emp_name
    public String getemp_name()
    {
        return emp_name;
    }

    // set method for emp_name to access
    // private variable emp_name
    public void setemp_name(String emp_name)
    {
        this.emp_name = emp_name;
    }
}
```

Output:

```
GeeksForGeeks
```

4. ***Using clone() method:*** It is used to make clone of an object. It is the easiest and most efficient way to copy an object. In code, **java.lang.Cloneable** interface must be implemented by the class whose object clone is to be created. If Cloneable interface is not implemented, clone() method generates **CloneNotSupportedException**.
   Syntax:

```
ClassName ReferenceVariable = (ClassName) ReferenceVariable.clone();
```

Example:

```
// java program to demonstrate
// object creation using clone() method

// employee class whose objects are cloned
class Employee implements Cloneable {
    int emp_id;
    String emp_name;

    // default constructor
    Employee(String emp_name, int emp_id)
    {
        this.emp_id = emp_id;
        this.emp_name = emp_name;
    }

    public Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
}

// driver class
public class Test {

    public static void main(String args[])
    {

        try {
            Employee ob1 = new Employee("Tom", 201);

            // Creating a new reference variable ob2
            // which is pointing to the same address as ob1
            Employee ob2 = (Employee)ob1.clone();

            System.out.println(ob1.emp_id + ", " + ob1.emp_name);
            System.out.println(ob2.emp_id + ", " + ob2.emp_name);
        }
        catch (CloneNotSupportedException c) {
            System.out.println("Exception: " + c);
        }
    }
}
```

Output:

```
201, Tom
201, Tom
```

5. ***Using deserialization:*** To deserialize an object, first implement a **serializable interface** in the class. No constructor is used to create an object in this method.

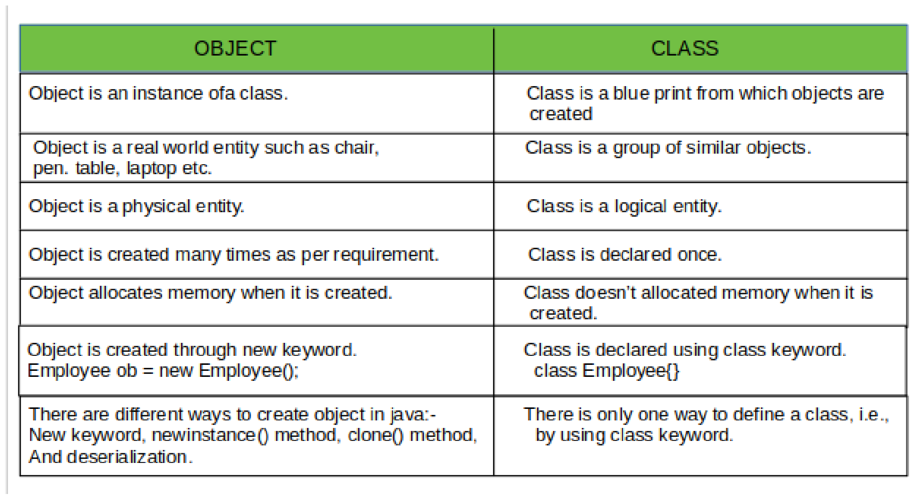
Syntax:

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream(FileName));
ClassName ReferenceVariable = (ClassName) in.readObject();
```

Example:

```
// Java code to demonstrate object
// creation by deserialization

import java.io.*;

// Base class
class Example implements java.io.Serializable {

    public int emp_id;
    public String emp_name;

    // Default constructor
    public Example(int emp_id, String emp_name)
    {
        this.emp_id = emp_id;
        this.emp_name = emp_name;
    }
}

// Driver class
class Test {
    public static void main(String[] args)
    {
        Example object = new Example(1, "geeksforgeeks");
        String filename = "file1.ser";

        // Serialization
        try {

            // Saving of object in a file
            FileOutputStream file1 = new FileOutputStream(filename);
            ObjectOutputStream out = new ObjectOutputStream(file1);

            // Method for serialization of object
            out.writeObject(object);

            out.close();
            file1.close();

            System.out.println("Object has been serialized");
        }

        catch (IOException ex) {
            System.out.println("IOException is caught");
        }

        Example object1 = null;

        // Deserialization
        try {

            // Reading object from a file
            FileInputStream file1 = new FileInputStream(filename);
            ObjectInputStream in = new ObjectInputStream(file1);

            // Method for deserialization of object
            object1 = (Example)in.readObject();

            in.close();
            file1.close();

            System.out.println("Object has been deserialized");
            System.out.println("Employee ID = " + object1.emp_id);
            System.out.println("Employee Name = " + object1.emp_name);
        }

        catch (IOException ex) {
            System.out.println("IOException is caught");
        }

        catch (ClassNotFoundException ex) {
            System.out.println("ClassNotFoundException is caught");
        }
    }
}
```

Differences between Objects and Classes



OBJECT	CLASS
Object is an instance ofa class.	Class is a blue print from which objects are created
Object is a real world entity such as chair, pen. table, laptop etc.	Class is a group of similar objects.
Object is a physical entity.	Class is a logical entity.
Object is created many times as per requirement.	Class is declared once.
Object allocates memory when it is created.	Class doesn't allocated memory when it is created.
Object is created through new keyword. Employee ob = new Employee();	Class is declared using class keyword. class Employee{}
There are different ways to create object in java:- New keyword, newinstance() method, clone() method, And deserialization.	There is only one way to define a class, i.e., by using class keyword.