09.22 lab3 quiz

# Class & Object

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

**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 (Refer [this](https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/) for details).
2. **Class name:** The name should begin with a 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, { }.

Constructors are used for initializing new objects. Fields are variables that provides the state of the class and its objects, and methods are used to implement the behavior of the class and its objects.

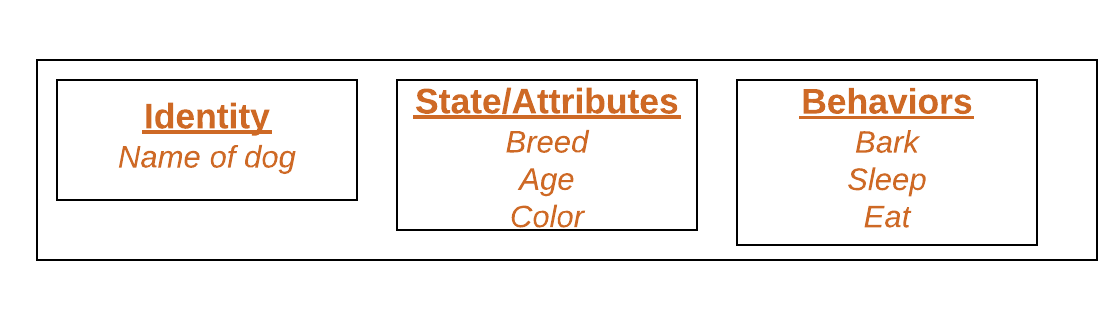
There are various types of classes that are used in real time applications such as [nested classes](https://www.geeksforgeeks.org/inner-class-java/), [anonymous classes](https://www.geeksforgeeks.org/anonymous-inner-class-java/), [lambda expressions](https://www.geeksforgeeks.org/lambda-expressions-java-8/).

**Object**

It is a basic unit of Object Oriented Programming and represents the real life entities.  A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

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

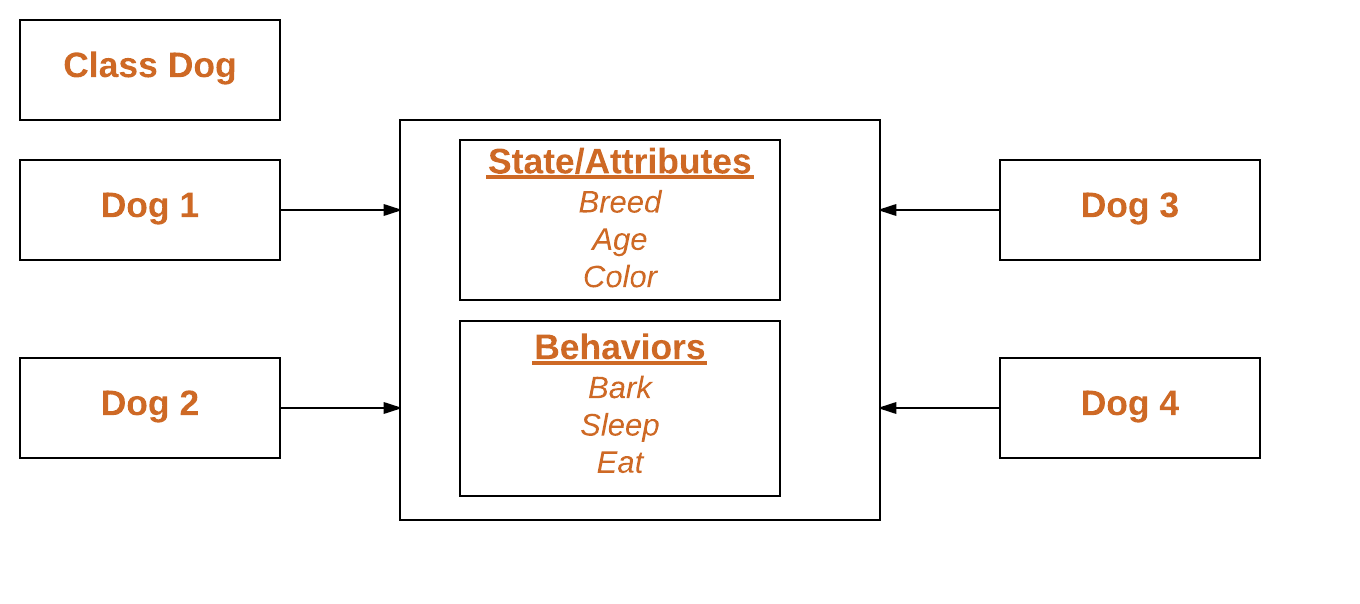
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Blank-Diagram-Page-1-5.png)

Objects correspond 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 :

[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Blank-Diagram-Page-1-3.png)

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**

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.

|  |
| --- |
| // 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());      }  } |

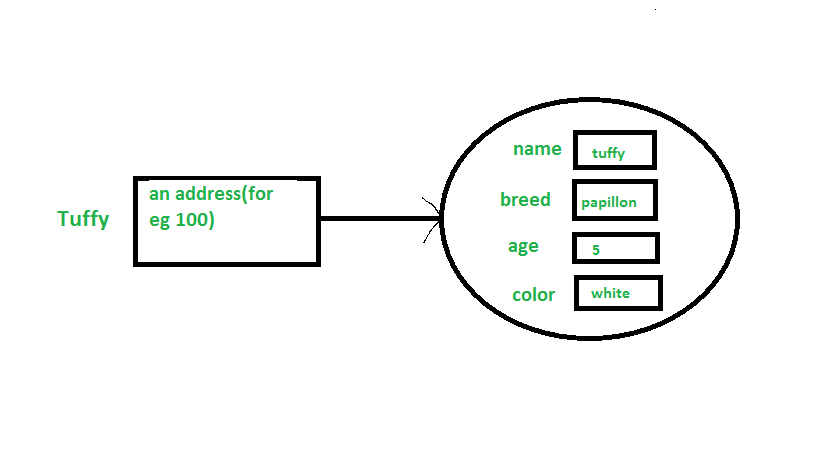
Run on IDE

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 :  
[](http://cdncontribute.geeksforgeeks.org/wp-content/uploads/Untitled5.png)

**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).

**Ways to create object of a class**

There are four ways to create objects in java.Strictly speaking there is only one way(by using new keyword),and the rest internally use new keyword.

* **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.We have to give the fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of 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. Refer [Serialization/De-Serialization in java](http://quiz.geeksforgeeks.org/serialization-in-java/)
* FileInputStream file = new FileInputStream(filename);
* ObjectInputStream in = new ObjectInputStream(file);

Object obj = in.readObject();

### 49) What is the final class?

If we make any class final, we can't inherit it into any of the subclasses.

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** Honda1();
8. honda.run();
9. }
10. }

### 58) What is the abstract class?

A class that is declared as abstract is known as an abstract class. It needs to be extended and its method implemented. It cannot be instantiated. It can have abstract methods, non-abstract methods, constructors, and static methods. It can also have the final methods which will force the subclass not to change the body of the method. Consider the following example.

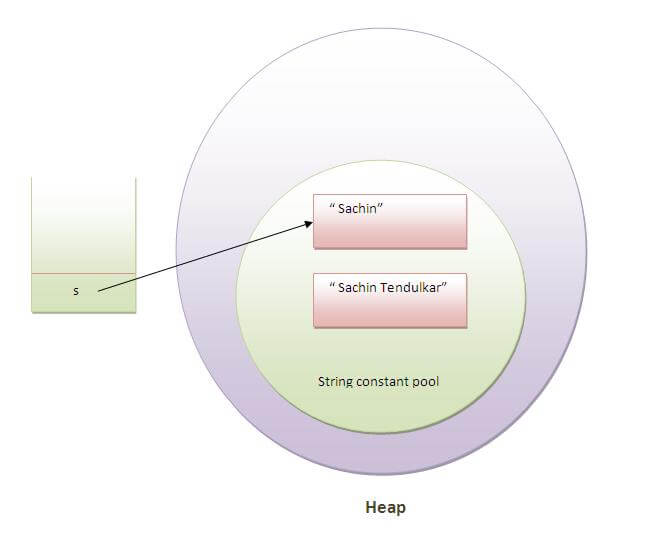
1. **abstract** **class** Bike{
2. **abstract** **void** run();
3. }
4. **class** Honda4 **extends** Bike{
5. **void** run(){System.out.println("running safely");}
6. **public** **static** **void** main(String args[]){
7. Bike obj = **new** Honda4();
8. obj.run();
9. }
10. }

### 68) When can an object reference be cast to an interface reference?

An object reference can be cast to an interface reference when the object implements the referenced interface.

85) Why are the objects immutable in java?

Because Java uses the concept of the string literal. Suppose there are five reference variables, all refer to one object "sachin". If one reference variable changes the value of the object, it will be affected by all the reference variables. That is why string objects are immutable in java.

   
More details.

# CONSTRUCTOR

15) What is the constructor?

The constructor can be defined as the special type of method that is used to initialize the state of an object. It is invoked when the class is instantiated, and the memory is allocated for the object. Every time, an object is created using the **new** keyword, the default constructor of the class is called. The name of the constructor must be similar to the class name. The constructor must not have an explicit return type.

[More details.](https://www.javatpoint.com/constructor)

16) What is the purpose of a default constructor?

The purpose of the default constructor is to assign the default value to the objects. The java compiler creates a default constructor implicitly if there is no constructor in the class.

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.

[More details.](https://www.javatpoint.com/constructor)

17) Does constructor return any value?

**Ans:** yes, The constructor implicitly returns the current instance of the class (You can't use an explicit return type with the constructor).[More details.](https://www.javatpoint.com/constructor)

18)Is constructor inherited?

No, The constructor is not inherited.

19) Can you make a constructor final?

No, the constructor can't be final.

# Null Pointer Exception

NullPointerException is a RuntimeException. In Java, a special [**null**](https://en.wikibooks.org/wiki/Java_Programming/Literals/null) value can be assigned to an object reference. NullPointerException is thrown when an application attempts to use an object reference that has the [**null**](https://en.wikibooks.org/wiki/Java_Programming/Literals/null) value. These include:

* Calling an instance method on the object referred by a null reference.
* Accessing or modifying an instance field of the object referred by a null reference.
* If the reference type is an array type, taking the length of a null reference.
* If the reference type is an array type, accessing or modifying the slots of a null reference.
* If the reference type is a subtype of Throwable, throwing a null reference.

|  |  |
| --- | --- |
| Example | **Code section 6.13: Null pointer.**  1 Object obj = **null**;  2 obj.toString(); *// This statement will throw a NullPointerException* |

The above code shows one of the pitfalls of Java and the most common source of bugs. No object is created and the compiler does not detect it. NullPointerException is one of the most common exceptions thrown in Java.

# Git Commend

<https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>

### git commend

$ git config --global alias.commend 'commit --amend --no-edit'

Ever commit and then immediately realize you’d forgotten to stage a file? Fret no more! git commend quietly tacks any staged files onto the last commit you created, re-using your existing commit message. So as long as you haven’t pushed yet, no-one will be the wiser.

$ git add Dockerfile  
$ git commit -m ‘Update Bitbucket pipeline with new Docker image’  
# (facepalm)  
$ git add bitbucket-pipelines.yml  
$ git commend

# Primitive & Reference Data Type

**Primitive types** are the basic **types** of **data**: byte , short , int , long , float , double , boolean , char .**Primitive** variables store **primitive** values. **Reference types** are any instantiable class as well as arrays: String , Scanner , Random , Die , int[] , String[] , etc.

<https://javarevisited.blogspot.com/2015/09/difference-between-primitive-and-reference-variable-java.html>

<http://pages.cs.wisc.edu/~bahls/cs302/PrimitiveVsReference.html>

## Comments

## Comments in Java Code

The bold characters in the following listing are comments.

**/\*\***

**\* The HelloWorldApp class implements an application that**

**\* simply displays "Hello World!" to the standard output.**

**\*/**

class HelloWorldApp {

public static void main(String[] args) {

System.out.println("Hello World!"); **//Display the string.**

}

}

The Java language supports three kinds of comments:

/\* text \*/

The compiler ignores everything from /\* to \*/.

/\*\* documentation \*/

This indicates a documentation comment (doc comment, for short). The compiler ignores this kind of comment, just like it ignores comments that use /\* and \*/. The JDK javadoc tool uses doc comments when preparing automatically generated documentation. For more information on javadoc, see the [Java tool documentation](http://java.sun.com/products/JDK/tools/index.html).

// text

The compiler ignores everything from // to the end of the line.

<https://www.geeksforgeeks.org/comments-in-java/>