Class And Object

- A class is a template for an object, and an object is an instance of a class.
- A class creates a new data type that can be used to create objects.
- It is a user-defined blueprint or prototype from which objects are created. For example, Student is a class while a particular student named Ravi is an object.
- A Class in Java can contain:
  - Data member (Variables)
  - Method
  - Constructor
  - Nested Class
  - Interface

Objects are characterized by three essential properties: state, identity, and behavior. The state of an object is a value from its data type. The identity of an object distinguishes one object from another. It is useful to think of an object's identity as the place where its value is stored in memory. The behavior of an object is the effect of data-type operations.

Dog tommy; // declare reference to object
tommy = new Dog();// allocate a Dog object
tommy.(it can access all the methods and variables from the Dog class)

The dot operator links the name of the object with the name of an instance variable. Although commonly referred to as the dot operator, the formal specification for Java categorizes the . as a separator. The 'new' keyword dynamically allocates (that is, allocates at run time) memory for an object & returns a reference to it. This reference is, more or less, the address in memory of the object allocated by new. This reference is then stored in the variable(i.e tommy in the given example above).

The first line declares tommy as a reference to an object of type Dog. At this point, tommy does not yet refer to anactual object. The next line allocates an object and assigns a reference to tommy. After the second line executes, you can use tommy as if it were a Dog object. But in reality, tommy simply holds, in essence, the memory address of the actual Dog object.

1. **Modifiers**: A class can be public or has default access
2. **Class keyword**: class keyword is used to create a class.
3. **Class name**: The name should begin with an initial letter (capitalized by convention).
4. **Superclass**: The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces:** 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.
6. **Body**: The class body is surrounded by braces, { }.

Variables

Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another. An instance variable can be declared public or private or default (no modifier). When we do not want our variable's value to be changed out-side our class we should declare them private. public variables can be accessed and changed from outside of the class.

Syntax

```
private int breed ;
{access modifier} {type} {name of the variable}
```

Methods

A method is a program module that contains a series of statements that carry out a task. To execute a method, you invoke or call it from the object created by use the dot operator or from another method. Any class can contain an unlimited number of methods, and each method can be called an unlimited number of times. The syntax to declare method is given below.

Syntax

```
public int addition(int a, int b) {
    return a + b;
}
```

Access Specifiers

Each object has members (members can be variable and methods) which can be declared to have specific access.

public: Members (variables, methods, and constructors) declared public (least restrictive) within a public class are visible to any class in the Java program, whether these classes are in the same package or in another package.

protected: The protected fields or methods, cannot be used for classes and Interfaces. Fields, methods and constructors declared protected in a super-class can be accessed only by subclasses in other packages. Classes in the same package can also access protected fields, methods and constructors as well, even if they are not a subclass of the protected member's class.

Default (no value):The default access level is declared by not writing any access modifier at all. Any class, field, method or constructor that has no declared access modifier is accessible only by classes in the same package.

private: The private (most restrictive) modifiers can be used for members but cannot be used for classes and Interfaces. Fields, methods or constructors declared private are strictly controlled, which means they cannot be accessed by anywhere outside the enclosing class.

Java has modifiers other than access modifiers listed below:

static: static can be used for members of a class. The static members of the class can be accessed without creating an object of a class.

final: This modifier is applicable to class, method, and variables. This modifier tells the compiler not to change the value of a variable once assigned. If applied to class, it cannot be sub-classed. If applied to a method, the method cannot be overridden in sub-class.

Modifier	class	constructor	method	Data/variables
public	Yes	Yes	Yes	Yes
protected		Yes	Yes	Yes
default	Yes	Yes	Yes	Yes
private		Yes	Yes	Yes
static			Yes	
final	Yes		Yes	