Classes and Objects

- Class is the collection of object.
- Class is not a real world entity. It is just a template or blueprint or prototype.
- Class does not occupy memory

This is a class declaration. Here, key word class is used to declare a class followed by class name MyClass.

Syntax:

class ***MyClass*** {

    // field, constructor, and

    // method declarations

}

The class body (the area between the braces) contains all the code that provides for the life cycle of the objects created from the class: constructors for initializing new objects, declarations for the fields that provide the state of the class and its objects, and methods to implement the behavior of the class and its objects.

You can provide more information about the class, such as the name of its superclass, whether it implements any interfaces, and so on, at the start of the class declaration. For example,

class ***MyClass*** extends ***MySuperClass*** implements ***YourInterface*** {

    // field, constructor, and

    // method declarations

}

means that **MyClass** is a subclass of **MySuperClass** and that it implements the **YourInterface** interface.

In general, class declarations can include these components, in order:

- Modifiers such as public, private, and a number of others that you will encounter later.
- The class name, with the initial letter capitalized by convention.
- The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
- 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.
- The class body, surrounded by braces, **{}**.

Creating Objects

A set of codes which perform a particular task is called object. It's can do the code reusability and code optimization.

- Object is an instance of class
- Object is real world entity.
- Object occupies memory.

Object consist of:

1. Identity: Name
2. State/Attribute: color, age
3. Behavior: eat, run, sleep

How to create an object:

- New keyword

Creating object using new Keyword

In Java, an object is created from a class. We have already created the class named **Animal**, so now we can use this to create objects.

To create an object of **Animal**, specify the class name, followed by the object name, and use the keyword new:

Example

Create an object called "**buzo**" and print the value of **age**:

```
public class Animal {
  int age = 5;

  public static void main(String[] args) {
    Animal buzo = new Animal();
    System.out.println(buzo.age);
  }
}
```

Point originOne = new Point();

Rectangle rectOne = new Rectangle();

The first line creates an object of the Point class, and the second creates an object of the Rectangle class.

Declaration: The code set in bold are all variable declarations that associate a variable name with an object type.

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

Initialization: The new operator is followed by a call to a constructor, which initializes the new object.

Multiple Objects

You can create multiple objects of one class:

Example

Create two objects of MyClass:

```
public class MyClass {
  int x = 5;

  public static void main(String[] args) {
    MyClass myObj1 = new MyClass();  // Object 1
    MyClass myObj2 = new MyClass();  // Object 2
    System.out.println(myObj1.x);
    System.out.println(myObj2.x);
  }
}
```

Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory/folder:

- MyClass.java
- OtherClass.java

```
public class MyClass {
  int x = 5;
```

```
}

class OtherClass {
  public static void main(String[] args) {
    MyClass myObj = new MyClass();
    System.out.println(myObj.x);
  }
}
```

Java Methods

- A **method** is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as **functions**.

Why use methods?

- To reuse code: define the code once, and use it many times.

Create a Method

A method must be declared within a class. It is defined with the name of the method, followed by parentheses **()**. Java provides some pre-defined methods, such as ***System.out.println(),*** but you can also create your own methods to perform certain actions:

Example

Create a method inside MyClass:

```
public class MyClass {
  static void myMethod() {
    // code to be executed
  }
}
```

Example Explained

- **`myMethod()`** is the name of the method
- **`static`** means that the method belongs to the MyClass class and not an object of the MyClass class. You will learn more about objects and how to access methods through objects later in this tutorial.
- **`void`** means that this method does not have a return value. You will learn more about return values later in this chapter

Call a Method

To call a method in Java, write the method's name followed by two parentheses **()** and a semicolon**;**

In the following example, **myMethod()** is used to print a text (the action), when it is called:

Example

Inside main, call the **myMethod()** method:

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}

// Outputs "I just got executed!"
```

A method can also be called multiple times:

Example

```
public class MyClass {
  static void myMethod() {
    System.out.println("I just got executed!");
  }

  public static void main(String[] args) {
    myMethod();
    myMethod();
    myMethod();
  }
}

// I just got executed!
// I just got executed!
// I just got executed!
```

Way to put the value in Variable

1. By Reference variable

2. By Method
3. By using Constructor

1. By reference Variable

```
Class Animal{
        String color;
        Int age;
Public static void main (String[] args){
        Animal buzo = new Animal();
        buzo.color = “Dog”;
        buzo.age =  10;
        System.out.println(buzo.color +” ”+ buzo.age);
}
}
```

2. By Method

```
Class Animal{
        String color;
        Int age;
void setObject(String s, int a){
        color = c;
        age = a;
}
void getObject {
        System.out.println(color +” ”+ age);
}
```

```
Public static void main (String[] args){
        Animal a = new Animal();
        a.setObject("Dog",10);
        a.getObject();
}
}
```

3. By using Constructor

```
Class Employee{
        String name;
        Int emp_id;
void setDetails(String n, int id){
        this.name = n;
        this.emp_id = id;
}
void getDetails {
        System.out.println(name +" "+emp_id);
}
}
Class Details{
    Public static void main (String[] args){
            Employee emp = new Employeel("Ram Hari",101);
            Employee emp2 = new Employeel("Hari Hari",102);
            emp.getDetails();
            emp2.getDetails();
    }
}
```

Static vs. Non-Static Method

You will often see Java programs that have either **static** or **public** attributes and methods.

In the example above, we created a static method, which means that it can be accessed without creating an object of the class, unlike public, which can only be accessed by objects:

Example

An example to demonstrate the differences between static and public **methods**:

```
public class MyClass {
  // Static method
  static void myStaticMethod() {
    System.out.println("Static methods can be called without
creating objects");
  }

  // Public method
  public void myPublicMethod() {
    System.out.println("Public methods must be called by
creating objects");
  }

  // Main method
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

// Create an object of MyClass
    MyClass myObj = new MyClass();

// Call the public method on the object
    myObj.myPublicMethod();
  }
}
```

Access Methods with an Object

Create a Car object named `myCar`. Call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program:

```
// Create a Car class
public class Car {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }
```

```
  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }

  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Car myCar = new Car();      // Create a myCar object
    myCar.fullThrottle();       // Call the fullThrottle() method
    myCar.speed(200);           // Call the speed() method
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```

Example explained

1) We created a custom `Car` class with the `class` keyword.
2) We created the `fullThrottle()` and `speed()` methods in the `Car` class.
3) The `fullThrottle()` method and the `speed()` method will print out some text, when they are called.
4) The `speed()` method accepts an `int` parameter called `maxSpeed` - we will use this in **8)**.
5) In order to use the `Car` class and its methods, we need to create an **object** of the `Car` Class.
6) Then, go to the `main()` method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).
7) By using the `new` keyword we created a `Car` object with the name `myCar`.
8) Then, we call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program using the name of the object (`myCar`), followed by a dot (`.`), followed by the name of the method (`fullThrottle();` and `speed(200);`). Notice that we add an `int` parameter of **200** inside the `speed()` method.

Using Multiple Classes

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory:

- Car.java
- OtherClass.java

```
public class Car {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
```

```
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}

class OtherClass {
  public static void main(String[] args) {
    Car myCar = new Car();      // Create a myCar object
    myCar.fullThrottle();       // Call the fullThrottle() method
    myCar.speed(200);           // Call the speed() method
  }
}
```

When both files have been compiled:

C:\Users*Your Name*>javac Car.java
C:\Users*Your Name*>javac OtherClass.java

Run the OtherClass.java file:

C:\Users*Your Name*>java OtherClass

And the output will be:

```
The car is going as fast as it can!
Max speed is: 200
```

Access Modifiers

Java's access modifiers are **public**, **private**, and **protected**. Java also defines a default access level.

Protected protected applies only when inheritance is involved.

public modifier—the field declared with public modifier is accessible from all classes.

private modifier—the field declared with public modifier is accessible only within its own class.

In the spirit of encapsulation, it is common to make fields private. This means that they can only be directly accessed from the **Bicycle** class.

```
public class Bicycle {
        //these fields can be accessed only from bicycle class
        private int cadence;
        private int gear;
        private int speed;
```

```
public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
}
/*But in case we need access to these values.
*This can be done indirectly by adding public methods as below
*/
public int getCadence() {
        return cadence;
}
public void setCadence(int newValue) {
        cadence = newValue;
}
public int getGear() {
        return gear;
}
public void setGear(int newValue) {
        gear = newValue;
}
public int getSpeed() {
        return speed;
}
public void applyBrake(int decrement) {
        speed -= decrement;
}
public void speedUp(int increment) {
        speed += increment;
}
}
```