Java Class

A class is a blueprint for the object. Before we create an object, we first need to define the class.

We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object.

Since many houses can be made from the same description, we can create many objects from a class.

Create a class in Java

We can create a class in Java using the class keyword. For example,

```
class ClassName {
  // fields
  // methods
}
```

Here, fields (variables) and methods represent the **state** and **behavior** of the object respectively.

- fields are used to store data
- methods are used to perform some operations

For our bicycle object, we can create the class as

```
class Bicycle {

  // state or field
  private int gear = 5;

  // behavior or method
  public void braking() {
    System.out.println("Working of Braking");
  }
}
```

In the above example, we have created a class named Bicycle. It contains a field named gear and a method named braking().

Here, Bicycle is a prototype. Now, we can create any number of bicycles using the prototype. And, all the bicycles will share the fields and methods of the prototype.

Note: We have used keywords `private` and `public`. These are known as access modifiers. To learn more, visit Java access modifiers.

Java Objects

An object is called an instance of a class. For example, suppose Bicycle is a class then MountainBicycle, SportsBicycle, TouringBicycle, etc can be considered as objects of the class.

Creating an Object in Java

Here is how we can create an object of a class.

```
className object = new className();

// for Bicycle class
Bicycle sportsBicycle = new Bicycle();

Bicycle touringBicycle = new Bicycle();
```

We have used the `new` keyword along with the constructor of the class to create an object. Constructors are similar to methods and have the same name as the class. For example, `Bicycle()` is the constructor of the Bicycle class. To learn more, visit Java Constructors.

Here, sportsBicycle and touringBicycle are the names of objects. We can use them to access fields and methods of the class.

As you can see, we have created two objects of the class. We can create multiple objects of a single class in Java.

Note: Fields and methods of a class are also called members of the class.

Access Members of a Class

We can use the name of objects along with the `.` operator to access members of a class. For example,

```
class Bicycle {

  // field of class
  int gear = 5;

  // method of class
  void braking() {
    ...
  }
}

// create object
Bicycle sportsBicycle = new Bicycle();

// access field and method
```

```
sportsBicycle.gear;
sportsBicycle.braking();
```

In the above example, we have created a class named Bicycle. It includes a field named gear and a method named `braking()`. Notice the statement,

```
Bicycle sportsBicycle = new Bicycle();
```

Here, we have created an object of Bicycle named sportsBicycle. We then use the object to access the field and method of the class.

- **sportsBicycle.gear** - access the field gear
- **sportsBicycle.braking()** - access the method `braking()`

We have mentioned the word **method** quite a few times. You will learn about Java methods in detail in the next chapter.

Now that we understand what is class and object. Let's see a fully working example.

Example: Java Class and Objects

```
class Lamp {

  // stores the value for light
  // true if light is on
  // false if light is off
  boolean isOn;

  // method to turn on the light
  void turnOn() {
    isOn = true;
    System.out.println("Light on? " + isOn);

  }

  // method to turnoff the light
  void turnOff() {
    isOn = false;
    System.out.println("Light on? " + isOn);
  }
}

class Main {
  public static void main(String[] args) {

    // create objects led and halogen
```

```
    Lamp led = new Lamp();
    Lamp halogen = new Lamp();

    // turn on the light by
    // calling method turnOn()
    led.turnOn();

    // turn off the light by
    // calling method turnOff()
    halogen.turnOff();
  }
}
```

Output:

Light on? true Light on? false

In the above program, we have created a class named Lamp. It contains a variable: isOn and two methods: `turnOn()` and `turnOff()`.

Inside the Main class, we have created two objects: led and halogen of the Lamp class. We then used the objects to call the methods of the class.

- **led.turnOn()** - It sets the isOn variable to `true` and prints the output.
- **halogen.turnOff()** - It sets the isOn variable to `false` and prints the output.

The variable isOn defined inside the class is also called an instance variable. It is because when we create an object of the class, it is called an instance of the class. And, each instance will have its own copy of the variable.

That is, led and halogen objects will have their own copy of the isOn variable.

Example: Create objects inside the same class

Note that in the previous example, we have created objects inside another class and accessed the members from that class.

However, we can also create objects inside the same class.

```
class Lamp {

  // stores the value for light
  // true if light is on
  // false if light is off
  boolean isOn;

  // method to turn on the light
  void turnOn() {
    isOn = true;
```

```
    System.out.println("Light on? " + isOn);

  }

  public static void main(String[] args) {

    // create an object of Lamp
    Lamp led = new Lamp();

    // access method using object
    led.turnOn();
  }
}
```

Output

Light on? true

Here, we are creating the object inside the `main()` method of the same class.