2.1 Java – Objects & Classes

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts:

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Parsing

In this chapter, we will look into the concepts - Classes and Objects.

- **Object -** Objects have states and behaviours.
  Example: A dog has states - colour, name, breed as well as behaviours – wagging the tail, barking, eating. An object is an instance of a class.

- **Class -** A class can be defined as a template/blueprint that describes the behaviour/state that the object of it's type support.

2.1.1 Classes:

- The introduction to object-oriented concepts in the lesson titled ***Object-oriented Programming Concepts*** used a bicycle class as an example, with racing bikes, mountain bikes, and tandem bikes as subclasses.
- Here is sample code for a possible implementation of a Bicycle class, to give you an overview of a class declaration.
- Subsequent sections of this lesson will back up and explain class declarations step by step.

- For the moment, don't concern yourself with the details.

Example:

```
public class Bicycle {
    // the Bicycle class has
    // three fields
    public int cadence;
    public int gear;
    public int speed;
```

2.1.2 Declaring Classes

You've seen classes defined in the following way:

Syntax:

```
class MyClass {
    // field, constructor, and
    // method declarations
}
```

This is a *class declaration*.

- 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 behaviour of the class and its objects.
- The preceding class declaration is a minimal one.
- It contains only those components of a class declaration that are required.
- 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**.

- You can also add modifiers like *public* or *private* at the very beginning- so you can see that the opening line of a class declaration can become quite complicated.
- The modifiers *public* and *private*, which determine what other classes can access MyClass, are discussed later in this lesson.
- The lesson on interfaces and inheritance will explain how and why you would use the *Extends* and *Implements* keywords in a class declaration. For the moment you do not need to worry about these extra complications.

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

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