CLASS

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. ... Modifiers: A class can be public or has default access (Refer this for details). class keyword: class keyword is used to create a class.

Java Classes/Objects

Java is an object-oriented programming language.
Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.
A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword `class`:

Main.java

Create a class named "`Main`" with a variable x:

```
public class Main {
  int x = 5;
}
```

Remember from the that a class should always start with an uppercase first letter, and that the name of the java file should match the class name.

Create an Object

In Java, an object is created from a class. We have already created the class named `Main`, so now we can use this to create objects.
To create an object of `Main`, specify the class name, followed by the object name, and use the keyword `new`:

Example

Create an object called "`myObj`" and print the value of x:

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

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

Multiple Objects

We can create multiple objects of one class:

Example

Create two objects of `Main`:

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

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

Multiple Objects

We can create multiple objects of one class:

Example

Create two objects of `Main`:

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

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

Second.java

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

When both files have been compiled:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Run the Second.java file:

```
C:\Users\Your Name>java Second
```

And the output will be:

```
5
```

Java Class Attributes

In the previous chapter, we used the term "variable" for `x` in the example (as shown below). It is actually an **attribute** of the class. Or you could say that class attributes are variables within a class:

Example

Create a class called "`Main`" with two attributes: `x` and `y`:

```
public class Main {
  int x = 5;
  int y = 3;
}
```

Another term for class attributes is **fields**.

Accessing Attributes

We can access attributes by creating an object of the class, and by using the dot syntax (`.`):

The following example will create an object of the `Main` class, with the name `myObj`. We use the `x` attribute on the object to print its value:

Example

Create an object called "`myObj`" and print the value of `x`:

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

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

Modify Attributes

We can also modify attribute values:

Example

Set the value of `x` to 40:

```
public class Main {
  int x;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 40;
    System.out.println(myObj.x);
```

```
  }
}
```

Or override existing values:

Example

Change the value of `x` to 25:

```
public class Main {
  int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // x is now 25
    System.out.println(myObj.x);
  }
}
```

If you don't want the ability to override existing values, declare the attribute as `final`:

Example

```
public class Main {
  final int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
    myObj.x = 25; // will generate an error: cannot assign a value to a final variable
    System.out.println(myObj.x);
  }
}
```

The `final` keyword is useful when you want a variable to always store the same value, like PI (3.14159...).
The `final` keyword is called a "modifier". You will learn more about these in the Java Modifiers Chapter.

Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

Example

Change the value of `x` to 25 in `myObj2`, and leave `x` in `myObj1` unchanged:

```
public class Main {
```

```
  int x = 5;

  public static void main(String[] args) {
    Main myObj1 = new Main();  // Object 1
    Main myObj2 = new Main();  // Object 2
    myObj2.x = 25;
    System.out.println(myObj1.x);  // Outputs 5
    System.out.println(myObj2.x);  // Outputs 25
  }
}
```

Multiple Attributes

You can specify as many attributes as you want:

Example

```
public class Main {
  String fname = "John";
  String lname = "Doe";
  int age = 24;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println("Name: " + myObj.fname + " " + myObj.lname);
    System.out.println("Age: " + myObj.age);
  }
}
```