HNDIT3012 - OBJECT ORIENTED PROGRAMMING

LECTURE 04 - CLASSES AND OBJECTS

CLASS VS OBJECT

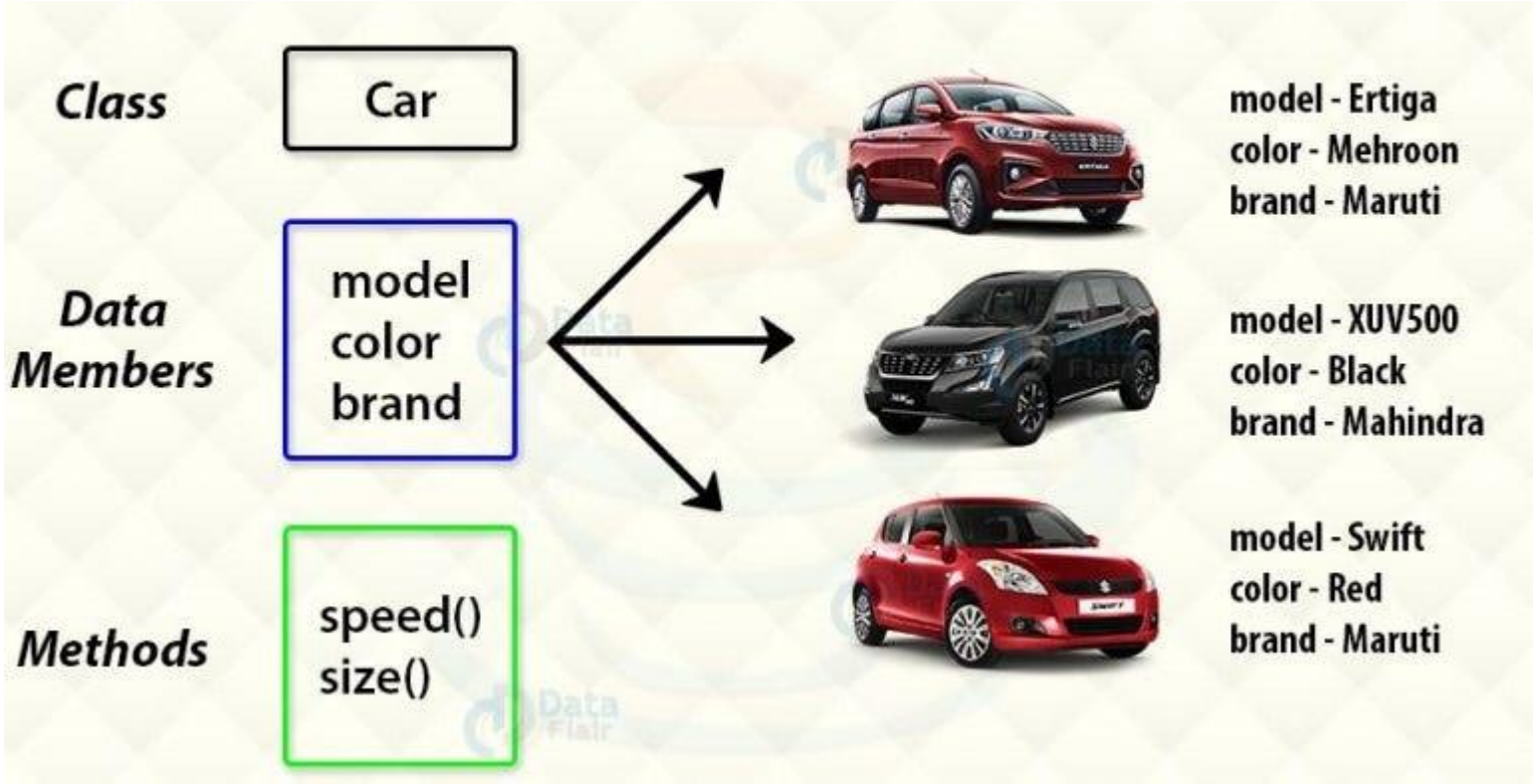
- **Object** – An object is an instance of a class.
  - Objects have states and behaviors.
  - Example: A dog has states - color, name, breed
  - behaviors – wagging the tail, barking, eating.

- **Class** – A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support.

CLASSES IN JAVA

- A class is a blueprint from which individual objects are created.

Class – Fruit
Objects – Apple, Banana, Mango



CREATING AN OBJECT

- a class provides the blueprints for objects. So basically, an object is created from a class.
- In Java, the **new** keyword is used to create new objects.
- There are three steps when creating an object from a class –
  - **Declaration** – A variable declaration with a variable name with an object type.
  - **Instantiation** – The 'new' keyword is used to create the object.
  - **Initialization** – The 'new' keyword is followed by a call to a **constructor.** This call initializes the new object.

CREATE AN OBJECT

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

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

MULTIPLE OBJECTS

```
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);
  }
}
```

JAVA CLASS ATTRIBUTES

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

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

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

MODIFY ATTRIBUTES

```
public class Main {
  int x;

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

FINAL KEY WORD

- If you don't want the ability to **override existing values**, declare the attribute as **final:**
- The **final** keyword is useful when you want a **variable to always store the same value**, like PI (3.14159...).

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

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

What happens???

QUESTION

- Create a class called student with attributes of id and name.
- Display the id and name of a student using an object.

```
class Student {
        int id;
        String name;

    public static void main(String args[])
    {
        // creating an object of Student
        Student s1 = new Student();
        System.out.println(s1.id);
        System.out.println(s1.name);
    }
}
```

QUESTION

- Create a class called person with the attributes fname, lname and age. Display the details of the person in the main method.

ANSWER

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

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

STATIC VS. PUBLIC

- You will often see Java programs that have either static or public attributes and methods.
- When created a static method, it can be accessed without creating an object of the class,
- public, which can only be accessed by objects:

```
public class Main {
  // 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

    Main myObj = new Main(); // Create an object of MyClass
    myObj.myPublicMethod(); // Call the public method
  }
}
```

CONSTRUCTORS

- Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a **default constructor** for that class.
- The main rule of **constructors** is that they should have the **same name as the class**.
- A class can have **more than one constructor.**
- However, constructors have **no explicit return type**.
- Java automatically provides a **default constructor** that initializes all member **variables to zero.**
- Once you define your own constructor, the default constructor is no longer used.

SYNTAX OF A CONSTRUCTOR

- Java allows two types of constructors
  - Default Constructor
  - Parameterized Constructor

•Default (No argument) Constructors

- This constructor does not accept any arguments. It is used to initialize the object with default values.

```
public class MyClass {
    public MyClass() {
        // constructor body
    }
}
```

PARAMETERIZED CONSTRUCTORS

- This constructor accepts one or more arguments.

- It is used to initialize the object with user-defined values.

```
public class MyClass {
    private int value;

    public MyClass(int value) {
        this.value = value;
    }
}
```

NO ARGUMENT CONSTRUCTORS

- Java does not accept any parameters in the default constructor, using these constructors the instance variables of a method will be **initialized with fixed values for all objects**.

```
Public class MyClass {
Int num;
MyClass() { // default constructor
num = 100;
}
}
```

You would call constructor to initialize objects as follows

```
public class MyClass {
public static void main(String args[]) {
MyClass t1 = new MyClass();
MyClass t2 = new MyClass();
System.out.println(t1.num + " " + t2.num);
}
}
```

PARAMETERIZED CONSTRUCTORS

- Parameters are added to a constructor in the same way that they are added to a method, just declare them inside the parentheses after the constructor's name.

```
// A simple constructor.
class MyClass {
int x;

// parameterized constructor
MyClass(int i ) {
x = i;
}
 }
```

```
public class MyClass {
public static void main(String args[]) {
MyClass t1 = new MyClass( 10 );
MyClass t2 = new MyClass( 20 );
System.out.println(t1.x + " " + t2.x); }
 }
```

EXAMPLE

```
public class Main {
  int modelYear;
  String modelName;

  public Main(int year, String name) { // constructor
    modelYear = year;
    modelName = name;
  }

  public static void main(String[] args) {
    Main myCar = new Main(1969, "Mustang"); //object creation
    System.out.println(myCar.modelYear + " " + myCar.modelName);
  }
}
```

JAVA THIS KEYWORD

- The this keyword refers to the current object in a method or constructor.
  - this can also be used to:
    - Invoke current class constructor
    - Invoke current class method
    - Return the current class object
    - Pass an argument in the method call
    - Pass an argument in the constructor call

QUESTION (PARAMETERIZED CONSTRUCTOR)

- Create a class called student with attributes of id and name. create a constructor and assign some value. Include a method to display id and name of two students using an objects.

```
class Student{
    int id;
    String name;

    Student(int i, String n){
    id = i;
    name = n;
    }
    void display(){
     System.out.println(id+" "+name);
     }

    public static void main(String args[]){
    Student s1 = new Student(001,"Kamal");
    Student s2 = new Student(002,"Saman");
    s1.display();
    s2.display();
   }
}
```

QUESTION

- Write a Java program to create a class called "Employee" with name, job title, and salary attributes.
  - Create a class as above.
  - Write the constructor.
  - Write a method to display employee details.
  - Create an employee object.
  - Display the details of that employee.

ANSWER

```
public class Employee {
private String name;
private String jobTitle;
private double salary;

public Employee(String name, String jobTitle, double salary) {
this.name = name;
this.jobTitle = jobTitle;
this.salary = salary;
}
public void printEmployeeDetails() {
System.out.println("Name: " + name);
System.out.println("Job Title: " + jobTitle);
System.out.println("Salary: " + salary);
}
```

ANSWER

```
public class Main {
public static void main(String[] args) {
Employee employee1 = new Employee(“Saman Perera", "HR Manager", 40000);
Employee employee2 = new Employee(“Kamal Siriwardana", "Software Engineer", 60000);
System.out.println("Employee Details:");
employee1.printEmployeeDetails();
employee2.printEmployeeDetails();
}
}
```