

Lecture 2

PROGRAMMING (2)

BY: DR. MOHAMMED ABDELFATTAH

Lecture Objectives

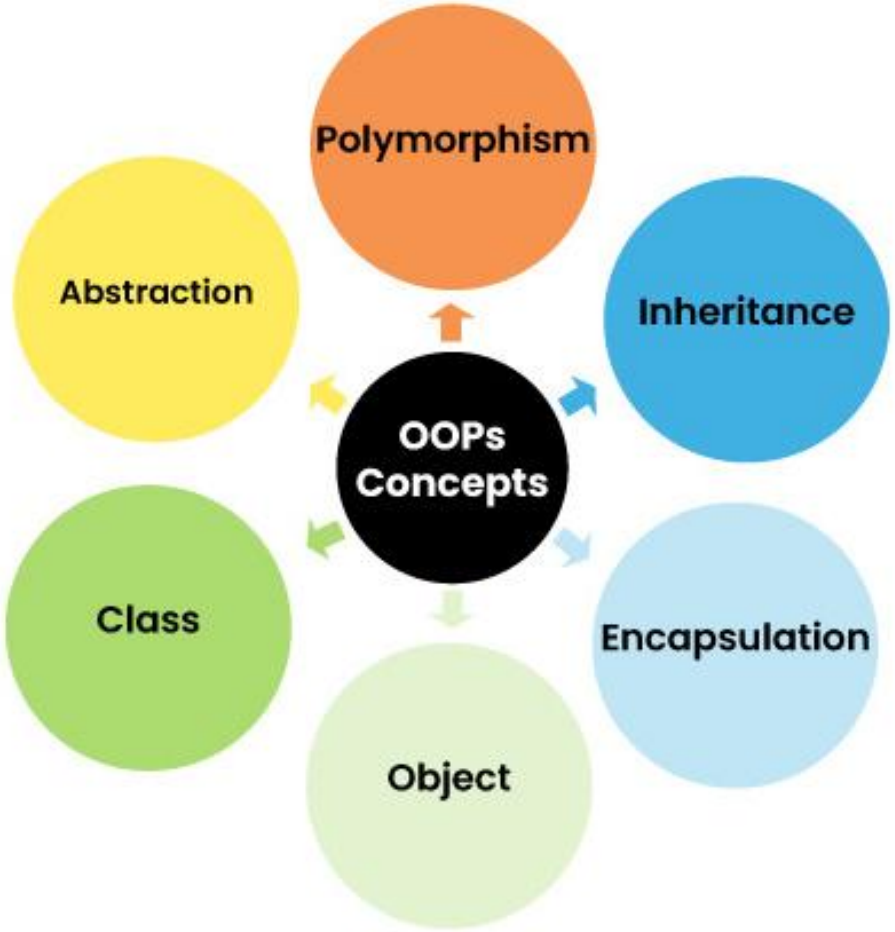
- **Overview for Main OOP Concepts:**
  - ✓ **Object.**
  - ✓ **Class.**
  - ✓ **Inheritance.**
  - ✓ **Polymorphism.**
  - ✓ **Abstraction.**
  - ✓ **Encapsulation.**
  - ✓ **Access Modifier.**
- **Advantages of OOPs.**
- **Classes:**
  - ✓ **Classes/Objects.**
  - ✓ **Multiple Objects.**
  - ✓ **Multiple Classes.**
  - ✓ **Class Attributes.**
  - ✓ **Accessing Attributes.**
  - ✓ **Modify Attributes.**
  - ✓ **Class Methods.**

Object-Oriented Programming (OOP)

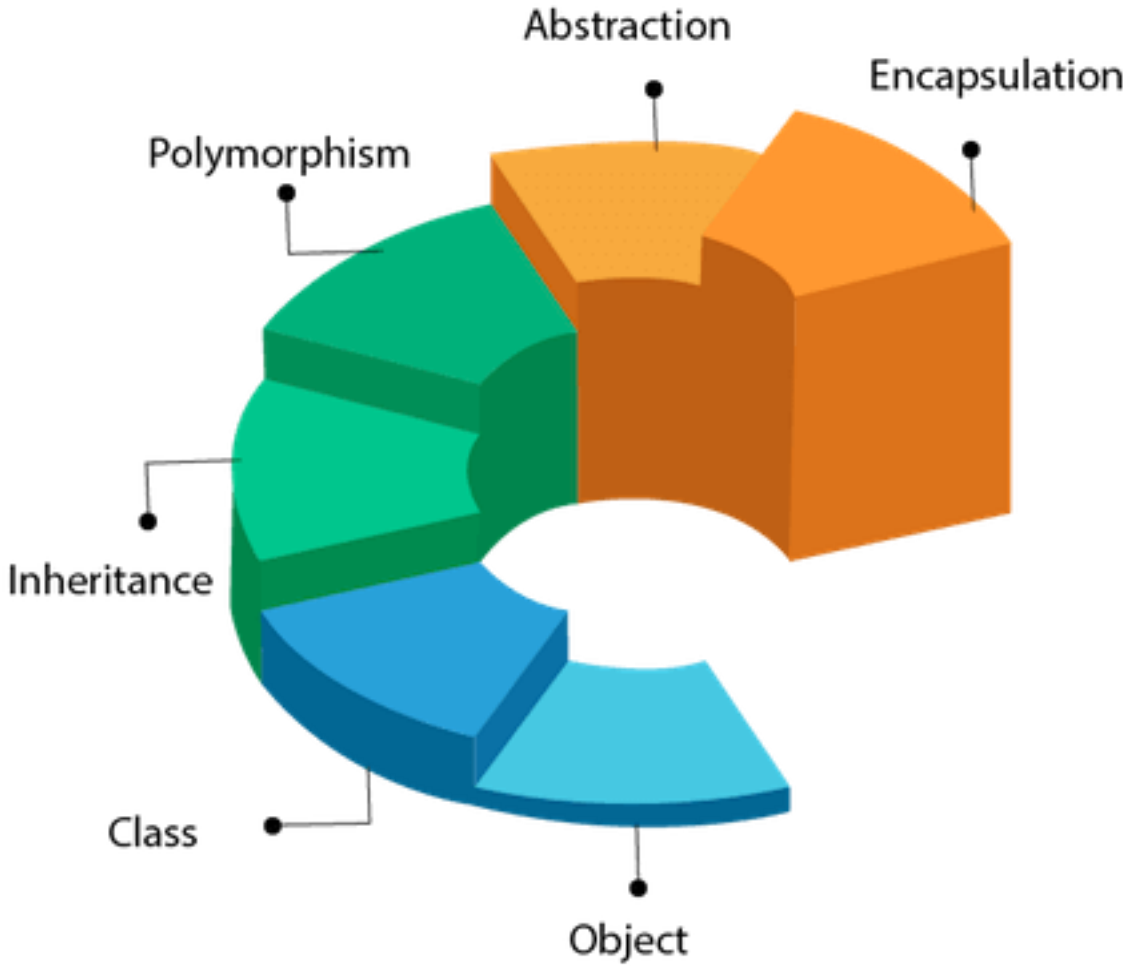
Object-oriented Programming (OOPs) means we organize our software as a combination of different types of objects that incorporate both data and behavior.

OOPs is a methodology that simplifies software development and maintenance by providing some rules.

- **Object**
- **Class**
- **Inheritance**
- **Polymorphism**
- **Abstraction**
- **Encapsulation**



OOP Concepts



Object

- **Object** means a real-world entity such as a pen, chair, table, computer, watch, etc.

- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.

- It simplifies software development and maintenance by providing some concepts.

Object

- Any entity that has **state** and **behavior** is known as an **object**. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- Object can be defined as an instance of a class.
- Object contains an address and takes up some space in memory.
- Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

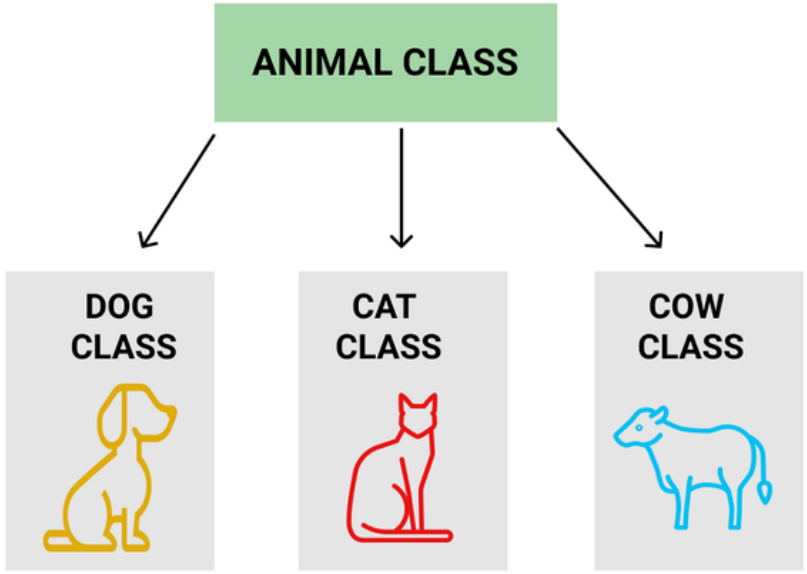


- **Example**: A dog is an object because it has:
  - **States** like color, name, breed, etc.
  - **Behaviors** like wagging the tail, barking, eating, etc.

Class

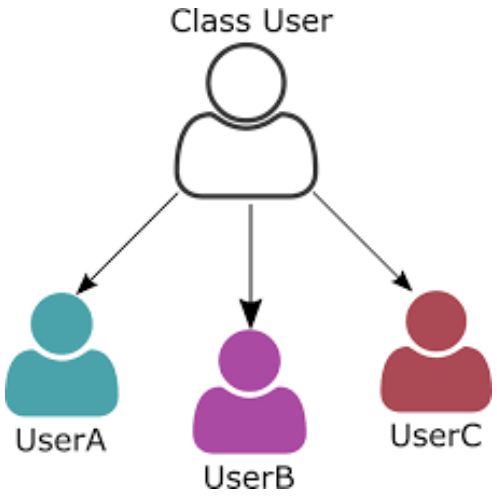
- Collection of objects is called class. It is a logical entity.

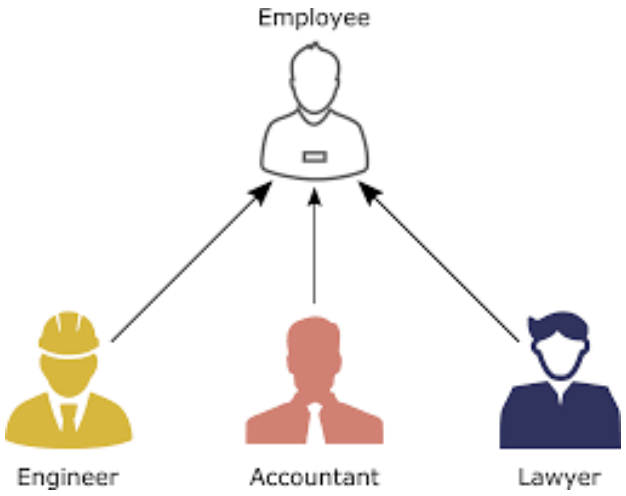
- A **class** can also be defined as a **blueprint** from which you can create an individual object.

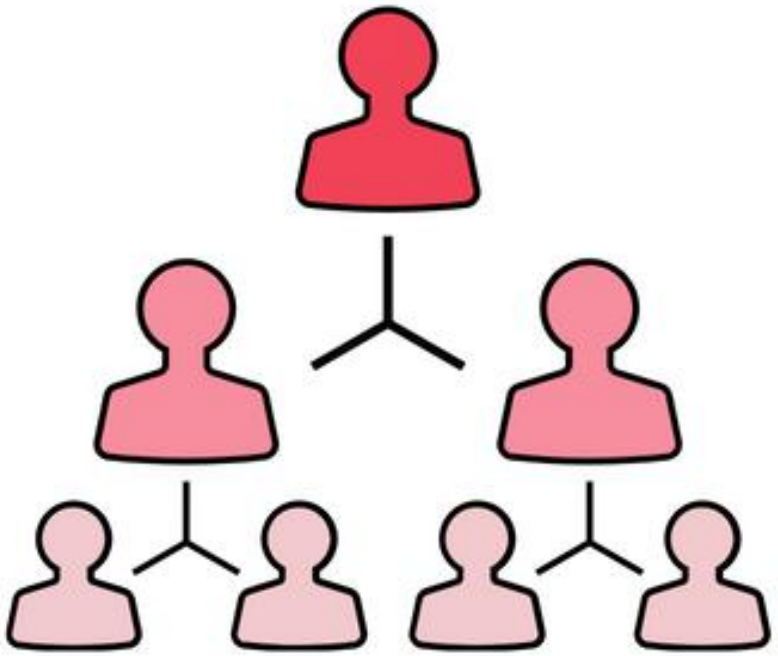


Inheritance

- When one **object acquires** all the properties and behaviors of a **parent object**, it is known as **inheritance**.
- It provides code reusability.
- It is used to achieve runtime polymorphism.







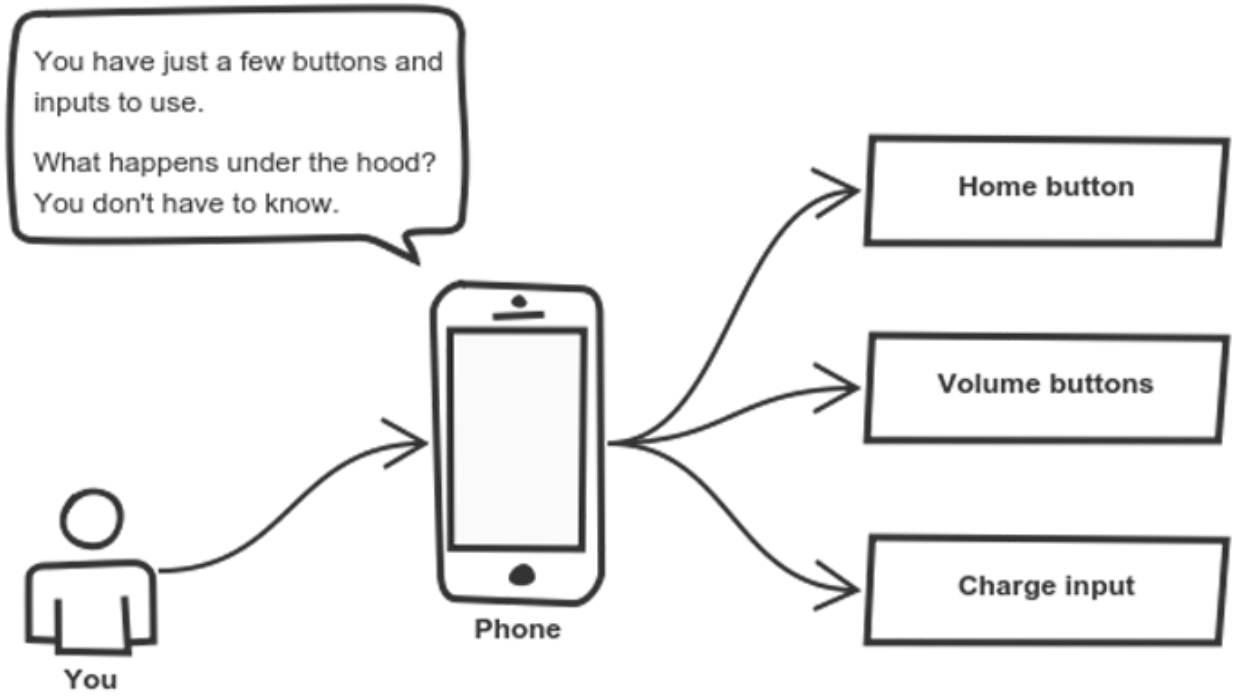
Polymorphism

- If one task is performed in different ways, it is known as polymorphism. For example, to:
  - Convince the customer differently,
  - Draw something, for example, shape, triangle, rectangle, etc.
- We use **method overloading** and **method overriding** to achieve polymorphism.
- Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

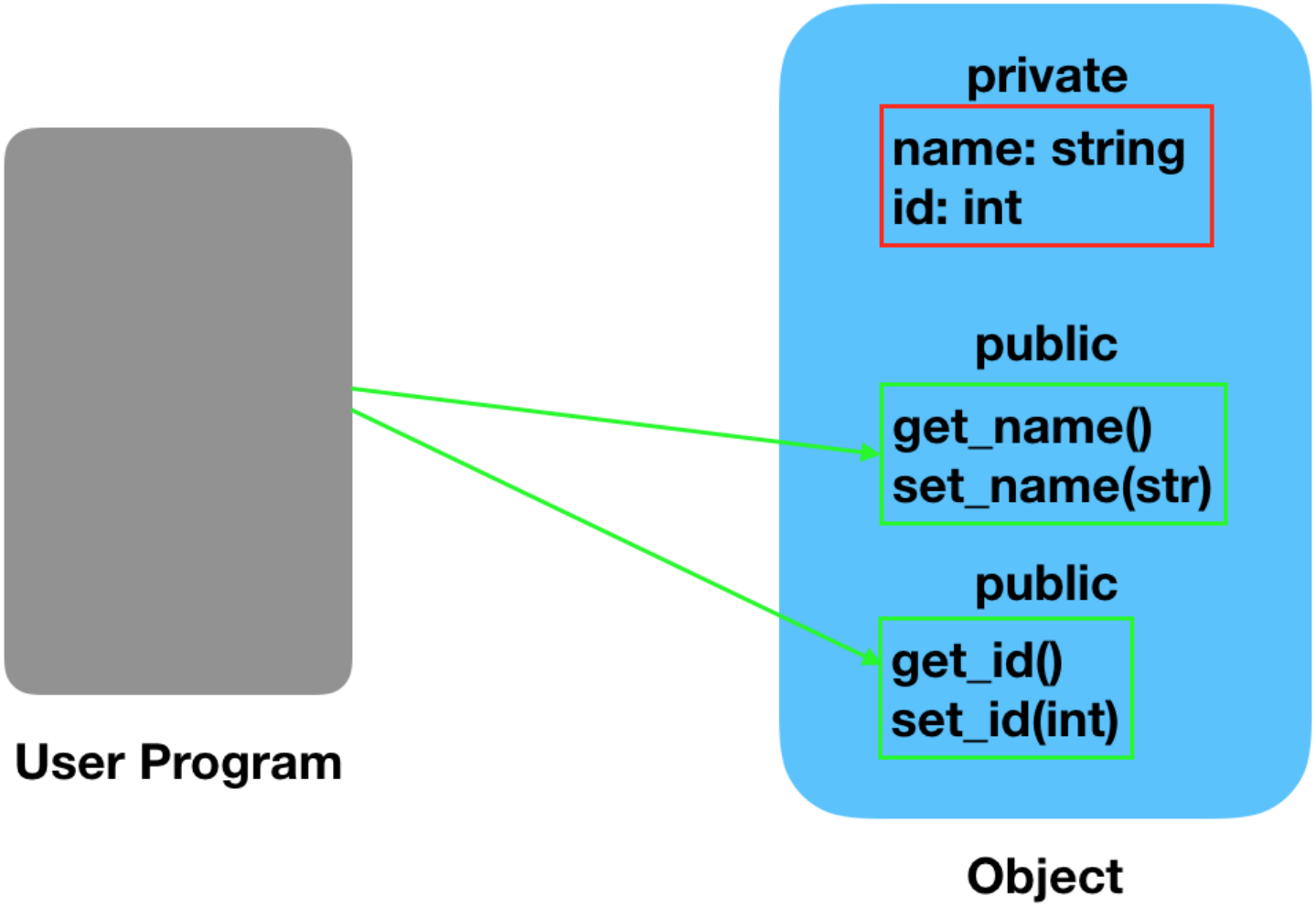


Abstraction

- Hiding internal details and showing functionality is known as abstraction.
- For example, phone call, we don't know the internal processing.
- We use **abstract class** and **interface** to achieve abstraction.

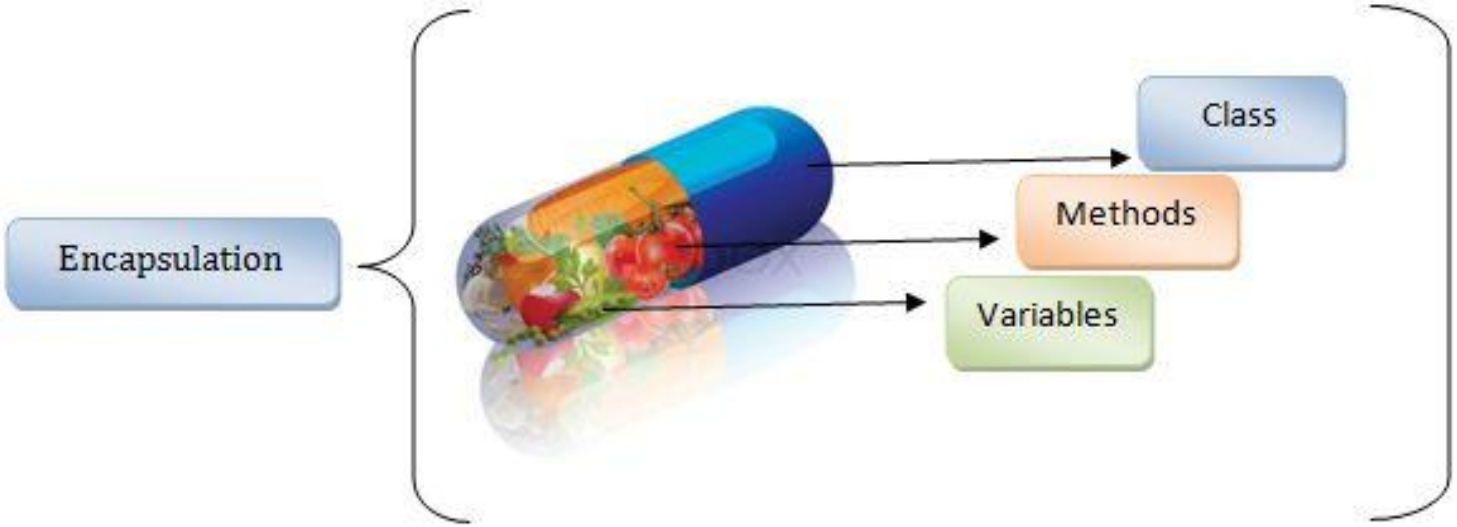


Abstraction



Encapsulation

- **Binding** (or wrapping) code and data together into a single unit are known as **encapsulation**.
- For example, a capsule, it is wrapped with different medicines.
- A java class is the example of encapsulation.
- Java is fully encapsulated class because all the data members are private here.



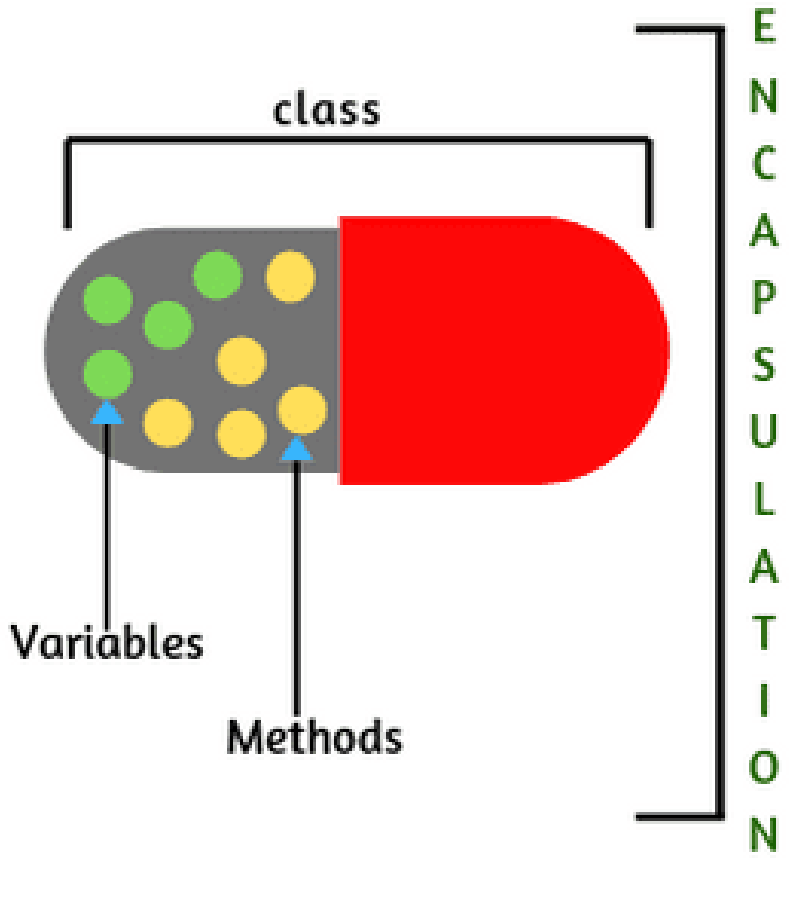
Encapsulation

```
class
{

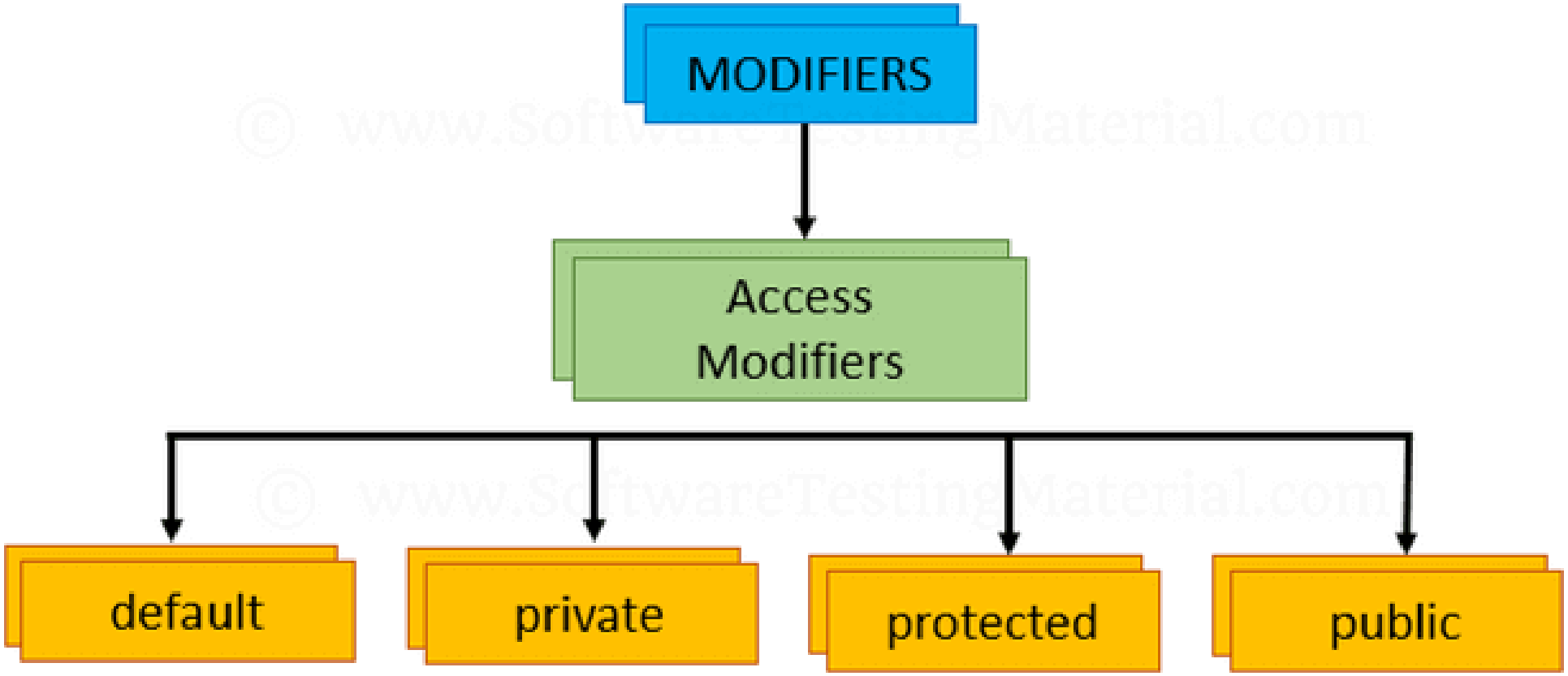
   data members
         +
   methods (behavior)

}
```

ENCAPSULATION



Access Modifier



© www.SoftwareTestingMaterial.com

Access Modifier

```
class Rectangle {
  int width, height;
  int CalcArea() {
    return width * height;
  }
}
```

Class definition

Fields (data)

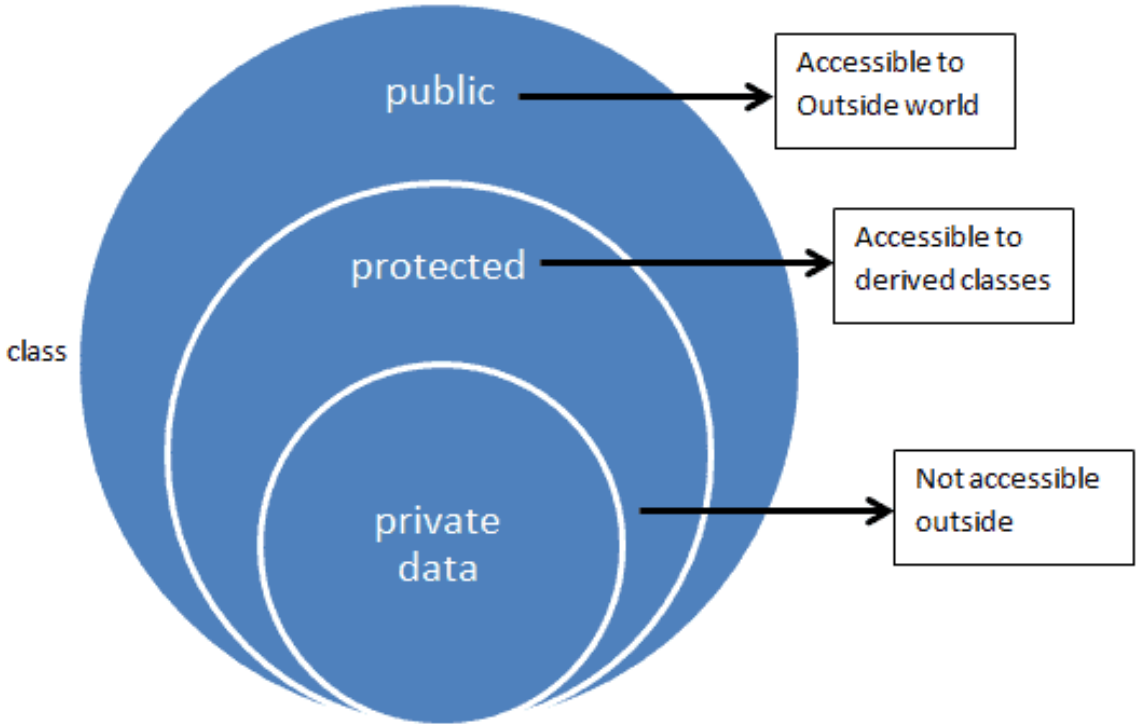
Methods (actions)

Objects

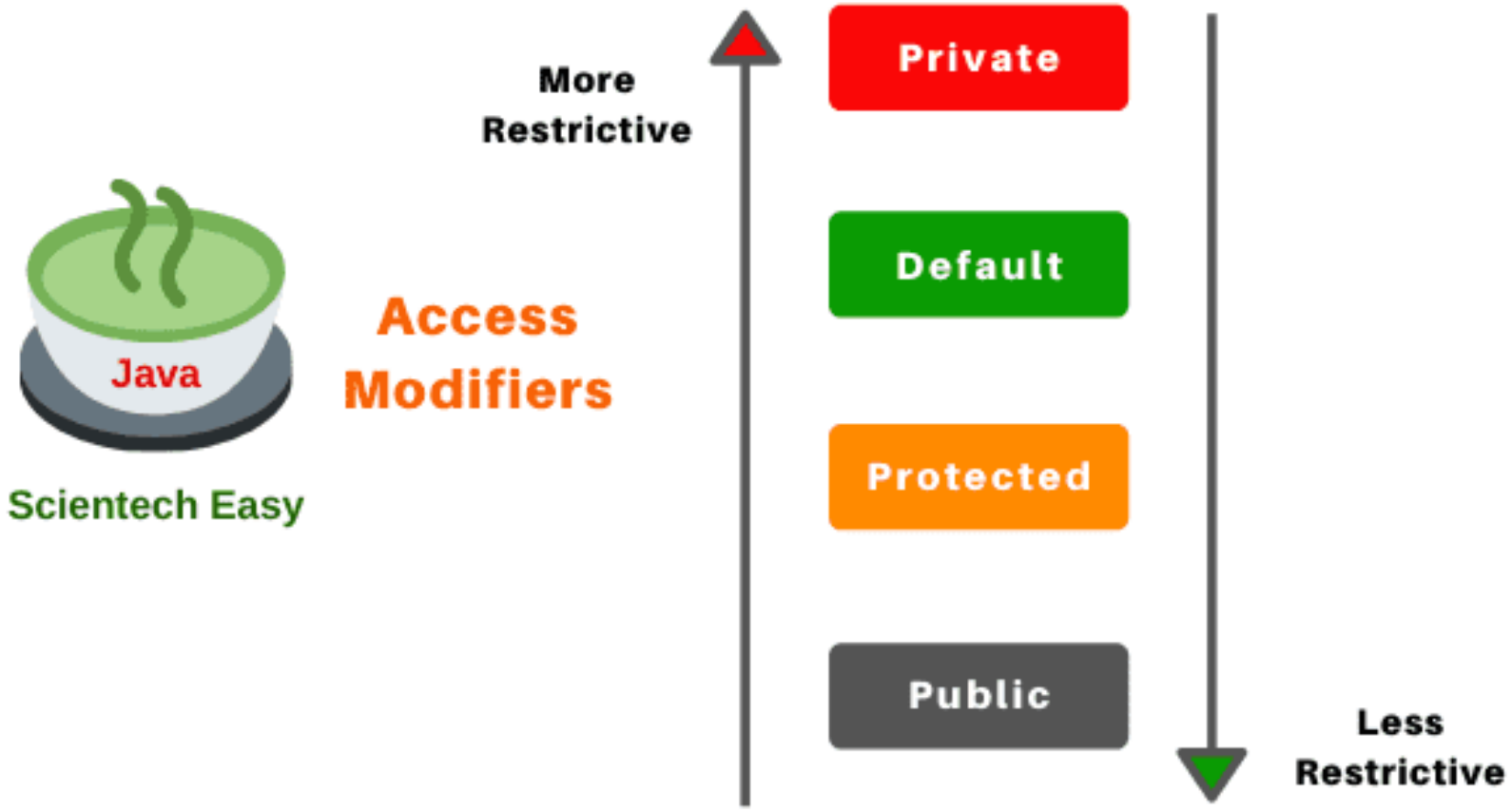
width = 5
height = 6

width = 7
height = 3

width = 6
height = 4



Access Modifier



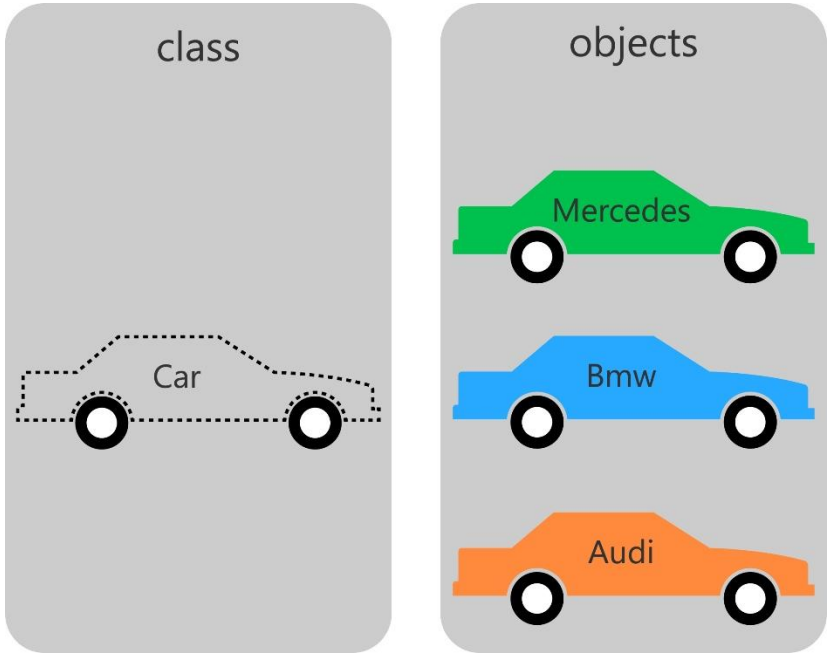
Advantages of OOPs

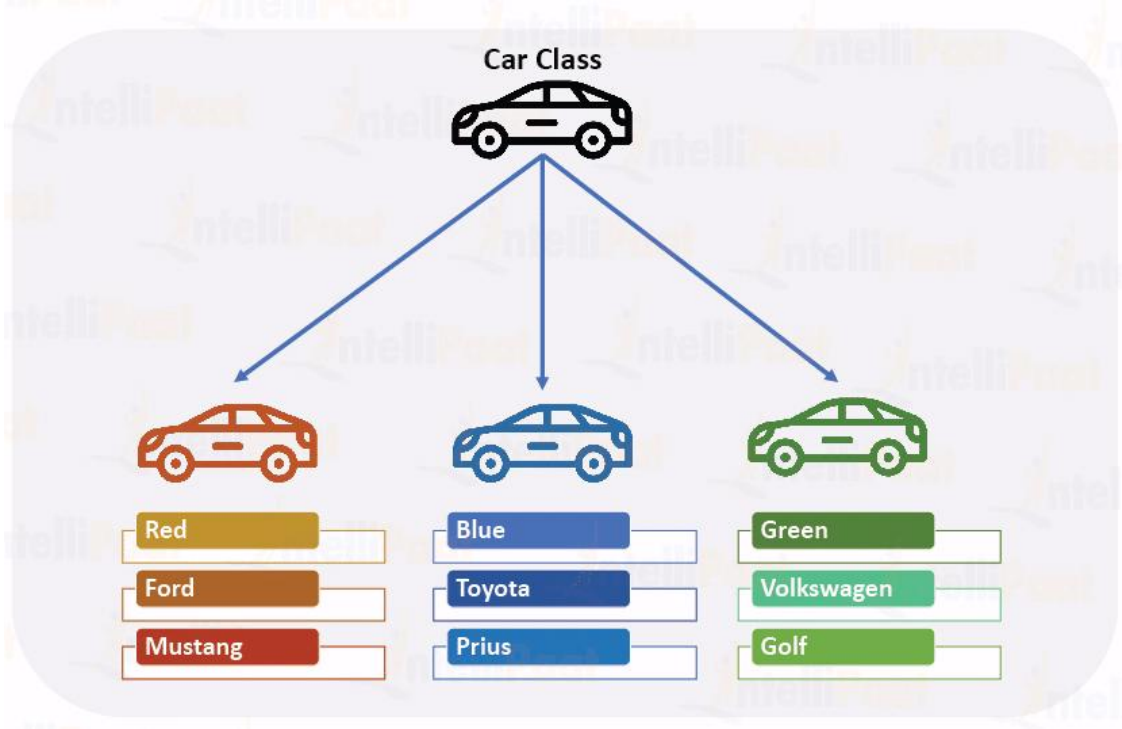
The following are the advantages of using the OOPs in Java:

- The implementations of OOPs concepts are easier.
- The execution of the OOPs is faster than procedural-oriented programming.
- OOPs provide code reusability so that a programmer can reuse an existing code.
- OOPs help us to keep the important data hidden.
- OOP helps to keep the Java code DRY "Don't Repeat Yourself“.
- OOP provides a clear structure for the programs
- OOPs makes development and maintenance easier, whereas, in a procedure-oriented programming language, it is not easy to manage if code grows as project size increases.

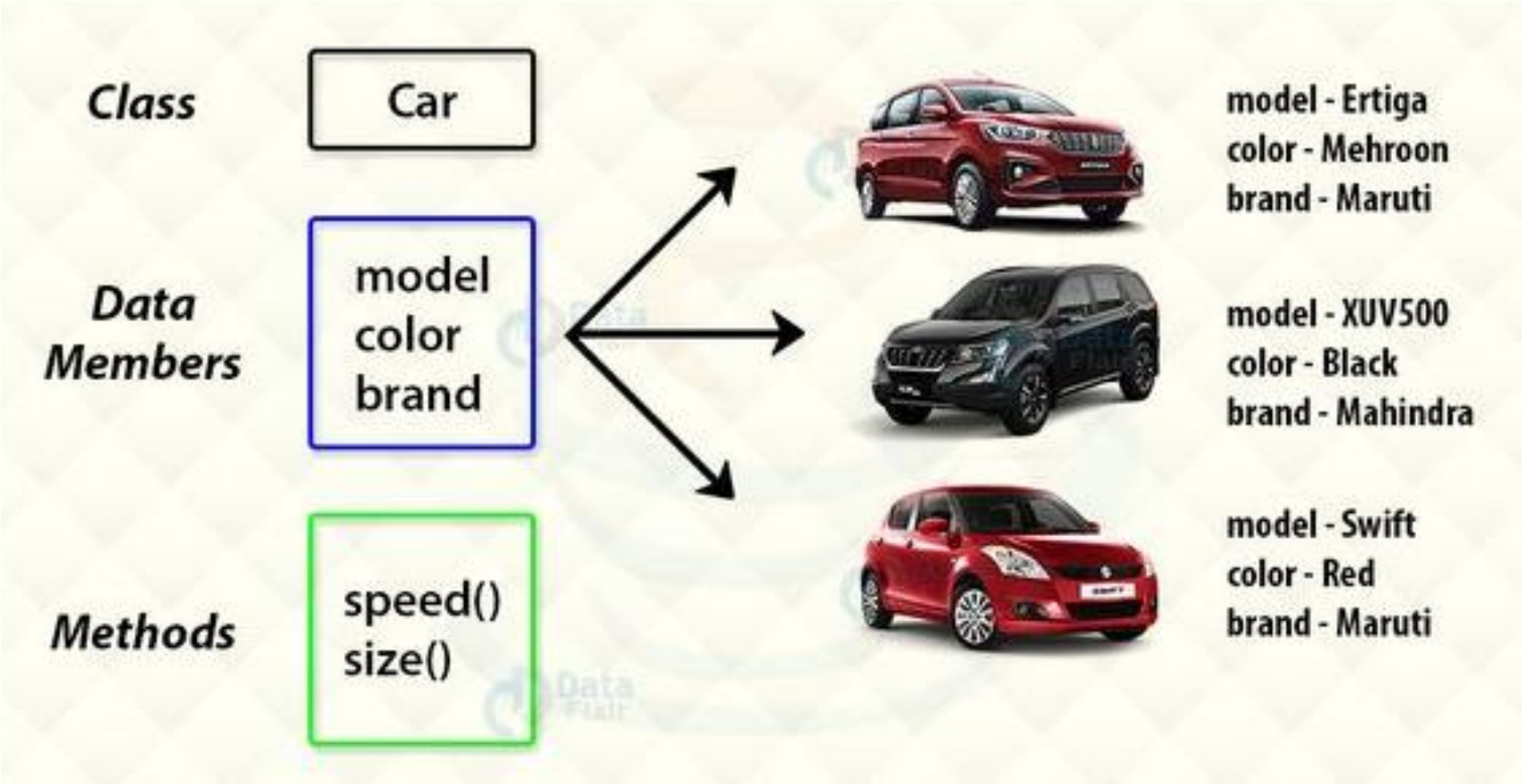
What are Classes and Objects?

- **Class** is a template for objects, and an **object** is an instance of a class.
- When the individual **objects** are created, they inherit all the variables and methods from the **class**.





What are Classes and Objects?



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.

Note: Class should always start with an uppercase first letter,

and that the name of the java file should match the class name.

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

Classes/Objects

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


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

```
// Output
5
```

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);
  }
}
```

```
// Output
5
5
```

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)).

Note: 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.

Main.java

Second.java

Multiple Classes

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

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

```
// Output
5
```

Class Attributes

Previously, 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.

Note: another term for class attributes is **fields**.

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

Accessing 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);
  }
}
```

```
// Output
5
```

Modify Attributes

You can also modify attribute values.

```
public class Main {
  int x;

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

```
// Output
40
```

Modify Attributes

Or override existing values.

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

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

```
// Output
40
```

Modify Attributes

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

```
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);
  }
}
```

```
//Output
Main.java:6: error: cannot assign a value to final variable x
   myObj.x = 25; // will generate an error: cannot assign a
value to a final variable
```

Modify Attributes

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...).

- The final keyword is called a "modifier".

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.

```
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
  }
}
```

```
// Output
5
25
```

Multiple Attributes

You can specify as many attributes as you want.

```
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);
  }
}
```

```
//Output
Name: John Doe
Age: 24
```

Class Methods

Methods are declared within a class, and that they are used to perform certain actions.

myMethod() prints a text (the action), when it is called. To call a method, write the method's name followed by two parentheses **()** and a semicolon**;**

```
public class Main
{
  static void myMethod()
  {
    System.out.println("Hello World!");
  }
}
```

Example

Inside main, call myMethod().

```
public class Main
{
    static void myMethod()
    {
      System.out.println("Hello World!");
    }

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

// Outputs
"Hello World!"

Thanks

References:
https://www.w3schools.com & https://www.tutorialspoint.com & https://www.javatpoint.com