پوهنتون کاردان
KARDAN UNIVERSITY

Object Oriented Programming (Java)

Classes & Objects

What is OOP?

- OOP stands for **Object-Oriented Programming**.
- **Object-oriented programming** has a sweeping impact because it appeals at multiple levels and promises faster and cheaper development and maintenance.
- The word **object-oriented** is the combination of two words i.e. **object** and **oriented**.
- The dictionary meaning of the object is an article or entity that exists in the real world.
- The meaning of oriented is interested in a particular kind of thing or entity.
- In layman's terms, it is a programming pattern that rounds around an object or entity are called **object-oriented programming.**

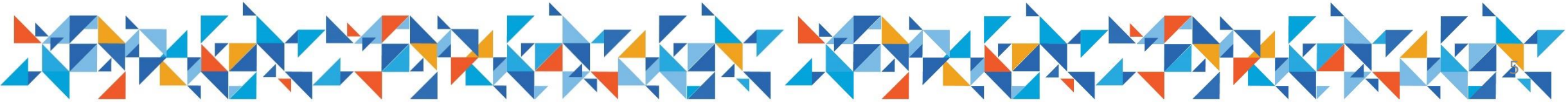
- The **object-oriented programming** is basically a computer programming design philosophy or methodology that organizes/ models software design around data, or objects rather than functions and logic.
- An object is referred to as a data field that has unique attributes and behavior.
- It is the most popular programming model among developers.
- It is well suited for programs that are large, complex, and actively updated or maintained.
- It simplifies software development and maintenance by providing major concepts such as **abstraction, inheritance, polymorphism**, and **encapsulation**. These core concepts support OOP.

- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

OOP advantages over procedural programming

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Tip: The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.



What are Classes and Objects?

- Classes and objects are the two main aspects of object-oriented programming.
- A class is a template for objects, and an object is an instance of a class

class	objects
Car	Volvo
	Audi
	Toyota

Class

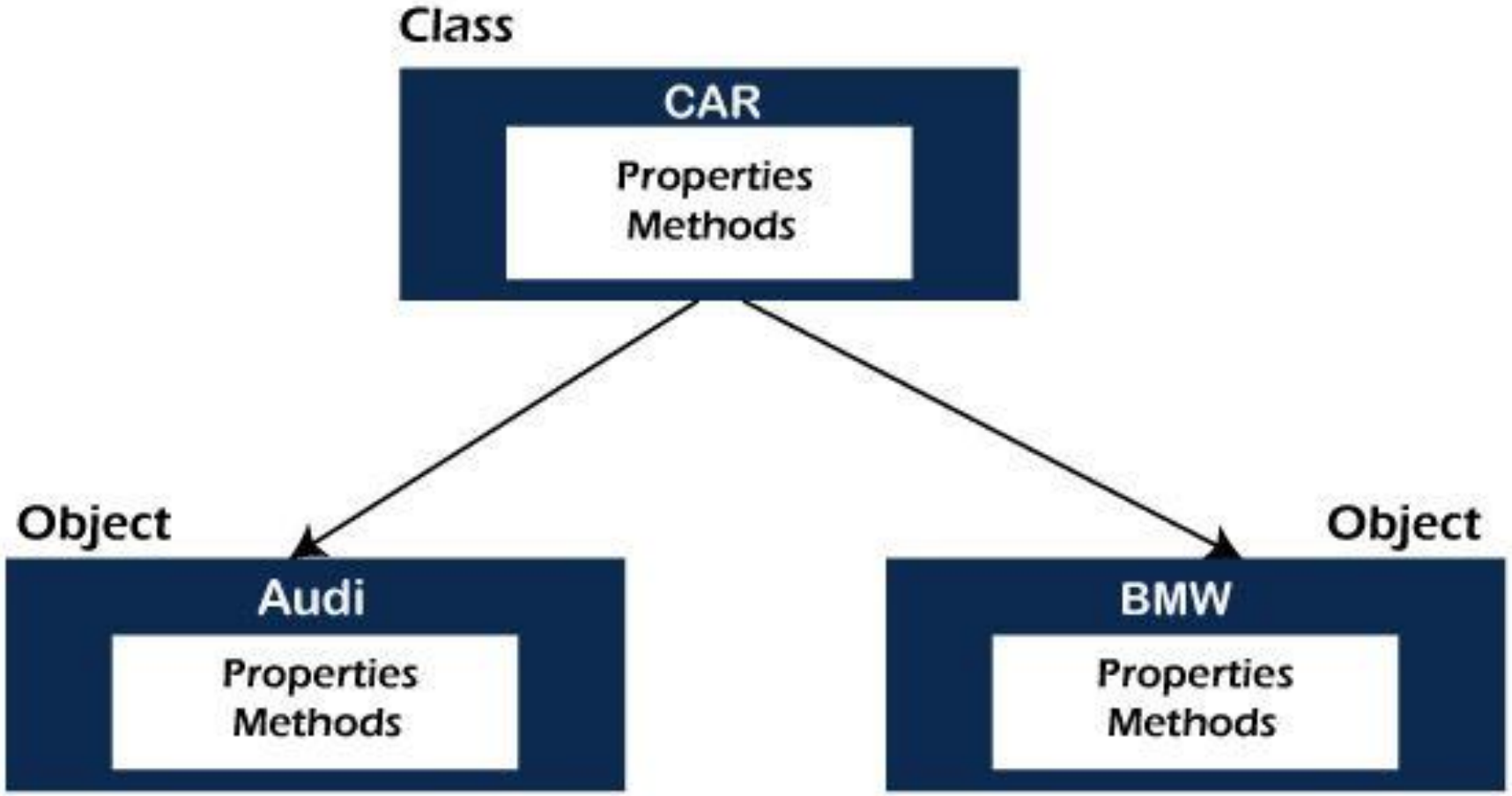
- A class is a blueprint or template of an object.
- It is a user-defined data type.
- Inside a class, we define variables, constants, member functions, and other functionality.
- it binds data and functions together in a single unit.
- It does not consume memory at run time.
- Note that classes are not considered as a data structure.
- It is a logical entity.
- It is the best example of data binding.
- Note that a class can exist without an object but vice-versa is not possible.

Object

- An object is a real-world entity that has attributes, behavior, and properties.

- It is referred to as an instance of the class.
- It contains member functions, variables that we have defined in the class.

- It occupies space in the memory.
- Different objects have different states or attributes, and behaviors.


Class
CAR
Properties
Methods
Object
Audi
Properties
Methods
Object
BMW
Properties
Methods

- 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.

Create a Class

```
public class Main {
    int x = 50;
}
```

Create an Object

- To create an object of Main, specify the class name, followed by the object name, and use the keyword new:

```
public class Main {
    int x = 50;

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

Multiple Objects

- You can create multiple objects of one class:

```
public class Main {
    int x = 50;

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

        Main obj2=new Main();
        System.out.println(obj2.x);

    }
}
```

Using 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)).
- Remember 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

Using Multiple Classes

```
public class Main {
    int x = 50;
}

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

        Main obj2=new Main();
        System.out.println(obj2.x);

    }

}
```

Java Class Attributes

- Class *attributes* are basically *variables* within a class.
- Another term for class attributes is ***fields***.

```
public class Main {
    int x = 50;
    int y =100;

}
```

Accessing Attributes

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

```
public class Main {
    int x = 50;

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

Modify Attributes

- You can also modify attribute values:

```
public class Main {
    int x;


    public static void main(String[] args) {
    Main obj=new Main();
    obj.x=200;
    System.out.println(obj.x);


    }

}
```

- Or override existing values:

```
public class Main {
    int x=25;


    public static void main(String[] args) {
        Main obj=new Main();
        obj.x=500;
        System.out.println(obj.x);


    }

}
```

Final Keyword

- Final keyword will not allow to override existing values.
- 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=25;


        public static void main(String[] args) {
        Main obj=new Main();
        obj.x=500;
        System.out.println(obj.x);


    }

}
```

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=25;


    public static void main(String[] args) {
    Main obj=new Main();
    Main obj2=new Main();

    obj2.x=500;

    System.out.println(obj.x);//25
    System.out.println(obj2.x);//500

}}
```

Multiple Attributes

- You can specify as many attributes as you want:

```
public class Main {
    String firstname="Muhammad";
    String lastname="Zubair";

    int age=25;


        public static void main(String[] args) {
        Main obj=new Main();

        System.out.println("Name is : "+ obj.firstname + " " + obj.lastname);
        System.out.println("Age is: "+ obj.age);
        }}
```

Java Class Methods

- Methods are declared within a class
- They are used to perform certain actions

```
public class Main {
static void mymethod(){
    System.out.println("Hello programmers!");

}

    public static void main(String[] args) {
        mymethod();//Methodcall or function call
    }}
```

Static vs. Public

- You will often see Java programs that have either *static* or *public* attributes and methods.

- we create a *static* method, which means that it can be accessed without creating an object of the class,
- unlike *public* method, which can only be accessed by creating objects

```
public class Main {

static void mystaticmethod(){
    System.out.println("This is a static method");
}

public void mypublicmethod(){
    System.out.println("This is a public method");
}


        public static void main(String[] args) {
            mystaticmethod();

           Main obj=new Main();
            obj.mypublicmethod();
        }}
```

Access Methods With an Object

```
public class CAR {

    public void speed(int maxspeed){
        System.out.println("The maximum speed of this car is: "+maxspeed);
    }
    public void brake(){
        System.out.println("The car will stop after pressing the brakes!");
    }

    public static void main(String[] args) {
        CAR obj=new CAR();//Creating an object of the class CAR

        obj.speed(220); // Function call for speed method
        obj.brake();    // Function call for brake method
    }
}
```

Java Constructors

- A constructor in Java is a **special method** that is used to initialize objects.
- The constructor is called when an object of a class is created.
- It can be used to set initial values for object attributes.

پوهنتون کاردان
KARDAN UNIVERSITY

```java
public class Main {
  int x;  // Create a class attribute

  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {

    Main myObj = new Main(); // Create an object of class Main (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x

  }}
```

Access Modifiers

1) public	The code is accessible for all classes
2) private	The code is only accessible within the declared class
3) protected	The code is accessible in the same package and **subclasses**.
4) default	The code is only accessible in the same package. This is used when you don't specify a modifier.

پوهنتون کاردان
KARDAN UNIVERSITY

Thank You…!