9.OOP Introduction

Sakib Abrar

CSE

Bangladesh University of Engineering & Technology

sakib.cghs@gmail.com

September 10, 2020

Overview

OOP Principles

- **Abstraction:** Abstraction means using simple things to represent complexity. We all know how to turn the TV on, but we don't need to know how it works in order to enjoy it.
- **Encapsulation:** This is the practice of keeping fields within a class private, then providing access to them via public methods. It's a protective barrier that keeps the data and code safe within the class itself.
- **Inheritance:** This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel.
- **Polymorphism:** This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is **method overloading**. The other form is **method overriding**.

Java Class

A class is a template definition of the methods and variables in a particular kind of object.

This is a blueprint for some objects that we will create later.

In stead of coding it whole everytime, we create a blueprint than reuse it whenerver we need to create an instance.

Java Object

An object is an instance of the class. Class is like a variable type and object is the variable.

Class is the template where we put our data the objects get created.

Example

```
class Car {
    int wheels = 4;
    int price = 100;
    String model = "Toyota";
}

public class OOPExamples {
    public static void main(String[] args) {
        Car amarGari = new Car();
        System.out.println(amarGari.model);
    }
}
```

Java Constructors

You don't always want your car model to be toyota.
You may want to be able to change your objects attributes while creation.
That was the whole point of the OOP. The template must be capable of creating different attribuited objects on the go.
And here comes the constructors to the rescue.

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.

Constructor Examples

```
class Car {
    int wheels;
    int price;
    String model;

    public Car(int wheels, int price, String model) {
        this.wheels = wheels;
        this.price = price;
        this.model = model;
    }
}
```

Constructor Examples

You can also overload constructors:

```
class Car {
    int wheels;
    int price;
    String model;

    public Car(int wheels, int price, String model) {
        this.wheels = wheels;
        this.price = price;
        this.model = model;
    }

    public Car(int price, String model) {
        this.wheels = 4;
        this.price = price;
        this.model = model;
    }
}
```

Default Constructor

If you don't implement any constructor in your class, the Java compiler inserts default constructor into your code on your behalf. You will not see the default constructor in your source code(the .java file) as it is inserted during compilation and present in the bytecode(.class file).

The default constructor is inserted by compiler and has no code in it, on the other hand we can implement no-arg constructor in our class which looks like default constructor but we can provide any initialization code in it.

Access Modifiers

- **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
- **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
- **Protected:** The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
- **Public:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table:

Most Restrictive ← → Least Restrictive

Access Modifiers ->	private	Default/no-access	protected	public
Inside class	Y	Y	Y	Y
Same Package Class	N	Y	Y	Y
Same Package Sub-Class	N	Y	Y	Y
Other Package Class	N	N	N	Y
Other Package Sub-Class	N	N	Y	Y

THE END