Monsoon 2019

Access Control Modifiers

Object Oriented Programming

by

Dr. Rajendra Prasath

Indian Institute of Information Technology

Sri City – 517 646, Andhra Pradesh, India

Recap: Method Overloading

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as method overloading

Why use Method Overloading in Java ?

- Suppose we have to perform addition of given number but there can be any number of arguments, if we write method such as a(int, int)for two arguments, b(int, int, int) for three arguments then it is very difficult for you and other programmer to understand purpose or behaviors of method they can not identify purpose of method.
- So use method overloading
- Example: Write
  - sum(int, int) for two arguments
  - sum(int, int, int) using method overloading concept.

Recap: Constructor Overloading

- Like methods, a constructor can also be overloaded.
- Overloaded constructors are differentiated on the basis of their type of parameters or number of parameters.
- Constructor overloading is not much different than method overloading.
- In case of method overloading we have multiple methods with same name but different signature, whereas in Constructor overloading we have multiple constructor with different signature but only difference is that **Constructor doesn't have return type** in Java.
- **Why do we Overload constructors ?**
  - Constructor overloading is done to construct object in different ways.

Recap: Overload Constructor

- This can be done by changing
  - Number of input parameters
  - Data-type of input parameters
  - Order/sequence of input parameters, if they are of different data-types
- Constructor Signature consists of
  - Name of the constructor which should be same as that of class name
  - number of input parameters
  - their data types
  - access modifiers like private, default, protected or public
- Access modifiers are not valid to consider in constructor overloading concept and in fact compiler throws exception if we overload constructor just by changing access modifiers keeping other things in constructor signature same

Default / Parameterized

Default Constructor	Parametrized Constructor
Takes no arguments	Takes one or more arguments
Compiler creates a default no-arguments constructor	If defined, then programmer needs to define default no-arg constructor
No need to pass any parameters while constructing new objects	At least one or more parameters needs to be passed while constructing new objects
Default constructor is used to initialize objects	Parameterized constructors are used to create distinct objects with different data

Java Copy Constructor

- There is no copy constructor in java. But, we can copy the values of one object to another

- There are many ways to copy the values of one object into another in java. They are:
  - By Constructor
  - By Assigning the Values of one object into Another
  - By clone() Method of Object class

- Let see first example of copy the values of one object into another using java constructor

Copy Constructor

- We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.
- Example:

```
Circle c1 = new Circle(10.0, 30.0, 2.0);
Circle c2 = new Circle (c1);
```

Here **new Circle(c1)** is a constructor overloading

```
Circle(Circle c) {
        this.x = c.x;
        this.y = c.y;
        this.r = c.r;
}
```

Point to Remember

- Name of the constructor should be same as that of class name
- Constructor do not have any return type unlike methods (not even void)
- Every concrete class and abstract class has a constructor
- Constructor in interfaces is not allowed
- It can have all Java statements and logic but should not return any value
- Constructor can have zero arguments which are called default constructor (or no-arg constructor)
- Constructor can have one or more input parameters - parameterized constructor

Point to Remember (contd.)

- Constructor cannot be inherited therefore it can not be overridden
- A class can have more than one constructor
- All four access modifiers; private, default, protected, public are allowed (no restriction on access modifiers)
- Private constructor are used for singleton design pattern
- Non-access modifier like static, final, synchronized, abstract, transient, volatile are not allowed
- Duplicate Constructors are not allowed (Compile time error)
- Only public, protected and private keywords are allowed before a constructor name. If you keep any other keyword before a constructor name, it gives compile time error

Point to Remember (contd.)

- Only public, protected and private keywords are allowed before a constructor name. If you keep any other keyword before a constructor name, it gives **compile time error**

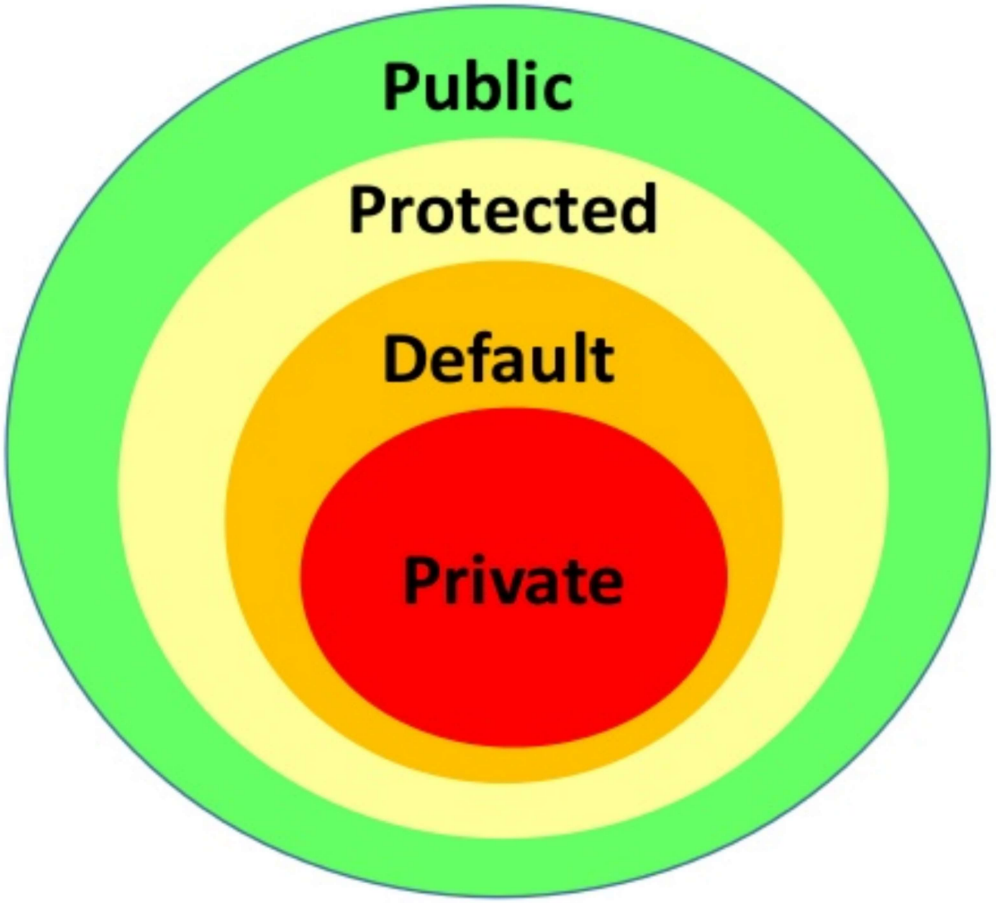
```
class A {
        final A() {
                //Constructor can not be final
        }
        static A() {
                //Constructor can not be static
        }
        abstract A() {
                //Constructors can not be abstract
        }
}
```

Constructor Chaining

- Constructor Chaining is the technique of creating an instance of a class with multiple constructors then using one constructors to call another.
- The primary use of constructor chaining is to make a program simpler, with fewer repeated lines of code

Access Modifiers

✧ 4 types of Java Access Modifiers



Access Modifiers in Java

- There are two types of modifiers in java: **access modifiers** and **non-access modifiers**
- The access modifiers in java specifies accessibility of a data member, method, constructor or class

- **There are 4 Types of Java Access Modifiers:**
  - private
  - default
  - protected
  - public
- There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient and so on

Access Control Modifiers

- Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors

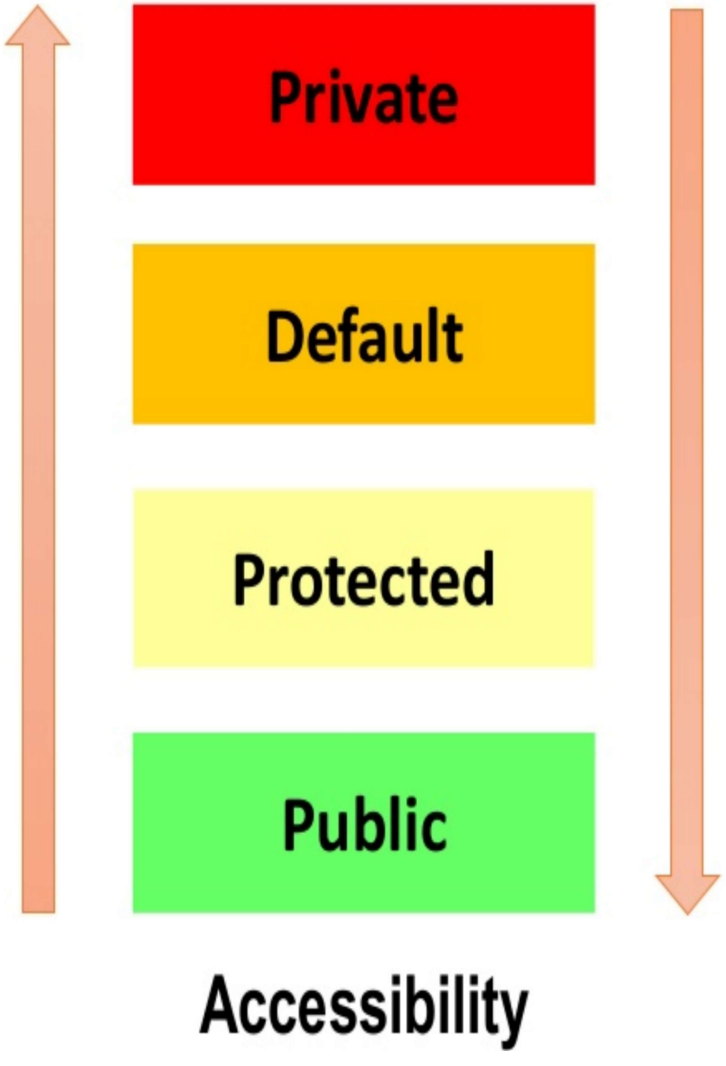
- The four access levels are –
  - Visible to the package, the default. No modifiers are needed.
  - Visible to the class only (private).
  - Visible to the world (public).
  - Visible to the package and all subclasses (protected)

Non-Access Modifiers

- Java provides a number of non-access modifiers to achieve many other functionality
  - The static modifier for creating class methods and variables.
  - The final modifier for finalizing the implementations of classes, methods, and variables.
  - The abstract modifier for creating abstract classes and methods.
  - The synchronized and volatile modifiers, which are used for threads

Access Modifiers in Java

More Restrictive



Less Restrictive

private Access Modifier

- The private access modifier is **accessible only within class**
- Methods, variables, and constructors that are declared private can only be accessed within the declared class itself.
- Private access modifier is the most restrictive access level
- Class and interfaces cannot be private.
- Variables that are declared private can be accessed outside the class, if public getter methods are present in the class.
- Using the private modifier is the main way that an object encapsulates itself and hides data from the outside world

private Access Modifier

✧ Example:

```
public class CodeTester {
    private int data = 10;

    public static void main(String args[]) {
        CodeTester ct = new CodeTester();
        System.out.println("Data is: " + ct.data);
    }
}
```

private Access Modifier

✧ Example:

```
class A {
        private int data = 20;
        private void message() {
                System.out.println("Hi Folks!!");
        }
}
public class CodeTester {
        public static void main(String args[]) {
                A obj = new A();
                System.out.println(obj.data); // Compile time error
                obj.data();     // Compile time error
        }
}
```

Use Public Methods: Use **getter** and **setter** methods

Role of private Constructor

- The private modifier when applied to a constructor works in much the same way as when applied to a normal method or even an instance variable
- Defining a constructor with the private modifier says that only the native class (as in the class in which the private constructor is defined) is allowed to create an instance of the class, and no other caller is permitted to do so
- There are two possible reasons - why one would want to use a private constructor?
  - The first is that you do not want any objects of your class to be created at all
  - The second is that you only want objects to be created internally – as in only created in your class

public Access Modifier

- The public access modifier is accessible everywhere. It has the widest scope among all other modifiers
- A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe
- However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses
- **Note:** About package we Learn Later in Details.

public Access Modifier

✧ Example:

```
class A {
        public int data = 40;
}

public class Test {
        public static void main(String args[]){
                A obj=new A();
                System.out.println("Data is: " + obj.data);
        }
}
```

protected Access Modifier

- The protected access modifier is accessible within package and outside the package but through inheritance only
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class
- Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it

protected Access Modifier

✧ Example:

```
class A {
        protected int data = 40;
}

public class Test {
        public static void main(String args[]) {
                A obj = new A();
                System.out.println("Data is: "+obj.data);
        }
}
```

default Access Modifier

- Default access modifier means we do not explicitly declare an access modifier for a class, field, method, and so on
- A variable or method declared without any access control modifier is available to any other class in the same package
- The fields in an interface are implicitly public static final and the methods in an interface are by default public
- The default modifier is accessible only within package

default Access Modifier

✧ Example:

```
class MyCalculator {
        int Add(int x,int y) {
                return x+y;
        }
}

class CodeTeser {
      public static void main(String arg[]) {
            MyCalculator M = new MyCalculator();
            //Calling Add() through Object
            System.out.println("The sum is : " + M.Add(10,20));
      }
}
```

We do not Explicitly Declare an Access Modifier

Access Modifiers in Java

- Allowed Levels of access modifiers

	public	private	protected	<unspecified>
Class	allowed	Not allowed	Not allowed	allowed
Constructor	allowed	allowed	allowed	allowed
Variable	allowed	allowed	allowed	allowed
Method	allowed	allowed	allowed	allowed

	Class	Subclass	Package	outside
public	allowed	allowed	allowed	allowed
private	allowed	Not allowed	Not allowed	Not allowed
protected	allowed	allowed	allowed	Not allowed
<unspecified>	allowed	Not allowed	allowed	Not allowed

this – keyword in java

- **this. (this dot)**
- used to differentiate variable of class and formal parameters of method or constructor
- "this" keyword are used for two purposes:
  - It always points to current class object.
  - Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity (no clarity between formal parameter and data members of the class)
  - Data members of the class must be preceded by "this"
  - Example: Define a Circle - center (x,y) and radius r

```
Circle (float x, float y, float r){
        x = x;
        y = y;
        r = r;
}
```

Exercise - 3

- **Practice Problems using various access modifiers**

  - Apply it for doing Basic Arithmetic operations
    - Addition
    - subtraction
    - Multiplication
    - Division
    - Modular division
    - Also mathematical functions

  - Develop an application **Basic Calculator** using method overloading and various access modifiers.

Assignments / Penalties

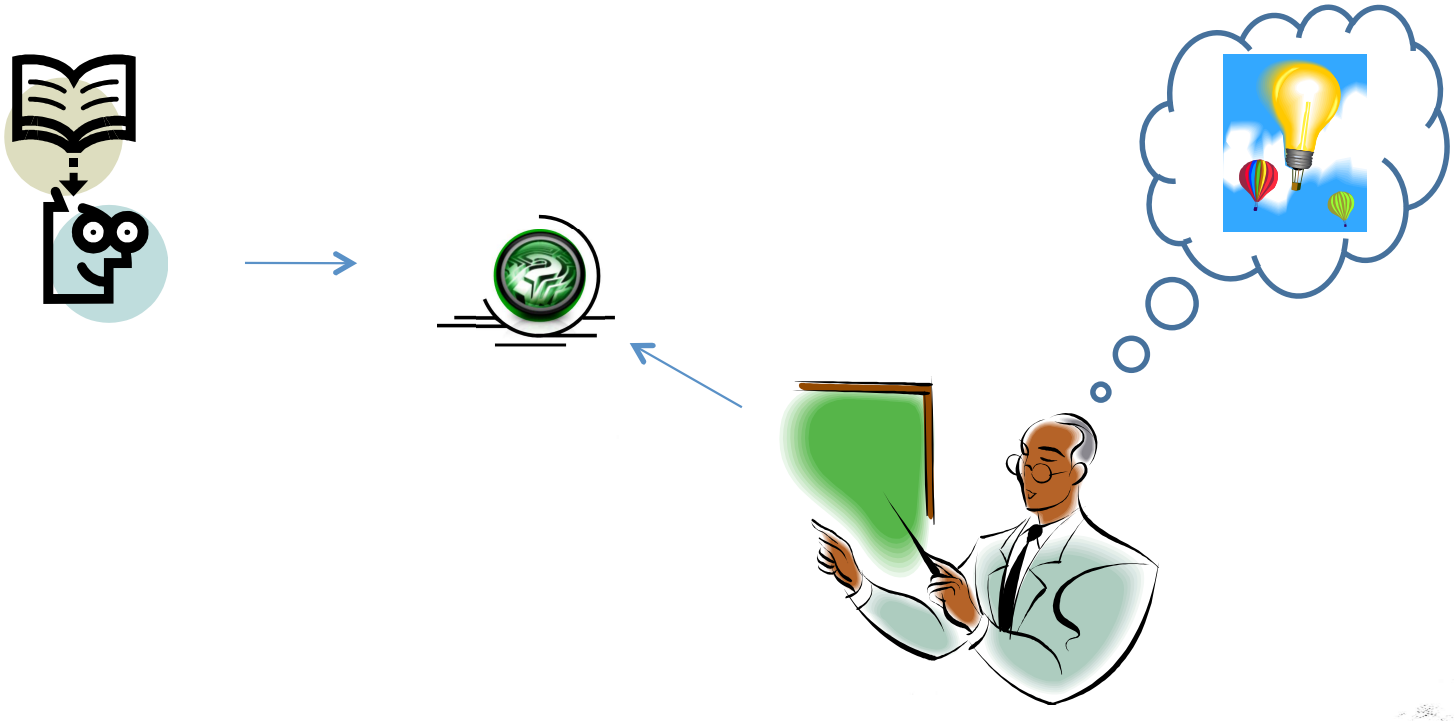
- Every Student is expected to complete the assignments and strictly follow a fair Academic Code of Conduct to avoid severe penalties

- Penalties would be heavy for those who involve in:
  - **Copy and Pasting** the code
  - **Plagiarism** (copied from your neighbor or friend – in this case, both will get "0" marks for that specific take home assignments)
  - If the candidate is **unable to explain his own solution**, it would be considered as a "copied case" !!
  - **Any other unfair means** of completing the assignments

Assistance

- You may post your questions to me at any time
- You may meet me in person on available time or with an appointment
- You may leave me an email any time
  (email is the best way to reach me faster)

Thanks …



… Questions ???