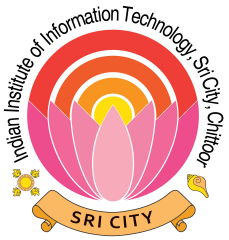


Monsoon 2019

Constructors

Object Oriented Programming

by

Dr. Rajendra Prasath

Indian Institute of Information Technology

Sri City – 517 646, Andhra Pradesh, India

Recap: Objects in JAVA ?

- An entity that has **state** and **behaviour** is known as an object
  - **Examples:** Chair, bike, marker, pen, table, car etc
  - It can be physical or logical
- An object has three characteristics:
  - **State:** represents data (value) of an object
  - **Behaviour:** represents the behaviour (functionality) of an object such as deposit, withdraw and so on
  - **Identity (Internally used):**
    - Signature (unique) of the object
    - Object identity is typically implemented via a unique ID
    - The value of the ID is not visible to the external user
    - But, Internally by JVM to identify each object uniquely

Recap: First Example

```
pubic class Increment {
        int myCount = 0;
        void increment ( ) {
                myCount = myCount + 1;
        }
        void print ( ) {
                System.out.println ("count = " + myCount);
        }
        public static void main(String[] args) {
                increment c1 = new Increment ( );
                c1.increment ( );    // c1's myCount is now 1
                c1.increment ( );    // c1's myCount is now 2
                c1.print();
                c1.myCount = 0;    // effectively re                r
                c1.print();
        }
}
```

Class Name

Variable

Method – Increment()

print() method

Main Method

Output is:
count = 2
count = 0

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: Static Binding - Example

- **Static Polymorphism** is also known as **compile time binding** or early binding

```
class Addition {
    void sum(int a, int b) {
        System.out.println(a+b);
    }
    void sum(int a, int b, int c) {
        System.out.println(a+b+c);
    }
    public static void main(String args[]) {
        Addition add = new Addition();
        add.sum(10, 20);
        add.sum(10, 20, 30);
    }
}
```

Output is: 30 60

Recap: Static Variable - Example

```
public class CodeTester {
    private static double salary;
    public static final String  DEPARTMENT = "CSE";
    public static void main(String args[]) {
        Salary = 1000;
        System.out.println(DEPARTMENT + " Avg.
Salary: " + salary);
    }
}
```

If variables are accessed from an outside class, the constant should be access as CodeTester.DEPARTMENT

Costructors

- **Constuctors in JAVA**
  - Constructor in Java is a special type of method that is used to initialize the object.
  - Java constructor is invoked at the time of object creation. It constructs the values, that is, data for the object

- **Rules for Creating Java Constructor**
  - Two rules that defines a constructor.
    - Constructor name must be same as class name
    - Constructor must have no explicit return type

Using Constructors - Some Rules

- **Important Rules to follow:**
  - Constructor Initializes an Object
  - Constructor cannot be called like methods
  - Constructors are called automatically as soon as object gets created
  - Constructor don't have any return Type (even Void)
  - Constructor name is same as that of "**Class Name**"
  - Constructor can accept parameter
  - Default constructor automatically called when object is created

Types of Constructors in JAVA

- **TWO Types**
  - Default Constructors
  - Parameterized Constructor

```
class Circle {
        float x, y;
        float radius;

        void Circle() {
            // Default
        }
}
```

```
class Circle {
        float x, y;
        float r;
        void Circle(float xc, float yc, float ra) {
            this.x = xc;
            this.y = yc;
            this.r = ra;
        }
}
```

Default Constructor

- **An Example**

```
public class Circle {
    float x, y, r;

    void Circle() {// Default Constructor
        System.out.println("New Circle is created!");
    }
    public static void main(String[] args) {
        Circle c = new Circle();
    }
}
```

How does JVM handles this?

- **Explanation**
  - First JVM searches for main() method in class whose access modifier is public
  - And this main() method should be static, as JVM invokes this method without instantiating the object
  - Return type should be void, as there is no need to send any return value to invoking JVM
  - So when JVM executes main() method, then below statement gets fired

    Circle circle = new Circle();
  - This statement in turn will invoke default constructor no argument constructor gets executed and prints the message inside default constructor
  - Finally, program exits with success

Default Constructor - Variation

```
public class Circle {
    float x, y, r;
    void Circle() {// Default Constructor
        x = 10.0;
        y = 20.0;
        r = 30.0;
    }
    public static void main(String[] args) {
        Circle c = new Circle();
    }
}
```

DC - Another Variation

```
public class Circle {
    float x, y, r;
    void Circle() {// Default Constructor
        setDimensions();
    }
    void setDimensions() {
            x = 10.0, y = 20.0, r = 30.0;
    }
    public static void main(String[] args) {
        Circle c = new Circle();
    }
}
```

DC – Runtime Variations

```
public class Circle {
        float x, y, r;
        void Circle() {    // Default Constructor
        }
        void setDimensions() {
                x = 10.0, y = 20.0, r = 30.0;
        }
        void setDimensions(float xc, float yc, float rad) {
                x = xc; y = yc; r = rad;
        }
        public static void main(String[] args) {
             Circle c = new Circle();
        }
}
```

Parameterized Constructor

- A constructor having parameters is known as **Parameterized Constructor**
- **Why do we need this?**
  - Distinct objects of the same class could be created during run time
  - Constructor can have values called **arguments**
  - Arguments can be of any type:
    - Primitive data types
    - Composite Data types OR
    - any objects
- **Constructor can take any number of arguments**

Points to remember

```
public class Circle {
    float x, y, r;
    public void Circle(float xc, float yc, float rad) {
        this.x = xc; this.y = yc; this.r = rad;
    }
    public int getRadius() { return r; }
    public static void main(String[] args) {
        Circle c = new Circle();
        System.out.print("Radius = " + c.getRadius());
    }
}
```

Compiler Error

If we do not define any constructor, compiler defines default one
If we define any constructor, compiler does not create it for us

Differences: Constructor/Method

Java Constructor	Java Method
Constructor is used to initialize the state of an object.	Method is used to expose behaviour of an object.
Constructor must not have return type.	Method must have return type.
Constructor is invoked implicitly.	Method is invoked explicitly.
The java compiler provides a default constructor if you don't have any constructor.	Method is not provided by compiler in any case.
Constructor name must be same as the class name.	Method name may or may not be same as class name.

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.

Constructor Overloading Needed?

Why Constructor Overloading is Required in Java?

- Constructor provides a way to create objects implicitly of any class using '**new**' keyword
- So, overloaded constructor serves many ways to create distinct objects using different types of data of same class

Example

- StringBuffer class has four overloaded constructors
- StringBuffer(String str) is one of the parametrized constructor which has a initial capacity of 16 plus length of the String supplied
- Use this constructor if you have initial string value
- Or else, if we don't have any idea about initial String to specify then simply use 1st overloaded constructor which has no argument (default constructor)

Ways to 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

Exercise - 2

- **Geometric Objects**
  - Create a set of Triangles
  - Three points are required for constructing a Triangle
  - Use the following to generate a list of triangles:
    - Method Overloading
    - Constructor Overloading

Assignments / Penalties

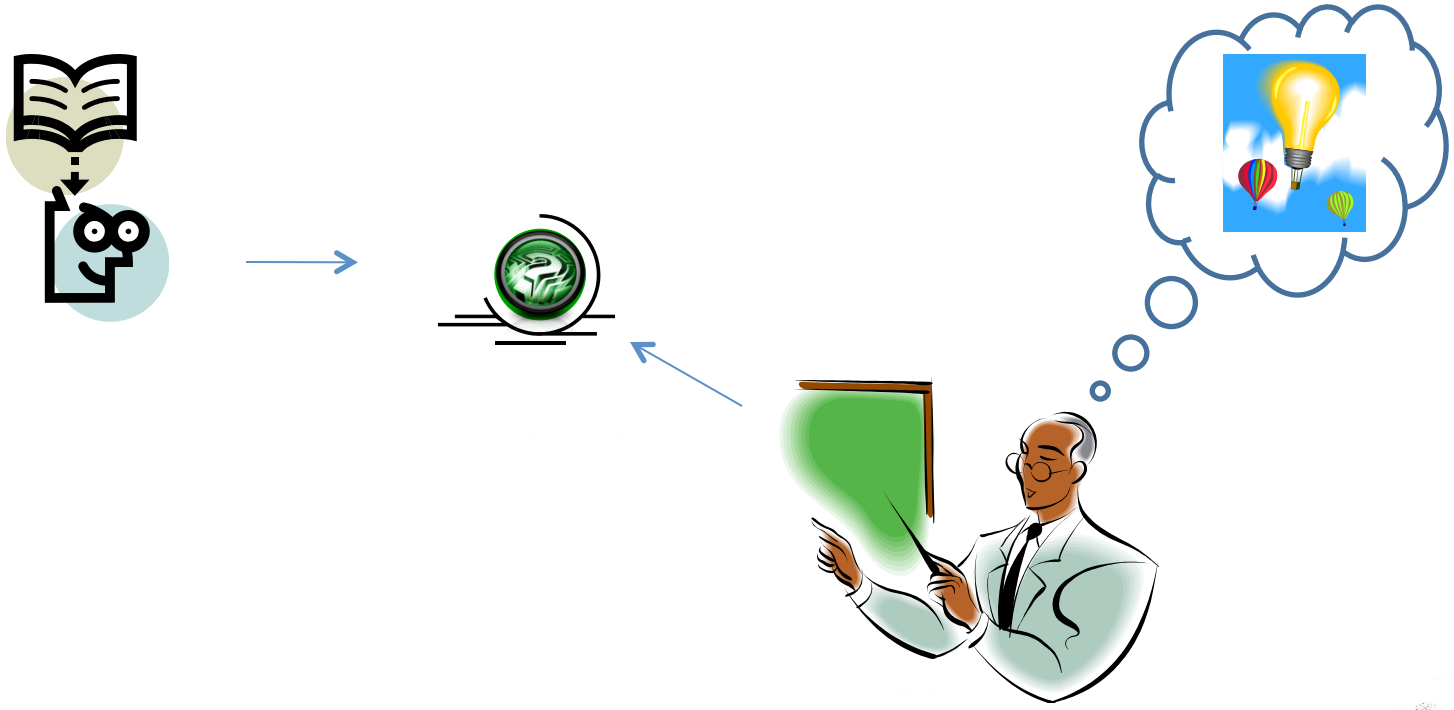
- Every Student is expected to complete the assignments and strictly follow a fair Academic Code of Conduct to avoid severe penalties

- 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 ???