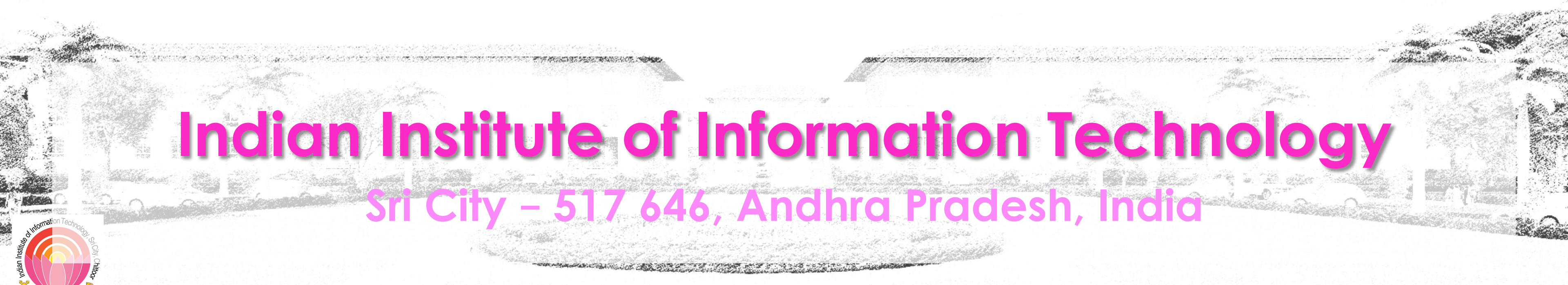
Monsoon 2019

Method Overloading

# 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: Remember 3 types

**Example: Create a Banking Software with functions like**

- Deposit
- Withdraw
- Show Balance

**Class and interface**

```
public class Employee  {
        //code snippet
}

interface Printable  {
        //code snippet
}
```

**Package and constant**

```
package com.bank;
class Employee  {
        //code snippet
}
class Employee  {
        //constant
        int MIN_AGE = 18;
        //code snippet
}
```

**Variables and Methods**

```
class Employee {
        //variable
        int id;
        //code snippet
}

class Employee  {
        //method
        void draw()  {
        //code snippet
        }
}
```

# 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: Copying of Objects

- **Shallow Copying vs Deep Copying**

  - Copying an object involves getting another object with the same properties of the original.

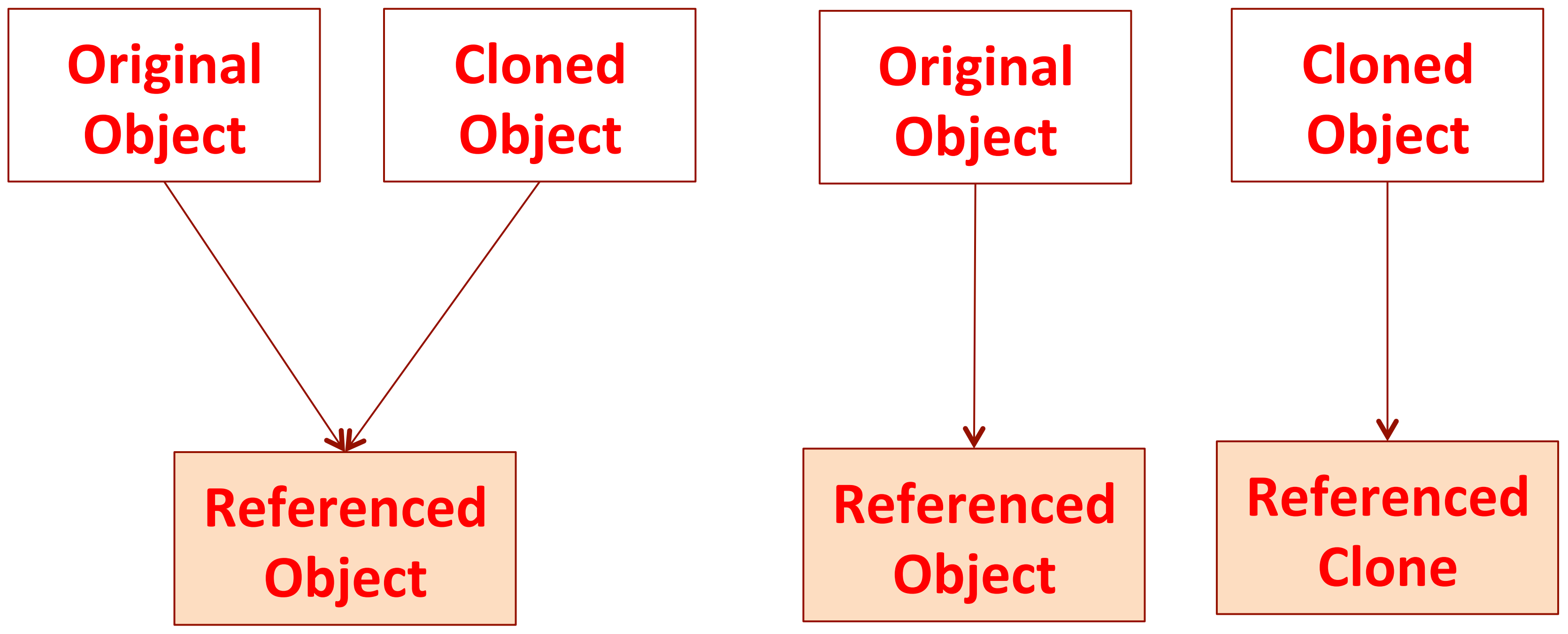
  - Here, there exists two ways:
    - two objects having their own set of properties (instance variables)

      OR

    - both objects referring the same location of properties.

## Copying Objects

- Shallow Clone
- Deep Clone



## Methods

- **What is a Method?**
  - A Collection of Statements that are grouped together to perform an operation / a specific task
  - Every JAVA program has at least one method – main()
- Examples:
  - System.**out**.println()
    - Needs execution of several steps before displaying the specified message.
  - next()
  - nextInt()
  - toString()

## Why do we use Methods

- **Modules**
  - Divide a large code into module
    - Easy to debug and maintain the code
- An Example:
  - You are writing a **Calculator** Program
  - Tasks:
    - Addition()
    - Subtration()
    - Multiplication()
    - Division()
    - OR Any other method defined as required

# Methods - Advantages

- **Methods**
  - Separate the **concept** (what is to be done?) from **Implementation** (How is it done?)
  - **Easier** to understand the programs
  - **Code Reuse**: Methods can be called several times in the same program
  - **Code Repetition:** Avoid writing code multiple times
  - **Reduce:** Complexity of the code could be reduced (How?)
  - **Maintainability:** Methods allow increased code maintainability

# Methods - Types

- JAVA allows **two types** of Methods:
  - **Library Methods or Pre-Defined Methods**
  - **User Defined Methods**

- **Pre-Defined Methods**
  - Rich Collection of JAVA Library (Classes)
    - java.util.*
    - java.io.*
    - java.lang.*
    - java.math.*

    - Use import the packages (or classes individually)

## Pre-Defined Method

- ✧ main() is a special method (The main execution starts here)

- ✧ **Example:**

```
public class Check {
   public static void main(String [] args {
      System.out.println("Hello world");
   }
}
```

## User Defined Method

- These Methods are created for specific tasks
- Example: create a method for add two number, say **sum()**
  - **sum()** is called user defined method
- Name of the method can be anything but it has to comply with JAVA Pogramming standards
- However, You can write any name of user defined method as you like or according to your requirements.

# User Defined Method - Features

- ✧ Set Your own method as per your needs.
- ✧ Method type may change as like return value types (int, char, double) method or only return type method like void.
- ✧ UDM (User Define Method) always accessed publicly, privately and protected formation.
- ✧ Be careful with the naming of methods

## UDM - Naming

- While defining a method, remember some points that are given below:
  - Do not use any reserved word that includes existing with the System method names.
  - The method name must be irreplaceable among user defined method for the number of arguments.
  - The method name can only comprehend letters, numbers, and the underscore (_).
  - But you must be remembering that the method name must start with a letter.
  - The method names which cannot be exceed 128 characters.

# Uer Defined Method - Example

Access Modifier

Return type

Method Name

Parameter(s) List

```
public static int sum(int a, int b) {

        // method body

}
```

One block of statements

Method body Contain set of Statements which Actually Perform by Method

# A Better Understanding of UDM

- **public** means that the method is visible and can be called from other objects of other types.
  - Other alternatives are private, protected, package and package-private. See here for more details.
- **static** means that the method is associated with the class, not a specific instance (object) of that class.
  - This means that you can call a static method without creating an object of the class.
- **int** means that the method has return value of integer type

# Type of the return value

- ✧ Method may or may not **return** a value

- ✧ **return** type may be **int**, **float**, **char** or any other valid data type
- ✧ The keyword is ‘return’ is used to **return some value** from the method
- ✧ The data type of the returning variable should **match** with returnvalue data type
- ✧ If the method **does not return** any value then keyword ‘**void**’ must be used

## Instance Method

- An Instance method is a method that act upon the instance variable
- To call the instance method, we should create an object of class
- instance method are called using object of class.
- There are two types of instance method
  - Accessor Method
  - Mutator Method

# Accessor Method

- These method accesses or reads instance variables but do not modify the instance variable - also called **getter** method
- **Example:**

```
class Test {
        int getId(){
                //getter method return id;
        }
        String getName(){
                //getter method return name;
        }
}
```

## Mutator Method

✧ These method not only reads but also modifies the instance variables - also called **setter** method

**Example:**

```
class Test {
        private int id;
        private String name;
        void setId(int id){//setter method
                this.id = id;
        }
        void setName(String name){
                this.name = name;
        }
}
```

## Passing Object as Parameter

✧ Object can also be passed as a parameter

**Example:**

```
class Square {
        int length;
        int width;
        Square(int l, int b) {
                length = l;
                width = b;
        }

        void area(Square s1) {
                int area = s1.length * s1.width;
                System.out.println("Area = " + area);
        }
}
```

## Passing Array as Parameter

- ✧ We can pass an array as a parameter to a method or a constructor
- ✧ The type of array we pass to the method must be assignment compatible to the formal parameter type
- ✧ The following code shows how to pass an Array as a Method Parameter

## Array as Parameter - Example

```
public class Test {
    public static void main(String[] args) {
        int[] num = { 1, 2 };
        System.out.println("Before swap");
        System.out.println("#1: " + num[0]);
        System.out.println("#2: " + num[1]);

        swap(num);

        System.out.println("After swap");
        System.out.println("#1: " + num[0]);
        System.out.println("#2: " + num[1]);
    }
}
```

# Method – swap()

```
public static void swap(int[] source) {
        if (source != null && source.length == 2) {
                // Swap the first and the second elements



                int temp = source[0];
                source[0] = source[1];
                source[1] = temp;
        }
}
```

## Passing 2D array

```
public class MatrixDiagonal {
        /* computes the sum of left diagonal elments */
        static int sumLeftDiagElements(int mat[][]) {
                int i, j, sum = 0;
                int rows = mat.length; // get the no. of rows
                int cols = mat[0].length; // get the no. of cols
                for ( i = 0; i < rows; i++ ) {
                        for ( j = 0; j < cols; j++ ) {
                                if ( i == j ) { // found a left diag. elem
                                 sum += mat[i][j];
                        }
                }
        }
        return sum;
}
```

## main() Method

```
public static void main(String args[]) {
        int mat[][] = {
                                { 2, 3, 8, 4 },
                                { 5, 1, 7, 3 },
                                { 9, 2, 6, 8 },
                                { 1, 4, 5, 7 }
                        };
        int ld_sum = sumLeftDiagElements(mat);

        System.out.println("Sum = " + ld_sum);
}
```

## 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.

## Method Overloading in Java

Method overloading is also known as Static Polymorphism

Points to Remember:

- ✧ **Static Polymorphism** is also known as **compile time binding** or early binding
- ✧ **Static binding** happens at compile time.
  - ✧ Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

# Static Binding – An 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**

## Method Overloading – Example

```
Class OverLoading {
      void display(int a) {
            System.out.println(a);
      }
      void display(char c) {
            System.out.println(c);
      }
      public static void main(String args[]) {
            OverLoading ol = new OverLoading();
            ol.display('10');
            ol.display(10);
            ol.display('A');
      }
}
```

## Method Overloading – Variation

```
Class OverLoading {
      void display(String name, int size) {
            System.out.println(name + " (" + size + ")" );
      }
      void display(int size, String name) {
            System.out.println(size + " (" + name + ")" );
      }
      public static void main(String args[]) {
            OverLoading ol = new OverLoading();
            ol.display("Vikas Jain", 10);
            ol.display(10, "Rahul Prasad");
      }
}
```

## Method Overloading Possible?

```
class Addition {
    int sum(int a, int b) {
        return a + b;
    }
    double sum(int a, int b) {
        return a + b;
    }
    public static void main(String args[]) {
        Addition add = new Addition();
        int result = add.sum(10, 20);
    }
}
```

Ambiguity!!
Compile Time Error !!!

## Overloading main() method?

- Can we overload the main() Method?
  - Yes. We can overload main() method.
  - A Java class can have any number of main() methods. But to run the java program, the class should have the UNIQUE signature as

    **public void main(String[] args)**
  - Any modification made to this signature will not affect the compilation. But we can not run the program (Again Run time error!!)

## Overloading main() method?

```
public class CodeTester {
    public static void main(String args[]) {
        System.out.println(“Main Method”);
    }
    void main(int args) {
        System.out.println(“First Override Method”);
    }
    double main(int a, double d) {
        System.out.println(“Second Override Method”);
        return d;
    }
}
```

## Types of Variables

- **3 Types of variables**
  - Local Variables
  - Instance Variables
  - Class / Static Variables

- **Scope of Variables**

# Local Variables

## Local Variables

- Declared in methods, constructors or blocks
- Created when the method is invoked and the variable is destroyed when the execution of the method or block is completed.
- Access Modifiers can not be used for local variables
- Local Variables are visible only within the method, constructor or block
- Implemented at Stack level internally
- No default value for local variables. So declare them with suitable initial values

## Local Variables – An example

```
public class CodeTester {

    public static void main(String args[]) {
        int a;     // Declare the varibles
        a = 10;  // Initialize the variables
        System.out.println(“ a = ” + a);
    }

}
```

## Local Variables – An example

```
public class CodeTester {

    public void getAge() {
        int age = 0;
        age = age + 7;
        System.out.println("Age = " + age);
    }
    public static void main(String args[]) {
        CodeTester ct = new CodeTester();
        ct.getAge();
    }
}
```

age is a local variable and its scope is within the method!!

## Static Variables

- Created when the program the program starts and destroyed when the program ends
- There exists only one copy of the class variables per class regardless of creating multiple instances
- Static variables are rarely used other than being constants – public / private / Final / Static
- Constant Variables never change from their initial values
- Static Variables are stored in static memory
- Visibility is similar to instance variables
- Default values are same as instance variables

## Static Variables – An 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

# Recap: Exercises

- **Create Geometric Objects**
  - Perform Basic Operations
  - Apply Transformation
  - Perform getter and setter
  - Extending the Object to Other Shapes

- **Bank Application**
  - Employee
  - Customer
  - ATM
  - Account details
    - Balance enquiry

# Exercise - 1

- **Problem: Sorting 1 million names based on Lexicographic ordering**
  - Generate One Millions Names randomly

    (As for the code o you own mechanisms)
  - T1: Perform lexicographic ordering of these names
    - Clearly describe the approach with the specific data structures with their computational complexity
  - T2: Apply any sorting algorithm that is defined in Collection.sort() Library.
  - Perform Code Profiling of T1 and T2 approaches.
  - Provide a report stating all your attempts highlighting what is the best approach and why is that approach the best approach? Justify your answer.

## Assignments / Penalties

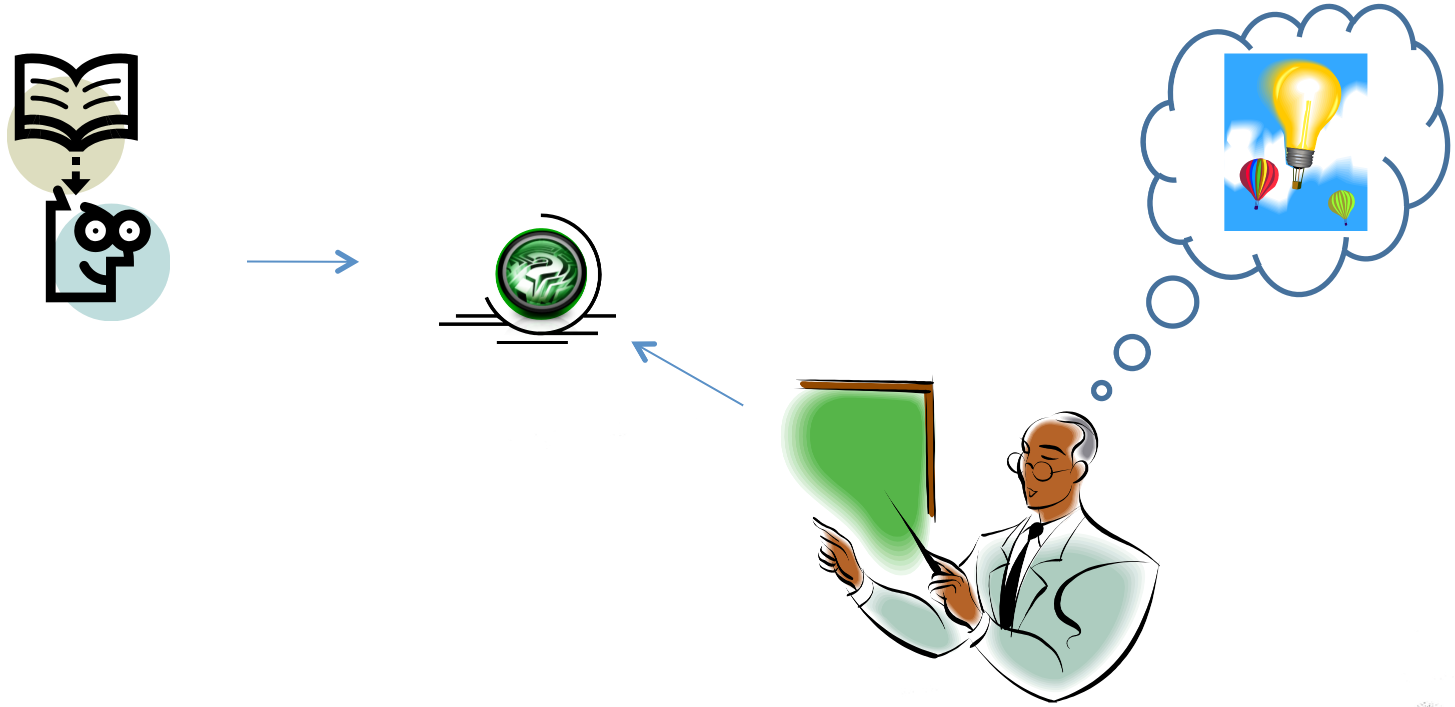
- Every Student is expected to complete the assignments and strictly follow a fair Academic Code of Conduct to avoid severe penalties

- 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 ???