# Java - Methods

Md. Mohsin Uddin

East West University

*mmuddin@ewubd.edu*

April 29, 2019

# Method Overloading : Part I

When a class has two or more methods by the same name but different parameters, it is known as method overloading.

```
public class ExampleOverloading {
    public static void main(String[] args) {
        int a = 11;
        int b = 6;
        double c = 7.3;
        double d = 9.4;
        int result1 = minFunction(a, b);
        // same function name with different parameters
        double result2 = minFunction(c, d);
        System.out.println("Minimum Value = " + result1);
        System.out.println("Minimum Value = " + result2);
    }

    // for integer
    public static int minFunction(int n1, int n2) {
        int min;
```

# Method Overloading : Part II

```
        if (n1 > n2)
            min = n2;
        else
            min = n1;

        return min;
    }

    // for double
    public static double minFunction(double n1, double n2) {
      double min;
        if (n1 > n2)
            min = n2;
        else
            min = n1;

        return min;
    }
}
```

# Method Overloading : Part III

When the above code is compiled and executed, it produces the following result:

```
Minimum Value = 6
Minimum Value = 7.3
```

# Using Command-Line Arguments

- A command-line argument is the information that directly follows the program's name on the command line when it is executed.
- To access the command-line arguments inside a Java program is quite easy.
- They are stored as strings in the String array passed to main( ).

# Using Command-Line Arguments

```
public class CommandLine {
    public static void main(String args[]) {
        for(int i = 0; i<args.length; i++) {
            System.out.println("args[" + i + "]: " +  args[i]);
        }
    }
}
```

Try executing this program as follows:
java CommandLine this is a command line 200 -100
When the above code is compiled and executed, it produces the following result:

```
args[0]: this
args[1]: is
args[2]: a
args[3]: command
args[4]: line
args[5]: 200
args[6]: -100
```

# The **this** keyword

- **this** is a keyword in Java which is used as a reference to the object of the current class, with in an instance method or a constructor.
- Differentiate the instance variables from local variables if they have same names, within a constructor or a method.

# The **this** keyword : Part I

```
public class This_Example {
    // Instance variable num
    int num = 10;
    This_Example() {
        System.out.println("Example program on keyword this");
    }
    This_Example(int num) {
        // Invoking the default constructor
        this();

        // Assigning the local variable num to the instance variab
        this.num = num;
    }

    public void greet() {
        System.out.println("Hi Welcome to CSE110");
    }

    public void print() {
```

# The **this** keyword : Part II

```
        // Local variable num
        int num = 20;

        // Printing the local variable
    System.out.println("local variable num is : "+num);

        // Printing the instance variable
    System.out.println("instance variable num is : "+this.num);

        // Invoking the greet method of a class
        this.greet();
    }

    public static void main(String[] args) {
        // Instantiating the class
        This_Example obj1 = new This_Example();

        // Invoking the print method
        obj1.print();
```

# The **this** keyword : Part III

```
        // Passing a new value to the num variable through parame
        This_Example obj2 = new This_Example(30);

        // Invoking the print method again
        obj2.print();
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
Example program on keyword this local variable num is : 20
instance variable num is : 10
Hi Welcome to CSE110
Example program on keyword this
local variable num is : 20
instance variable num is : 30
Hi Welcome to CSE110
```

# Variable Arguments(var-args) : Part I

```
public class VarargsDemo {
    public static void main(String args[]) {
        // Call method with variable args
              printMax(34, 3, 3, 2, 56.5);
        printMax(new double[]{1, 2, 3});
    }
    public static void printMax( double... numbers) {
        if (numbers.length == 0) {
            System.out.println("No argument passed");
            return;
        }
        double result = numbers[0];
        for (int i = 1; i < numbers.length; i++)
        if (numbers[i] > result)
        result = numbers[i];
        System.out.println("The max value is " + result);
    }
}
```

# Variable Arguments(var-args) : Part II

When the above code is compiled and executed, it produces the following result:

```
The max value is 56.5
The max value is 3.0
```

# References



DEITEL, Java How to Program, 11/e


Java: the complete reference, Herbert Schildt, McGraw-Hill Education Group