Methods

Introduction to Programming I

Lecture 3

Introduction to Methods

Agenda

- Week 3 Lecture Coverage
  - Introduction to Methods
  - Why Use Methods?
  - Method Definition
  - Accessing a Method
  - Pass by value
  - Pass by reference
  - Arguments to Main Method
  - Returning from a Method
  - Pure and Impure Methods
  - Method Overloading

Review of Previous Week

- Introduction to Constructors
- Need for Constructor
- Declaration & Definition of Constructors
- Difference between Constructors & methods
- Types of Constructors
- The *this* keyword
- Constructor Overloadding
- Calling a Constructor Inside Another Constructor

Let’s get started with Lecture 3

An Introduction to **Methods**

Introduction - Methods

- A program is a set of instructions given to computer.
- These instructions initiate some action and hence sometimes called executable instructions.
- In Java programs, the executable instructions are specified through methods or functions.
- Method is a sequence of some declaration statements and executable statements.

Introduction - Methods

- In other programs, methods are known as *functions, procedures, subprograms* or *subroutines*.
- In Java, which is strictly object-oriented, any action can take place through *methods* or *functions.*
- These methods have to exist as a part of a class.

Why Use Methods?

1. To cope with complexity
2. To hide details
3. To reuse

Method Definition

- In Java, a method must be defined before it is used anywhere in the program.
- The general form of a method definition is as given below:

```
 [ access-specifier] [modifier] return-type
method-name(paramter list){
      body of the method
 }
```

Method Definition

- access specifier: can be either **public** or **private** or protected
- modifier: can be one of: **final**, **native, synchronized, transient, volatile**
- return_type: specifies the type of value that the return statement of the method returns. It may be any valid data_type. If no value is being returned, it should be **void**.
- method_name: It has to be valid Java identifier.

Accessing a Method

- A method is *called (or invoked, or executed)* by providing the method name, followed by the parameters being sent enclosed in parentheses.
- For instance, to invoke a function whose prototype looks like

  ```
  float area(float, float);
  ```

  the method call statement may look like this:

  ```
  area(x, y);
  ```

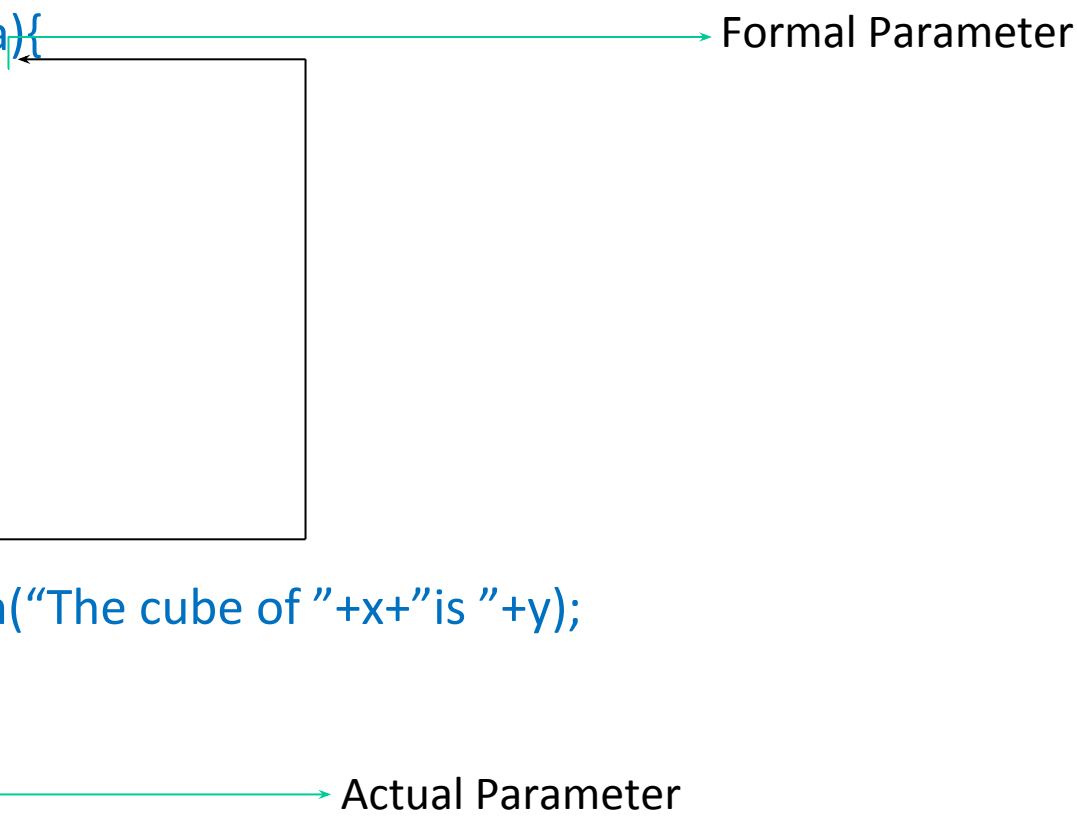
  where *x,y* have to be **float** variables.

Accessing a Method

- Whenever a *method call statement* is encountered, the control is transferred to the function, the statements in the method-body are executed, and then the control returns to the statement following the function call.

Method Call Demo

```
public class First{
    static float cube(float a){
        float n=a*a;
        return n;
    }
    public void firstTest(){
        float x=7.5f, y=0;
        y=cube(x);
        System.out.println("The cube of "+x+"is "+y);
    }
}
```



Any Questions?

Pass By Value(Call By Value)

- The call by value method copies the values of *actual parameters* into the *formal parameters*, that is, the method creates its own copy of argument values and then uses them.
- The previous example can be used as a demo.

Pass By Reference (Call By Reference)

- In call by value, the called method creates a new set of variables and copies the values of arguments into them.
- The method does not have access to the original variables (actual parameters) and can only work on the copies of values it created.
- Passing arguments by value is useful when the original values are not to be modified.

Pass By Reference (Call By Reference)

- Call by reference passes a ***reference*** to the original variable.
- Remember that a *reference* stores a memory location of a variable.
- Provision of the reference variables in Java permits the passing of parameters to methods by reference.

Pass By Reference (Call By Reference)

- When a method is called by reference, then, the *formal parameters* become *references* to the *actual parameters* in the calling method.
- This means that in the call by reference method, the called method does not create its own copy of original values, rather, it refers to the original values only by different names, i.e. the references.

Pass By Reference (Call By Reference)

- Thus the called method works with the original data and any change in the values gets reflected to the data.
- In Java, all primitive types are passed by value and all reference types (objects, arrays) are passed by reference.

Arguments to main Method

- Just as you pass arguments to methods in Java, you can also pass arguments to main method.

  public static void main(String[] args)

- Here, String[] args is an array of strings passed to the main method as an argument.
- Suppose the class is named *MyApplication*, then the program is run from the command linen as

  java MyApplication

Arguments to main Method

- if it is instead run with the command:

  java MyApplication first second "third fourth" fifth

  then the following assignments are made:

  args[0] receives *first*

  args[1] recieves *second*

  args[2] receives third fourth

  args[3] receives *fifth*

Arguments to main Method

- In BlueJ, when you run **main** method, you can type the arguments at the dialog that appears after clicking at **void main(args)** option on the shortcut menu of initial class.
- The arguments to main() method i.e., args[0], args[1], ... etc are String type. To convert them to another type you can use any of the following syntax:
- To convert args[1] to **integer** type, use

```
int ag1 = Integer.parseInt(args[1]);
```

Arguments to main Method

- To convert args[2] to **float** type, use

  float ag2 = Float.parseFloat(args[2]);

- To convert args[0] to **double** type, use

  double ag3 = Double.parseDouble(args[0]);

Returning From A Method

- As invoking a method is important, returning from a method is equally important as it not only terminates the method's execution but also passes the control back to the calling method.
- A method terminates when either a **return** statement is encountered or the last statement in the method is executed.

Returning From A Method

- Generally, a **return** statement is used to terminate a method whether or not it returns a value.
- A method can return only a single value, although it can have multiple return statements.

Returning From A Method

```
public class Try{
    public boolean positive(){
        boolean res;
        res=(x>=0)? true: false;
        return res;
    }
}
```

Pure And Impure Methods

- A *pure* method is the one that takes objects and/or primitive data types as arguments but does not modify the objects.
- The return value of a pure method is either a primitive or a new object created inside the method.
- An *impure* method, on the other hand changes/modifies the state of received object.

Example of Pure Method

```
public static boolean after(Time time1, Time time2)
{
    boolean result;
    result = (time1.hour>time2.hour)?true: false;
    return result;
}
```

It is pure function as it is not modifying arguments.

Example of Impure Method

```
public static void increment(Time time, double secs){
    time.second +=secs;
    time.minute -=60;
    time.hour +=1;
}
```

It is impure function as it is modifying arguments received.


ANY
QUESTIONS
?
www.rekruitin.com
ReKruiTIn.com

Method Overloading

- When several method declarations are specified for a single method name in the same scope, the method name is said to be overloaded.
- In a simplified version, if a class have multiple methods by same name but different parameters, it is known as **Method Overloading**.

Method Overloading

- For example,

  float divide(int a, int b){ ... }

  float divide(float x, float y) { ... }

  that is, **divide()** taking two **int** arguments is different from **divide()** taking two **float** arguments. This is Method Overloading.

Method Overloading

- There are two ways to overload the method in java
  - By changing the data type (earlier example)
  - By changing number of arguments

```
void sum(int a,int b) { ... }
void sum(int a,int b,int c) { ... }
```

Summary: Week 3 Lecture

- Introduction to Methods
- Why Use Methods?
- Method Definition
- Accessing a Method
- Pass by value
- Pass by reference
- Arguments to Main Method
- Returning from a Method
- Pure and Impure Methods
- Method Overloading

What to Expect: Week 3 Lab

- Discussion about the topics covered in the lecture.
- Provide practical questions based on constructor so it is vital that you go through the lecture slide!!


Thank you