Programming Fundamentals I
Java Fundamentals

Nguyen Ngoc Minh

Saigon Institute of Technology
Readings

• Readings for these lecture notes
  • Textbook:
    • Tony Gaddis & Godfrey, Starting Out with Java From Control Structures through Data Structures
  • Reference:
    • Cay S. Horstmann and Gary Cornell, Core Java™

• These lecture notes contain material © Tony Gaddis, and Godfrey Muganda, 2007
Chapter Objectives

Upon completion of this chapter, you should be able to:

• Understand why and when we use method.
• Decompose your program into smaller pieces.
• Understand passing arguments: by value and by reference
Outline

• Introduction to Methods
• Passing Arguments to a Method
• More About Local Variables
• Returning a Value from a Method
• Problem Solving with Methods
Introduction to Methods

• Methods are commonly used to break a problem down into small manageable pieces. This is called *divide and conquer*.

• Methods simplify programs. If a specific task is performed in several places in the program, a method can be written once to perform that task, and then be executed anytime it is needed. This is known as *code reuse*.
Introduction to Methods
void Methods and Value-Returning Methods

• A void method is one that simply performs a task and then terminates.

  System.out.println(“Hi!”);

• A value-returning method not only performs a task, but also sends a value back to the code that called it.

  int number = Integer.parseInt(“700”);
Introduction to Methods

Defining a void Method

• To create a method, you must write a definition, which consists of a header and a body.

• The **method header**, which appears at the beginning of a method definition, lists several important things about the method, including the method’s name.

• The **method body** is a collection of statements that are performed when the method is executed.
Introduction to Methods
Two Parts of Method Declaration

```java
public static void displayMessage()
{
    System.out.println("Hello");
}
```
### Introduction to Methods

**Parts of a Method Header**

<table>
<thead>
<tr>
<th>Method Modifiers</th>
<th>Return Type</th>
<th>Method Name</th>
<th>Parentheses</th>
</tr>
</thead>
<tbody>
<tr>
<td>public static</td>
<td>void</td>
<td>displayMessage</td>
<td>()</td>
</tr>
</tbody>
</table>

```java
public static void displayMessage()
{
    System.out.println("Hello");
}
```
Introduction to Methods

Parts of a Method Header

• Method modifiers
  • public—method is publicly available to code outside the class
  • static—method belongs to a class, not a specific object.

• Return type—void or the data type from a value-returning method

• Method name—name that is descriptive of what the method does

• Parentheses—contain nothing or a list of one or more variable declarations if the method is capable of receiving arguments.
Introduction to Methods
Calling a Method

- A method executes when it is called.
- The main method is automatically called when a program starts, but other methods are executed by method call statements.
  
  ```
  displayMessage();
  ```
- Notice that the method modifiers and the void return type are not written in the method call statement. Those are only written in the method header.
Calling a Method (Example)

```java
public class SimpleMethod {

    public static void main(String[] args) {
        System.out.println("Hello from the main method.");
        displayMessage();
        System.out.println("Back in the main method.");
    }

    private static void displayMessage() {
        System.out.println("Hello from the displayMessage method.");
    }

}```
public class LoopCall {
    public static void main(String[] args) {
        System.out.println("Hello from the main method.");
        for(int i=1; i <5 ; i++)
            displayMessage();
        System.out.println("Back in the main method.");
    }

    private static void displayMessage() {
        System.out.println("Hello from the displayMessage method.");
    }
}
public class DeepAndDeeper {
    public static void main(String[] args) {
        System.out.println("I am starting in main.");
        deep();
        System.out.println("Now I am back in main.");
    }

    private static void deep() {
        System.out.println("I am now in deep.");
        deeper();
        System.out.println("Now I am back in deep.");
    }

    private static void deeper() {
        System.out.println("I am now in deeper.");
    }
}
Introduction to Methods

Documenting Methods

• A method should always be documented by writing comments that appear just before the method’s definition.

• The comments should provide a brief explanation of the method’s purpose.

• The documentation comments begin with /** and end with */.
Outline

• Introduction to Methods
• **Passing Arguments to a Method**
• More About Local Variables
• Returning a Value from a Method
• Problem Solving with Methods
Introduction to Methods
Passing Arguments to a Method

• Values that are sent into a method are called arguments.
  System.out.println(“Hello”);
  number = Integer.parseInt(str);

• The data type of an argument in a method call must correspond to the variable declaration in the parentheses of the method declaration. The parameter is the variable that holds the value being passed into a method.
Introduction to Methods
Passing Arguments to a Method (Example)

```java
public class PassArg {
    public static void main(String[] args) {
        int x = 10;
        System.out.println("I am passing values to displayValue.");
        displayValue(5);
        displayValue(x);
        displayValue(x * 4);
        displayValue(Integer.parseInt("700"));
    }

    private static void displayValue(int num) {
        System.out.println("The value is " + num);
    }
}
```
Introduction to Methods
Passing Arguments to a Method (Example)

displayValue(5);

The argument 5 is copied into the parameter variable num.

public static void displayValue(int num) {
    System.out.println("The value is " + num);
}

The method will display The value is 5
Introduction to Methods
Argument and Parameter Data Type Compatibility

• When you pass an argument to a method, be sure that the argument’s data type is compatible with the parameter variable’s data type.

• Java will automatically perform widening conversions, but narrowing conversions will cause a compiler error.

  ```java
  double d = 1.0;
  displayValue(d);
  ```
The argument 5 is copied into the num1 parameter.

The argument 10 is copied into the num2 parameter.

```java
showSum(5, 10); //NOTE: Order matters!

public static void showSum(double num1, double num2)
{
    double sum; //to hold the sum
    sum = num1 + num2;
    System.out.println("The sum is "+ sum);
}
```
Introduction to Methods
Arguments are Passed by Value

- In Java, all arguments of the primitive data types are passed by value, which means that only a copy of an argument’s value is passed into a parameter variable.
- A method’s parameter variables are separate and distinct from the arguments that are listed inside the parentheses of a method call.
- If a parameter variable is changed inside a method, it has no affect on the original argument.
Arguments are Passed by Value (Example)

```java
public class PassByValue {

    public static void main(String[] args) {
        int number = 99;
        System.out.println("number is " + number);
        changeMe(number);
        System.out.println("number is " + number);
    }

    private static void changeMe(int myValue) {
        System.out.println("I am changing the value ");
        myValue = 0;
        System.out.println("Now the value is " + myValue);
    }
}
```
Introduction to Methods
Passing String Object References to a Method

• Recall that a class type variable does not hold the actual data item that is associated with it, but holds the memory address of the object. A variable associated with an object is called a reference variable.

• When an object, such as a String is passed as an argument, it is actually a reference to the object that is passed.
**Introduction to Methods**

Passing a Reference as an Argument

```java
showLength(name);

public static void showLength(String str)
{
    System.out.println(str + " is " + str.length() + " characters long.");
    str = "Joe"; // see next slide
}
```

Both variables reference the same object

The address of the object is copied into the `str` parameter.
Introduction to Methods
Strings are Immutable Objects

- Strings are immutable objects, which means that they cannot be changed. When the line
  
  \[
  \text{str} = \text{“Joe”};
  \]

  is executed, it cannot change an immutable object, so creates a new object.

  The name variable holds the address of a String object

  \[
  \text{address} \rightarrow \text{“Warren”}
  \]

  The \text{str} variable holds the address of a different String object

  \[
  \text{address} \rightarrow \text{“Joe”}
  \]

- **Example:** [PassString.java](#)
Introduction to Methods

Strings are Immutable Objects (Example)

```java
public class PassString {

    public static void main(String[] args) {
        String name = "Shakespeare";
        System.out.println("In main, the name is " + name);
        changeName(name);
        System.out.println("Back in main, the name is " + name);
    }

    private static void changeName(String str) {
        str = "Harry";
        System.out.println("In changeName, the name is now " + str);
    }
}
```
Introduction to Methods
@param Tag in Documentation Comments

• You can provide a description of each parameter in your documentation comments by using the @param tag.

• General format

  @param parameterName Description

• All @param tags in a method’s documentation comment must appear after the general description. The description can span several lines.
Outline

• Introduction to Methods
• Passing Arguments to a Method
• More About Local Variables
• Returning a Value from a Method
• Problem Solving with Methods
More About Local Variables

- A local variable is declared inside a method and is not accessible to statements outside the method.

- Different methods can have local variables with the same names because the methods cannot see each other’s local variables.

- A method’s local variables exist only while the method is executing. When the method ends, the local variables and parameter variables are destroyed and any values stored are lost.

- Local variables are not automatically initialized with a default value and must be given a value before they can be used.
More About Local Variables (Example)

```java
public class LocalVars {

    public static void main(String[] args) {
        texas();
        california();
    }

    private static void california() {
        int birds = 5000;
        System.out.println("In texas there are " + birds + " birds.");
    }

    private static void texas() {
        int birds = 3000;
        System.out.println("In California there are " + birds + " birds.");
    }

}
```
More About Local Variables (Example)

```java
public class UnInitLocalVar {

    public static void main(String[] args) {
        System.out.println("Result: " + isValid(10));
    }

    public static boolean isValid(int a){
        boolean result;
        if(a > 10)
            result = true;
        //else if(a <= 10)
        //    result = false;
        else
            result = false;
        return result;
    }
}
```

What happen if we replace line 14, 15 by line 12, 13 (uncomment) ?
Outline

• Introduction to Methods
• Passing Arguments to a Method
• More About Local Variables
• Returning a Value from a Method
• Problem Solving with Methods
Returning a Value from a Method

- Data can be passed into a method by way of the parameter variables. Data may also be returned from a method, back to the statement that called it.

  \[
  \text{int num} = \text{Integer.parseInt(“700”);}
  \]

- The string “700” is passed into the parseInt method.
- The int value 700 is returned from the method and stored into the num variable.
Returning a Value from a Method
Defining a Value-Returning Method

```java
public static int sum(int num1, int num2)
{
    int result;
    result = num1 + num2;
    return result;
}
```

The return statement causes the method to end execution and it returns a value back to the statement that called the method.

This expression must be of the same data type as the return type.
Returning a Value from a Method

Calling a Value-Returning Method

total = sum(value1, value2);

public static int sum(int num1, int num2)
{
    int result;
    result = num1 + num2;
    return result;
}
Returning a Value from a Method

Data Type Compatibility

• The return’s statement’s expression must be compatible with the data type specified in the method header.

• Java will automatically widen the value of the return expression, but will not narrow it automatically.
Returning a Value from a Method
@return Tag in Documentation Comments

• You can provide a description of the return value in your documentation comments by using the @return tag.

• General format

  @return Description

• The @return tag in a method’s documentation comment must appear after the general description. The description can span several lines.
Returning a Value from a Method

Returning a booleanValue

Frequently, we need to write methods to test arguments for validity and return true or false.

```java
Public static boolean isValid(int number) {
    boolean status;
    if(number >= 1 && number <= 100)
        status = true;
    else
        status = false;
    Return status;
}
```

Calling code:
```java
int value = 20;
If(isValid(value))
    System.out.println("The value is within range");
else
    System.out.println("The value is out of range");
```
Returning a Value from a Method
Returning a Reference to a String Object

customernname = fullName(“John”, “Martin”);

public static String fullName(String first, String last)
{
    String name;
    name = first + “ “ + last;
    return name;
}

Example:  ReturnString.java

Local variable name holds the reference to the object. The return statement sends a copy of the reference back to the call statement and it is stored in customernname.
Outline

• Introduction to Methods
• Passing Arguments to a Method
• More About Local Variables
• Returning a Value from a Method
• Problem Solving with Methods
Problem Solving with Methods

- A large, complex problem can be solved a piece at a time by methods.
- The process of breaking a problem down into smaller pieces is called *functional decomposition*.
- If a method calls another method that has a throws clause in its header, then the calling method should have the same throws clause.
Example: How to implement a program based on following brief pseudo-code model of algorithm:

*Ask the user to enter the name of the file*

*Get the total of the sales amounts in the file.*

*Calculate the average daily sales*

*Display the total and average daily sales*
Calling Methods that Throw Exceptions

• Note that the main() and getTotalSales() methods in SalesReport.java have a throws IOException clause.
• Java requires all methods that use a BufferedReader object to throw or handle IOException.
• You will learn how to handle exceptions in Chapter 12.
• For now, understand that Java required any method that interacts with an external entity, such as the file system to either throw an exception to be handled elsewhere in your application or to handle the exception locally.