CSEN 202 – Introduction to Computer Programming

Lecture 4: Methods

Prof. Dr. Slim Abdennadher,
Dr. Wael Abouelsaadat and
Dr Mohammed Abdel Megeed Salem
slim.abdennadher@guc.edu.eg

German University Cairo, Faculty of Media Engineering and Technology

March 4 - March 9, 2017

Control-flow constructs

- Conditional
  - `if`-statement,
  - `switch`-statement,
  - conditional expression.
- Iterative
  - `while`-loop,
  - `do-while`-loop,
  - `for`-iteration

Today's topic

methods

The first java program

```
public class Hello {

  public static void main(String[] args) {
    // display a greeting in the console window
    System.out.println("Hello, World!");
  }

}
```

- This code defines a class named `Hello`.
- The method `main` is the code that runs when you execute the program

Program complexity

- As programs become more complex programmers must structure programs in such a way as to effectively manage the complexity.
- The trick to managing complexity is to break down the problem into more manageable pieces.
- The problem is ultimately solved by putting these pieces together to form the complete solution.

Monolithic code

- As the number of statements with a method increases, the method can become unmanageable.
- The code within such a method that does all the work by itself is called monolithic code.
- Monolithic code that is long and complex is undesirable:
  - It is difficult to write correctly.
  - It is difficult to debug.
  - It is difficult to extend.

Divide and conquer

- A programmer can decompose a complicated method into several simpler methods.
- The original method can then do its job by delegating its work to these other methods.
- The original method can be thought as a "work coordinator".
- Advantages:
  - Methods bundle functionality into reusable parts.
  - The same method may be used in numerous places within a program
  - If the method is written properly, it may be able to be reused in other programs as well.

Terminology of sub-routines

We distinguish:

- Procedure (sub-routine). A coherent, closed, part of a program that provides specific, reusable, functionality.
- Function. A procedure with zero or more inputs (arguments) and zero or one output (return value)
- Method. A function that is associated with a class or object and in general has side effects on the class or object. We distinguish
  - Class methods, and
  - Instance methods (explained later)

Example

```
public class Hello {

  public static void main(String[] args) {
    // display a greeting in the console window
    System.out.println("Hello, World!");
  }

}
```

- The method `main` is a class method (keyword: **`static`**)

The two sides of a method

There are two aspects to every Java method:

- Method Definition: The definition of a method provides the code that determine the method's behavior
- Method Invocation: A method is used within a program via a method invocation.

Every method has exactly one definition but may have many invocations.

Later in this semester we might talk about method declarations

Method definition

```
public static type name (parameter list) {
   Method body
}
```

- The access specifier **`public`** denotes the visibility of the method (to be treated later)
- The reserved word **`static`** denotes that the method is a class method
- The type indicates the type of the value that the method returns. The reserved word **`void`** indicates that the method does not return a value.

Method definition

```
public static type name (parameter list) {
   Method body
}
```

- The name is an identifier
- The parameter list is a comma separated list of pairs of the form:
  type name
  where type is recognized Java type (like **`int`**, **`double`**, `String`, *etc.*) and name is an identifier representing a parameter; the parameter list may be empty
- The Method body contains the code that defines the actions of the method.

Communicating with methods

The principle is similar to a mathematical function:

$$f\ \overbrace{(a, b)}^{\text{parameters}} = \underbrace{\sqrt{a^2 + b^2}}_{\text{“body”}}$$

- Parameters communicate information into methods.
- Parameters are a way to hand over values to a method.
- The formal parameters—the variables declared in the method header—are assigned the values of
- the actual parameters, (*i. e.*, the values provided to the message sent).

Communicating with methods

The principle is similar to a mathematical function:

$$f\ \overbrace{(a, b)}^{\text{parameters}} = \underbrace{\sqrt{a^2 + b^2}}_{\text{“body”}}$$

- One piece of information can be communicated back in the form of a return value.

A simple class

```
public class Greeter {
  public String sayHello () {
    String message = "Hello, World!";
    return message;
  }
}
```

- An access specifier: **`public`**
- The return type of the method: `String`
- The name of the method: `sayHello`
- A list of parameters of the method, enclosed in parentheses: `sayHello` method has no parameters
- The body of the method: a sequence of statements enclosed in braces

Another example

- Method definition:

```
public class Square {
  public static int square (int x) {
    x *= x;
    return x;
  }
```

- Method invocation:

```
  public static void main (String[] args) {
    System.out.println (square (4));
    System.out.println (square (12));
  }
}
```

Discussion

- Once a method has been defined it can be used.
- A method is invoked or called.
- The calling code passes the necessary parameters required by the method.
- At the time of the method invocation the values of the actual parameters are assigned to the corresponding formal parameters.

Methods invocation example: Palindrome

```
public class Palindrome {
  public static void main (String[] args) {
    System.out.println (reverse ("GUC"));
    System.out.println (palindrome ("rats␣live␣on␣no␣evil␣star"));
  }
  public static String reverse (String s) {
    String r = "";
    for (int i = s.length () - 1; i >= 0; i--)
      r += s.charAt (i);
    return r;
  }
  public static boolean palindrome (String s) {
    String r = reverse (s);
    return s.equals (r);
} }
```

Passing by value: swap-method

```
import java.util.Scanner;

public class Swapper {
  public static void main (String[] args) {
    Scanner sc = new Scanner (System.in) ;
    System.out.print ("Enter a: ") ;
    int a = sc.nextInt ();
    System.out.print ("Enter b: " ) ;
    int b = sc.nextInt ();
    swap (a, b);
    System.out.println ("a: " + a + "\n" + "b: " + b);
  }
  public static void swap (int x, int y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
} }
```

Passing by value: swap-method

```
  public static void swap (int x, int y) {
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
} }
```

- You may think that this method swaps `a` and `b` but actually it doesn't. After the termination of the call, `a` remains with value 2 and b with value 3.
- If a variable is passed by value to a specific function then only a copy of the variable is passed, which means the original variable does not change after the call is terminated!

Parameters: call by value

- The formal parameters act as placeholders for the values that are passed as arguments when the method is invoked.
- Each parameter has a type and a name which is used in the method body.
- When the method is invoked, the number of arguments must match the number parameters and each argument must have a type compatible to the declared type of the parameter.
- All arguments passed call-by-value: the method receives and works on the value of the argument, not on its address in memory.
- Thus, a method call cannot change the value of a variable used as argument.

Overloaded methods

- In Java, a class can have multiple methods with the same name.
- When two or more methods in a class have the same name, the method is said overloaded.
- The methods must be different somehow, or else the compiler would not associate a call to a particular method definition.

The method signature

- The compiler identifies a method by more than its name.
- A method is uniquely identified by its signature.
- A method signature consists of
  - the method's name and
  - its parameter list
- If the parameter types do not match exactly, both in number and position, then the method signatures are different.

Example

`System.out.println (int)` —prints a number
`System.out.println (String)` —prints text

Example: Overloading

1. `static void f() { ... }`
   This version has no parameters, so its signature differs from all the others which each have at least one parameter.
2. `static void f(int x) { ... }`
   This version differs from version 3, since its single parameter is an int, not a double.
3. `static void f(double x) { ... }`
   This version differs from version 2, since its single parameter is a double, not an int.

Example: Overloading

4 `static void f(int x, double y) {...}`
This version differs form version 5 because, even though versions 4 and 5 have the same number of parameters with the same types, the order of the types is different.

5 `static void f(double x, int y) {...}`

Next week's topics

- Recursion