

CSC 125

Object Oriented Programming

Ch06_Methods

Dr. Fadi Alzhouri



What is a Method?

- A set of statements to perform a specific task.
- A block of code which only runs when it is called.
- aka: **functions**, modules, **procedures**, subroutine, or subprograms)
- **Why?**: To encapsulate functionality and promote code reuse.
 - It is a technique of writing a piece of code once and using it multiple times.
 - Avoiding repetition
- Programs can be quite large; we need to break them down into smaller functions (methods).
- Methods call other methods to complete specific tasks.

A view of methods

```
public class Main
{
    public static int add(int a, int b) {
        // Method 1 body
    }

    public static int subtract(int a, int b) {
        // Method 2 body
    }

    public static void main(String[] args) {
        Statement 1;
        int sum1 = add(num1, num2); // Calling method 1
        Statement 3;
        int diff = subtract(num1, num2); // Calling method 2
        Statement 5;
        int sum2 = add(num2, num3); // Calling method 1
    }
}
```



A view of methods: Example 1

- Exercise:
- Write a program to find the sum of integers from 1 to 10, 20 to 37, and 35 to 49, respectively.

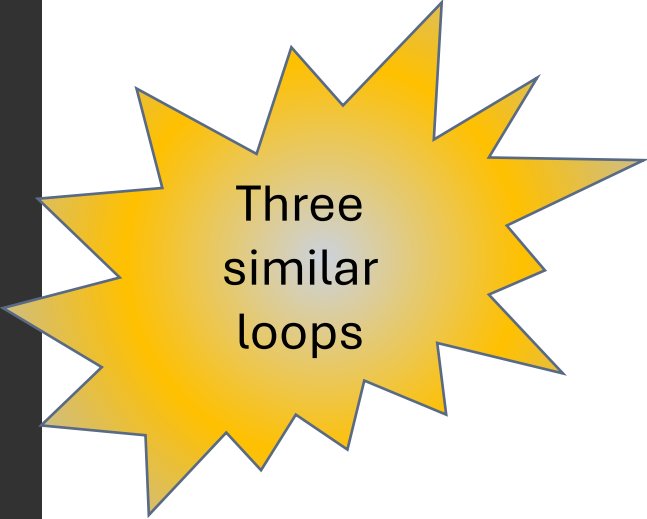
A view of methods: Example 1 (cont.)

```
public class SumOfIntegers {  
    public static void main(String[] args) {  
  
        int sum1 = 0;  
        for (int i = 1; i <= 10; i++) {  
            sum1 += i;  
        }  
        System.out.println("Sum of integers from 1 to 10: " + sum1);  
  
        int sum2 = 0;  
        for (int i = 20; i <= 37; i++) {  
            sum2 += i;  
        }  
        System.out.println("Sum of integers from 20 to 37: " + sum2);  
  
        int sum3 = 0;  
        for (int i = 35; i <= 49; i++) {  
            sum3 += i;  
        }  
        System.out.println("Sum of integers from 35 to 49: " + sum3);  
    }  
}
```

```
Sum of integers from 1 to 10: 55  
Sum of integers from 20 to 37: 513  
Sum of integers from 35 to 49: 630
```

A view of methods: Example 1 (cont.)

```
public class SumOfIntegers {  
    public static void main(String[] args) {  
  
        int sum1 = 0;  
        for (int i = 1; i <= 10; i++) {  
            sum1 += i;  
        }  
        System.out.println("Sum of integers from 1 to 10: " + sum1);  
  
        int sum2 = 0;  
        for (int i = 20; i <= 37; i++) {  
            sum2 += i;  
        }  
        System.out.println("Sum of integers from 20 to 37: " + sum2);  
  
        int sum3 = 0;  
        for (int i = 35; i <= 49; i++) {  
            sum3 += i;  
        }  
        System.out.println("Sum of integers from 35 to 49: " + sum3);  
    }  
}
```

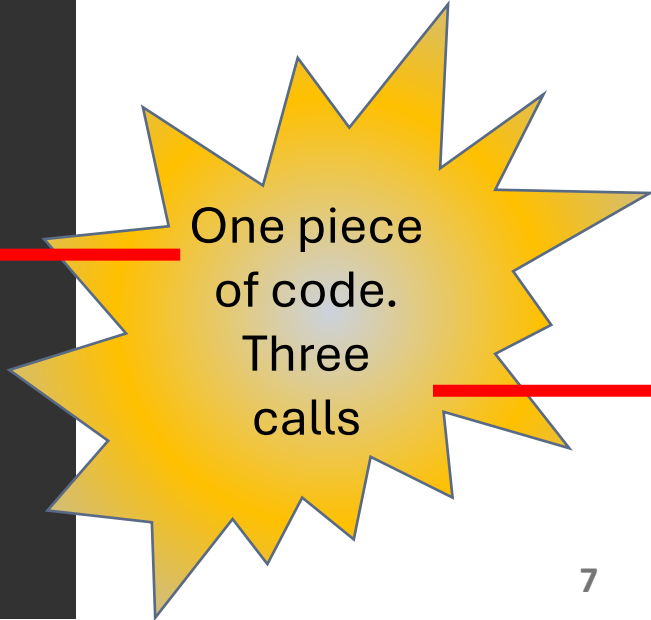


Three
similar
loops

A view of methods: Example 1 (cont.)

```
public class SumOfIntegers {  
    public static void main(String[] args) {  
  
        int sum1 = sumRange(1, 10);  
        int sum2 = sumRange(20, 37);  
        int sum3 = sumRange(35, 49);  
  
        System.out.println("Sum of integers from 1 to 10: " + sum1);  
        System.out.println("Sum of integers from 20 to 37: " + sum2);  
        System.out.println("Sum of integers from 35 to 49: " + sum3);  
    }  
}
```

```
public static int sumRange(int start, int end) {  
    int sum = 0;  
    for (int i = start; i <= end; i++) {  
        sum += i;  
    }  
    return sum;  
}
```



One piece
of code.
Three
calls

Using Methods

- Method's motivations?
- **Divide and Conquer**: Divide the problem into smaller pieces, and you conquer the complexity of the problem.
- **Reusability**: Can be used in more than one place in a program or in different programs.
- **Simplicity**: Simplify code maintenance.

Methods Types

1) **Predefined** Methods (Built-in)

- Methods that the Java **API** provides.
- Examples:
 - `System.out.println()`
 - `String.length()`

2) **User-Defined** Methods

- Void Methods (nonvalue-returning): Methods that **do not return** a value.
- Non-Void Methods (Value-returning): Methods that **return** a value.
 - 1) have a data type
 - 2) return only one value (thing) to the caller

Predefined Methods (Built-in)

- They simplify programming by offering commonly used functionalities.
- Example I: **Math methods**
 - The Math class provides methods for performing basic numeric operations.

Method	Description
<code>exp(double a)</code>	Returns Euler's number e raised to the power of the specified double value.
<code>pow(double a, double b)</code>	Returns the value of the first argument raised to the power of the second argument.
<code>sqrt(double a)</code>	Returns the square root of the specified double value. If the value is negative, NaN is returned.
<code>ceil(double a)</code>	Returns the smallest (closest to negative infinity) double value that is greater than or equal to the argument and is equal to a mathematical integer.

Predefined Methods (Built-in) (cont.)

Method	Description
<code>floor(double a)</code>	Returns the largest (closest to positive infinity) double value that is less than or equal to the argument and is equal to a mathematical integer.
<code>round(double a)</code>	Returns the closest integer to the argument, rounding up if the fractional part is 0.5 or greater.
<code>random()</code>	Returns a double value greater than or equal to 0.0 and less than 1.0, representing a pseudo-random number.

Exercise:

```
public class Main
{
    public static void main(String[] args) {

        System.out.println(Math.pow(2.0, 3.0));

        double sqrtResult = Math.sqrt(16.0);
        System.out.println("Math.sqrt(16.0) = " + sqrtResult);

        long roundResult = Math.round(3.5);
        System.out.println("Math.round(3.5) = " + roundResult);

        System.out.println(Math.random());
    }
}
```

```
8.0
Math.sqrt(16.0) = 4.0
Math.round(3.5) = 4
0.8661053575567472
```

Predefined Methods (Built-in) (cont.)

- Example I: **String methods**
 - The String class provides methods for manipulating strings.
 - We discussed several String methods in Chapter 5.
 - Such as:
 - `length()`
 - `charAt()`

```
String text = "Welcome to Java";  
System.out.println(text.length());  
System.out.println(text.charAt(3));
```



```
15  
c
```

User-Defined Methods

- To create your own methods (functions) follow this syntax:

```
accessModifier returnType methodName (parameter list) {  
    // Method body  
    // Code to be executed  
    return value; // (optional, if returnType is not void)  
}
```

Method header

Method body

- Parameter list: Comma-separated list of parameters
 - Data type needed for each parameter
 - If no parameters, leave a blank

Invoke (call) a Method

- Two parties are involved in using a method: a **caller** who calls the method, and the **method** called (aka. callee).
- The **caller** passes **argument**(s) to the method (function).
- The method receives these argument(s) and assigns them to the **parameters**.
- Executes instructions in the method's body.
- The **callee** returns a piece of result back to the **caller** (in case of non-void methods).

Invoke (call) a Method (cont.)

- How to invoke (call) the method.

```
int x=3;  
System.out.println(pow2(x));
```

Call the method
X is argument

```
public static int pow2(int y) {  
    return y*y;  
}
```

Define the function
Y is parameter

Returns data and control goes to the function's caller

Invoke (call) a Method (cont.)

- When you call a method, the arguments you provide must match the method's parameters in the following ways:
 1. **Order**: The arguments must be in the same order as the parameters defined in the method signature.
 2. **Number**: The number of arguments must match the number of parameters.
 3. **Compatible Type**: The types of arguments must be compatible with the types of the parameters.

Exercise 1: non-void method

```
/**
 * *****
 * Methods
 * Author: Dr. Fadi Alzhouri
 * Example 1: invoke (call) a method
 * *****
 */
public class Main
{
    public static void main(String[] args) {
        int i=5 , j=2;
        int m=max(i,j);
        System.out.println("The maximum number is "+ m);
    }


    public static int max(int i, int j){
        if(i > j)
            return i;
        else
            return j;
    }
}
```

Non-void method

The maximum number is 5

Exercise 2: void method

```
/* ****  
Methods  
Author: Dr. Fadi Alzhouri  
Example 2: void method  
**** */  
public class Main  
{  
    public static void main(String[] args) {  
        int i=5 , j=2;  
        max(i,j);  
    }  
  
    public static void max(int i, int j){  
        if(i > j)  
            System.out.println("The maximum number is "+ i);  
        else  
            System.out.println("The maximum number is "+ j);  
    }  
}
```



The maximum number is 5

Return:

- A **return** statement is not needed for a **void** method, but it can be used for **terminating** the method and returning to the method's caller.

```
public static void main(String[] args) {  
    double x=-4;  
    System.out.println("Enter positive number: " + x);  
    check(x);  
}  
  
public static void check(double i){  
    if(i >0 )  
        System.out.println(Math.sqrt(i));  
    else  
        return;  
}
```

It doesn't return any value, it just terminates the method.

Pass by value (copy)

- Pass-by-Value is used for primitive-type arguments.
- All primitive data types (like int, char, float, etc.) are passed by value.
- When an argument is passed into a method, a **copy** is created and passed into the parameters' method.
- The **invoked** method works on the **cloned copy**, and **cannot modify** the original copy.

References

- **Introduction to Java Programming, Brief Version, Global Edition, 11th edition**, Published by Pearson (June 21, 2018) © 2018