Methods

CSE 114, Computer Science 1
SUNY Korea

Pradipta De
pradipta.de@sunykorea.ac.kr

Acknowledgement for the slides: Dr. Paul Fodor (CS Stony Brook)
Opening Problem

Find multiple sums of integers:
- from 1 to 10,
- from 20 to 30,
- from 35 to 45,
...
Opening Problem

```java
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```
public static int sum(int i1, int i2) {
    int sum = 0;
    for (int i = i1; i <= i2; i++)
        sum += i;
    return sum;
}

public static void main(String[] args) {
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 30 is " + sum(20, 30));
    System.out.println("Sum from 35 to 45 is " + sum(35, 45));
}
Why write methods?

- To shorten your programs
  - avoid writing identical code twice or more
- To modularize your programs
  - fully tested methods can be trusted
- To make your programs more:
  - readable
  - reusable
  - testable
  - debuggable
  - extensible
  - adaptable
Rule of thumb

- If you have to perform some operation in more than one place inside your program, make a new method to implement this operation and have other parts of the program use it.
Defining Methods

A method is a collection of statements that are grouped together to perform an operation.

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

**Method Header:**
- **Modifier:** `public`
- **Return Value Type:** `int`
- **Method Name:** `max`
- **Formal Parameters:** `(int num1, int num2)`

**Method Body:**
- **Variable:** `result`
- **Control Flow:** if-else
- **Statements:**
  - `if (num1 > num2)`
  - `result = num1;`
  - `else`
  - `result = num2;`
- **Return Statement:** `return result;`

**Invoke a Method:**
- **Expression:** `int z = max(x, y);`
- **Actual Parameters:** `(x, y)`
method signature is the combination of the method name and the parameter list.
Formal Parameters

The variables defined in the method header are known as **formal parameters**.

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
```

**Invoking a method**

```
int z = max(x, y);
```
Actual Parameters

When a method is invoked, you pass a value to the parameter: actual parameter or argument.

```
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) 
        result = num1;
    else 
        result = num2;
    return result;
}
```

```
int z = max(x, y);
```
A method may return a value.

The **returnValueType** is the data type of the value the method returns.

If the method does not return a value, the **returnValueType** is the keyword **void**.
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);
  System.out.println(
    "The maximum between " + i + 
    " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
  int result;
  if (num1 > num2)
    result = num1;
  else
    result = num2;
  return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + ", "+ j + " is "+ k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) result = num1;
    else result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) 
        result = num1;  
    else 
        result = num2;  
    return result;  
}
Trace Method Invocation

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
```

Reduction:
1. **Declare variable `result`**
2. Define it in `max` method
3. Use `result` in `main` method
Trace Method Invocation

```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
```

(num1 > num2) is true since num1 is 5 and num2 is 2
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}

class Solution {
    public static int max(int num1, int num2) {
        int result;
        if (num1 > num2) {
            result = num1;
        } else {
            result = num2;
        }
        return result;
    }
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+i+" and "+j+" is "+k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
public static void main(String[] args) {  
  int i = 5;  
  int j = 2;  
  int k = max(i, j);  
  System.out.println("The maximum between " + i + " and " + j + " is " + k);  
}

// Method definition
public static int max(int num1, int num2) {  
  int result;  
  if (num1 > num2)  
    result = num1;  
  else  
    result = num2;  
  return result;  
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between \" + i + " and \" + j + \" is \" + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
Return value in value returning method

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else if (n < 0)
        return -1;
}
```

Java Compiler shows error

Correct Code
CAUTION: all execution paths are required for a value-returning method.

The method shown below has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else if (n < 0)
        return -1;
}
```

To fix this problem, delete `if (n < 0)` in (a), so that the compiler will see a `return` statement to be reached regardless of how the `if` statement is evaluated.

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else
        return -1;
}
```
Method Abstraction

API = the method body is a black box that contains the detailed implementation for the method.
Benefits of Methods

• Write a method once and reuse it anywhere.
• Information hiding:
  • Hide the implementation from the user.
• Reduce complexity.
Generate Javadoc for your project in Eclipse with:

1. Project -> Generate Javadoc
2. In the "Javadoc command" field - browse to find javadoc.exe
   - On the computers in the lab that is C: \ Program Files \ Java \ jdk1.8.0 \ bin \ javadoc.exe
   - On other computers it would be <path_to_jdk_directory>\bin\javadoc.exe
3. Check the box next to the project/package/file for which you are creating the javadoc
4. In the "Destination" field browse to find the desired destination (for example, the doc directory of the current project).
5. Leave everything else as it is.
6. Click "Finish" and open "index.html"
Class pattern

java.lang.Object

pattern

extends java.lang.Object

Constructor Summary

Constructors

Constructor and Description

pattern()
Methods are executed using a stack data structure.

<table>
<thead>
<tr>
<th>(a) The main method is invoked.</th>
<th>(b) The max method is invoked.</th>
<th>(c) The max method is being executed.</th>
<th>(d) The max method is finished and the return value is sent to k.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Space required for the main method</strong></td>
<td><strong>Space required for the max method</strong></td>
<td><strong>Space required for the main method</strong></td>
<td><strong>Space required for the main method</strong></td>
</tr>
<tr>
<td>k: 2</td>
<td>num2: 2</td>
<td>result: 5</td>
<td>k: 5</td>
</tr>
<tr>
<td>i: 5</td>
<td>num1: 5</td>
<td>num1: 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>num2: 2</td>
<td>j: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num1: 5</td>
<td>i: 5</td>
</tr>
<tr>
<td>Stack is empty</td>
<td>Space required for the main method</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>k: 2</td>
<td>j: 5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i: 5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Methods are executed using a stack data structure.
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) 
        result = num1;
    else 
        result = num2;
    return result;
}
The main method is invoked.

j: 2
i: 5

j is declared and initialized

public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) 
        result = num1;
    else 
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```java
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```

(num1 > num2) is true

Space required for the main method

<table>
<thead>
<tr>
<th>k:</th>
<th>result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>j: 2</td>
<td>num2: 2</td>
</tr>
<tr>
<td>i: 5</td>
<td>num1: 5</td>
</tr>
</tbody>
</table>

The max method is invoked.

result:
num2: 2
num1: 5
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);

    System.out.println(
        "The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println(
        "The maximum between " + i + " and " + j + " is " + k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between "+ i + " and "+ j + " is "+ k);
}

public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
Call-by-value

• Method arguments are *copies of the original data*.

• **Consequence?**
  • methods cannot assign (“=” new values to arguments and affect the original passed variables.

• **Why?**
  • changing argument values changes the copy, not the original.
The main method is invoked

The values of num1 and num2 are passed to n1 and n2. Executing swap does not affect num1 and num2.

The swap method is invoked

Space required for the swap method

temp:
n2: 2
n1: 1

Space required for the main method

num2: 2
num1: 1

The swap method is finished

Space required for the main method

num2: 2
num1: 1

The main method is finished

Stack is empty

Call-by-value
public class Test {
    public static void main(String[] args) {
        nPrintln(5, "Welcome to Java!");
    }

    public static void nPrintln(String message, int n) {
        for (int i = 0; i < n; i++)
            System.out.println(message);
    }
}
Method Overloading

- Use the same name for multiple methods
- Use different signatures for methods with same name
- Ex:
  1. `public static int max(int num1, int num2)`
  2. `public static double max(double num1, double num2)`

- The method “max” called with `int` parameters invokes `max(int num1, int num2)`
- The method “max” called with `double` parameters invokes `max(double num1, double num2)`

- Overloaded methods must have different parameter lists.
- The Java compiler finds the method that best matches a method invocation.
Ambiguous Invocation

• Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
• This is referred to as ambiguous invocation.
• Ambiguous invocation is a compilation error.
public class AmbiguousOverloading {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }

    public static double max(int num1, double num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }

    public static double max(double num1, int num2) {
        if (num1 > num2)
            return num1;
        else
            return num2;
    }
}
A local variable: a variable defined inside a method.

Scope: the part of the program where the variable can be referenced.

The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

You can declare a local variable with the same name multiple times in different non-nesting blocks in a method, but you cannot declare a local variable twice in nested blocks.
Scope of Local Variables

- A variable declared in the initial action part of a for loop header has its scope in the entire loop.
- A variable declared inside a for loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

```java
public static void method1() {
    for (int i = 1; i < 10; i++) {
        int j;
    }
}
```

The scope of `i` is limited within the for loop body.
The scope of `j` is within the block that contains it.
// Fine with no errors
public static void correctMethod() {
    int x = 1;
    int y = 1;
    // i is declared
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    // i is declared again
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
Scope of Local Variables

It is fine to declare i in two non-nesting blocks

```java
public static void method1() {
    int x = 1;
    int y = 1;

    for (int i = 1; i < 10; i++) {
        x += i;
    }

    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```

It is wrong to declare i in two nesting blocks

```java
public static void method2() {
    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++) {
        sum += i;
    }
}
```
// With no errors
public static void incorrectMethod() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        int x = 0;
        x += i;
    }
}

Scope of Local Variables