Methods

CSE 114, Computer Science 1
Stony Brook University
http://www.cs.stonybrook.edu/~cse114
Opening Problem

Find multiple sums of integers:
- from 1 to 10,
- from 20 to 30,
- from 35 to 45,
...
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.
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;
}
```

Define a method

- **Modifier**
- **Return value**
- **Type**
- **Method name**
- **Formal parameters**

Invoke a method

```java
int z = max(x, y);
```
**Method signature** is the combination of the method name and the parameter list.

```
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);
```
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;
}
```

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

- **Modifier**: The access level of the method (public, static, int).
- **Return Value**: The type of the method's return value.
- **Method Name**: The name of the method.
- **Formal Parameters**: The variables defined in the method header.
- **Method Signature**: The combination of the method name and its formal parameters.
- **Parameter List**: The list of actual parameters passed to the method.
- **Method Body**: The code block that defines the method's behavior.

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
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);
```
Return Value Type

A method may return a value.

The return ValueType is the data type of the value the method returns.

If the method does not return a value, the return ValueType is the keyword void.

```
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);
```
Calling Methods

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 "+ i + " is "+ k);
}

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

pass the value of i

pass the value of j
public static void main(String[] args) {
    int i = 5;
    int i = 2;
    int k = max(i, i);

    System.out.println("The maximum between " + i + " and " + i + " 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

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 " + i + " 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 " + i + " 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 i = 2;
    int k = max(i, i);

    System.out.println("The maximum between " + i + ", " + i + " 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);
}

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 i = 2;
    int k = max(i, i);
    System.out.println("The maximum between " + i + " and " + i + " 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 " + i + " is " + k);
}

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

return result, which is 5
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 " + i + " is " + k);
}
```

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

return max(i, j) and assign the return value to k
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;
}
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"
public class pattern
extends java.lang.Object

Constructor Summary

Constructors

Constructor and Description

pattern()
A return statement is 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;
}
```

(a) Should be

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

(b)

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.
Methods are executed using a stack data structure.

(a) The main method is invoked.

(b) The max method is invoked.

(c) The max method is being executed.

(d) The max method is finished and the return value is sent to k.

(e) The main method is finished.
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 i = 2;
    int k = max(i, i):
    System.out.println("The maximum between " + i + " and " + i + " is " + k):
}
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 " + i + " is " + k);
}

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

```
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 " + i + " is " + k);
}

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

Invoke max(i, j)

The main method is invoked.

Space required for the main method
- k: 
- j: 2
- i: 5

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
Trace Call Stack

```
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:
}
```

pass the values of i and j to num1 and num2

num1: 5
num2: 2

Space required for the main method

k: 5
j: 2
i: 5

The max method is invoked.
Trace Call Stack

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;
}

pass the values of i and j to num1 and num2

The max method is invoked.

Space required for the main method

result:
num2: 2
num1: 5

k:
j: 2
i: 5

(c) Pearson Education, Inc. & Paul Fodor (CS Stony Brook)
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
public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, i):

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

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

    return result:
}
The max method is invoked.

Space required for the max method:
- result: 5
- num2: 2
- num1: 5

Space required for the main method:
- k: 5
- j: 2
- i: 5

The max method is invoked.

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

    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 Call Stack

```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:
}
```

Execute print statement

Space required for the main method
- k: 5
- j: 2
- i: 5

The main method is invoked.
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 values of num1 and num2 are passed to n1 and n2. Executing swap does not affect num1 and num2.
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;
    }
}
Scope of Local Variables

• 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`  
The scope of `j`
// 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;
    }
}
Static methods

• Remember the main method header?
  
  public static void main(String[] args)

• What does static mean?
  
  • associates a method with a particular class name
  
  • any method can call a static method either:
    
    • directly from within same class OR
    
    • using class name from outside class
Generating Random Characters

(int)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1)

all numeric operators can be applied to the char operands

The char operand is cast into a number if the other operand is a number or a character. So, the preceding expression can be simplified as follows:

'a' + Math.random() * ('z' - 'a' + 1)

So a random lowercase letter is

(char)('a' + Math.random() * ('z' - 'a' + 1))