Methods

Introduction to Programming and Computational Problem Solving - 2
CSE 8B
Lecture 6
Announcements

• Assignment 2 is due today, 11:59 PM
  – Upgrade beginning Apr 22, 12:01 PM
• Assignment 3 will be released today
  – Due Apr 26, 11:59 PM
• Educational research study
  – Apr 21, weekly survey
Variable and method names

• Naming convention: Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name
  – For example, the variables $radius$ and $area$, and the method $computeArea$. 
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;
}
```

int z = max(x, y);
Method signature

• The *method signature* is the combination of the method name and the parameter list.

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

Define a method

<table>
<thead>
<tr>
<th>method header</th>
<th>return value type</th>
<th>method name</th>
<th>formal parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifier</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Invoke a method

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

<table>
<thead>
<tr>
<th>method signature</th>
<th>actual parameters (arguments)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CSE 8B, Spring 2023 5
Formal parameters

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

• When a method is invoked, you pass a value to the parameter
  – This value is referred to as actual parameter or argument
Pass by value

• Java uses **pass by value** to pass arguments to a method

• For example, modifying `num1` does not modify `x`

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

```java
int z = max(x, y);
```
Return value type

• A method may return a value
• The *return value type* is the data type of the value the method returns
  – If the method does not return a value, the *return value type* is the keyword *void*

```java
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 return statement is required for a 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;
}
```

Delete `if (n < 0)` in (a), so the compiler will see a return statement is reached regardless of how the `if` statement is evaluated.
Reuse methods from other classes

• One of the benefits of methods is for reuse
  – Call (i.e., invoke) a static method using ClassName.methodName

• Calling a method executes the code in the method
Instance methods vs static methods

• Instance methods can only be invoked from a specific instance of a class
  – The syntax to invoke an instance method is
    referenceVariable.methodName(arguments)
  – For example, the simple String methods
    String message = "Welcome to Java";
    int messageLength = message.length(); // Not String.length()

• Static methods can be invoked without using an object (i.e., they are not tied to a specific instance)
  – All static methods are non-instance methods
  – The syntax to invoke a static method is
    ClassName.methodName(arguments)
  – For example, all the methods defined in the Math class are static methods
    double x = -1.2345;
    double absx = Math.abs(x); // Not x.abs()
Reuse methods from other classes

• For example, the max method is member of the class TestMax
• The max method can be invoked from any class besides TestMax
• If you create a new class Test, you can invoke the max method using TestMax.max

```java
public class TestMax {
    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.

```
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);
}
```

```java
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.

```
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;
}
```

j is declared and initialized

j: 2
i: 5
The main method is invoked.

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

The main method is invoked.

```
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.

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

The main method is invoked.
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 max method is invoked.

pass the values of i and j to num1 and num2

Space required for the main method

num1: 5
num2: 2
k: 
j: 2
i: 5

The max method is invoked.

CSE 8B, Spring 2023
The max method is invoked.

```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;
}
```

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

The max method is invoked.
The max method is invoked.

```
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
k: 2
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, 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:
}
Modularizing code

- Methods can be used to reduce redundant coding and enable code reuse
- Methods can also be used to modularize code and improve the quality of the program
Overloading methods

• Overloading methods enable you to define the methods with the same name as long as their parameter lists are different

• For example, overloading the `max` method

```java
public static double max(double num1, double num2) {
    if (num1 > num2)
        return num1;
    else
        return num2;
}
```
Ambiguous invocation

- The Java compiler determines which method to use based on the method signature.
- 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 compile error.
Scope of local variables

• A local variable is a variable defined inside a method
• Scope is 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
Method abstraction

- You can think of the method body as a black box that contains the detailed implementation for the method (i.e., encapsulation)
Benefits of methods

• Write a method once and reuse it anywhere
• Information hiding
  – Hide the implementation from the user
• Reduce complexity
Stepwise refinement

• The concept of method abstraction can be applied to the process of developing programs.

• When writing a large program, you can use the “divide and conquer” strategy, also known as *stepwise refinement*, to decompose it into subproblems.

• The subproblems can be further decomposed into smaller, more manageable problems.
Example design diagram

- printCalendar (main)
  - readInput
  - printMonth
    - printMonthTitle
      - getMonthName
    - printMonthBody
      - getTotalNumOfDays
        - getNumOfDaysInMonth
          - isLeapYear

March 2014
Su Mo Tu We Th Fr Sa
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
Bottom-up implementation

• Bottom-up approach is to implement one method in the structure chart at a time from the bottom to the top
• For each method implemented, write a test program to test it
Top-down implementation

- Top-down approach is to implement one method in the structure chart at a time from the top to the bottom.
- Stubs can be used for the methods waiting to be implemented:
  - A stub is a simple but incomplete version of a method.
  - The use of stubs enables you to test invoking the method from a caller.
- In the example, implement the main method first and then use a stub for the printMonth method:
  - For example, let printMonth display the year and the month in the stub.
Implementation

• Both top-down and bottom-up methods are fine
• Both approaches implement the methods incrementally and help to isolate programming errors and makes debugging easy
• Sometimes, they can be used together
Stepwise refinement

• Simpler program
• Reusing methods
• Easier developing, debugging, and testing
• Better facilitating teamwork
Next Lecture

• Loops and recursion