Loops and Methods

Introduction to Programming and Computational Problem Solving - 2

CSE 8B

Lecture 4
Announcements

• Assignment 1 is due today, 11:59 PM
• Assignment 2 will be released today
  – Due Apr 13, 11:59 PM
• Reading
  – Liang
    • Chapters 5 and 6
Loops

• while loops
• do-while loops
• for loops
while loops

• Executes statements repeatedly while the condition is true

```java
while (loop-continuation-condition) {
    // loop-body
    Statement(s);
}
```
while loops

```java
int count = 0;
while (count < 100) {
    System.out.println("Welcome to Java");
    count++;
}
```
Ending a loop with a sentinel value

• Often the number of times a loop is executed is not predetermined
• You may use an input value to signify the end of the loop
• Such a value is known as a sentinel value
• For example, a program reads and calculates the sum of an unspecified number of integers. The input 0 signifies the end of the input.
do-while loops

• Execute the loop body first, then checks the loop continuation condition

do {
    // Loop body
    Statement(s);
} while (loop-continuation-condition);
for loops

• A concise syntax for writing loops

for (initial-action; loop-continuation-condition; action-after-each-iteration) {
  // loop body
  Statement(s);
}

(a)
for loops

```java
int i;
for (i = 0; i < 100; i++) {
    System.out.println("Welcome to Java!");
}
```
for loops

• The initial-action in a for loop can be a list of zero or more comma-separated expressions
• The action-after-each-iteration in a for loop can be a list of zero or more comma-separated statements
• However, it is best practice (less error prone) **not to use comma-separated** expressions and statements

```java
for (int i = 0, j = 0; (i + j < 10); i++, j++) {
    // Do something
}
```
Loops and floating-point accuracy

- Remember, calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy.
- As such, do not use floating-point values for equality checking in a loop control.

```java
double sum = 0;
double item = 1;
while (item != 0) { // No guarantee item will be 0
    sum += item;
    item -= 0.1;
}
System.out.println(sum);
```
Infinite loops

• If the loop-continuation-condition in a for loop is omitted, it is implicitly true

(a) 
for (; ; ) {
  // Do something
}

(b) 
while (true) {
  // Do something
}
Loops

- The three forms of loop statements, *while*, *do-while*, and *for*, are expressively equivalent
  - You can write a loop in any of these three forms

```plaintext
while (loop-continuation-condition) {
  // Loop body
}
```

**Equivalent**

```plaintext
for ( ; loop-continuation-condition; ) {
  // Loop body
}
```

```plaintext
for (initial-action;
     loop-continuation-condition;
     action-after-each-iteration) {
  // Loop body;
}
```

**Equivalent**

```plaintext
initial-action;
while (loop-continuation-condition) {
  // Loop body;
  action-after-each-iteration;
}
```
Loops

• Use the loop form that is most intuitive and comfortable
  – A `for` loop may be used if the number of repetitions is known
  – A `while` loop may be used if the number of repetitions is not known
  – A `do-while` loop can be used to replace a `while` loop if the loop body must be executed before testing the continuation condition
public class TestBreak {
    public static void main(String[] args) {
        int sum = 0;
        int number = 0;

        while (number < 20) {
            number++;
            sum += number;
            if (sum >= 100)
                break;
        }
        System.out.println("The number is "+ number);
        System.out.println("The sum is "+ sum);
    }
}
public class TestContinue {
  public static void main(String[] args) {
    int sum = 0;
    int number = 0;

    while (number < 20) {
      number++;
      if (number == 10 || number == 11)
        continue;
      sum += number;
    }

    System.out.println("The sum is " + sum);
  }
}
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 signature:
```
int z = max(x, y);
```

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

Method invocation:
The 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*
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`

```
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`

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

Define a method

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

Invoke a method

actual parameters (arguments)
return statement

• 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;
}
```

(a)

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
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;
    }
}
```
Trace code

```
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:
}
```

The main method is invoked.

i is declared and initialized.
The main method is invoked.

```java
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);  
}
```

```
j is declared and initialized
```

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

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

Declare \( k \)

---

```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;
}
```
The main method is invoked.

Space required for the main method

k: 
j: 2
i: 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;
}
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;
}
```

The max method is invoked.

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

result:
- num2: 2
- num1: 5

Declare result.
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:
}
```

(result: num2: 2 num1: 5)

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

(num1 > num2) is true

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

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

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

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;
}```
The max method is invoked.

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

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

Return result and assign it to k.

```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;
}
```
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
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;
    }
}
```
Scope of local variables

// 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

// With errors
public static void incorrectMethod() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        int x = 0;
        x += i;
    }
}

Compile error: duplicate local variable
Method abstraction

- You can think of the method body as 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
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

getStartDay

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
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
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
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

• Single-dimensional arrays
• Multidimensional arrays
• Reading
  – Liang
    • Chapters 7 and 8