Re: Objectives in Loops

- Sequence and selection aside, we need repetition (loops)
- To use while, do-while, and for loop statements to control the repetition of statements
- To understand the flow of control in loop statements
- To use Boolean expressions to control loop statements
- To write nested loops
- To know the similarities and differences of three types of loops
- To implement program control with break and continue
Objectives in Methods

- To declare methods, invoke methods, and pass arguments to a method
- To use method overloading and know ambiguous overloading
- To determine the scope of local variables
- To learn the concept of method abstraction
- To know how to use the methods in the Math class
- To design and implement methods using stepwise refinement

What is a Method?

- A method is a collection of statements that are grouped together to perform an operation (for reuse).
  - `System.out.println(“Hello World!”);`
  - `String input = JOptionPane.showInputDialog(“Enter the number:”);`
  - Example: create a method to find which of two integers is bigger.

Define a method

Invoke a method
Two Method Types

- **Query**: A method may return a value. The `returnValueType` is the data type of the value the method returns
  - Example: public static int max(int num1, int num2)
    ```java
textValue = max(number1, number2);
```
  - String input = JOptionPane.showInputDialog(“Enter the number.”);

- **Command**: A method does not return a value, the `returnValueType` is the keyword `void`.
  - Example: public static void main(String[] args)
    ```java
    System.out.println(“Hello World!”);
    ```

1. Every parameter in a method must be declared to be of some specific data type
2. Every Java method must have a specific type that it returns

void Methods

Listing 5.2 Print the grade for a given score

```java
public class Instructor {
    public static void main(String[] args) {
        printGrade(78.5);
    }

    public static void printGrade(double score) {
        if (score >= 90.0) {
            System.out.println('A');
        } else if (score >= 80.0) {
            System.out.println('B');
        } else if (score >= 70.0) {
            System.out.println('C');
        } else if (score >= 60.0) {
            System.out.println('D');
        } else {
            System.out.println('F');
        }
    }
}
```
nonvoid Methods

Listing 5.1 Testing the `max` method

```java
public class TestMax {
    /** Main method */
    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);
    }

    /** Return the max between two numbers */
    public static int max(int num1, int num2) {
        int result;
        if (num1 > num2)
            result = num1;
        else
            result = num2;
        return result; // how to simplify it using the conditional operator?
    }
}
```

Calling Methods, cont.

Caution: the actual arguments must match the parameters in order, number, and compatible types, as defined in the method signature.

* Can the arguments of a method have the same name as its parameters?
CAUTION

A return statement is required for a nonvoid method. The following method is logically correct, but it 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 the code. The compiler can see a return statement to be reached regardless of the if comparison.

Call Stacks

The main method is invoked. 

The max method is invoked.

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

The main method is finished.
Passing Parameters

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

Suppose you invoke the method using
nPrintln("Welcome to Java", 5);
What is the output?

Suppose you invoke the method using
nPrintln("Computer Science", 15);
What is the output?

Pass by Value in Java

Listing 5.2 Testing Pass by value; swap two integer values

```java
public class TestPassByValue {
    /** Main method */
    public static void main(String[] args) {
        // Declare and initialize variables
        int num1 = 1;
        int num2 = 2;
        System.out.println("Before method invoking, num1 is " + num1 + " and num2 is " + num2);
        swap(num1, num2);  // invoke the swap method
        System.out.println("After method invoking, num1 is " + num1 + " and num2 is " + num2);
    }

    /** Swap two variables */
    public static void swap(int n1, int n2) {
        System.out.println("Inside the swap method");
        System.out.println("Before swapping n1 is " + n1 + " n2 is " + n2);
        // Swap n1 with n2
        int temp = n1;
        n1 = n2;
        n2 = temp;
        System.out.println("After swapping n1 is " + n1 + " n2 is " + n2);
    }
}
```

What output you expect?
Stack of the Pass by Value

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

What else we can do in Java?

Overloading Methods

- method overloading: two methods have the same name but different parameter lists within one class (make program more readable)

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

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

public static double max(double num1, double num2,
                          double num3) {
    return max(max(num1, num2), num3);
}
```

How the compiler determines which one to use?

* caution: we cannot overload methods based on return type or modifiers.
Ambiguous Invocation

An ambiguous invocation: there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match; a compilation error.

```java
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

- The scope of a variable: the part of the program where the variable can be referenced/accessed
  - a local variable: a variable defined inside a method
  - 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.

```java
public static void method1() {
    int k;
    
    for (int i = 1; i < 10; i++) {
        
        int j;        // What is the scope of k?
        
        
    }
}
```
Scope of Local Variables, cont.

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*.

```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 fine to declare `i` in two non-nesting blocks.

```java
public static void method2()
{
    int i = 1;
    int sum = 0;
    for (int i= 1; i < 10; i++)
    { sum += i;
    }
}
```

It is wrong to declare `i` in two nesting blocks.

---

The Math Class

- **Class constants:**
  - PI
  - E (the base of natural logarithms)

- **Class methods (static):**
  - Exponent Methods
  - Rounding Methods
  - min, max, abs, and random Methods
  - and more

All Math methods are static methods (vs. instant methods)
**Exponent Methods**

- `exp(double a)`
  Returns \( e \) raised to the power of \( a \).
- `log(double a)`
  Returns the natural logarithm of \( a \).
- `log10(double a)`
  Returns the 10-based logarithm of \( a \).
- `pow(double a, double b)`
  Returns \( a \) raised to the power of \( b \).
- `sqrt(double a)`
  Returns the square root of \( a \).

**Examples:**
- `Math.exp(1)` returns 2.71
- `Math.log(2.71)` returns 1.0
- `Math.pow(2, 3)` returns 8.0
- `Math.pow(3, 2)` returns 9.0
- `Math.pow(3.5, 2.5)` returns 22.91765
- `Math.sqrt(4)` returns 2.0
- `Math.sqrt(10.5)` returns 3.24

---

**Rounding Methods**

- `double ceil(double x)`
  \( x \) rounded up to its nearest integer. This integer is returned as a double value.
- `double floor(double x)`
  \( x \) is rounded down to its nearest integer. This integer is returned as a double value.
- `double rint(double x)`
  \( x \) is rounded to its nearest integer. If \( x \) is equally close to two integers, the even one is returned as a double.
min, max, and abs

- \( \text{max}(a, b) \) and \( \text{min}(a, b) \)
  - Overloaded methods; return the maximum or minimum of two parameters (int, long, float, or double).
- \( \text{abs}(a) \)
  - Returns the absolute value of the parameter.
- \( \text{random()} \)
  - Returns a random double value in the range [0.0, 1.0); excluding 1.0

How to generate random numbers, say 0 ~ 9; 0 ~ 10; 50 ~ 100; a ~ a + b?

Tips:
- \((\text{int}) (\text{Math.random()} * 10)\)
- \((\text{int}) (\text{Math.random()} * 11)\)
- \(50 + (\text{int}) (\text{Math.random()} * 50)\)
- \(a + \text{Math.random()} * b\)

You can view the complete documentation for the Math class online from http://java.sun.com/j2se/1.5.2/docs/api/index.htm.

---

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.
NOTE: One of the benefits of methods is for reuse.

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 `ClassName.methodName` (e.g., `TestMax.max`).

If you create a new class `Grader`, you can invoke the `printGrade` method using `Instructor.printGrade`.

*Note that those methods must be defined using `public static` modifiers (called static methods)*

---

**Re: void Methods**

Listing 5.2 Print the grade for a given score

```java
public class Instructor {
    public static void main(String[] args) {
        printGrade(78.5);
    }

    public static void printGrade(double score) {
        if (score >= 90.0) {
            System.out.println('A');
        } else if (score >= 80.0) {
            System.out.println('B');
        } else if (score >= 70.0) {
            System.out.println('C');
        } else if (score >= 60.0) {
            System.out.println('D');
        } else {
            System.out.println('F');
        }
    }
}
```

* Method abstraction is crucial to programming and problem solving. printGrade() implementation can be changed by the instructor; but a grader can reuse it as long as the method signature is the same.*
Case Study: Generating Random Characters

public class RandomCharacter {
    /** Generate a random character between ch1 and ch2 */
    public static char getRandomCharacter(char ch1, char ch2) {
        return (char)(ch1 + Math.random() * (ch2 - ch1 + 1));
    }

    /** Generate a random lowercase letter */
    public static char getRandomLowerCaseLetter() {
        return getRandomCharacter('a', 'z');
    }

    /** Generate a random uppercase letter */
    public static char getRandomUpperCaseLetter() {
        return getRandomCharacter('A', 'Z');
    }

    /** Generate a random digit character */
    public static char getRandomDigitCharacter() {
        return getRandomCharacter('0', '9');
    }

    /** Generate a random character */
    public static char getRandomCharacter() {
        return getRandomCharacter(' ', '￿');
    }
}

Case Study: Method Reuse

public class TestRandomCharacter {
    /** Main method */
    public static void main(String args[]) {
        final int NUMBER_OF_CHARS = 175;
        final int CHAR_PER_LINE = 25;

        // Print random low case characters between 'a' and 'z', 25 chars per line
        for (int i = 0; i < NUMBER_OF_CHARS; i++) {
            char ch = RandomCharacter.getRandomLowerCaseLetter();
            if ((i + 1) % CHAR_PER_LINE == 0) {
                System.out.println(ch);
            } else {
                System.out.print(ch);
            }
        }
    }
}

How to compile and run an application with multiple classes?
Stepwise Refinement (Reading)

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 sub-problems. The sub-problems can be further decomposed into smaller, more manageable problems.

PrintCalendar Case Study

Let us use the PrintCalendar example to demonstrate the stepwise refinement approach.
### Package

There are three reasons for using packages:

- **To avoid naming conflicts.** When you develop reusable classes to be shared by other programmers, naming conflicts often occur. To prevent this, put your classes into packages so that they can be referenced through package names.
- **To distribute software conveniently.** Packages group related classes so that they can be easily distributed.
- **To protect classes.** Packages provide protection so that the protected members of the classes are accessible to the classes in the same package, but not to the external classes.

#### Putting classes into a package:

```java
package packageName;
```

#### Using classes in a package:

```java
import javax.swing.*;
import javax.swing.JOptionPane;
```
Exercise: Computing the sum of a geometric series

° Suppose we wish to compute the sum of a geometric series:
  \[ \frac{1}{2} + (\frac{1}{2})^2 + (\frac{1}{2})^3 + \ldots + (\frac{1}{2})^n. \]

° At your desk, write a function \textit{geometricSum} that takes the value \( n \) (type \textit{int}) as its input parameter and returns the sum of the geometric series (as a \textit{double}).

° We shall embed the function \textit{geometricSum} into an interesting application whose specification is given as follows.

  - Specification: write a short application class, \textit{GeometricSeries},
    that outputs a table with consecutive values of \( n \) in its first column and the sum of the geometric series \( (\frac{1}{2}) + (\frac{1}{2})^2 + \ldots + (\frac{1}{2})^n \) in its second column. Continue this table until the sum of the geometric series equals to or greater than 1.

Clearly we do not know how many rows this table shall have in advance (theoretically, indefinite if we had unlimited precision).

<table>
<thead>
<tr>
<th>( n )</th>
<th>Geometric Sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>0.875</td>
</tr>
</tbody>
</table>

Reading/Homework

° Review chapter 5 of the textbook: 5.1 – 5.12

° Preview chapter 6 of the textbook

° Do review questions: 5.1, 5.2, 5.5, 5.7, 5.11, 5.13, 5.15