Lecture 9: Loops + Methods

CS 170, Section 000, Fall 2009
24 September 2009
Lecture Plan

• Logistics
• Lab 4 questions?
• Loops
  – Recap from last lecture
  – break/continue
  – more examples/case studies

Methods
• defining (declaring)
• calling (invoking)
• parameters (arguments)
Logistics

• Midterm 1: Postponed to Tuesday, October 6th
  – Will include material in Chapters 1 through 5

• Optional Review/Seminar (Q/A) session:
  – Monday afternoon, October 5th

• Homework 1 solutions to be posted by end of week
  – (not everyone has submitted yet)

• Homework 2 to be assigned today, due Wed 9/30
  – Solutions to be posted by end of week (in time for review session)
Lab 4 questions?
Loops Recap: while loop

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

int count = 0;
while (count < 2) {
    System.out.println("Welcome to Java!");
    count++;
}
System.out.print("Enter int value (0 to exit): ");
int data = input.nextInt();

int sum = 0;
while (data != 0) {// Keep reading until input=0
    sum += data;
    System.out.print("Enter value (0 to exit): ");
data = input.nextInt();
}
System.out.println("The sum is "+ sum);
Loops Recap: **do-while Loop**

```plaintext
do {
    // Loop body;
    **Statement(s);**
} while (loop-continuation-condition);
```
Recap: for Loops

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

for (int i=0; i<2; i++) {
    System.out.println("Welcome to Java!");
}
Recap: Which Loop to Use?

- while and for loops are pre-test loops, do-while are post-test loops
- The three loops are expressively equivalent.

A while loop in (a) below can always be converted into the for loop in (b):

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

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

A for loop in (a) below can generally be converted into the while loop in (b) except in certain special cases (see Review Question 3.19 for one of them):

```
for (initial-action;
    loop-continuation-condition;
    action-after-each-iteration) {
    // Loop body;
}
```
```
Equivalent
```
initial-action;
while (loop-continuation-condition) {
    // Loop body;
    action-after-each-iteration;
}
```
Recap: Nested Loops

- **Print Triangle**
- **Outer** loop for triangle rows
- **Inner** loop for triangle columns for each row

```
for (int i = 1; i <= n; i++)
{
    for (int j = 1; j <= i; j++)
    {
        System.out.print("*");
    }
    System.out.println();
}
```

Exercise: Modify code to print a square instead of a triangle

```
for (int i = 1; i <= n; i++)
{
    int j;
    // draw one row
    for (int j = 1; j <= n - i; j++)
    {
        System.out.print(" ");
    }
    for (j; j <= n; j++) print ("*");
    System.out.println();
    J = n - i
}
```
Using **break** and **continue**

- **break** – ends the **innermost loop** that contains it. Breaks out of a loop. (Can be used in loops and switch statements)

- **continue** – ends the current iteration. Continues to the next iteration of innermost loop.

- Normally used with if statement to break or continue the loop in a certain condition

- **You can always write a program using loops without break and continue**
break/continue Examples

• Using break
  Adds the integers from 1 to 20 in this order until the sum is greater than or equal to 100.
  TestBreak.java

• Using continue
  Adds all integers from 1 to 20 except 10 and 11.
  TestContinue.java
Example: Finding the Greatest Common Divisor

Problem: Write a program that prompts the user to enter two positive integers and finds their greatest common divisor. Suppose you enter two integers 4 and 2, their greatest common divisor is 2. Suppose you enter two integers 16 and 24, their greatest common divisor is 8.

How do you find the greatest common divisor?

approach: Let the two input integers be $n_1$ and $n_2$. You know number 1 is a common divisor, but it may not be the greatest common divisor. So you can check whether $k$ (for $k = 2, 3, 4, \text{ and so on}$) is a common divisor for $n_1$ and $n_2$, until $k$ is greater than $n_1$ or $n_2$. 

9/24/2009
GCD (cont’d): Pseudocode

\[
\begin{align*}
\text{n1, n2 : gcd} & = 1; \\
\text{for } k : 1, 2, 3 \ldots \text{ min}(n1, n2), \\
\text{check-if } & (n1 \mod k = 0 \land n2 \mod k = 0) \\
\text{gcd} & = k; \\
\text{print}(gcd); 
\end{align*}
\]
GCD (cont’d): code

/home/cs170000/inclass/sept24/GreatestCommonDivisor.java
Loops Questions?
Methods

Chapter 5
Motivating Example

• Often we need to find the maximum between two numbers

```java
int result;
if (num1 > num2)
    result = num1;
else
    result = num2;
```

• A method is a construct for grouping statements together to perform a function.

• A method is defined and implemented once but can be used repeatedly

• A method can be used as a blackbox
Method Abstraction

- Method abstraction 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.
Defining and Using Methods

- Define a method – give a definition of what the method is to do
  
  ```java
  modifier returnType methodName(list of parameters) {
    collection of statements;
  }
  ```

- Call or invoke a method – use a method
  
  ```java
  methodName(list of parameters)
  ```
Return Value Type

• A method may return a value (int, double, char, String, ...) – *value-returning method*

• A method may perform desired operations without returning a value (void) – *void method*
Method Signature

- **Method signature**
  - 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;
}
```

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

Define a method

Invoke a method

actual parameters (arguments)
Parameters

- The variables defined in the method header are known as *formal parameters*.
- When invoking a method, a value is passed to the parameter and this value is referred to as *actual parameter or argument*.
- The arguments must match the parameters in order, number, and compatible type.
- Parameters are optional.
Method Body

- The method body contains a collection of statements

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

Invoke a method

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

• The main method is a method that’s invoked by JVM
• The main method’s header is always the same
  
  ```java
  public static void main(String[] args) {
    //method body
  }
  ```

  The statements in main method may invoke other methods that are defined in the class or in other classes

  ```java
  System.out.println("Hello World!");
  int number = (int) (Math.random()*100);
  ```
Example

- A program that defines and uses a method `max` to return the larger of the two `int` values

**TestMax.java**

```java
int max(int n1, int n2) {
    int result;
    if (n1 > n2) result = n1;
    else result = n2;
    return result;
}
```
Calling Methods

```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;
}
```
```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;
}
```
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;
}
```
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;
}
(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 + ", " + 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

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

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 Stacks

- Each time a method is invoked, the system stores parameters and variables in an area of memory, known as a stack.
- When a method calls another method, the caller's stack space is kept intact.
  - New space is created to handle the new method call.
- When a method finishes, its associated space is released.
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;
}
```

i 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;
}
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;
}
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;
}
CAUTION

- A return statement is required for a value-returning method
- Compiler error when it is possible that a method does not return a value

```java
public static int sign(int x) {
    if (x < 0) {
        return -1;
    } else {
        return 1;
    }
}
```
Reuse Methods from Other Classes

• A method can be invoked from other classes
• Invoking a static method defined in other classes

ClassName.methodName (list of parameters)

• Example
  int number = (int) (Math.random() * 100);

• If you create a new class Test, you can invoke the max method using
  int number = Test.max(3, 5);
void Method Example

- If a method does not return any value, it is said to return **void** type. The method is called a void method.
- The main method is a void method:

  ```java
  public static void main(String[] args) {
      System.out.println("Hello world!");
  }
  ```

- Program example – defines and uses a method which prints the grade for a given score

```
/home/cs170000/inclass/sept24/TestVoidMethod.java
```

Passing Parameters

- When calling a method, the arguments must match the parameters in order, number, and compatible type.

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

public static void main(String[] args) {
    nPrintln("Hello!", 3);
    nPrintln("So that’s how the methods work", 10);
}
```

- When invoking a method, the value of the argument is passed to the parameter. The variable itself is not affected. This is referred to as **pass-by-value**.
Pass by Value – Trace call stack

```
public static void main(String[] args) {
    int x = 3;
    someMethod(x);
    System.out.println("x in main method: " + x);
}

public static void someMethod(int x) {
    x++;
    System.out.println("x in someMethod: " + x);
}
```
Pass by Value

```java
public static void main(String[] args) {
    int x = 3;
    someMethod(x);
    System.out.println("x in main method: " + x);
}

public static void someMethod(int x) {
    x++;
    System.out.println("x in someMethod: " + x);
}
```
Pass by Value

```java
public static void main(String[] args) {
    int x = 3;
    someMethod(x);
    System.out.println("x in main method: "+ x);
}

public static void someMethod(int x) {
    x++;
    System.out.println("x in someMethod: "+ x);
}
```
Pass by Value

The following code demonstrates the concept of pass by value:

```java
public static void main(String[] args) {
    int x = 3;
    someMethod(x);
    System.out.println("x in main method: " + x);
}

public static void someMethod(int x) {
    ++x;
    System.out.println("x in someMethod: " + x);
}
```
Pass by Value

public static void main(String[] args) {
    int x = 3;
    someMethod(x);
    System.out.println("x in main method: " + x);
}

public static void someMethod(int x) {
    x++;
    System.out.println("x in someMethod: " + x);
}
Program Example - Pass by Value

• The program demonstrates the effect of pass-by-value
• It creates a method for swapping two variables
• The values of the arguments are not changed after the method is invoked
• What if we change the method to

```java
swap(int num1, int num2)
```

/home/cs170000/inclass/sept24/TestPassByValue.java
Program Example: GCD with Methods

• Approach:
  – two input integers \( n_1 \) and \( n_2 \).
  – You know number 1 is a common divisor, but it may not be the greatest common divisor.
  – So you can check whether \( k \) (for \( k = 2, 3, 4, \) and so on) is a common divisor for \( n_1 \) and \( n_2 \), until \( k \) is greater than \( n_1 \) or \( n_2 \).

• Which methods do we need?
  \[\text{isCommonDivisor}(n_1, n_2, k);\]
Homework 2 (due Wed Sept 30)

- Problem 1: E (Loop exercise)
- Problem 2: Lucky
- Problem 3: Credit Card check sum (Luhn) algorithm
Problem 1

Write a program called E (in file E.java) which uses ONE loop to calculate an approximation of the value of $e$ (Euler’s number) using following formula:

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{n!}$$  \hspace{1cm} (1)

Hint: Value of next fraction can be computed using value of previous one:

$$\frac{1}{(n+1)!} = \frac{1}{n!} \cdot \frac{1}{n+1}$$  \hspace{1cm} (2)

```
% java E
Enter number of fractions used in formula for computing e: 100
2.7182818284590455
```

Note: Please, make sure that you are using doubles, not floats. Numerical precision is very important in this assignment.
Problem 2

Write a program called Lucky which implements the following bidding game:

1. Player starts with $1000.
2. On a turn the player makes a bid between 0 and all his money.
3. The player then wins or loses with 50% odds (use Math.random()) which results in the player’s money increasing or decreasing by the bid amount.
4. The game keeps asking the player for bids until either the player bids $0, or loses all his money.
5. If the player enters a bid lower than 0 or higher than his(her) total amount of money, the game ignores it and asks the player to enter another bid.
Problem 3

– Input: 13-19 digit number (hint: use \texttt{long} data type)
– if checksum is 0, print “This is a valid credit card number.” else, print “This is invalid credit card number.”
– Implementation:
  • \textbf{Use Methods}
  • \textbf{Input as Long} (not as String)
  • \textbf{Hints:} use integer division and remainder in a \texttt{loop} to get k-th digit; encapsulate it as a METHOD to make code more easy
Problem 3 (cont’d)

- **Luhn’s algorithm:**
  1. Counting from the check digit, which is the rightmost, and moving left, double the value of every second digit.
  2. Sum the digits of the products together with the undoubled digits from the original number.
  3. If the total ends in 0 (put another way, if the total modulo 10 is congruent to 0), then the number is valid according to the Luhn formula; else it is not valid.

- **Example:** if the account number is 49927398716, it will be validated as follows:
  - Double every second digit, from the rightmost: (1×2) = 2, (8×2) = 16, (3×2) = 6, (2×2) = 4, (9×2) = 18
  - Sum all the *individual* digits (digits in parentheses are the products from Step 1): 6 + (2) + 7 + (1+6) + 9 + (6) + 7 + (4) + 9 + (1+8) + 4 = 70
  - Take the sum modulo 10: 70 mod 10 = 0; the account number is valid

- You may test your code against reference implementation:
  - `/home/cs170000/inclass/sept24/Luhn.class`