Lecture 10: Methods (cont’d)

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

- HW1 Solution

- Methods
  - Recap
  - defining (declaring)
  - calling (invoking)
  - parameters (arguments)
  - overloading

- HW2 questions
Logistics

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

• Optional Review/Seminar (Q/A) session:
  – Monday afternoon, October 5\textsuperscript{th}
  – Time: TBD now

\*NO lab on 10/6

Room TBA \rightarrow 6pm
HW1 Solution

- /home/cs170000/share/hws/hw1/solutions/
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)
```

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

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

Define a method

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

Invoke a method

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

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

- The method body contains a collection of statements

```java
// Define a method
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);
```

- Actual parameters (arguments)
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;
}
```
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.
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;
}
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.
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 {  // must return something
        return 1;
    }
}
```
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**.
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/sept29/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?**
  \[
  \text{isCommonDivisor}(n_1, n_2, k);
  \]
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.
Scope of Local Variables

```java
public static void main(String[] args) {
    int a = 0;
    . . .
    if (a == 1) {
        int x = 0;
        . . .
        System.out.println("x=" + x);
    }
}
```

---

```java
public static void main(String[] args) {
    int a = 0;
    . . .
    if (a == 1) {
        int x = 0;
        //do some calculations on x
    }
    int x = 2;
}
```

Error

Okay
Scope of Local Variables

```java
public static void method1() {
    for (int i = 1; i < 10; i++) {
        int j;
    }
}
```

The scope of `i`

The scope of `j`
Scope of Local Variables

It is fine to declare i in two non-nesting blocks

```java
class Example1 {
    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
class Example2 {
    public static void method2() {
        int i = 1;
        int sum = 0;
        for (int i = 1; i < 10; i++) {
            sum += i;
        }
    }
}
```
What does the following method print?

```java
public static void method() {
    int sum = 0;  // not affected
    for (int i = 0; i < 5; i++) {
        int sum = 0;  // Error: Multiple definitions of 'sum'
        sum += i;
    }
    System.out.println("sum: " + sum);
}
```
Overloading methods

• Method overloading: multiple methods can have the same name but different parameter lists
• Compiler determines which method is used based on the method signature (method name and parameters)
Overloading Methods

```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;
}

max(1, 3);
max(1.0, 3.0);
max(1.0, 3);
```
Example of overloading: randomChar

/home/cs170000/inclass/sept29/RandomCharacter/
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:

\[ e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \cdots + \frac{1}{n!} \]  

(1)

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

\[ \frac{1}{(n+1)!} = \frac{1}{n!} \cdot \frac{1}{n+1} \]  

(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 long data type)
- if checksum is 0, print “This is a valid credit card number.” else, print “This is invalid credit card number.”
- Implementation:

  • Use Methods
  • Input as Long (not as String)
  • Hints: use integer division and remainder in a 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.java