Lecture 11: Methods (wrap-up)

CS 170, Section 000
1 October 2009
Lecture Plan

- Logistics

- Common errors in using variables, loops, methods
  - Program design using methods
    - top-down
    - bottom-up
    - case studies

- Midterm coverage
Logistics

- **Midterm 1:** Tuesday, October 6\(^\text{th}\)
  - Will include material in Chapters 1 through 5
  - Exact sections covered spec’d later; also to be posted

- **Homework 2** solutions to be posted by end of week (in time for you to study for midterm).

- **Optional** Review/Seminar (come with questions)
  - Monday afternoon, October 5\(^\text{th}\)
  - 4:15pm-6pm in room W301 (next door)
Common Errors in Variables, Loops, Methods

• Variables
  ▪ Common errors: scope, assignment, logic/semantics

▪ Methods
  ▪ Common errors: declaring and implementing, calling, parameters

▪ Loops
  ▪ common errors: terminating conditions, incrementing vs. decrementing counter
Variables: defining vs. using

- `int Number; // definition`
- `Number = 12345; // usage` ❌
- Every variable that you use in a Java program must be defined **before** its first usage.
- Example: compute area (area=pi*r*r)

```java
double area, pi, r;
r = 5; pi = 3.14;
//area = pi * r * r;
*^area = Area(pi, r);
```
Variables: defining vs. using

- int Number; // definition
- Number = 12345; // usage

- Every variable that you use in a Java program must be defined before its first usage.

- Example:

  ```java
  float pi, radius, area; // define variables
  area = pi * radius * radius;
  // get value of pi, radius
  // compute result of expression
  // store result in area
  ```
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.
public static void main(String[] args) {
    int a = 0;
    ...  
    if (a == 1) {
        int x = 0;
        ...
    }
    System.out.println("x=" + x);
}

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

public static void method1() {
    
    for (int i = 1; i < 10; i++) {
        
        The scope of i
        int j;
        
        The scope of j
    }

    
}
Scope of Local Variables

It is fine to declare \texttt{i} in two non-nesting 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 wrong to declare \texttt{i} in two nesting blocks

```java
public static void method2() {
    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++)
        sum += 1;

    }
```
What does the following method print

```java
public static void method() {
    for (int i = 0; i < 5; i++) {
        int sum = 0;
        sum += i;
    }
    int sum; // 0
    System.out.println("sum: " + sum);
}
```
Loop Examples

• Increment vs. Decrement counter

```java
for (int i = 0; i < 10; i++) {
    print spaces
    for (int j = 0; j < 10 - i; j++)
        print(" ");
}
```

10/2/2009 CS170, Section 000, Fall 2009
Nested Loop Examples

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

```
[ ]
[ ] [ ]
[ ] [ ] [ ]
[ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ]
[ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

9/24/2009  CS170, Section 000, Fall 2009
Program Design Using Methods

- statement = word
- method = sentence
- simple program/procedure = paragraph (HWs)
Methods: defining vs. calling

- Define methods in outermost scope of a class, not inside another method!

```java
class foo{
    public static void main(...){
        int a = getDigit(...);
    }

    public static int getDigit(...) {
    }
}
```
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);
```
void quadruple(int x, int y){
    x *= y;
    y *= y;
}

main(..){
    int n1=1, n2=2;
    quadruple (n1, n2);
    // 4*1, 4*2;
    n1; // 1
    n2; // 8
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 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
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 to decompose it into subproblems.
- The subproblems can be further decomposed into smaller, more manageable problems.
Write a program that reads year and month and prints out a calendar.

Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 2
February 2009

-----------------------------
Sun Mon Tue Wed Thu Fri Sat
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

PrintCalendarIncomplete.java
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 2
February 2009

Sun Mon Tue Wed Thu Fri Sat
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
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 2
February 2009

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 2
February 2009

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>
Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 2
February 2009

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>
Design Diagram

Enter full year (e.g., 2001): 2009
Enter month in number between 1 and 12: 2
February 2009

<table>
<thead>
<tr>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
<td>27</td>
<td>28</td>
</tr>
</tbody>
</table>

How to get the start day for the first date in a month, assuming we know that Jan 1, 1800, was Wednesday?
Implementation

- **Top-down**
  - Implement one method in the structure chart at a time from the top to bottom
  - Stubs can be used for the methods waiting to be implemented

- **Bottom-up**
  - 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
Midterm Coverage

Ch 1: all except 1.5
Ch 2: all except 2.9. 2.16
Ch 3: 11 3.8 4.41
Ch 4: all except 4.10
Ch 5: no 5.9

Sample Midterm Sunday night