CS1001

Lecture 13
Overview

- Java Programming
Goals

- Understand the basics of Java programming
Assignments

- Brooksheer: Ch 4, Ch 5 (Read)
- Read linked documents on these slides (slides will be posted in courseworks)
Objectives:

- Learn to distinguish the required syntax from the conventional style
- Learn when to use comments and how to mark them
- Review reserved words and standard names
- Learn the proper style for naming classes, methods, and variables
- Learn to space and indent blocks of code
Comments

- Comments are notes in plain English inserted in the source code.

- Comments are used to:
  - document the program’s purpose, author, revision history, copyright notices, etc.
  - describe fields, constructors, and methods
  - explain obscure or unusual places in the code
  - temporarily “comment out” fragments of code
Formats for Comments

- A “block” comment is placed between /* and */ marks:
  ```
  /* Exercise 5-2 for Java Methods
  Author: Miss Brace
  Date: 3/5/2010
  Rev. 1.0 */
  ```

- A single-line comment goes from // to the end of the line:
  ```
  wt *= 2.2046; // Convert to kilograms
  ```
Reserved Words

- In Java a number of words are reserved for a special purpose.
- Reserved words use only lowercase letters.
- Reserved words include:
  - primitive data types: `int`, `double`, `char`, `boolean`, etc.
  - storage modifiers: `public`, `private`, `static`, `final`, etc.
  - control statements: `if`, `else`, `switch`, `while`, `for`, etc.
  - built-in constants: `true`, `false`, `null`
- There are about 50 reserved words total.
Programmer-Defined Names

- In addition to reserved words, Java uses standard names for library packages and classes:
  - String, Graphics, javax.swing, JApplet, JButton, ActionListener, java.awt

- The programmer gives names to his or her classes, methods, fields, and variables.
Names (cont’d)

- **Syntax:** A name can include:
  - upper- and lowercase letters
  - digits
  - underscore characters

- **Syntax:** A name cannot begin with a digit.

- **Style:** Names should be descriptive to improve *readability.*
Names (cont’d)

- Programmers follow strict style conventions.

- Style: Names of classes begin with an **uppercase** letter, subsequent words are capitalized:
  
  ```java
  public class FallingCube
  ```

- Style: Names of methods, fields, and variables begin with a lowercase letter, subsequent words are capitalized.

  ```java
  private final int delay = 30;
  public void dropCube()
  ```
Names (cont’d)

- Method names often sound like verbs:
  setBackground, getText, dropCube, start

- Field names often sound like nouns:
  cube, delay, button, whiteboard

- Constants sometimes use all caps:
  PI, CUBESIZE

- It is OK to use standard short names for temporary “throwaway” variables:
  i, k, x, y, str
Syntax vs. Style

- **Syntax** is part of the language. The compiler checks it.
- **Style** is a convention widely adopted by software professionals.
- The main purpose of style is to improve the **readability of programs**.
Syntax

- The compiler catches syntax errors and generates error messages.
- Text in comments and literal strings within double quotes are excluded from syntax checking.
- Before compiling, carefully read your code a couple of times to check for syntax and logic errors.
Syntax (cont’d)

- Pay attention to and check for:
  - matching braces `{ }`, parentheses `( )`, and brackets `[ ]`
  - missing and extraneous semicolons
  - correct symbols for operators
    +, -, =, <, <=, ==, ++, &&, etc.
  - correct spelling of reserved words, library names and programmer-defined names, including case
Syntax (cont’d)

Common syntax errors:

- Spelling (p → P, if → If)
- Missing closing brace
- Extraneous semicolon
- Missing semicolon
Style

- Arrange code on separate lines; insert blank lines between fragments of code.
- Use comments.
- Indent blocks within braces.
### Style (cont’d)

**Before:**

```java
public boolean moveDown() {
    if (cubeY < 6 * cubeX) {
        cubeY += yStep;
        return true;
    } else
    return false;
}
```

**Compiles fine!**

---

**After:**

```java
public boolean moveDown() {
    if (cubeY < 6 * cubeX) {
        cubeY += yStep;
        return true;
    } else
    return false;
}
```

Compiles fine!
public void fill (char ch) {
    int rows = grid.length, cols = grid[0].length;
    int r, c;
    for (r = 0; r < rows; r++) {
        for (c = 0; c < cols; c++) {
            grid[r][c] = ch;
        }
    }
}
Blocks, Indentation

- Java code consists mainly of declarations and control statements.
- Declarations describe objects and methods.
- Control statement describe actions.
- Declarations and control statements end with a semicolon.
- No semicolon is used after a closing brace (except certain array declarations).
Blocks, Indentation (cont’d)

- Braces divide code into nested blocks.
- A block in braces indicates a number of statements that form one *compound* statement.
- Statements inside a block are indented, usually by two spaces or one tab.
public void fill (char ch)
{
    int rows = grid.length, cols = grid[0].length;
    int r, c;

    for (r = 0; r < rows; r++)
    {
        for (c = 0; c < cols; c++)
        {
            grid[r][c] = ch;
        }
    }
}
Review:

- Name as many uses of comments as you can.
- Explain the difference between syntax and style.
- Why is style important?
- Roughly how many reserved words does Java have?
Review (cont’d):

- Explain the convention for naming classes, methods and variables.
- Which of the following are syntactically valid names for variables: `C, _denom_, my.num, AvgScore`? Which of them are in good style?
- What can happen if you put an extra semicolon in your program?
- What are braces used for in Java?
- Is indentation required by Java syntax or style?
Objectives:

- Review primitive data types
- Learn how to declare fields and local variables
- Learn about arithmetic operators, compound assignment operators, and increment / decrement operators
- Learn how to avoid common mistakes in arithmetic
Variables

- A variable is a “named container” that holds a value.
- \( q = 100 - q; \)

means:
1. Read the current value of \( q \)
2. Subtract it from 100
3. Move the result back into \( q \)

\[\begin{align*}
\text{mov } & ax, q \\
\text{mov } & bx, 100 \\
\text{sub } & bx, ax \\
\text{mov } & q, bx
\end{align*}\]
Variables (cont’d)

- Variables can be of different data types: `int`, `char`, `double`, `boolean`, etc.
- Variables can hold objects; then the type is the class of the object.
- The programmer gives names to variables.
- Names usually start with a lowercase letter.
Variables (cont’d)

- A variable must be declared before it can be used:

  
<table>
<thead>
<tr>
<th>Type</th>
<th>Name(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>count</td>
</tr>
<tr>
<td>double</td>
<td>x, y</td>
</tr>
<tr>
<td>JButton</td>
<td>go</td>
</tr>
<tr>
<td>FallingCube</td>
<td>cube</td>
</tr>
<tr>
<td>String</td>
<td>firstName</td>
</tr>
</tbody>
</table>

Declarations
Variables (cont’d)

- The assignment operator = sets the variable’s value:

  ```java
  count = 5;
  x = 0;
  go = new JButton("Go");
  firstName = args[0];
  ```

- A variable can be initialized in its declaration:

  ```java
  int count = 5;
  JButton go = new JButton("Go");
  String firstName = args[0];
  ```
Variables (cont’d)

- Each variable has a *scope* — the area in the *source code* where it is “visible.”

- If you use a variable outside its scope, the compiler reports a syntax error.

- Variables *can* have the same name. **Caution:** use only when their scopes do not intersect.
Fields vs. Local Variables

- Fields are declared outside all constructors and methods.
- Local variables are declared inside a constructor or a method.
Fields vs. Local Variables (cont’d)

- Fields are usually grouped together, either at the top or at the bottom of the class.

- The scope of a field is the whole class.
Variables (cont’d)

- **Common mistakes:**

  ```java
  public void SomeMethod (...) {
      int x;
      ...  // x = 5; // should be: x = 5;
      ...
  }
  ```

  Variable declared twice — syntax error
Primitive Data Types

- int
- double
- char
- boolean

- byte
- short
- long
- float

Used in Java Methods
Constants

'A', '+', '\n', '\t' // char

-99, 2010, 0 // int

0.75, -12.3, 8., .5 // double
Constants (cont’d)

- Symbolic constants are initialized `final` variables:

```java
private final int delay = 30;
private final double aspectRatio = 0.7;
```
Constants (cont’d)

- Why use symbolic constants?
  - easier to change the value throughout, if necessary
  - easy to change into a variable
  - more readable, self-documenting code
  - additional data type checking
Arithmetic

- Operators: +, −, /, *, %
- The precedence of operators and parentheses work the same way as in algebra.
- \[ m \ % \ n \] means the remainder when \( m \) is divided by \( n \) (e.g. 17 % 5 is 2).
- % has the same rank as / and *
- Same-rank binary operators are performed in order from left to right.
Arithmetic (cont’d)

- The type of the result is determined by the types of the operands, not their values; this rule applies to all intermediate results in expressions.

- If one operand is an int and another is a double, the result is a double; if both operands are ints, the result is an int.
Arithmetic (cont’d)

- **Caution:** if `a` and `b` are `ints`, then `a / b` is truncated to an `int`...

  
  
  \[
  \begin{align*}
  17 / 5 & \text{ gives } 3 \\
  3 / 4 & \text{ gives } 0
  \end{align*}
  \]

- ...even if you assign the result to a `double`:

  
  ```
  double ratio = 2 / 3;
  ```

  The `double` type of the result doesn’t help: ratio still gets the value `0.0`. 
Arithmetic (cont’d)

- To get the correct double result, use double constants or the cast operator:

  double ratio = 2.0 / 3;
  double ratio = 2 / 3.0;
  double factor = (double) m / (double) n;
  double factor = m / (double) n;
  double r2 = k / 2.0;
  double r2 = (double) k / 2;
**Arithmetic (cont’d)**

- **Caution:** the range for **ints** is from $-2^{31}$ to $2^{31}-1$ (about $-2 \cdot 10^9$ to $2 \cdot 10^9$)

- Overflow is **not** detected by the Java compiler or interpreter

<table>
<thead>
<tr>
<th>$n$</th>
<th>$10^n$</th>
<th>$n!$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>100000000</td>
<td>40320</td>
</tr>
<tr>
<td>9</td>
<td>10000000000</td>
<td>362880</td>
</tr>
<tr>
<td>10</td>
<td>1410065408</td>
<td>36288000</td>
</tr>
<tr>
<td>11</td>
<td>1215752192</td>
<td>39916800</td>
</tr>
<tr>
<td>12</td>
<td>-727379968</td>
<td>479001600</td>
</tr>
<tr>
<td>13</td>
<td>1316134912</td>
<td>1932053504</td>
</tr>
<tr>
<td>14</td>
<td>276447232</td>
<td>1278945280</td>
</tr>
</tbody>
</table>
Arithmetic (cont’d)

- Use compound assignment operators:
  
  - $a = a + b; \rightarrow a += b;$
  - $a = a - b; \rightarrow a -= b;$
  - $a = a * b; \rightarrow a *= b;$
  - $a = a / b; \rightarrow a /= b;$
  - $a = a \% b; \rightarrow a \%= b;$

- Use increment and decrement operators:
  
  - $a = a + 1; \rightarrow a++;$
  - $a = a - 1; \rightarrow a--;$

  *Do not use these in larger expressions*
Review:

- What is a variable?
- What is the type of variable that holds an object?
Review (cont’d):

- What is the range for `ints`?
- When is a cast to `double` used?
- Given
  ```java
  double dF = 68.0;
  double dC = 5 / 9 * (dF - 32);
  ```
  what is the value of `dC`?
- When is a cast to `int` used?
- Should compound assignment operators be avoided?