CSE 142, Summer 2015

Lecture 2: Static Methods
Expressions

reading: 1.4 – 2.1
The Mechanical Turk
Escape Characters

System.out.println("ab\"\\\\\\\\V\td");

Output: ab"\\\\\\\\d
Algorithms

• **algorithm**: A list of steps for solving a problem.

• Example algorithm: "Bake sugar cookies"
  1. Mix the dry ingredients.
  2. Cream the butter and sugar.
  3. Beat in the eggs.
  4. Stir in the dry ingredients.
  5. Set the oven temperature.
  6. Set the timer for 10 minutes.
  7. Place the cookies into the oven.
  8. Allow the cookies to bake.
  9. Spread frosting and sprinkles onto the cookies.
Style

Correct Recipe with Bad Style:

- Recipe Works Correctly
- Hard to understand or modify

5. Set the oven temperature. 8. Spread frosting and sprinkles onto the cookies. 6. Set the timer for 10 minutes. 3. Beat in the eggs. 1. Mix the dry ingredients. 2. Cream the butter and sugar. 7. Allow the cookies to bake. 4. Stir in the dry ingredients. 6.5. Place the cookies into the oven.
Style

• **Better Style**: Split into coherent tasks.

1 **Make the batter**.
   - Mix the dry ingredients.
   - Cream the butter and sugar.
   - Beat in the eggs.
   - Stir in the dry ingredients.

2 **Bake the cookies**.
   - Set the oven temperature.
   - Set the timer for 10 minutes.
   - Place the cookies into the oven.
   - Allow the cookies to bake.

3 **Decorate the cookies**.
   - Mix the ingredients for the frosting.
   - Spread frosting and sprinkles onto the cookies.
   ...

Copyright 2010 by Pearson Education
Recall: structure, syntax

```java
public class name {
    public static void main(String[] args) {
        statement;
        statement;
        ...
        statement;
    }
}
```

- Every executable Java program consists of a **class**, that contains a **method** named **main**, that contains the **statements** (commands) to be executed.
Static methods

- **static method**: A named group of statements.
  - denotes the *structure* of a program
  - eliminates *redundancy* by code reuse

- **procedural decomposition**: dividing a problem into methods

- Writing a static method is like adding a new command to Java.
Declaring a method

*Gives your method a name so it can be executed*

- **Syntax:**

  ```java
public static void name() {
    statement;
    statement;
    ...
    statement;
  }
```

- **Example:**

  ```java
  public static void printWarning() {
    System.out.println("This product causes cancer");
    System.out.println("in lab rats and humans.");
  }
  ```
Calling a method

Executes the method's code

• Syntax:

    name();

    • You can call the same method many times if you like.

• Example:

    printWarning();

• Output:

    This product causes cancer in lab rats and humans.
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }
    public static void message1() {
        System.out.println("This is message1.");
    }
    public static void message2() {
        System.out.println("This is message2.");
        message1();
        System.out.println("Done with message2.");
    }
}

• Output:
  This is message1.
  This is message2.
  This is message1.
  Done with message2.
  Done with main.
When to use methods

- Place statements into a static method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should not create static methods for:
  - An individual `println` statement.
  - Only blank lines. (Put blank `printlns` in `main`.)
  - Unrelated or weakly related statements.
    (Consider splitting them into two smaller methods.)
Static methods question

- Write a program to print these figures using methods.
Output structure

The structure of the output:
- initial "egg" figure
- second "teacup" figure
- third "stop sign" figure
- fourth "hat" figure

This structure can be represented by methods:
- egg
- teaCup
- stopSign
- hat
Output redundancy

The redundancy in the output:

- egg top: reused on stop sign, hat
- egg bottom: reused on teacup, stop sign
- divider line: used on teacup, hat

This redundancy can be fixed by methods:

- eggTop
- eggBottom
- line
Data and expressions

reading: 2.1
Data types

- Internally, computers store everything as 1s and 0s
  
  104 → 01101000  
  "hi" → 0110100001101001  
  h → 01101000  

- How are h and 104 differentiated?

- **type**: A category or set of data values.  
  - Constrains the operations that can be performed on data  
  - Many languages ask the programmer to specify types  
  - Examples: integer, real number, string
Java's primitive types

- **primitive types**: 8 simple types for numbers, text, etc.
- Java also has **object types**, which we'll talk about later

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>integers (up to $2^{31} - 1$)</td>
<td>42, -3, 0, 926394</td>
</tr>
<tr>
<td>double</td>
<td>real numbers (up to $10^{308}$)</td>
<td>3.1, -0.25, 9.4e3</td>
</tr>
<tr>
<td>char</td>
<td>single text characters</td>
<td>'a', 'X', '?', '\n'</td>
</tr>
<tr>
<td>boolean</td>
<td>logical values</td>
<td>true, false</td>
</tr>
</tbody>
</table>

- Why does Java distinguish integers vs. real numbers?
Integer or real number?

- Which category is more appropriate?

<table>
<thead>
<tr>
<th>integer (int)</th>
<th>real number (double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Temperature in degrees Celsius</td>
<td>7. Number of miles traveled</td>
</tr>
<tr>
<td>2. The population of lemmings</td>
<td>8. Number of dry days in the past month</td>
</tr>
<tr>
<td>3. Your grade point average</td>
<td>9. Your locker number</td>
</tr>
<tr>
<td>4. A person's age in years</td>
<td>10. Number of seconds left in a game</td>
</tr>
<tr>
<td>5. A person's weight in pounds</td>
<td>11. The sum of a group of integers</td>
</tr>
<tr>
<td>6. A person's height in meters</td>
<td>12. The average of a group of integers</td>
</tr>
</tbody>
</table>

Expressions

- **expression**: A value or operation that computes a value.
  - Examples: $1 + 4 \times 5$
    
    $(7 + 2) \times 6 / 3$
    
    42
  
  - The simplest expression is a *literal value*.
  - A complex expression can use operators and parentheses.
Arithmetic operators

- **operator**: Combines multiple values or expressions.
  - `+`: addition
  - `-`: subtraction (or negation)
  - `*`: multiplication
  - `/`: division
  - `%`: modulus (a.k.a. remainder)

As a program runs, its expressions are **evaluated**.

- `1 + 1` evaluates to `2`
- `System.out.println(3 * 4);` prints `12`
  - How would we print the text `3 * 4`?
Integer remainder with %

- The % operator computes the remainder from integer division.
  - $14 \% 4 \text{ is } 2$
  - $218 \% 5 \text{ is } 3$

\[
\begin{array}{c}
4 \) 14 \\
\underline{12} \\
2
\end{array}
\quad
\begin{array}{c}
5 \) 218 \\
\underline{20} \\
18 \\
15 \\
3
\end{array}
\]

- Applications of % operator:
  - Obtain last digit of a number: $230857 \% 10 \text{ is } 7$
  - Obtain last 4 digits: $658236489 \% 10000 \text{ is } 6489$
  - See whether a number is odd: $7 \% 2 \text{ is } 1$, $42 \% 2 \text{ is } 0$
Precedence

- **precedence**: Order in which operators are evaluated.
- Generally operators evaluate left-to-right.
  - $1 - 2 - 3$ is $(1 - 2) - 3$ which is $-4$

- But $* / \%$ have a higher level of precedence than $+ -$
  - $1 + 3 \times 4$ is $13$
  - $6 + 8 / 2 \times 3$
  - $6 + 4 \times 3$
  - $6 + 12$ is $18$

- Parentheses can force a certain order of evaluation:
  - $(1 + 3) \times 4$ is $16$

- Spacing does not affect order of evaluation
  - $1 + 3 \times 4 - 2$ is $11$
Precedence examples

\[
1 \times 2 + 3 \times 5 \mod 4
\]

\[
\frac{2}{+} \quad 3 \times 5 \mod 4
\]

\[
\frac{2}{+} \quad 15 \mod 4
\]

\[
\frac{2}{+} \quad 3
\]

\[
5
\]
Precedence questions

- What values result from the following expressions?

  - $9 \div 5$
  - $695 \% 20$
  - $7 + 6 \times 5$
  - $7 \times 6 + 5$
  - $248 \% 100 \div 5$
  - $6 \times 3 - 9 \div 4$
  - $(5 - 7) \times 4$
  - $6 + (18 \% (17 - 12))$
Real numbers (type double)

- Examples: 6.022, -42.0, 2.143e17
  - Placing .0 or . after an integer makes it a double.

- The operators + - * / % () all still work with double.
  - / produces an exact answer: 15.0 / 2.0 is 7.5
  - Precedence is the same: () before * / % before + -
Real number example

\[ 2.0 \times 2.4 + 2.25 \times 4.0 / 2.0 \]

\[ \begin{array}{c}
 4.8 + 2.25 \times 4.0 / 2.0 \\
 4.8 + 9.0 / 2.0 \\
 4.8 + 4.5 \\
 9.3 
\end{array} \]
Mixing types

- When `int` and `double` are mixed, the result is a `double`.
  - `4.2 * 3` is `12.6`

- The conversion is per-operator, affecting only its operands.
  - `7 / 3 * 1.2 + 3 / 2` becomes `3.4`
  - `2.0 + 10 / 3 * 2.5 - 6 / 4` becomes `8.5`

- `3 / 2` is `1` above, not `1.5`.

Copyright 2010 by Pearson Education
String concatenation

- **string concatenation**: Using + between a string and another value to make a longer string.

  - "hi" + "hello" is "hihello"
  - "hello" + 42 is "hello42"
  - 1 + "abc" + 2 is "1abc2"
  - "abc" + 1 + 2 is "abc12"
  - 1 + 2 + "abc" is "3abc"
  - "abc" + 9 * 3 is "abc27"
  - "1" + 1 is "11"
  - 4 - 1 + "abc" is "3abc"

- Use + to print a string and an expression's value together.

  - System.out.println("Grade: " + (95.1 + 71.9) / 2);
  - **Output**: Grade: 83.5