Building Java Programs

Chapter 1
Lecture 1-2: Static Methods
reading: 1.4 - 1.5

Using comments

- Where to place comments:
  - at the top of each file (a "comment header")
  - at the start of every method (soon later)
  - to explain complex pieces of code

- Comments are useful for:
  - Understanding larger, more complex programs.
  - Multiple programmers working together, who must understand each other's code.

Comments

- comment: A note written in source code by the programmer to describe or clarify the code.
- Comments are not executed when your program runs.

- Syntax:
  // comment text, on one line
  /* comment text; may span multiple lines */

- Examples:
  // This is a one-line comment.
  /* This is a very long multi-line comment. */

Comments example

/* Remy Student, CS 101, Fall 2019
   * This program prints lyrics about ... something. */

public class maxtanka {
    public static void main(String[] args) {
        // First verse
        System.out.println("maxtanka");
        System.out.println("It's hang a dang diggy diggy");
        System.out.println();
        // second verse
        System.out.println("Diggy said the hokey");
        System.out.println("rais up jump the hokey");
    }
}

Static methods

reading: 1.4
self-check: 16-25
exercises: #5-10
videos: Ch. 1 #1

Algorithms

- algorithm: A list of steps for solving a problem.
- Example algorithm: "Bake sugar cookies"
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.
  - Set the oven temperature.
  - Set the timer.
  - Place the cookies into the oven.
  - Allow the cookies to bake.
  - Spread frosting and sprinkles onto the cookies.
  - ...
Problems with algorithms

- **lack of structure**: Many tiny steps, tough to remember.
- **redundancy**: Consider making a double batch...
  - Mix the dry ingredients.
  - Cream the butter and sugar.
  - Beat in the eggs.
  - Stir in the dry ingredients.
  - Set the oven temperature.
  - Set the timer.
  - Place the first batch of cookies into the oven.
  - Allow the cookies to bake.
  - Set the timer.
  - Place the second batch of cookies into the oven.
  - Allow the cookies to bake.
  - Mix ingredients for frosting.
  - ...
Design of an algorithm

// This program displays a delicious recipe for baking cookies.
public class CookieRecipe {  
  public static void main(String[] args) {  
    System.out.println("Step 1: Make the dough.");  
    System.out.println("Step 2: Add the dry ingredients.");  
    System.out.println("Step 3: Mix the egg and sugar.");  
    System.out.println("Step 4: Bake the cookies.");  
    System.out.println("Step 5: Let cookies cool.");  
    System.out.println("Step 6: Decorate the cookies.");  
  }  
}  

Declaring a method

// Gives your method a name so it can be executed
public static void printWarning() {  
  System.out.println("Warning: This product causes cancer.");  
  System.out.println("in lab rats and humans.");  
}  

Calling a method

// Executes the method's code
public static void name() {  
  System.out.println("This product causes cancer in lab rats and humans.");  
}  

Example:

public static void main(String[] args) {  
  printWarning();  
}  

Output:

This product causes cancer
in lab rats and humans.

Final cookie program

// This program displays a delicious recipe for baking cookies.
public class CookieRecipe {  
  public static void main(String[] args) {  
    System.out.println("Step 1: Make the dough.");  
    System.out.println("Step 2: Add the dry ingredients.");  
    System.out.println("Step 3: Mix the egg and sugar.");  
    System.out.println("Step 4: Bake the cookies.");  
    System.out.println("Step 5: Let cookies cool.");  
    System.out.println("Step 6: Decorate the cookies.");  
  }  
}  

Methods calling methods

public class MethodExample {  
  public static void main(String[] args) {  
    System.out.println("This is message.");  
    System.out.println("This is message.");  
    System.out.println("This is message.");  
    System.out.println("This is message.");  
  }  
}  

Control flow

// When a method is called, the program's execution...
public class MethodExample {  
  public static void main(String[] args) {  
    System.out.println("This is message.");  
    System.out.println("This is message.");  
    System.out.println("This is message.");  
    System.out.println("This is message.");  
  }  
}  

...
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 println in main.)
  - Unrelated or weakly related statements. (Consider splitting them into two smaller methods.)

Drawing complex figures with static methods

**reading**: 1.5

(Ch. 1 Case Study: DrawFigures)

**exercises**: #7-9

**videos**: Ch. 1 #2

Static methods question

- Write a program to print these figures using methods.

Development strategy

**First version (unstructured):**

- Create an empty program and `main` method.
- Copy the expected output into it, surrounding each line with `System.out.println` syntax.
- Run it to verify the output.

Development strategy 2

**Second version (structured, with redundancy):**

- Identify the structure of the output.
- Divide the `main` method into static methods based on this structure.

Program version 1

```java
public class Figures {
    public static void main(String[] args) {
        System.out.println("/");
        System.out.println("/");
        System.out.println("/");
        System.out.println("/");
    }
}
```
Output structure

The structure of the output:
- Initial "egg" figure
- Second "tea cup" figure
- Third "stop sign" figure
- Fourth "hat" figure

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

Program version 2

```java
public class Figures{
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    public static void egg() {
        // Code for egg
    }

    public static void teaCup() {
        // Code for tea cup
    }

    public static void stopSign() {
        // Code for stop sign
    }

    public static void hat() {
        // Code for hat
    }
}
```

Program version 2, cont'd.

```java
public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    public static void egg() {
        // Code for egg
    }

    public static void teaCup() {
        // Code for tea cup
    }

    public static void stopSign() {
        // Code for stop sign
    }

    public static void hat() {
        // Code for hat
    }
}
```

Development strategy 3

Third version (structured, without redundancy):
- Identify redundancy in the output, and create methods to eliminate as much as possible.
- Add comments to the program.

Output redundancy

The redundancy in the output:
- Egg top: reused on stop sign, hat
- Egg bottom: reused on tea cup, stop sign
- Divider line: used on tea cup, hat

This redundancy can be fixed by methods:
- eggTop
- eggBottom
- line

Program version 3

```java
// Egg
// Code for egg...

// Tea cup
// Code for tea cup...

// Stop sign
// Code for stop sign...

// Hat
// Code for hat...
```
Program version 3, cont'd.

// Draws a teetop figure
public static void teetop() {
    System.out.println("/");
    System.out.println("/");
}

// Draws a stop sign figure
public static void stopSign() {
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
}

// Draws a figure that looks sort of like a hat.
public static void hat() {
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
    System.out.println("/");
}

// Draws a line of dashes
public static void line() {
    System.out.println("---------------");
}