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. */`
Using comments

- Where to place comments:
  - at the top of each file (a "comment header")
  - at the start of every method (seen 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 example

```java
/* Suzy Student, CS 101, Fall 2019
This program prints lyrics about ... something. */

public class BaWitDaBa {
    public static void main(String[] args) {
        // first verse
        System.out.println("Bawitdaba");
        System.out.println("da bang a dang diggy diggy");
        System.out.println();

        // second verse
        System.out.println("diggy said the boogy");
        System.out.println("said up jump the boogy");
    }
}
```
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.
  - ...

---

bye
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.
  - ...

Structured algorithms

- **structured algorithm**: Split into coherent tasks.

1. **Make the cookie 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.
   - Place the cookies into the oven.
   - Allow the cookies to bake.

3. **Add frosting and sprinkles**.
   - Mix the ingredients for the frosting.
   - Spread frosting and sprinkles onto the cookies.
   - ...
Removing redundancy

- A well-structured algorithm can describe repeated tasks with less redundancy.

1 Make the cookie batter.
   - Mix the dry ingredients.
   - ...

2a Bake the cookies (first batch).
   - Set the oven temperature.
   - Set the timer.
   - ...

2b Bake the cookies (second batch).

3 Decorate the cookies.
   - ...

A program with redundancy

```java
public class BakeCookies {
    public static void main(String[] args) {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
        System.out.println("Set the oven temperature.");
        System.out.println("Allow the cookies to bake.");
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```
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.

Using static methods

1. Design the algorithm.
   - Look at the structure, and which commands are repeated.
   - Decide what are the important overall tasks.

2. **Declare** (write down) the methods.
   - Arrange statements into groups and give each group a name.

3. **Call** (run) the methods.
   - The program’s *main* method executes the other methods to perform the overall task.
Design of an algorithm

// This program displays a delicious recipe for baking cookies.
public class BakeCookies2 {
    public static void main(String[] args) {
        // Step 1: Make the cake batter.
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");

        // Step 2a: Bake cookies (first batch).
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 2b: Bake cookies (second batch).
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");

        // Step 3: Decorate the cookies.
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}

Declaring a method

Gives your method a name so it can be executed

- Syntax:

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

- Example:

    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:**
  
  ```java
  name();
  ```

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

- **Example:**

  ```java
  printWarning();
  ```

- **Output:**

  This product causes cancer in lab rats and humans.

Program with static method

```java
public class FreshPrince {
    public static void main(String[] args) {
        rap(); // Calling (running) the rap method
        System.out.println();
        rap(); // Calling the rap method again
    }

    // This method prints the lyrics to my favorite song.
    public static void rap() {
        System.out.println("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

**Output:**

Now this is the story all about how
My life got flipped turned upside-down

Now this is the story all about how
My life got flipped turned upside-down
Final cookie program

// This program displays a delicious recipe for baking cookies.
public class BakeCookies3 {
    public static void main(String[] args) {
        makeBatter();
        bake(); // 1st batch
        bake(); // 2nd batch
        decorate();
    }
    // Step 1: Make the cake batter.
    public static void makeBatter() {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
    }
    // Step 2: Bake a batch of cookies.
    public static void bake() {
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
    }
    // Step 3: Decorate the cookies.
    public static void decorate() {
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}

Methods calling methods

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.");
    }
    public static void message3() {
        System.out.println("This is message3.");
        message1();
        System.out.println("Done with message3.");
    }
}

• Output:
  This is message1.
  This is message2.
  This is message1.
  Done with message2.
  Done with main.
Control flow

- When a method is called, the program's execution...
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.

```java
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.");
    }
}
```

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.
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.

Program version 1

```java
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println(" \ / ");
        System.out.println(" \______/ ");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println("|  STOP  |");
        System.out.println(" \ / ");
        System.out.println(" \______/ ");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println("+--------+");
    }
}
```
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.

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
public class Figures2 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    public static void egg() {
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println("\ / ");
        System.out.println(" \______/ ");
        System.out.println();
    }

    public static void teaCup() {
        System.out.println("\ / ");
        System.out.println(" \______/ ");
        System.out.println("+--------+");
        System.out.println();
    }

    public static void stopSign() {
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println("|  STOP  |");
        System.out.println("\ / ");
        System.out.println(" \______/ ");
        System.out.println();
    }

    public static void hat() {
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
        System.out.println("+--------+");
        System.out.println();
    }
}

Program version 2, cont'd.

...
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 teacup, stop sign
- divider line: used on teacup, hat

This redundancy can be fixed by methods:
- eggTop
- eggBottom
- line
Program version 3

// Sury Student, CSE 138, Spring 2094
// Prints several figures, with methods for structure and redundancy.
public class Figures3 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    // Draws the top half of an egg figure.
    public static void eggTop() {
        System.out.println("  ______");
        System.out.println(" /      ");
        System.out.println("/        ");
    }

    // Draws the bottom half of an egg figure.
    public static void eggBottom() {
        System.out.println("\ / ");
        System.out.println(" \______/ ");
    }

    // Draws a complete egg figure.
    public static void egg() {
        eggTop();
        eggBottom();
        System.out.println();
    }

    // Draws a teacup figure.
    public static void teaCup() {
        eggBottom();
        line();
        System.out.println();
    }

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

    // Draws a figure that looks sort of like a hat.
    public static void hat() {
        eggTop();
        line();
    }

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

Program version 3, cont'd.

...