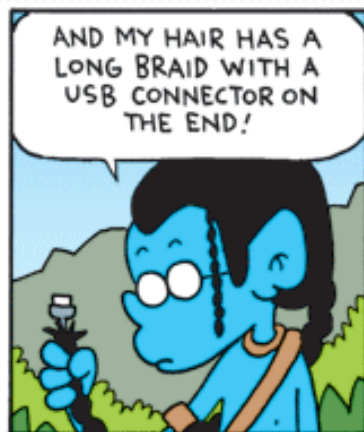
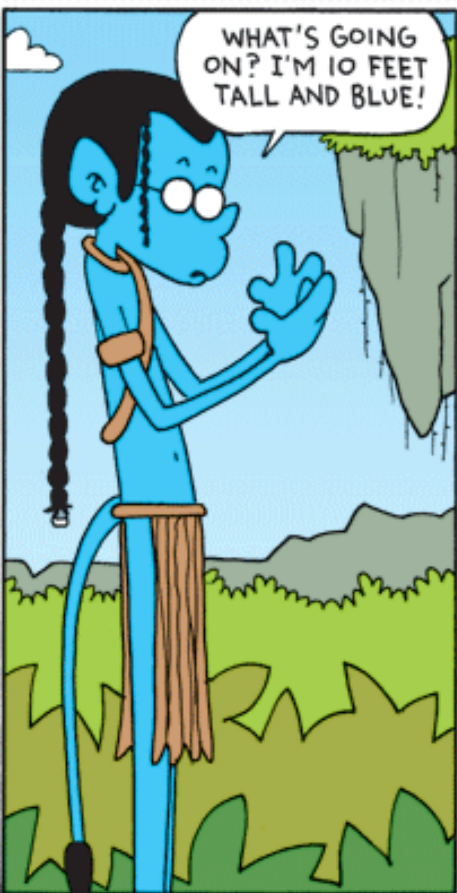


Building Java Programs

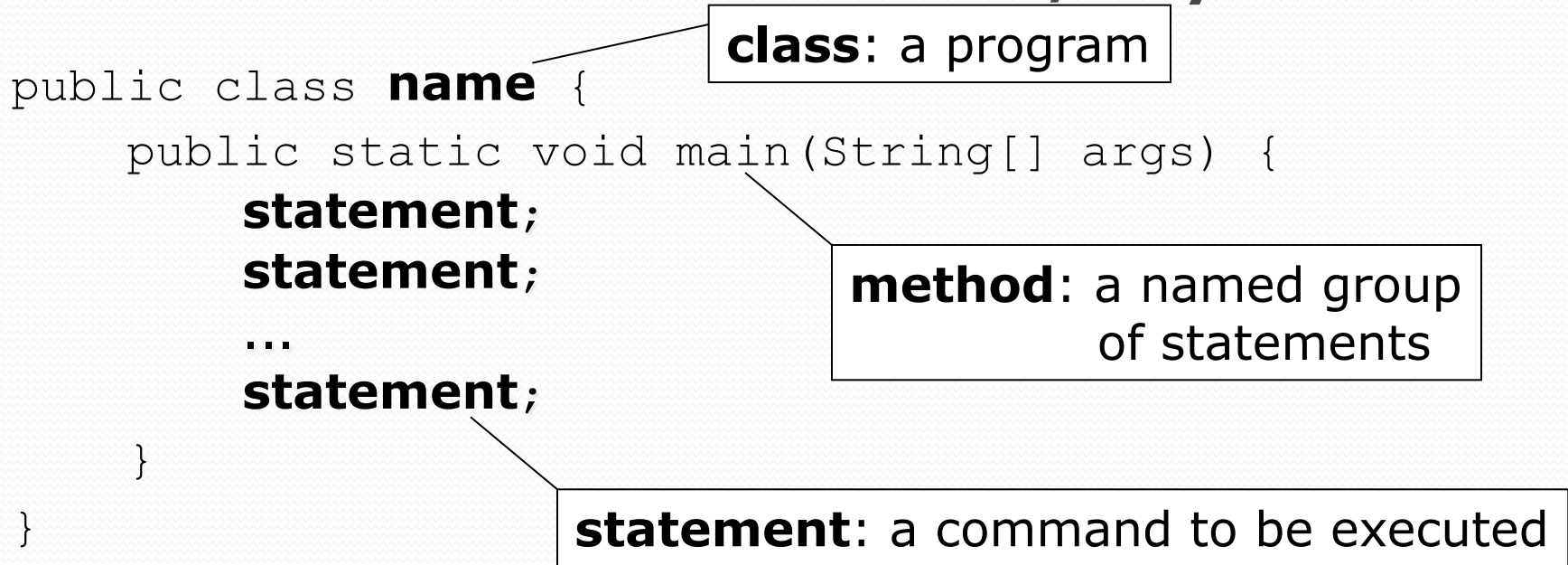
Chapter 1

Lecture 1-2: Static Methods

reading: 1.4 - 1.5



Recall: structure, syntax



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

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`
 - or,
 - `/* comment text; may span multiple lines */`
- Examples:
 - `// This is a one-line comment.`
 - `/* This is a very long
multi-line comment. */`

Comments example

```
/* Suzy Student, CSE 142, Fall 2019  
   Displays lyrics*/
```

```
public class Lyrics {  
    public static void main(String[] args) {  
        // first line  
        System.out.println("When I first got into magic");  
        System.out.println("it was an underground phenomenon");  
        System.out.println();  
  
        // second line  
        System.out.println("Now everybody's like");  
        System.out.println("pick a card, any card");  
    }  
}
```


Static methods

reading: 1.4

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 for 10 minutes.
 - 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 steps; tough to follow.
- *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 for 10 minutes.
 - Place the first batch of cookies into the oven.
 - Allow the cookies to bake.
 - Set the timer for 10 minutes.
 - 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 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.

...

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 for 10 minutes.
- ...

2b Bake the cookies (second batch).

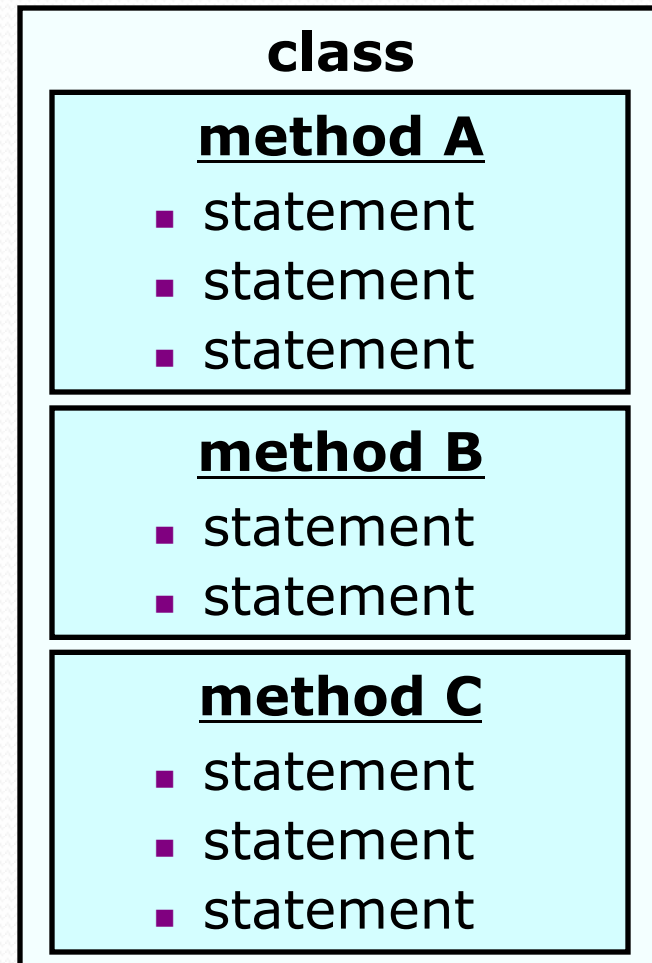
- Repeat Step 2a

3 Decorate the cookies.

- ...

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** (think about) 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.

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:

name ();

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

- Example:

```
printWarning();
```

- Output:

```
This product causes cancer  
in lab rats and humans.
```

Program with static method

```
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
```

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.");
    }
}
```

- **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.

```
public class MethodsExample {  
    public static void main(String[] args) {  
        message1 () ;  
  
        message2 () ;  
  
        System.out.println("...")  
    }  
  
    ...  
}
```

```
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 message1() {  
    System.out.println("This is message1.");  
}
```

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

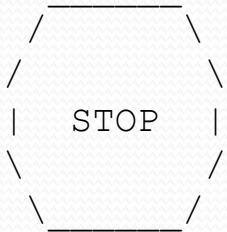
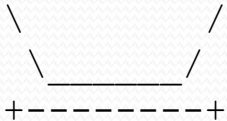
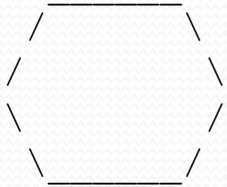
Drawing complex figures with static methods

reading: 1.5

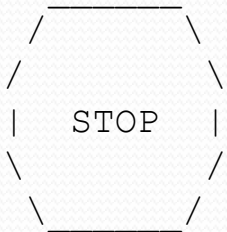
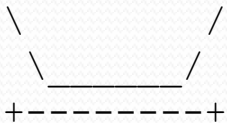
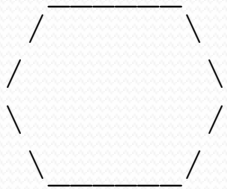
(Ch. 1 Case Study: DrawFigures)

Static methods question

- Write a program to print these figures using methods.



Development strategy



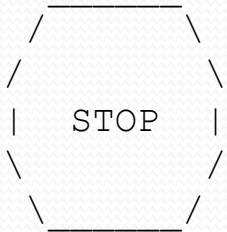
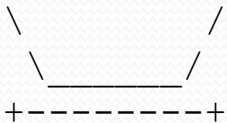
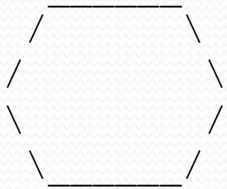
First version (unstructured):

- 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

```
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();
        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("+-----+");
    }
}
```

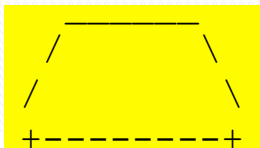
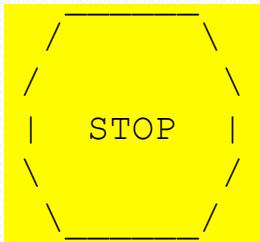
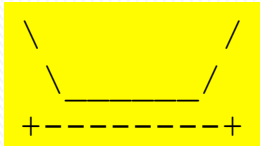
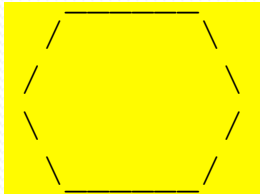
Development strategy 2



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

Program version 2

```
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();
    }
    ...
}
```

Program version 2, cont'd.

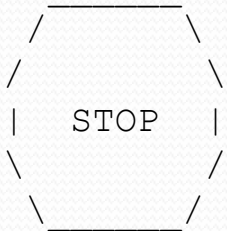
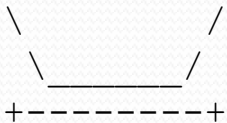
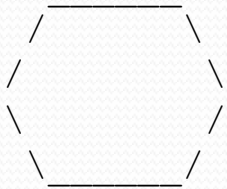
...

```
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("+-----+");  
}
```

```
}
```

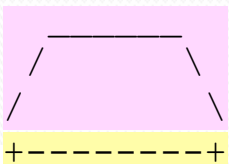
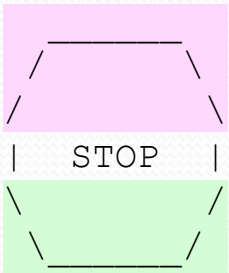
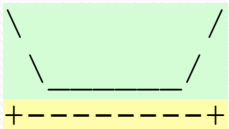
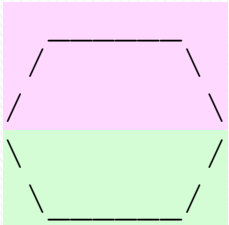

Development strategy 3



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

```
// Suzy 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 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();
    }
}
```

Program version 3, cont'd.

...

// 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("+-----+");  
}
```

```
}
```