# Topic 3
# static Methods and Structured Programming

"The cleaner and nicer the program,

the faster it's going to run.

And if it doesn't, it'll be easy to make it fast."

-Joshua Bloch

Based on slides by Marty Stepp and Stuart Reges
from http://www.buildingjavaprograms.com/

# Clicker 1

- What is the name of the method that is called when a Java program starts?

A. main

B. static

C. void

D. println

E. class

# 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. */
```

# 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.

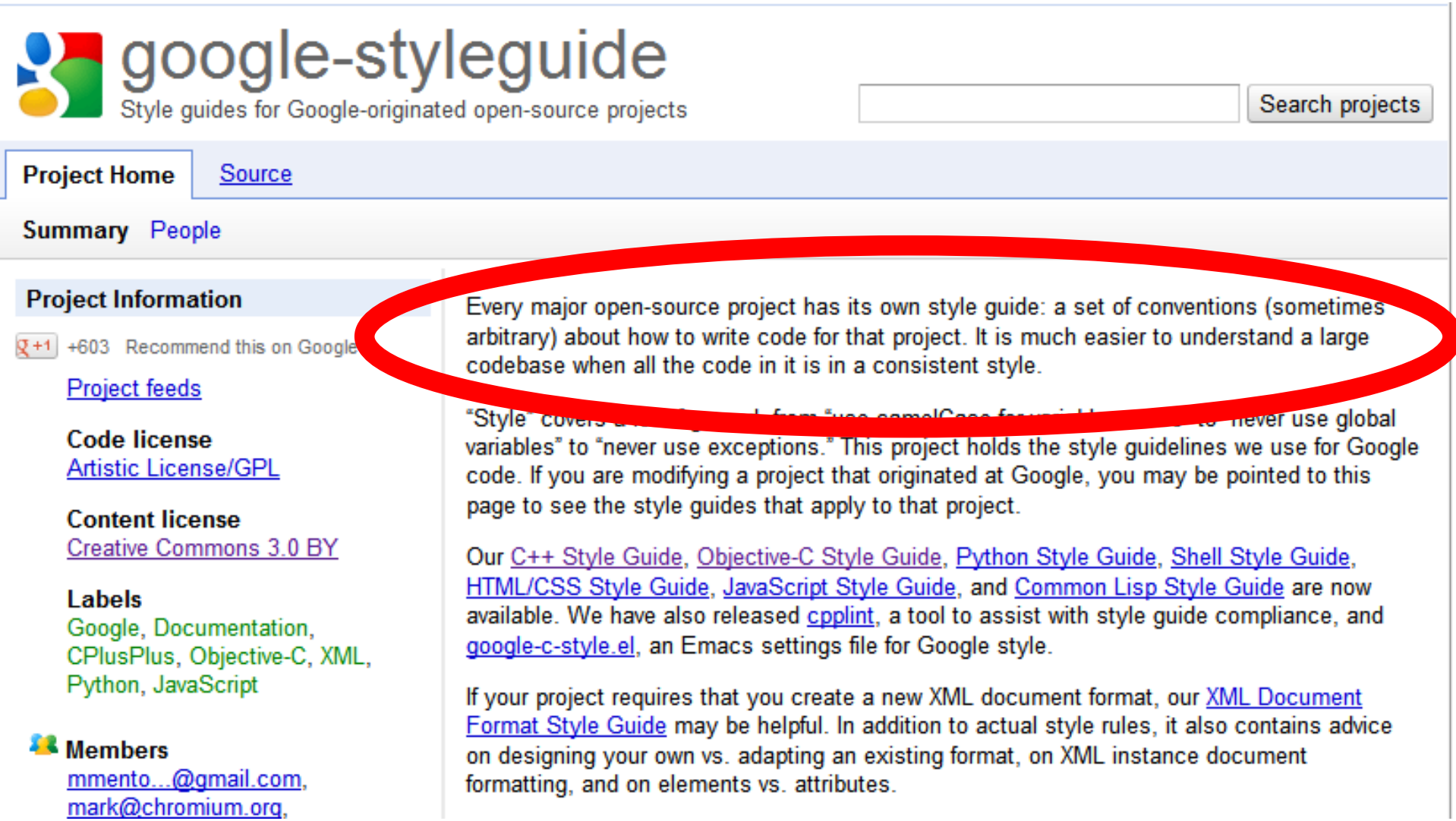
# Comments example

```
/* 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");
    }
}
```

# Program Hygiene - a.ka. Style

- Provide a structure to the program
- Eliminate redundant code
- Use spaces judiciously and **consistently**
- Indent properly
- Follow the naming conventions
- Use comments to describe code behavior
- Follow a brace style
- Good software follows a style guide
  - See links on assignment page

# Google C++ Style Guide



google-styleguide

Style guides for Google-originated open-source projects

Search projects

Project Home | Source

Summary | People

**Project Information**

+1 +603 Recommend this on Google

Project feeds

**Code license**
Artistic License/GPL

**Content license**
Creative Commons 3.0 BY

**Labels**
Google, Documentation, CPlusPlus, Objective-C, XML, Python, JavaScript

**Members**
mmento...@gmail.com, mark@chromium.org,

Every major open-source project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style.

"Style" covers [illegible] never use global variables" to "never use exceptions." This project holds the style guidelines we use for Google code. If you are modifying a project that originated at Google, you may be pointed to this page to see the style guides that apply to that project.

Our C++ Style Guide, Objective-C Style Guide, Python Style Guide, Shell Style Guide, HTML/CSS Style Guide, JavaScript Style Guide, and Common Lisp Style Guide are now available. We have also released cpplint, a tool to assist with style guide compliance, and google-c-style.el, an Emacs settings file for Google style.

If your project requires that you create a new XML document format, our XML Document Format Style Guide may be helpful. In addition to actual style rules, it also contains advice on designing your own vs. adapting an existing format, on XML instance document formatting, and on elements vs. attributes.

http://code.google.com/p/google-styleguide/

# Google C++ Style Guide

## Local Variables

▽ Place a function's variables in the narrowest scope possible, and initialize variables in the declaration.

C++ allows you to declare variables anywhere in a function. We encourage you to declare them in as local a scope as possible, and as close to the first use as possible. This makes it easier for the reader to find the declaration and see what type the variable is and what it was initialized to. In particular, initialization should be used instead of declaration and assignment, e.g.

```
int i;
i = f();      // Bad -- initialization separate from declaration.
```

```
int j = g();  // Good -- declaration has initialization.
```

```
vector<int> v;
v.push_back(1);  // Prefer initializing using brace initialization.
v.push_back(2);
```

```
vector<int> v = {1, 2};  // Good -- v starts initialized.
```

# Why Worry About Program Hygiene ?

- Programmers build on top of other's code all the time.
  - Computer Scientists and Software developers spend as much time maintaining code as they do creating new code
  - You shouldn't waste time deciphering what a method does.

- You should spend time on thinking and coding. You should **NOT** be wasting time looking for that missing closing brace.

- "Code is read more often than it is written."
  - *Guido Van Rossum* (Creator of the Python Language)

# 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.
  - Mix ingredients for frosting.
  - ...

# Problems with algorithms

- *lack of structure*: Many small 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 oven temperature.
  - 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 solution 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

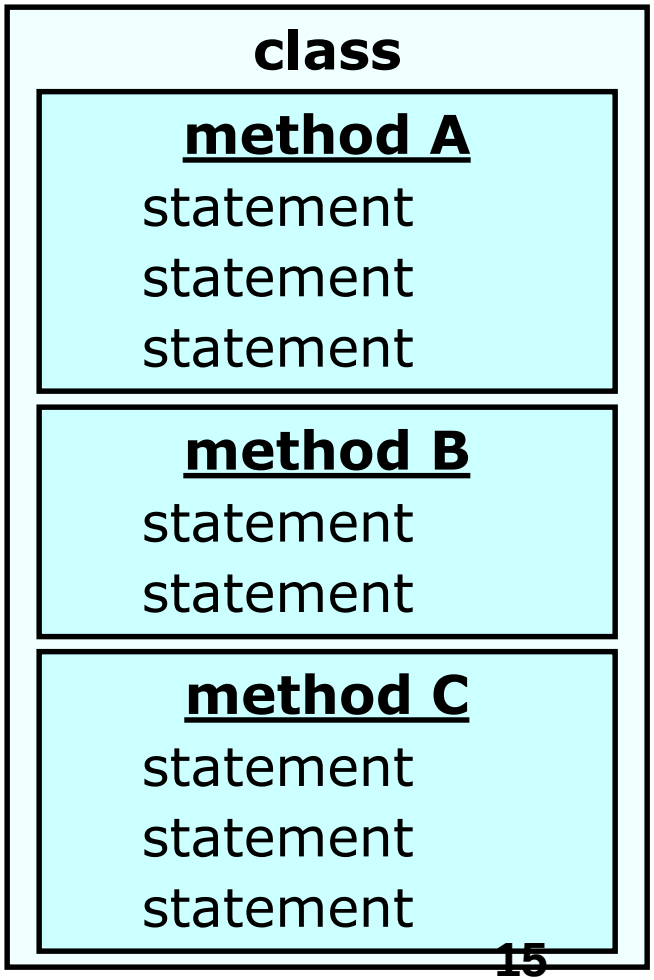
```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies {

    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("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("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
  - a way to *manage complexity*

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



- Building complex systems is hard
- Some of the most complex systems are software systems

| YEAR | COMPANY | OUTCOME (COSTS IN US $) |
|---|---|---|
| 2005 | Hudson Bay Co. [Canada] | Problems with inventory system contribute to $33.3 million* loss. |
| 2004–05 | UK Inland Revenue | Software errors contribute to $3.45 billion* tax-credit overpayment. |
| 2004 | Avis Europe PLC [UK] | Enterprise resource planning (ERP) system canceled after $54.5 million† is spent. |
| 2004 | Ford Motor Co. | Purchasing system abandoned after deployment costing approximately $400 million. |
| 2004 | J Sainsbury PLC [UK] | Supply-chain management system abandoned after deployment costing $527 million.† |
| 2004 | Hewlett-Packard Co. | Problems with ERP system contribute to $160 million loss. |
| 2003–04 | AT&T Wireless | Customer relations management (CRM) upgrade problems lead to revenue loss of $100 million |
| 2002 | McDonald's Corp. | The Innovate information-purchasing system canceled after $170 million is spent. |
| 2002 | Sydney Water Corp. [Australia] | Billing system canceled after $33.2 million† is spent. |
| 2002 | CIGNA Corp. | Problems with CRM system contribute to $445 million loss. |
| 2001 | Nike Inc. | Problems with supply-chain management system contribute to $100 million loss. |
| 2001 | Kmart Corp. | Supply-chain management system canceled after $130 million is spent. |

- http://spectrum.ieee.org/computing/software/why-software-fails

# Using static methods

1. **Design** the algorithm.
   - Look at the structure, and which commands are repeated.
   - Decide what are the important overall tasks.
   - Good programmers do this BEFORE writing any code

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:

  ***<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
```

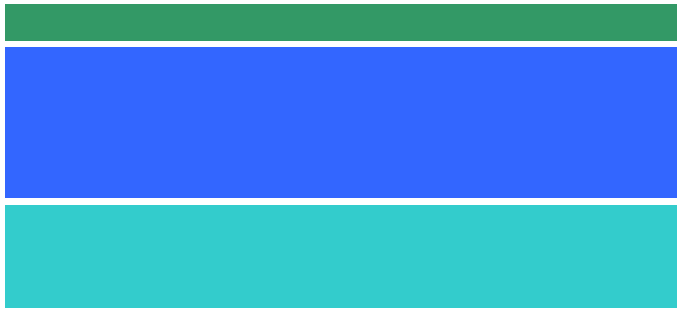
# Final cookie program

```
// 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 cookie 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.");
    }
}
```

# Summary: Why methods?

- Makes code easier to read by capturing the structure of the program
  - `main` should be a good summary of the program

```
public static void main(String[] args) {
```



```
}
```

**Note:** Longer code doesn't necessarily mean worse code!!!

```
public static void main(String[] args) {
```

```
}

public static ...                (...) {
```

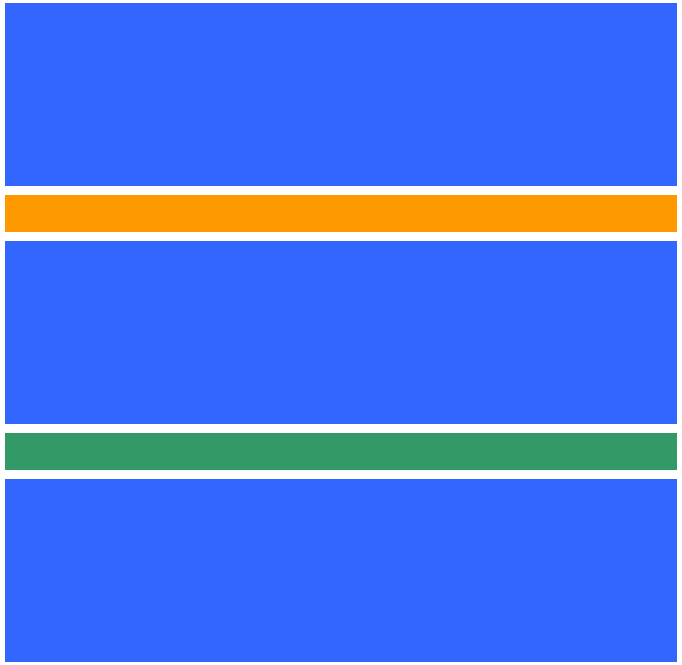
```
}

public static ...                (...) {
```

```
}
```

# Summary: Why methods?

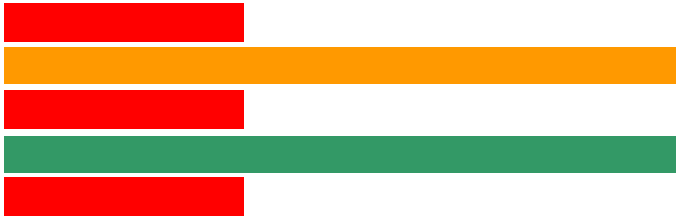
- Eliminate redundancy

```
public static void main(String[] args) {
```



```
}
```

```
public static void main(String[] args) {
```



```
}
```

```
public static ...
```

```
(...) {
```

```
}
```

# 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("Dor
    }

    ...
}
```

```
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.");
}
```

# Clicker 2 - How many lines of output with visible characters does the following program produce?

```
public class MethodCalls {
    public static void main(String[] args ) {
        a();
        b();
        c();
        c();
    }

    public static void a() {    System.out.println("A");    }

    public static void b() {
        System.out.println("B");
        a();
        System.out.println("B");
    }

    public static void c() {
        a();
        b();
        System.out.println("C");
        b();
    }
}
```

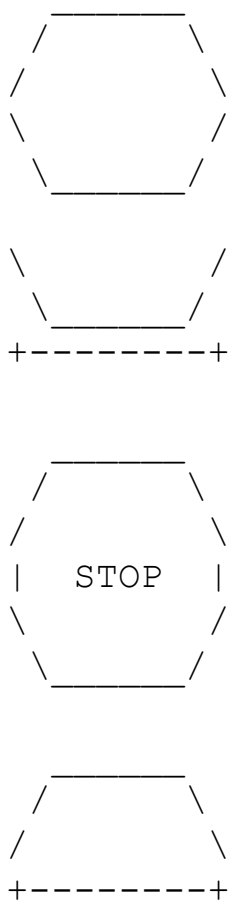
A. 3  B. 4  C. 8  D. 12  E. 20

# Drawing complex figures with static methods

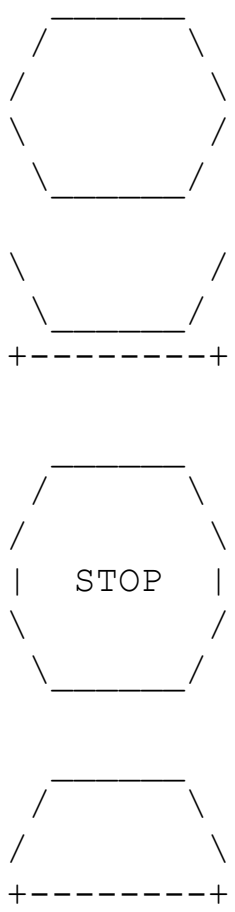
**reading: 1.5**
(Ch. 1 Case Study:
`DrawFigures`)

# Static methods question

- Write a program to print these figures.



# 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.

**Clicker 3 - Are there repeated sections of output for this program?**

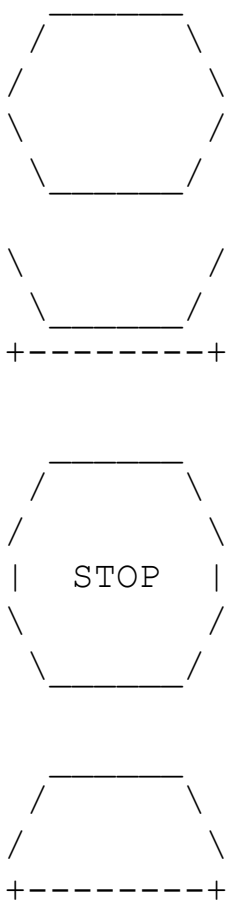
**A. No**

**B. Yes**

# 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

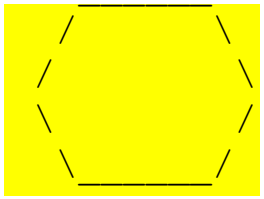


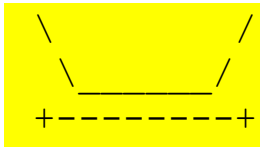

Second version (structured, with redundancy):

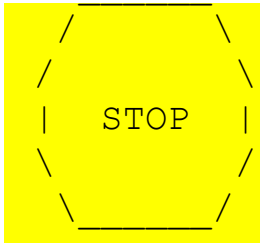
Identify the structure of the output.

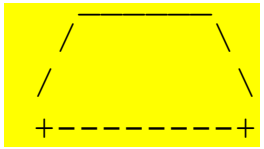
Divide the `main` method into static methods based on this structure.

# 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
```

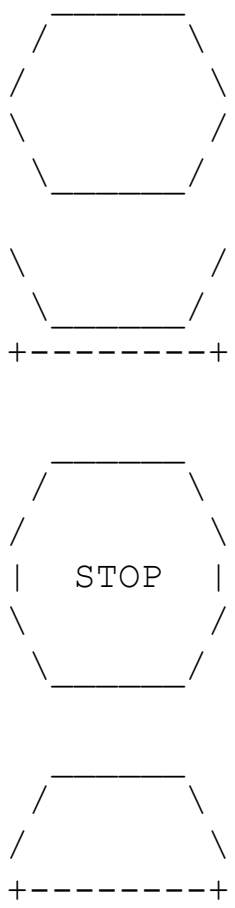
# Program version 2

```
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

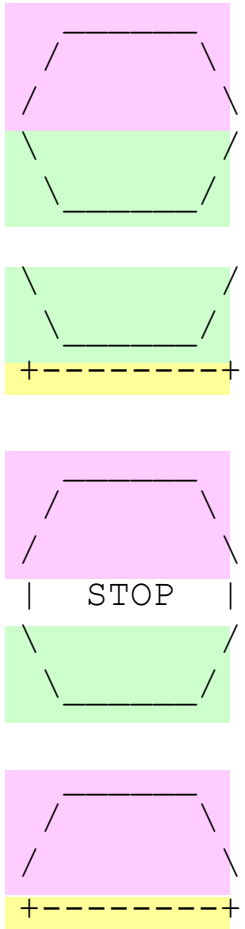



Third version (structured, without redundancy):

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