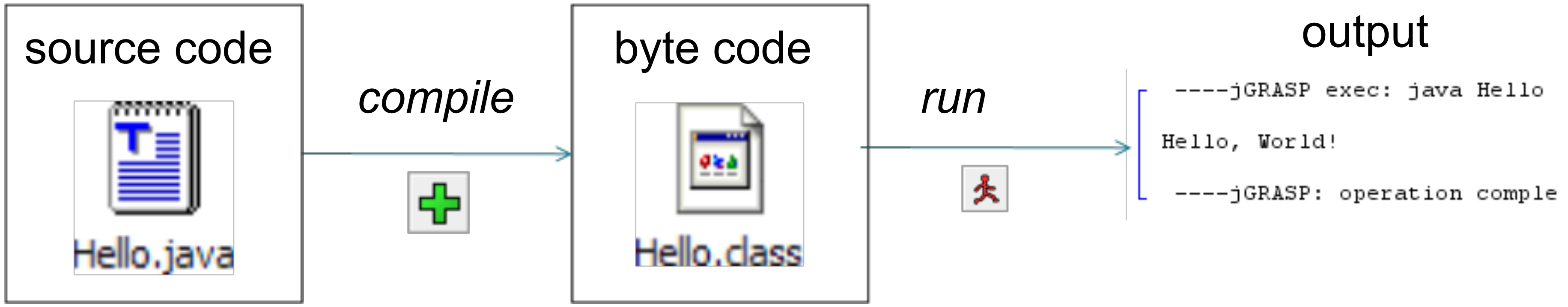
Entering the world of Javatar

FoxTrot
by Bill Amend
WHAT'S GOING ON? I'M 10 FEET TALL AND BLUE!
AND MY HAIR HAS A LONG BRAID WITH A USB CONNECTOR ON THE END!
I WONDER WHAT WOULD HAPPEN IF I PLUGGED IT IN TO THAT COMPUTER...
IT'S LIKE OUR MINDS ARE ONE! I CAN SPEAK ITS LANGUAGE! THIS IS AWESOME!
OOO – IT WANTS ME TO WRITE SOME OBJECT-ORIENTED CODE...
I HAD THAT "JAVATAR" DREAM AGAIN LAST NIGHT.
YOU SAY THAT LIKE I'M SUPPOSED TO BE JEALOUS.
www.foxtrot.com
twitter: @billamend
©2010 Bill Amend / Dist. by Universal Press Syndicate
GOCOMICS.
GET A LAUGH!

Compiling/running programs

1. *Write* it.
   - **code** or **source code**: The set of instructions in a program.
2. *Compile* it.
   - **compile**: Translate a program from one language to another.
   - **byte code**: The Java compiler converts your code into a format named *byte code* that runs on many computer types.
3. *Run* (execute) it.
   - **output**: The messages printed to the user by a program.



```
 ----jGRASP exec: java Hello
Hello, World!
 ----jGRASP: operation comple
```

Syntax

- Set of legal structures and commands that can be used in a language
  - Semicolons ;
  - curly braces { }
  - **Identifiers**: names of the commands and such…
- Compiler checks syntax, gives errors

```
Hello.java:3: ';' expected
              System.out.println("Hello, world!!")
                                                  ^

1 error
```

Names and identifiers

- You must give your program a name.

```
public class GangstaRap {
```

  - Naming convention: capitalize each word (e.g. `MyClassName`)
  - Your program's file must match exactly (`GangstaRap.java`)
    - includes capitalization (Java is "case-sensitive")

- **identifier**: A name given to an item in your program.
  - must start with a letter or _ or $
  - subsequent characters can be any of those or a number
    - legal: `_myName` `TheCure` `ANSWER_IS_42` `$bling$`
    - illegal: `me+u` `49ers` `side-swipe` `Ph.D's`

Keywords

- **keyword**: An identifier that you cannot use because it already has a reserved meaning in Java.

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	**public**	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	**static**	**void**
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

System.out.println

- A statement that prints a line of output on the console.
  - pronounced "print-linn"
  - sometimes called a "println statement" for short
  - A newline is included at the end of each println

- Two ways to use System.out.println :
  - System.out.println("**text**");
    Prints the given message as output.

  - System.out.println();
    Prints a blank line of output.

Strings

- Sequence of characters
- Enclosed in double quotes "This is enclosed in double quotes"
- Some special characters must be *escaped* using the backslash: '****'
  - For example: `\"  \t  \n  \\`
    - Double quote: `\"`
    - Tab: `\t`
    - Newline: `\n`

Reality check

- What is the output of the following println statements?

```
System.out.println("\ta\tb\tc");
System.out.println("\\\\");
System.out.println("'");
System.out.println("\"\"\"");
System.out.println("C:\nin\the downward spiral");
```

- Write a println statement to produce this output:

```
/ \ // \\ /// \\\
```

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”
Martin Golding

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

- Use them to explain the tricky bits of your code
- Lets others know what’s on your mind
- Use appropriately: if something is not obvious from the syntax, it’s best to add a comment
- Can use both flavors: `/* … */` and `//`

Good:

```
// Prints a greeting
```

Bad:

```
/* This is my super awesome program
that uses the println statement to go
ahead and display a friendly message
to the user because it's a convention
that was started long ago.*/
```

Static Methods & Decomposition

Subset of the Supplement Lesson slides from: Building Java Programs, Chapter 1
by Stuart Reges and Marty Stepp (http://www.buildingjavaprograms.com/)

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 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 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

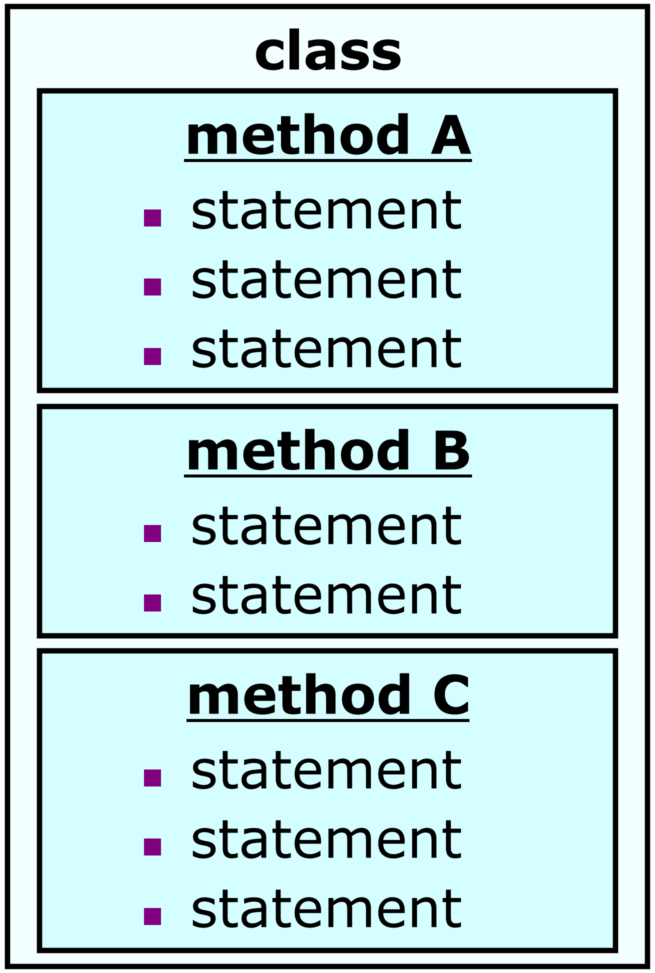
```
public class BakeCookies {
    public static void main(String[] args) {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
        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

- 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:

  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 StartToday20110912 {
     public static void main(String[] args) {

          first();
          cheer();
          first();
          cheer();
          second();
     }

     public static void cheer() {
          System.out.println("G!H!S!");
     }

     public static void first() {
          System.out.println("Y-E-L-L Bulldogs Yell!");
     }

     public static void second() {
          System.out.println("Everybody Screams");
     }
  }
```

Output:

```
Y-E-L-L Bulldogs Yell!
G!H!S!
Y-E-L-L Bulldogs Yell!
G!H!S!
Everybody Screams!
```

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

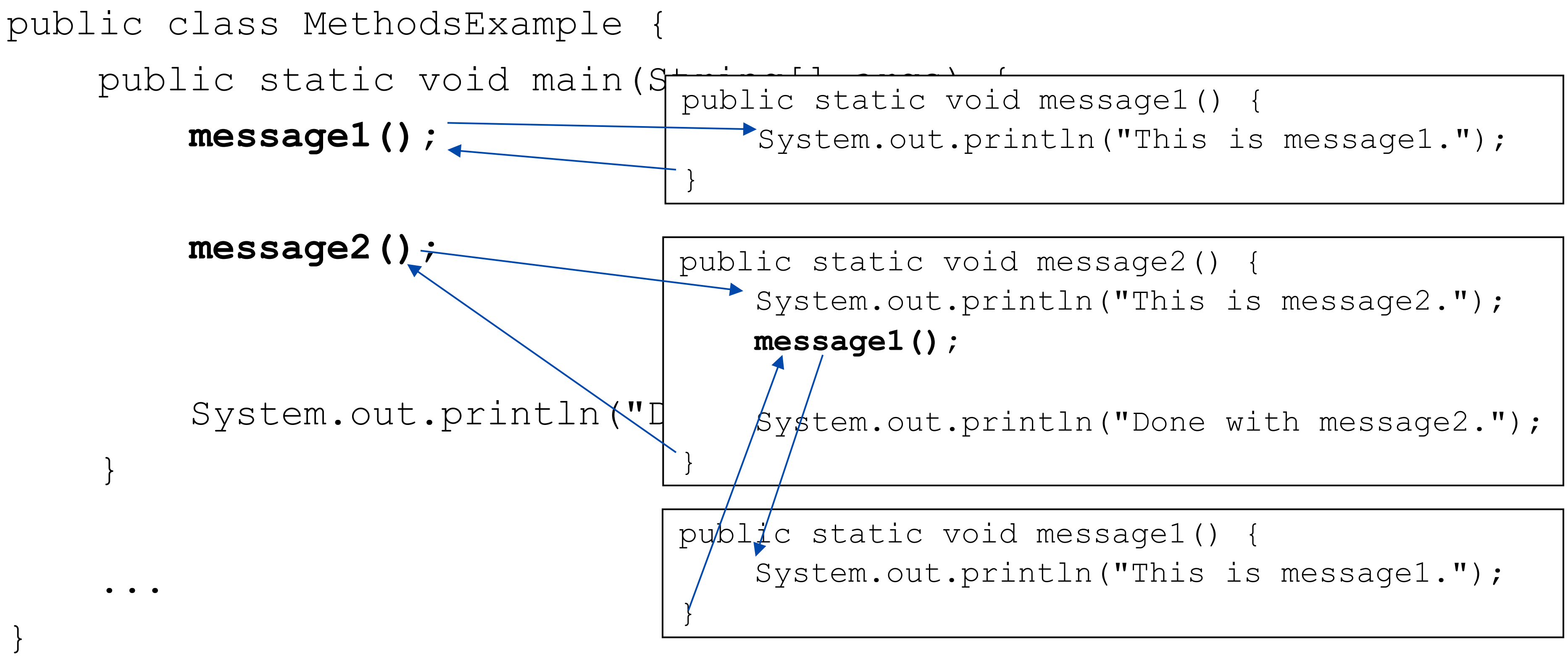
```
public class MethodsExample {
    public static void main(String[] args) {
        message1();
        message2();
        System.out.println("Done with main.");
    }

    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.");
    }
}
```

- 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(S
        message1();

        message2();


        System.out.println("D
    }

    ...
}
```

```
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

Static methods question

- Write a program to print these figures using methods.

```
   ______
  /      \
 /        \
 \        /
  \______/

 \        /
  \______/
 +--------+

   ______
  /      \
 /        \
 |  STOP  |
 \        /
  \______/

   ______
  /      \
 /        \
 +--------+
```

Development strategy

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+


  ______
 /      \
/        \
|  STOP  |
\        /
 \______/


  ______
 /      \
/        \
+--------+
```

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

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

Second version (structured, with redundancy):

- Identify the structure of the output.

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

Output structure

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

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

```
  ______
 /      \
/        \
\        /
 \______/

\        /
 \______/
+--------+

  ______
 /      \
/        \
|  STOP  |
\        /
 \______/

  ______
 /      \
/        \
+--------+
```

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

```
  ______
 /      \
/        \
\        /
 \______/
```

```
\        /
 \______/
+--------+
```

```
  ______
 /      \
/        \
|  STOP  |
\        /
 \______/
```

```
  ______
 /      \
/        \
+--------+
```

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