Building Java Programs

A Back to Basics Approach

CS 210

CHAPTER 1

INTRODUCTION TO JAVA PROGRAMMING

Please download the PPT, and use Slide Show for a better viewing experience

Winnie Li

Topics will be covered

CS 210

- What is Computer Science?
- What is Programming?
- Basic Java Program (with `println` statements)
  - Structure
  - Identifier names
  - Syntax
  - Strings
  - Comments
- Static Methods and Algorithms
  - Structured Version
  - Unstructured With Redundancy Version
  - Unstructured Without Redundancy Version

What is Computer Science?

Computers? Sciences? Programming?
Late lonely nights in front of computer? Or…

- Computer Science (from Wikipedia)

  The study of theoretical foundations of information and computation and their implementation and application in computer systems.

- Algorithmic Thinking

  Algorithm is… a step-by-step procedure for solving a problem or accomplishing some end *especially by a computer*

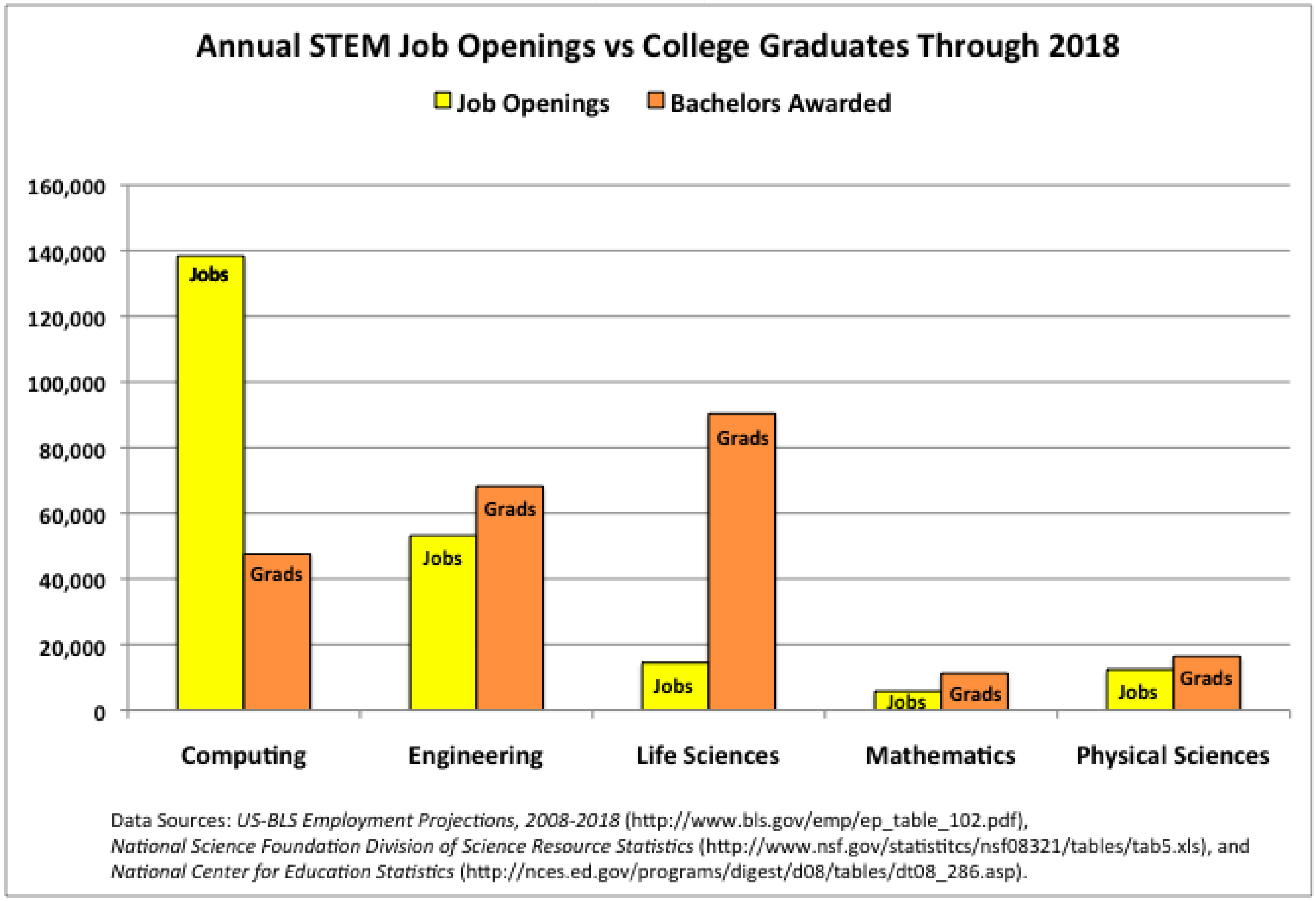
- Computer Engineering (CSE)
  - Overlap with CS and EE; emphasizes hardware

Many subfields

- Graphics, Computer Vision
- Artificial Intelligence
- Scientific Computing
- Robotics
- Databases, Data Mining
- User Interface
- Natural Language Processing …

CS Job Market

CS 210



Data Sources: *US-BLS Employment Projections, 2008-2018* (http://www.bls.gov/emp/ep_table_102.pdf), *National Science Foundation Division of Science Resource Statistics* (http://www.nsf.gov/statistitcs/nsf08321/tables/tab5.xls), and *National Center for Education Statistics* (http://nces.ed.gov/programs/digest/d08/tables/dt08_286.asp).

What is programming?

- **program**: A set of instructions to be carried out by a computer.

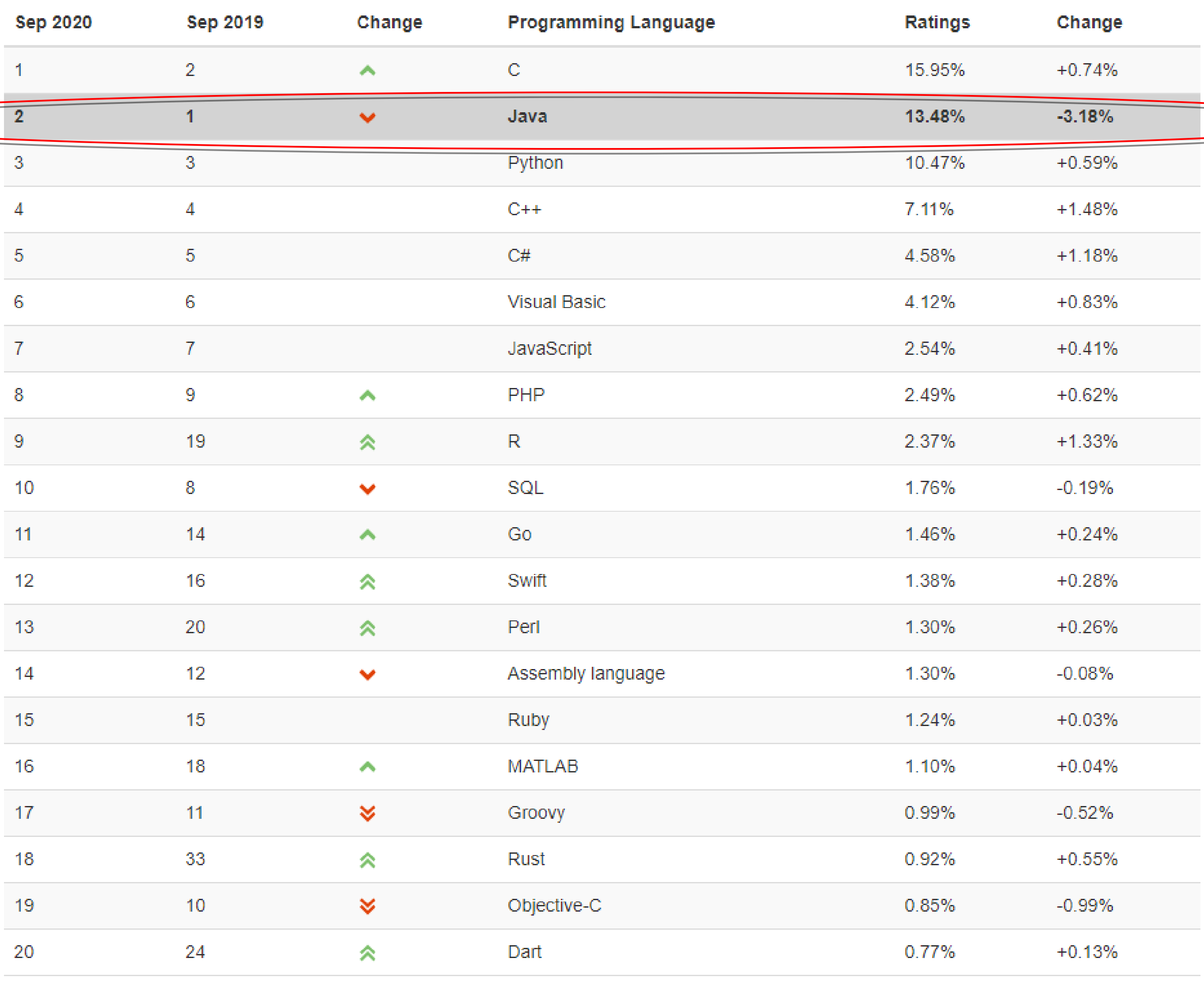
- **program execution**: The act of carrying out the instructions contained in a program.

- **programming language**: A systematic set of rules used to describe computations in a format that is editable by humans.
  - This textbook teaches programming in a language named Java.

Some modern languages

- *procedural languages*: programs are a series of commands
  - **Pascal** (1970): designed for education
  - **C** (1972): low-level operating systems and device drivers
- *functional programming*: functions map inputs to outputs
  - **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
  - **Smalltalk** (1980): first major object-oriented language
  - **C++** (1985): "object-oriented" improvements to C
    - successful in industry; used to build major OSes such as Windows
  - **Java** (1995): designed for embedded systems, web apps/servers
    - Runs on many platforms (Windows, Mac, Linux, cell phones...)
    - The language taught in this textbook

TIOBE Programming Community Index



Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	15.95%	+0.74%
2	**1**	↓	**Java**	**13.48%**	**-3.18%**
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	↑	PHP	2.49%	+0.62%
9	19	⇈	R	2.37%	+1.33%
10	8	↓	SQL	1.76%	-0.19%
11	14	↑	Go	1.46%	+0.24%
12	16	⇈	Swift	1.38%	+0.28%
13	20	⇈	Perl	1.30%	+0.26%
14	12	↓	Assembly language	1.30%	-0.08%
15	15		Ruby	1.24%	+0.03%
16	18	↑	MATLAB	1.10%	+0.04%
17	11	⇊	Groovy	0.99%	-0.52%
18	33	⇈	Rust	0.92%	+0.55%
19	10	⇊	Objective-C	0.85%	-0.99%
20	24	⇈	Dart	0.77%	+0.13%

Myths

- CS/CSE majors spend lives in dark offices alone writing code
- CS/CSE is only about lines and lines of details
- CS/CSE is not creative
- CS/CSE is only for people that want to work at large software companies
- All the jobs moved to Bangalore
- …

See more at: http://www.cs.washington.edu/whycse

Should you take this course?

- NO
  - “I hate computers.”
  - “I don’t pay attention to details.”
    - Programming is fairly detail-oriented.
  - “I refuse to think logically.”
  - “I want to take an easy class.”
    - Hard for those who find difficulty in logical thinking and who don’t pay attention to details.

Should you take this course?

- PROBABLY NOT
  - “I want free gourmet meals and to make lots of money by working for Google.”
  - “World of Warcraft rocks hardcore!”
  - “Everyone, look at my Facebook farm!”
  - “I’ve programmed before, so I can just coast through this for an easy A!“

Should you take this course?

- YES
  - “I have to take this class.”
    - Is this the only reason? Are you pursuing the right major?
  - “I like to solve problems.”
  - “Computers and robots are going to take over the world. I want to befriend them so that my life will be spared.”

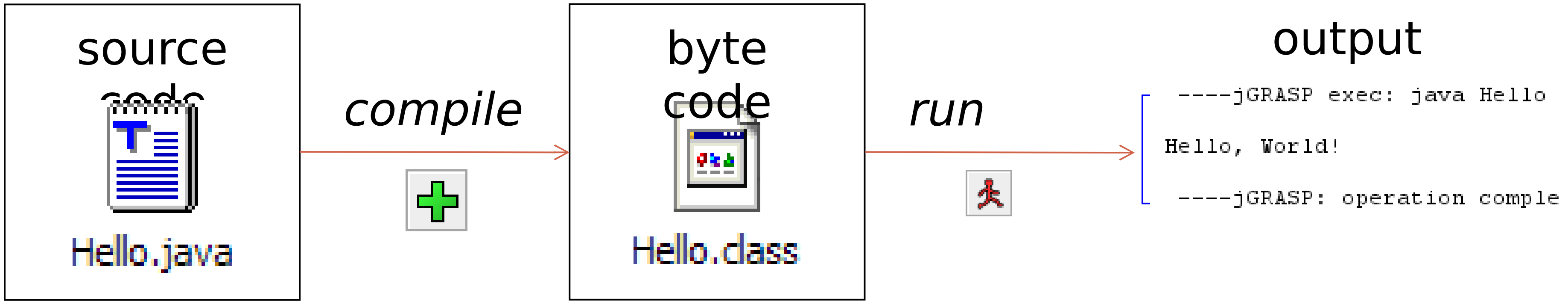
Basic Java programs with `println` statements

CS 210

STRUCTURE
IDENTIFIER NAMES
SYNTAX
STRINGS
COMMENTS

Compile/run a Java program

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
```

Example Java program

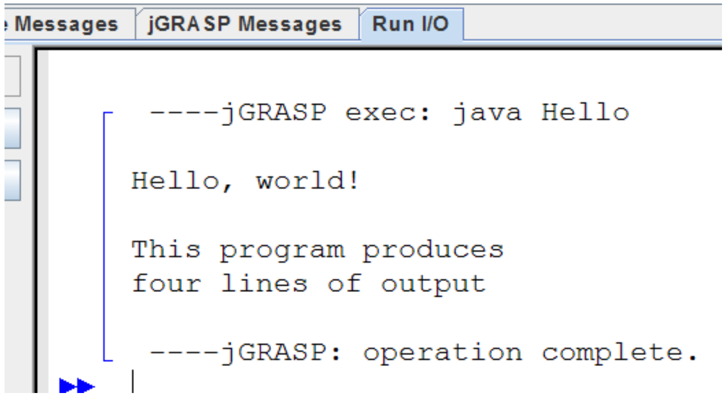
CS 210

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

- Its output:

```
Hello, world!

This program produces
four lines of output
```

- **console**: Text box into which the program's output is printed.



```
----jGRASP exec: java Hello

Hello, world!

This program produces
four lines of output

----jGRASP: operation complete.
```

Structure of a Java program

CS 210

```
public class Name {
    public static void main(String[] args) {
        statement;
        statement;
        ...
        statement;
    }
}
```

class: a program

method: a named group of statements

statement: a command to be executed

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

Names and identifiers

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
...
```

- You must give your program a name.

  `public class MyProgram {`

  - Naming convention: Capitalize each word (e.g. `MyClassName`)
  - Your program's file must match exactly (`MyProgram.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`
    - **Why?**

Keywords

CS 210

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
     ...
```

- **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

CS 210

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello,
          world!");
        …
```

- A statement that prints a line of output on the console.
  - pronounced "print-linn"
  - sometimes called a "println statement" for short

- Two ways to use `System.out.println` :

  - `System.out.println(`“**Your Message**`");`
    Prints the given message as output.

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

Syntax

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello,
          world!");
    }
}
```

- **syntax**: The set of legal structures and commands that can be used in a particular language.
  - Every basic Java statement ends with a semicolon `;`
  - The contents of a class or method occur between `{` and `}`

- **syntax error (compiler error)**: A problem in the structure of a program that causes the compiler to fail.
  - Missing semicolon
  - Too many or too few `{ }` braces
  - Illegal identifier for class name
  - Class and file names do not match… and many more

Syntax error example

```
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!")
4      }
5  }
```

- **Compiler output:**

```
Hello.java:2: <identifier> expected
    pooblic static void main(String[] args) {
           ^
Hello.java:3: ';' expected
}
^
2 errors
```

  - The compiler shows the line number where it found the error.
  - The error messages can be tough to understand!

Strings

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello,
          world!");
        …
```

- **string**: A sequence of characters to be printed.
  - Starts and ends with a " quote " character.
    - The quotes do not appear in the output.
  - Examples:

```
"hello"
"This is a string.  It's fairly long!"
```

- Restrictions:
  - May not span multiple lines.

```
"This is not
a legal String."
```

  - May not contain a " character.

```
"This is not a "legal" String either."
```

Escape sequences

- **escape sequence**: A special sequence of characters used to represent certain special characters in a string.

`\t`	tab character (Note: use FOUR spaces to represent)
`\n`	new line character
`\"`	quotation mark character
`\\`	backslash character

  - Example:

```
System.out.println("\\hello\nhow\tare \"you\"?\\\\");
```

  - Output:

```
\hello
how    are "you"?\\
```

Exercise 1

CS 210

- #1 - #5: What is the output of the following `println` statements?

```
1. System.out.println("a\tb\tc");
2. System.out.println("\\\\");
3. System.out.println("'");
4. System.out.println("\"\"\"");
5. System.out.println("C:/in\the downward spiral");
```

- #6: Write a `println` statement to produce this output.

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

Fill in your answers…

- Output of each `println` statement:

- `println` statement to produce the line of output:

Exercise 2

- What `println` statements will generate this output?

```
This program prints a
quote from the Gettysburg Address.

"Four score and seven years ago,
our 'fore fathers' brought forth on
this continent a new nation."
```

- What `println` statements will generate this output?

```
A "quoted" String is
'much' better if you learn
the rules of "escape sequences."

Also, "" represents an empty String.
Don't forget: use \" instead of " !
'' is not the same as "
```

Fill in your answers…

- `println` statements to generate the output:

- `println` statements to generate the output:

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

```
/* Maroon 5 Fan Student, CS 210, Winter 2012
   This program prints lyrics about ... Moves Like Jagger */

public class MovesLikeJagger {
    public static void main(String[] args) {
        // first verse
        System.out.println("Just shoot for the stars");
        System.out.println("If it feels right");
        System.out.println("Then aim for my heart");
       …
        System.out.println();

        // second verse
        System.out.println("Maybe it's hard");
        System.out.println("When you feel like you're broken
  and scarred");
        System.out.println("Nothing feels right");
       …
        System.out.println();
    }
}
```

Static methods and algorithms

CS 210

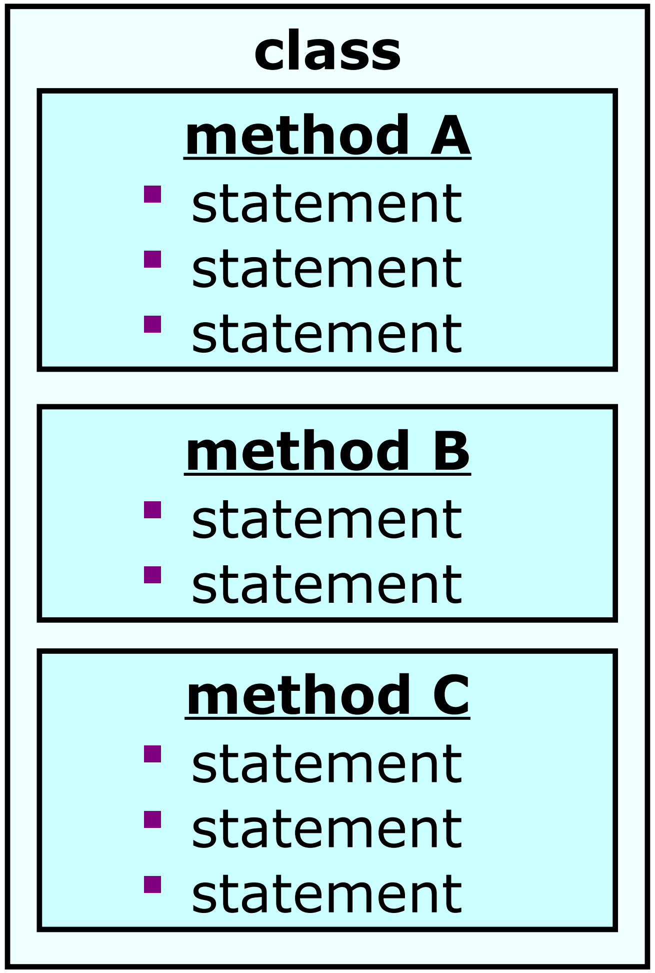
STATIC METHODS
ALGORITHMS
BAKE COOKIES EXAMPLE

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.

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 printClassSchedule() {
    System.out.println("CS 210 Fundamentals of CS I");
    System.out.println("MW 10:30am to 12:20pm");
}
```

Calling a method

Executes the method's code

- Syntax:

  name();

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

- Example:

  ```
  printClassSchedule();
  ```

  - Output:

  ```
  CS 210 Fundamentals of CS I
  MW 10:30am to 12:20pm
  ```

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:**

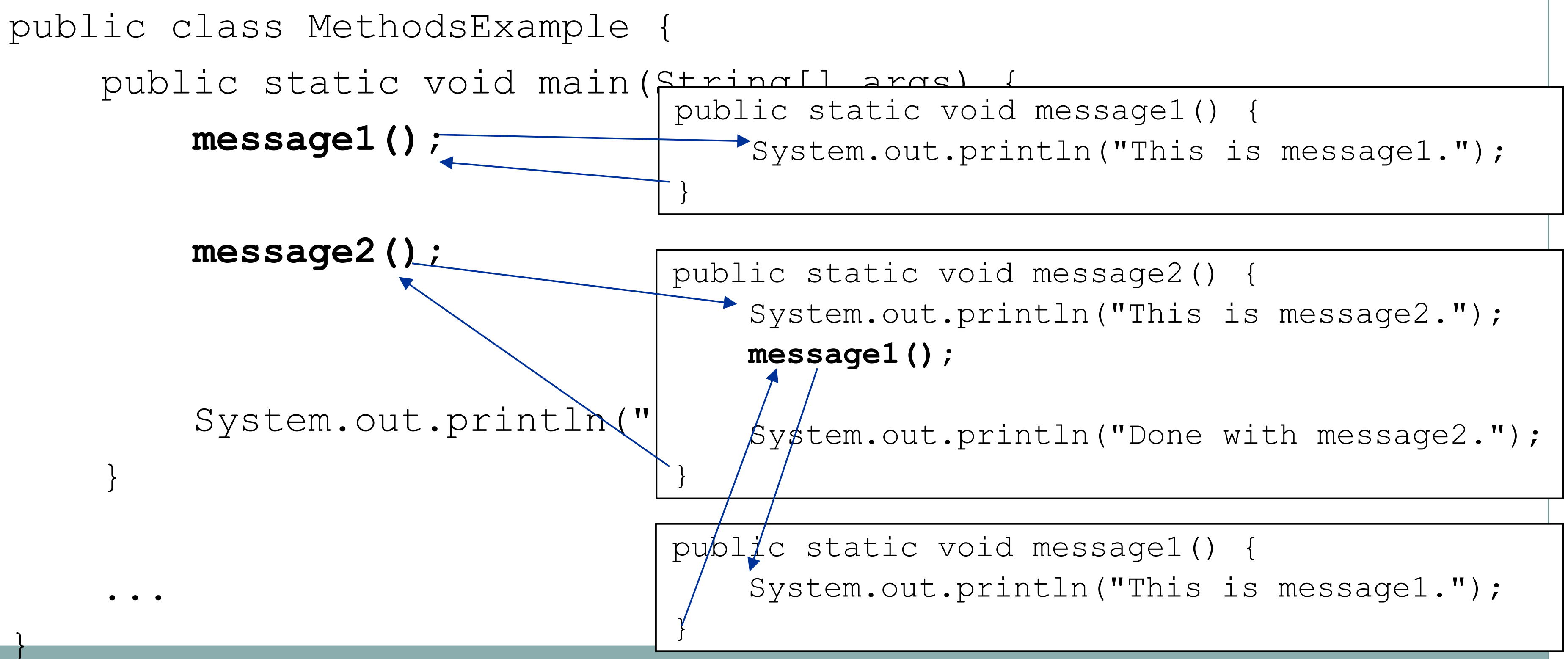
```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with main.
```

Control flow

CS 210

play Slide Show for better displays

- 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.");
}
```

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.
  - ...

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.");
    }
}
```

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.");
    }
}
```

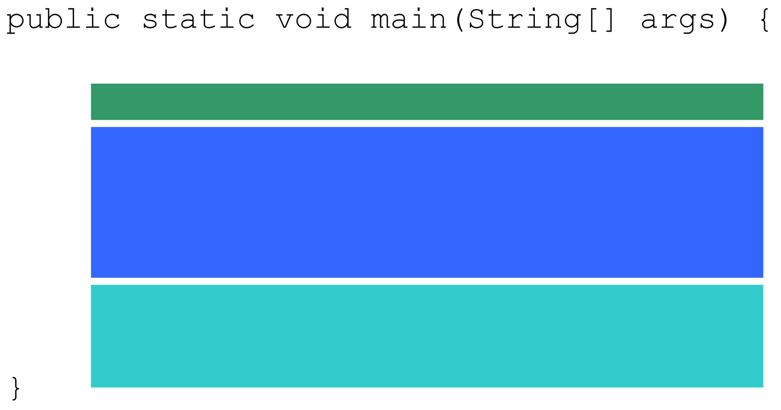
Final cookie program

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

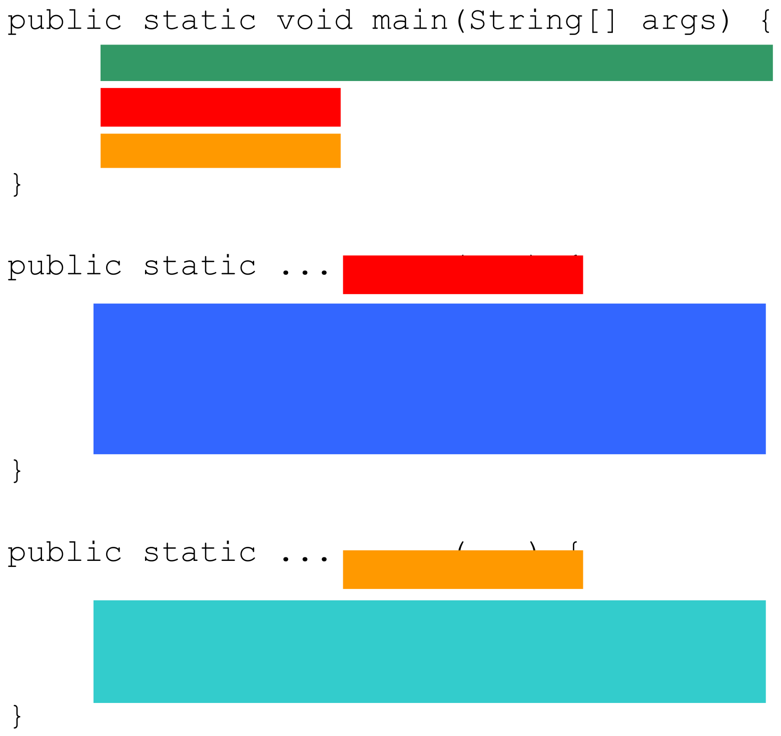
Why methods part I?

CS 210

- 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) {

}
```



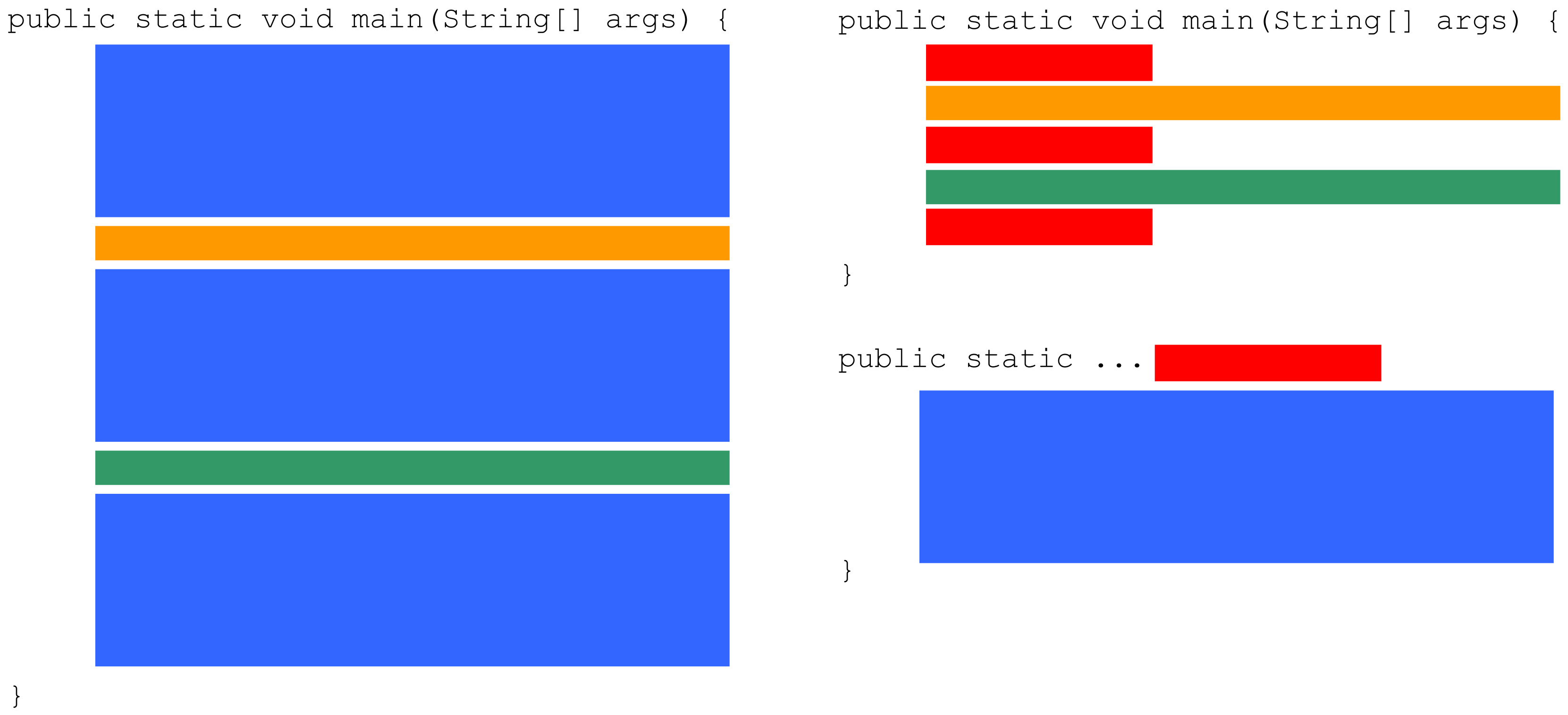
```
public static void main(String[] args) {

}

public static ...

}

public static ...

}
```

Note: Longer code doesn't necessarily mean worse code

Why methods part II?

CS 210

- Eliminate redundancy



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

}
```

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

}

public static ...

}
```

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

CS 210

UNSTRUCTURED VERSION
STRUCTURED WITH REDUNDANCY VERSION
STRUCTURED WITHOUT REDUNDANCY VERSION

Static methods question

- Write a program to print these figures using methods.

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

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

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

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

Development strategy

CS 210

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

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

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

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

<u>First version (unstructured):</u>

- 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

CS 210

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

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

  ______
 /      \
/        \
|  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

CS 210

```
  _______
 /       \
/         \
\         /
 \_______/

\         /
 \_______/
+---------+

  _______
 /       \
/         \
|  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.

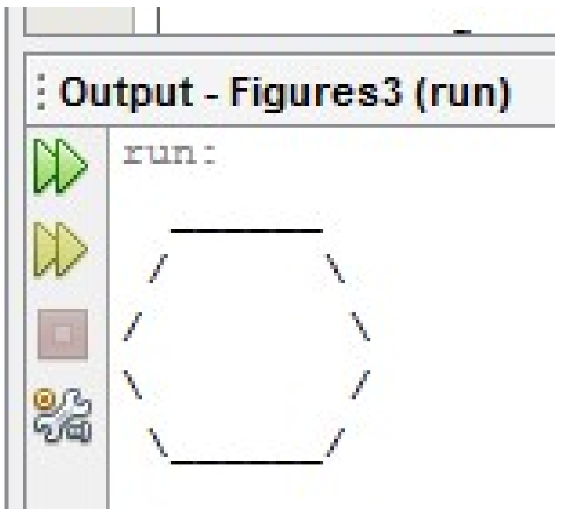
```
    ...
    // Draws a teacup figure.
    public static void teaCup() {
        eggBottom();
        line();
        System.out.println();
    }

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

Program version 3 – Implementation Steps

Step 3: Re[illegible] method at a time, keep [illegible] move on…

play Slide Show for better displays

```
// CS 210  Spring 2013 Section OAS
// Student Name
// Prints several figures, with methods for structure and redundancy.

public class Figures3 {
    public static void main(String[] args) {
      // TODO (fill in your                      xact figure as specified
   in the problem
    }
}
    // Draws a complete e
    public static void eg
        eggTop();
        eggBottom();
        System.out.printl
    }
   // 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(" \\______/");
    }
```



```
run:
   ______
  /      \
 /        \
 \        /
  \______/
```

Continue

A word about style

- Structure your code properly
- Eliminate redundant code
- Use spaces judiciously and **consistently**
- Indent properly
- Follow the naming conventions
- Use comments to describe code behavior

Why style?

- Programmers build on top of other's code all the time.
  - You should **NOT** waste time deciphering what a method does.
- You should spend time on thinking or coding.
  - You should **NOT** be wasting time looking for that missing closing brace.
- So CODE WITH STYLE!

The End

CS 210

CHAPTER 1

INTRODUCTION TO JAVA PROGRAMMING

Winnie Li