

# **Building Java Programs**

## **Chapter 1**

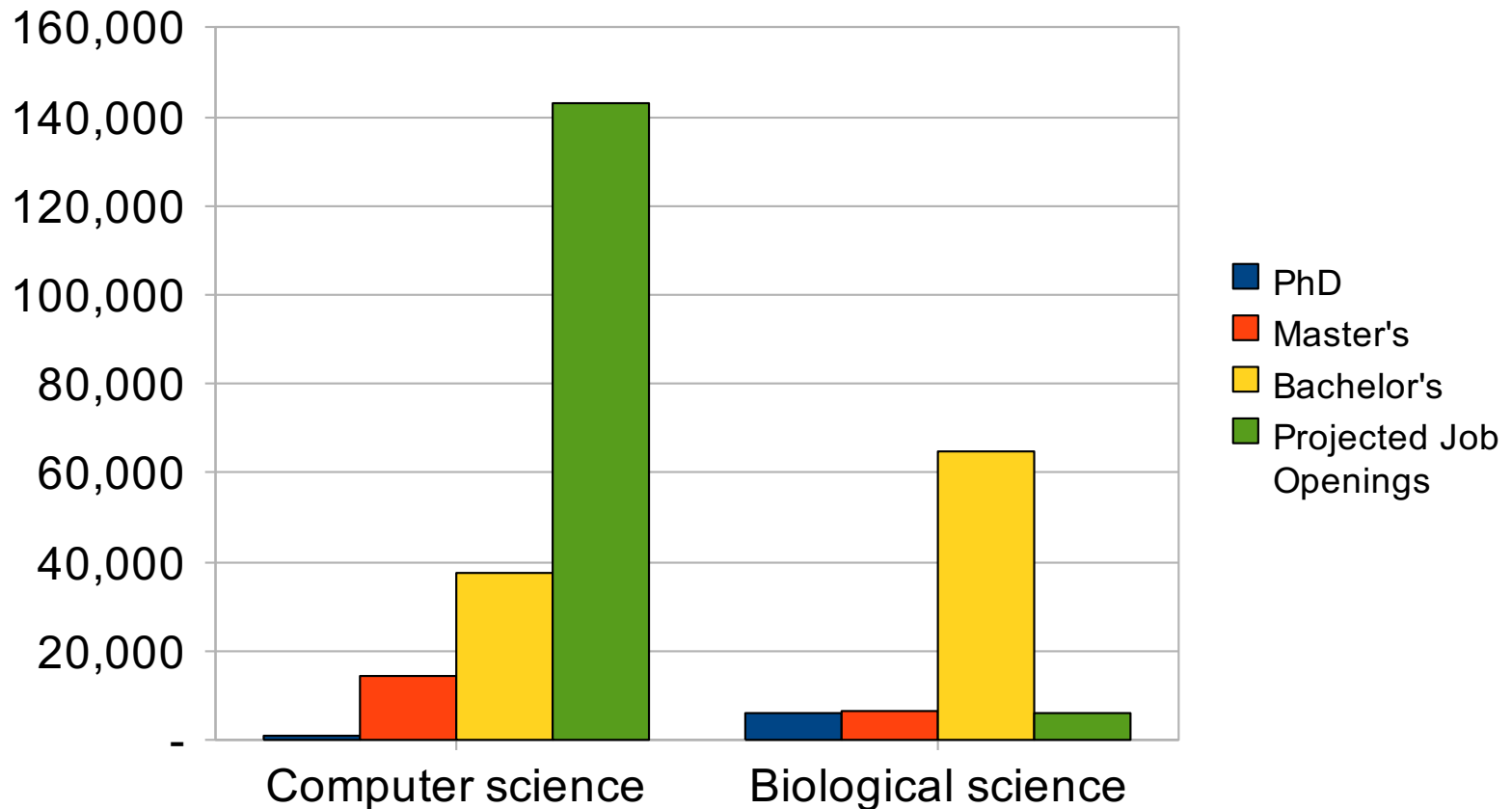
Introduction to Java Programming

Copyright (c) Pearson 2013.  
All rights reserved.

# What is computer science?

- Computer Science
  - The study of theoretical foundations of information and computation and their implementation and application in computer systems. -- Wikipedia
  - Many subfields
    - Graphics, Computer Vision
    - Artificial Intelligence
    - Scientific Computing
    - Robotics
    - Databases, Data Mining
    - Computational Linguistics, Natural Language Processing ...
- Computer Engineering
  - Overlap with CS and EE; emphasizes hardware

# The CS job market



SOURCES: Tabulated by National Science Foundation/Division of Science Resources Statistics; data from Department of Education/National Center for Education Statistics: Integrated Postsecondary Education Data System Completions Survey; and NSF/SRS: Sur

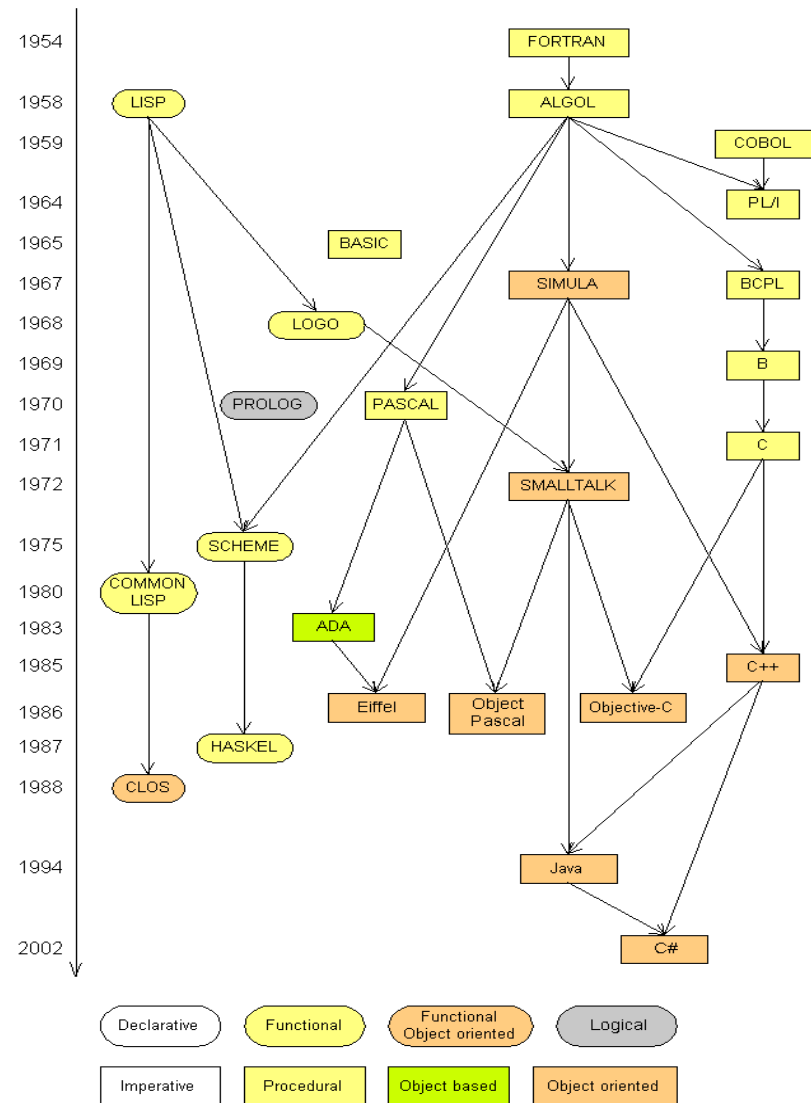
# What is programming?

- **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.



# Programming languages

- Some influential ones:
  - FORTRAN
    - science / engineering
  - COBOL
    - business data
  - LISP
    - logic and AI
  - BASIC
    - a simple language



# 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

# **Basic Java programs with `println` statements**

# Compile/run a program

## 1. Write it.

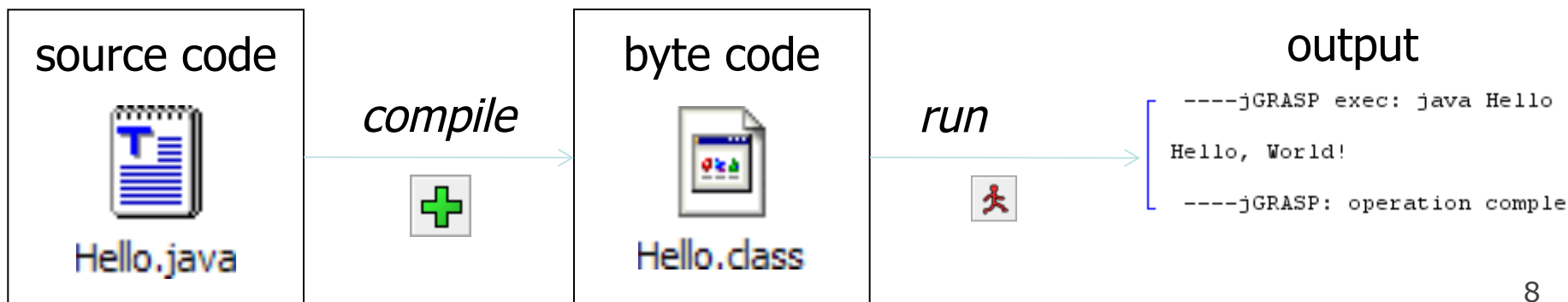
- **code** or **source code**: The set of instructions in a program.

## 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.





# A Java program

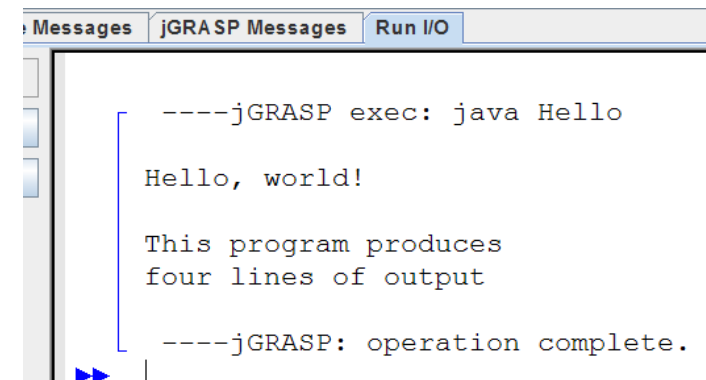
```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
        System.out.println();  
        System.out.println("This program produces");  
        System.out.println("four lines of output");  
    }  
}
```

- **Its output:**

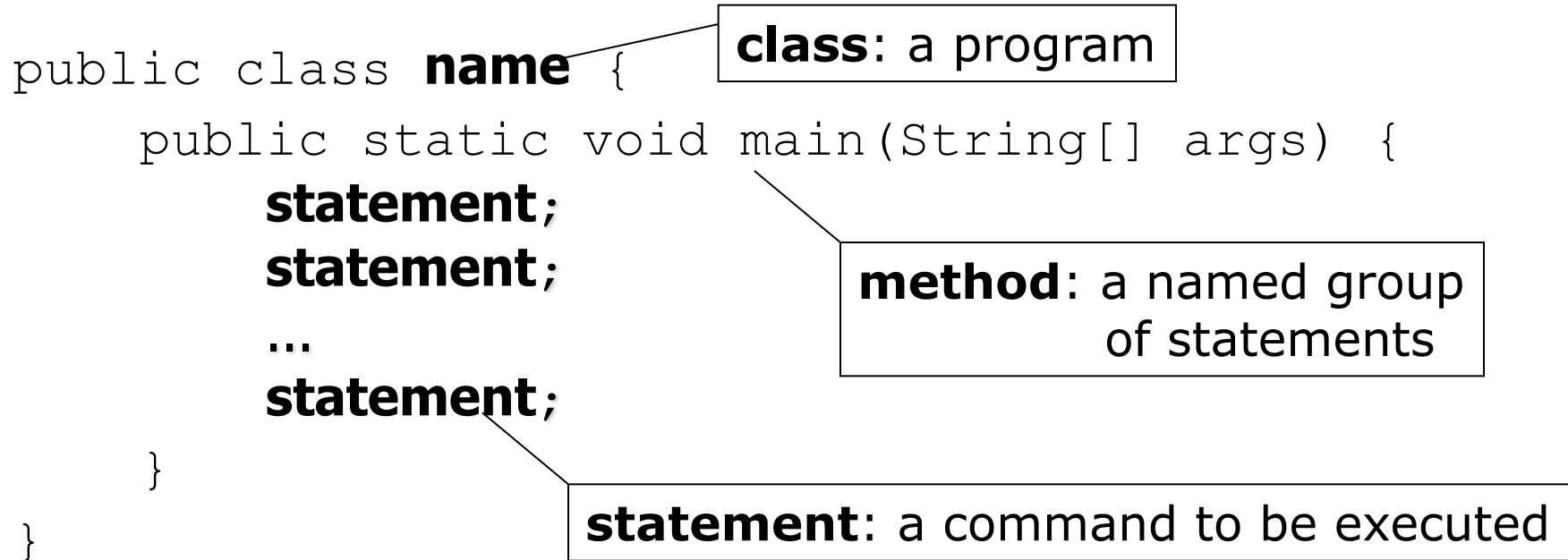
Hello, world!

This program produces  
four lines of output

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



# Structure of a Java program



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

# System.out.println

- 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("text") ;`  
Prints the given message as output.
  - `System.out.println() ;`  
Prints a blank line of output.

# 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	<b>public</b>	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	<b>static</b>	<b>void</b>
char	finally	long	strictfp	volatile
<b>class</b>	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

# Syntax

- **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
  - ...

# 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

- **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 very 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.

<code>\t</code>	tab character
<code>\n</code>	new line character
<code>\"</code>	quotation mark character
<code>\\</code>	backslash character

- **Example:**

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

- **Output:**

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

# Questions

- 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:

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

# Answers

- Output of each `println` statement:

```
      a      b      c
\\
'
""
C:
in      he downward spiral
```

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

```
System.out.println("/  \\  //  \\\\  ///  \\\\\\\");
```

# Questions

- 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 "

# Answers

- `println` statements to generate the output:

```
System.out.println("This program prints a");  
System.out.println("quote from the Gettysburg Address.");  
System.out.println();  
System.out.println("\Four score and seven years ago,");  
System.out.println("our 'fore fathers' brought forth on");  
System.out.println("this continent a new nation.\");
```

- `println` statements to generate the output:

```
System.out.println("A \"quoted\" String is");  
System.out.println("'much' better if you learn");  
System.out.println("the rules of \"escape sequences.\");  
System.out.println();  
System.out.println("Also, \"\" represents an empty String.");  
System.out.println("Don't forget: use \"\" instead of \" !");  
System.out.println("' is not the same as \");
```

# 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");
    }
}
```



# **Static methods**

# 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.");  
    }  
}
```

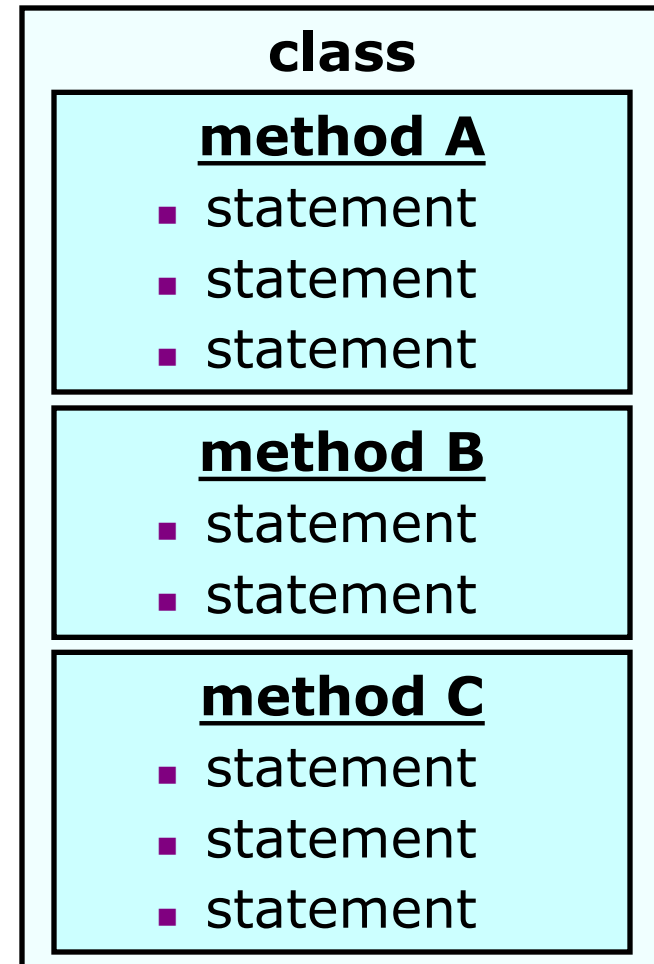
# Static methods

- **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 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.");  
    }  
}
```

- **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("Done with message1.");  
    }  
    ...  
}
```

```
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

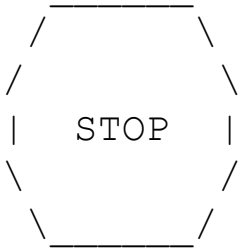
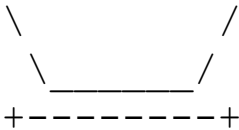
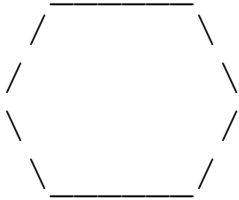
- Place statements into a static method if:
  - The statements are related structurally, and/or
  - The statements are repeated.
- You should not create static methods for:
  - An individual `println` statement.
  - Only blank lines. (Put blank `println`s in `main`.)
  - Unrelated or weakly related statements.  
(Consider splitting them into two smaller methods.)



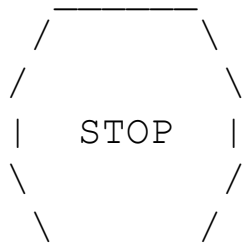
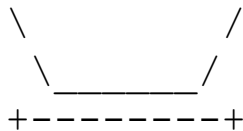
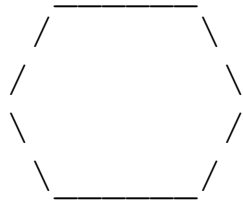
# **Drawing complex figures with static methods**

# Static methods question

- Write a program to print these figures using methods.



# Development strategy



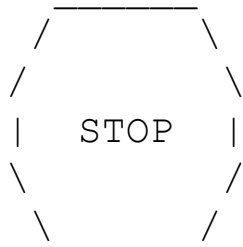
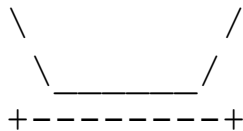
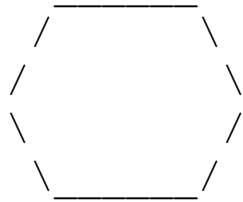
## First version (unstructured):

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

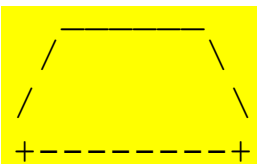
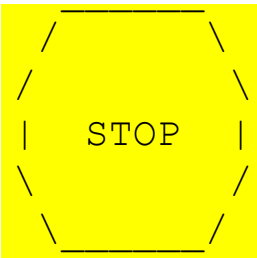
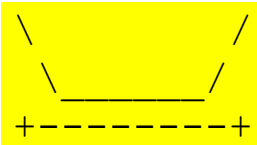
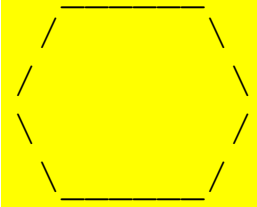
# Development strategy 2



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.

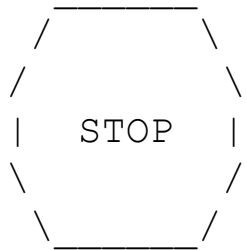
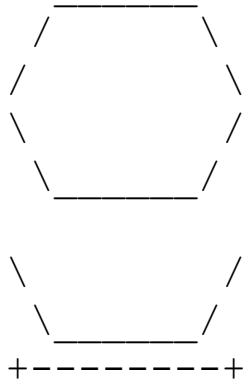
...

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



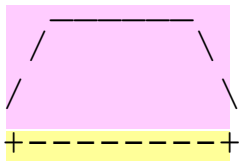
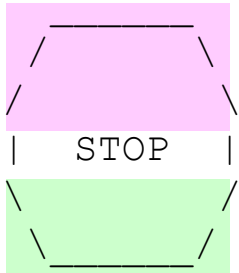
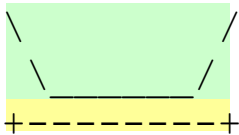
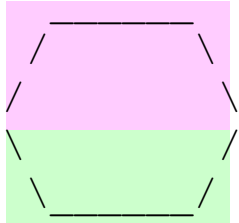
# Development strategy 3



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