Java Basics

CSCI 121: Data Structures

START RECORDING

Outline

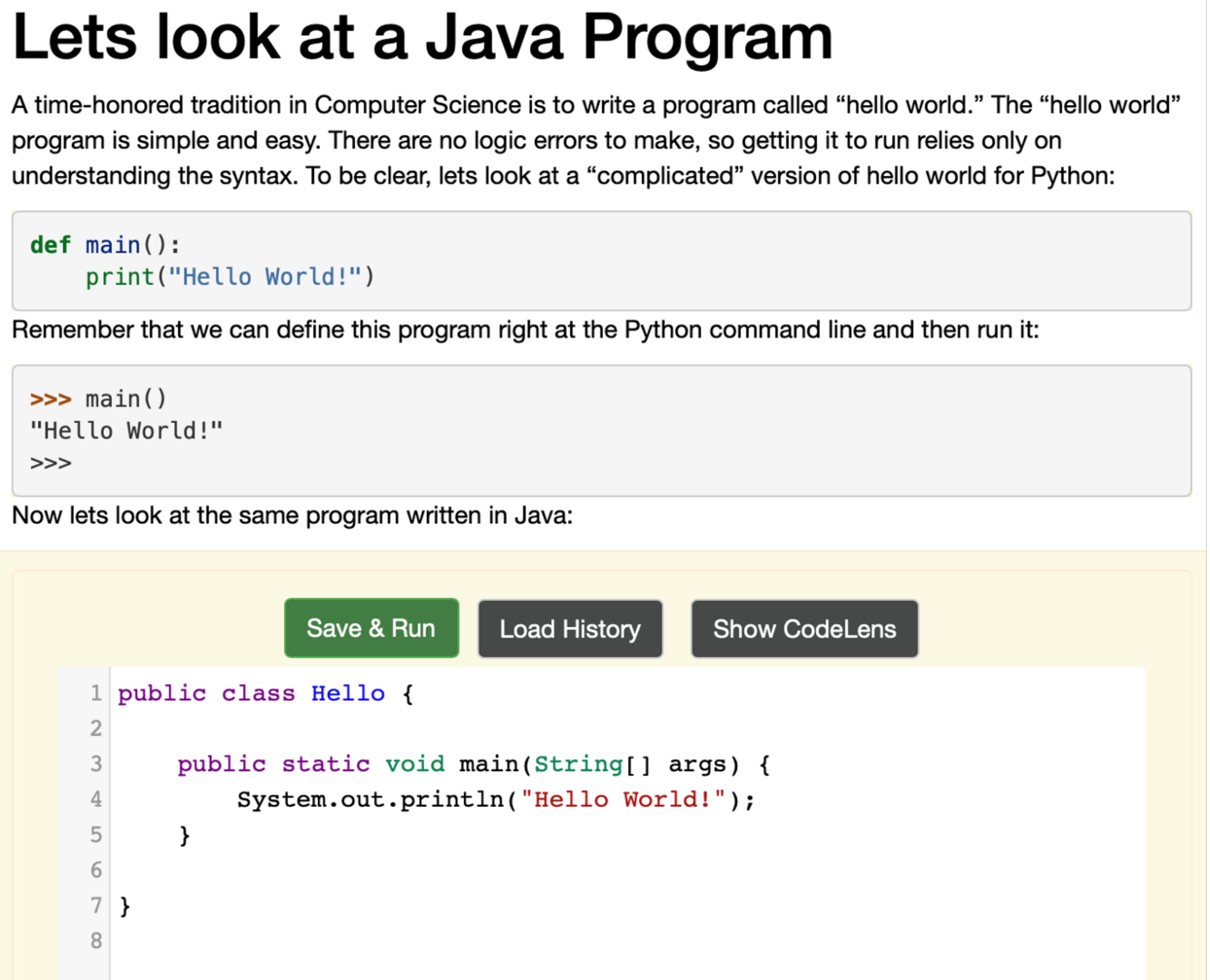
- About the course
- Textbooks
- Related course: CSCI 122, “Introduction to Discrete Structures”
- Why Java?
- Program development

About the Course

- Instructor: Dr. Peter Story
    - Happy to answer emails, meet before/after class, etc.
    - Office hours: Monday 3pm-4pm and Friday 1:30pm-2:30pm in BP334
- Website: https://cs.clarku.edu/~cs121/
    - Check frequently!
    - Syllabus, schedule, assignments, readings, lecture slides, etc.
- Canvas: resources I can’t post on the course website
- Attendance quizzes:
    - Arrive 5 minutes early! Attendance quizzes will be handed out exactly at class time.

Textbooks

- Main textbook:
    - “Algorithms,” 4th edition, by R. Sedgewick and K. Wayve
- Virtual textbook for Java:
    - “Java for Python Programmers,” by Brad Miller
    - Interactive
    - Open-source: if you find a problem, let me know, and we can fix it!



Lets look at a Java Program

A time-honored tradition in Computer Science is to write a program called “hello world.” The “hello world” program is simple and easy. There are no logic errors to make, so getting it to run relies only on understanding the syntax. To be clear, lets look at a “complicated” version of hello world for Python:

```python
def main():
    print("Hello World!")
```

Remember that we can define this program right at the Python command line and then run it:

```
>>> main()
"Hello World!"
>>>
```

Now lets look at the same program written in Java:

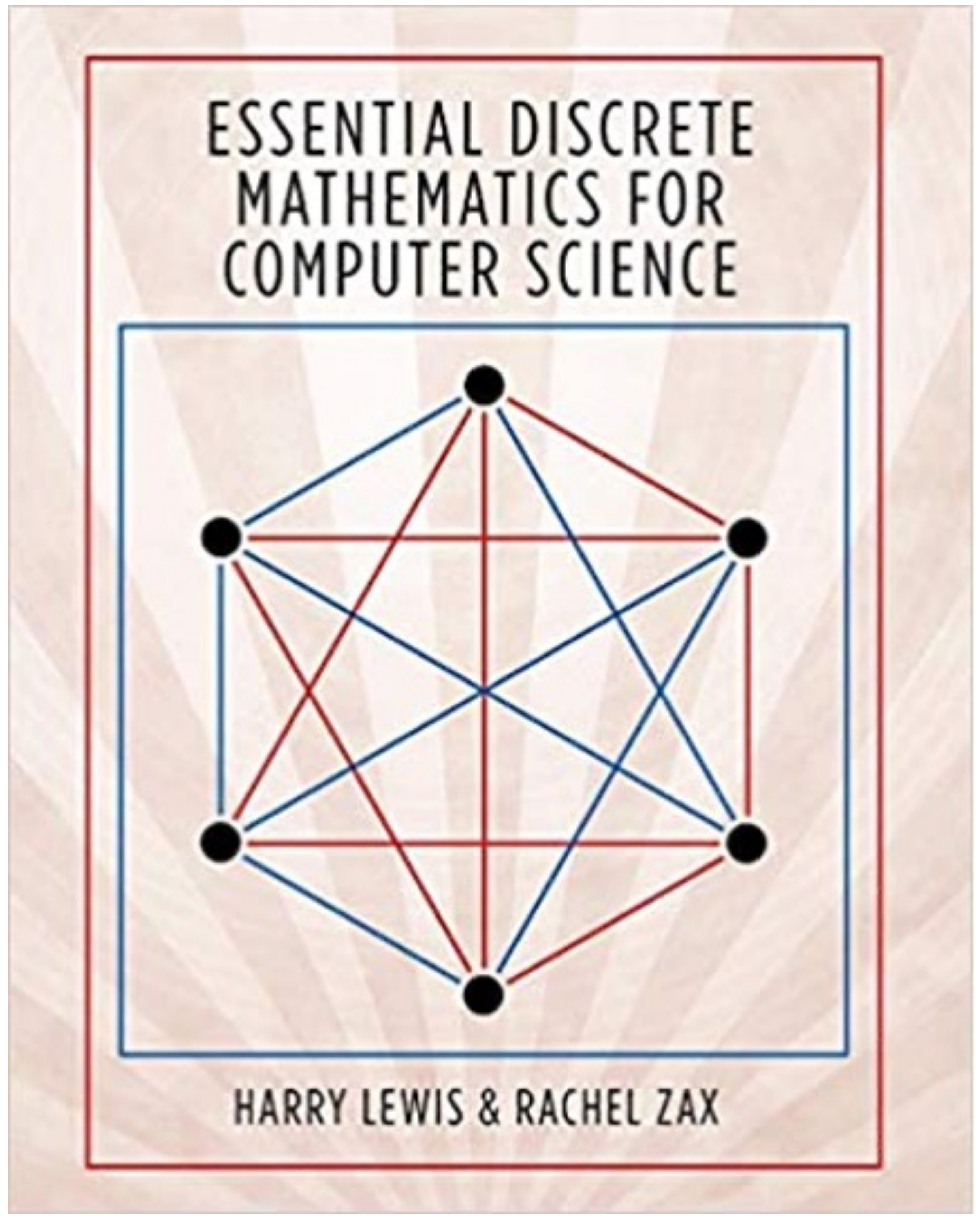
```java
public class Hello {

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

}

```

Related Course:
CSCI 122 Introduction to Discrete Structures



Section 1: MWF 9am-10:15am

Section 2: MWF 10:25am-11:40am

- Required for Computer Science (CS) Major and Minor, and the Data Science Major's CS Track
- Follows curriculum guidelines from ACM and IEEE
- Covers discrete topics essential for CS and DS
  - Foundational Concepts and Proof Techniques
  - Undirected and Directed Graphs
  - Order Notations and Counting
  - Discrete Probability, if time permits
- Teaches mathematical reasoning, and help students learn to think and prove formally and precisely – invaluable skills
- In-class problem sessions, active learning

Website Tour

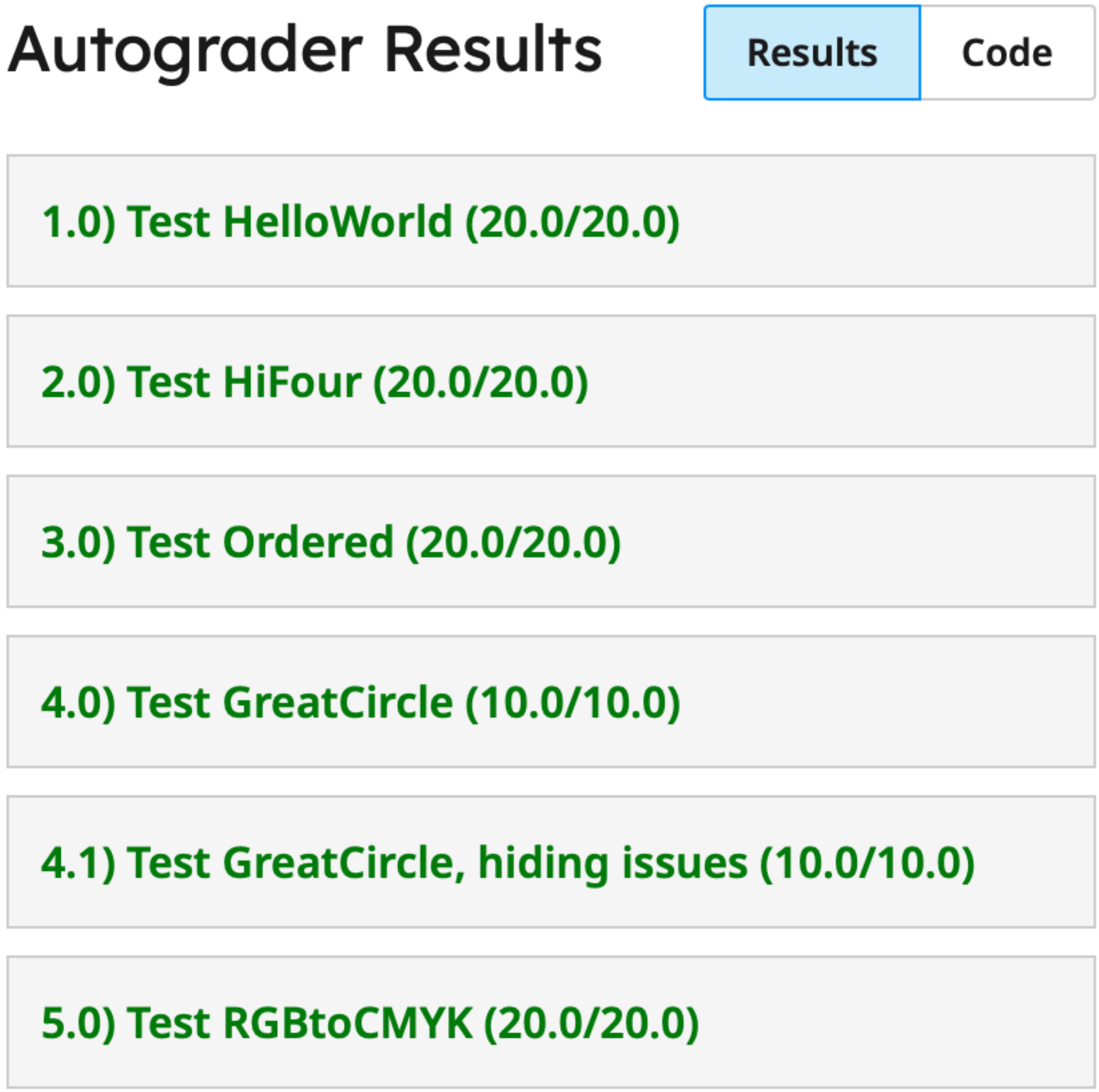
Gradescope

- Submit homework and labs via Gradescope

- Code is tested automatically

    - Resubmissions allowed until the deadline

    - But don't become dependent on Gradescope finding your bugs – later assignments will give less descriptive messages!



1.0) Test HelloWorld (20.0/20.0)

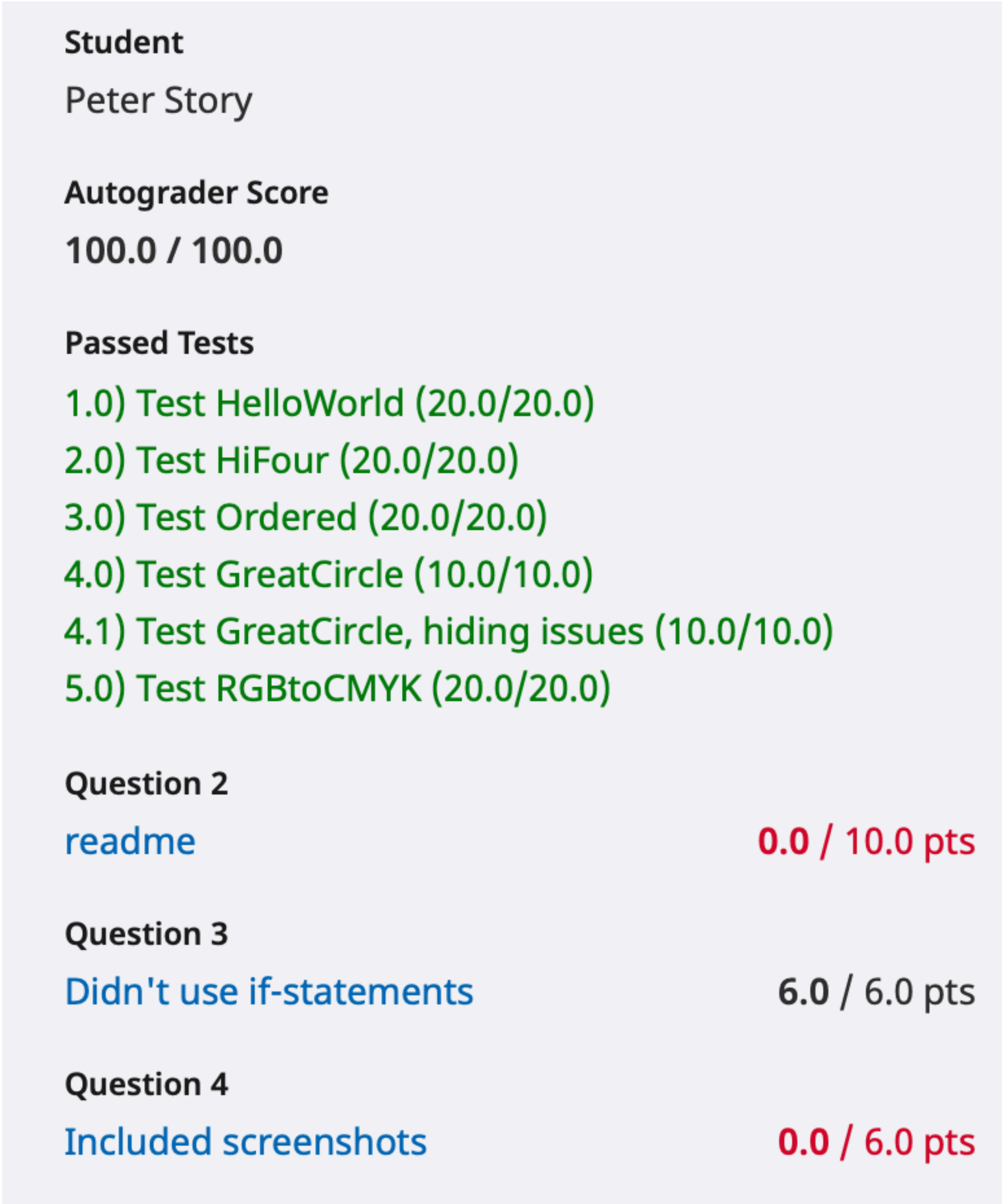
2.0) Test HiFour (20.0/20.0)

3.0) Test Ordered (20.0/20.0)

4.0) Test GreatCircle (10.0/10.0)

4.1) Test GreatCircle, hiding issues (10.0/10.0)

5.0) Test RGBtoCMYK (20.0/20.0)



Student
Peter Story

Autograder Score
100.0 / 100.0

Passed Tests
1.0) Test HelloWorld (20.0/20.0)
2.0) Test HiFour (20.0/20.0)
3.0) Test Ordered (20.0/20.0)
4.0) Test GreatCircle (10.0/10.0)
4.1) Test GreatCircle, hiding issues (10.0/10.0)
5.0) Test RGBtoCMYK (20.0/20.0)

Question 2
readme 0.0 / 10.0 pts

Question 3
Didn't use if-statements 6.0 / 6.0 pts

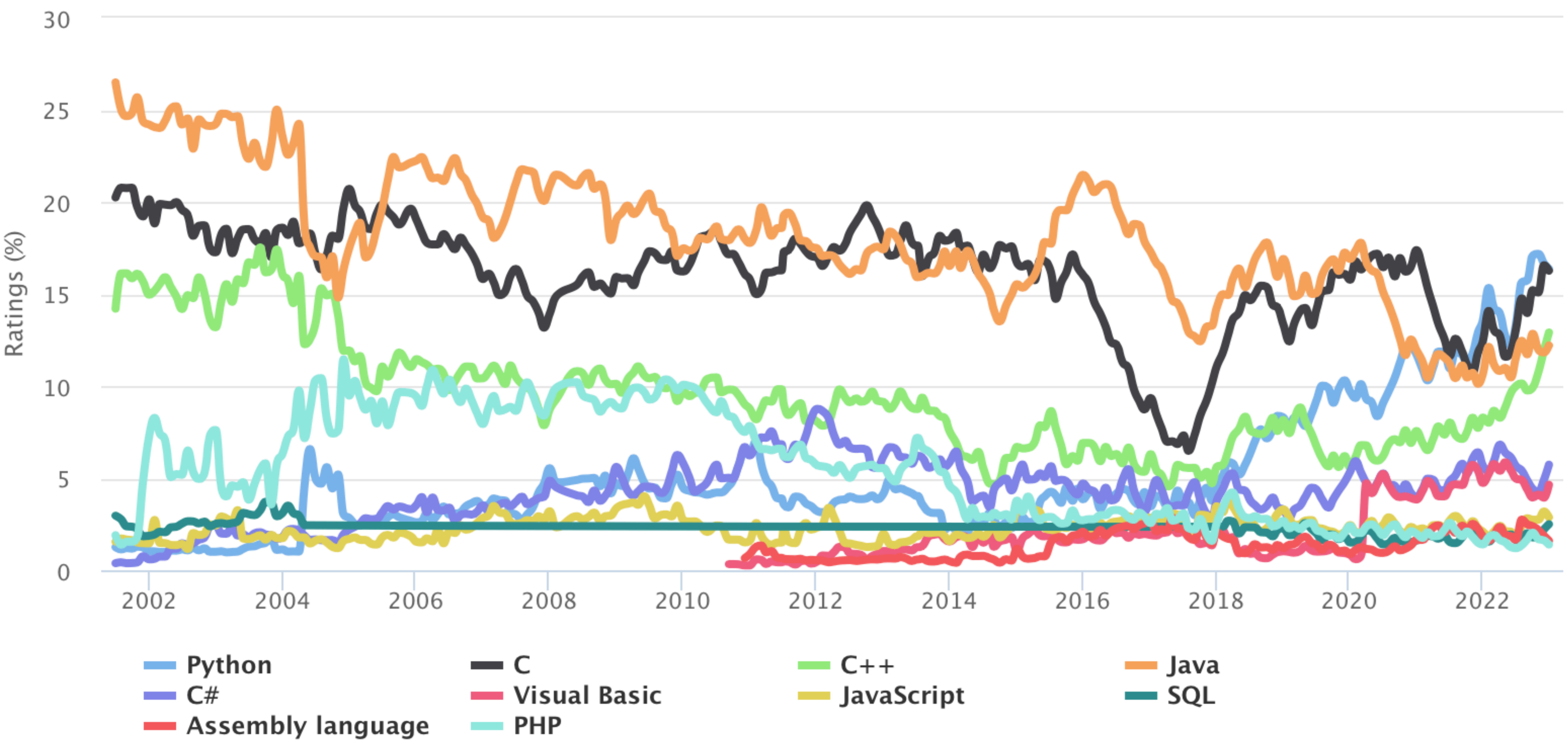
Question 4
Included screenshots 0.0 / 6.0 pts

Questions?

Why Java?

Why Java?

- Why not just teach data structures using Python (or C++)?
- **Learning new languages and tools is a skill.** Many differences between programming in Java and Python. For example:
    - Compilation:
        - Java: A distinct compilation step, prior to execution. Many automatic checks.
        - Python: JIT-compiler
    - Exceptions:
        - Java support explicit exceptions, which must be handled by method callers
    - Syntax: many subtle differences
    - Types:
        - Java is “strongly typed”
        - Python uses dynamic typing



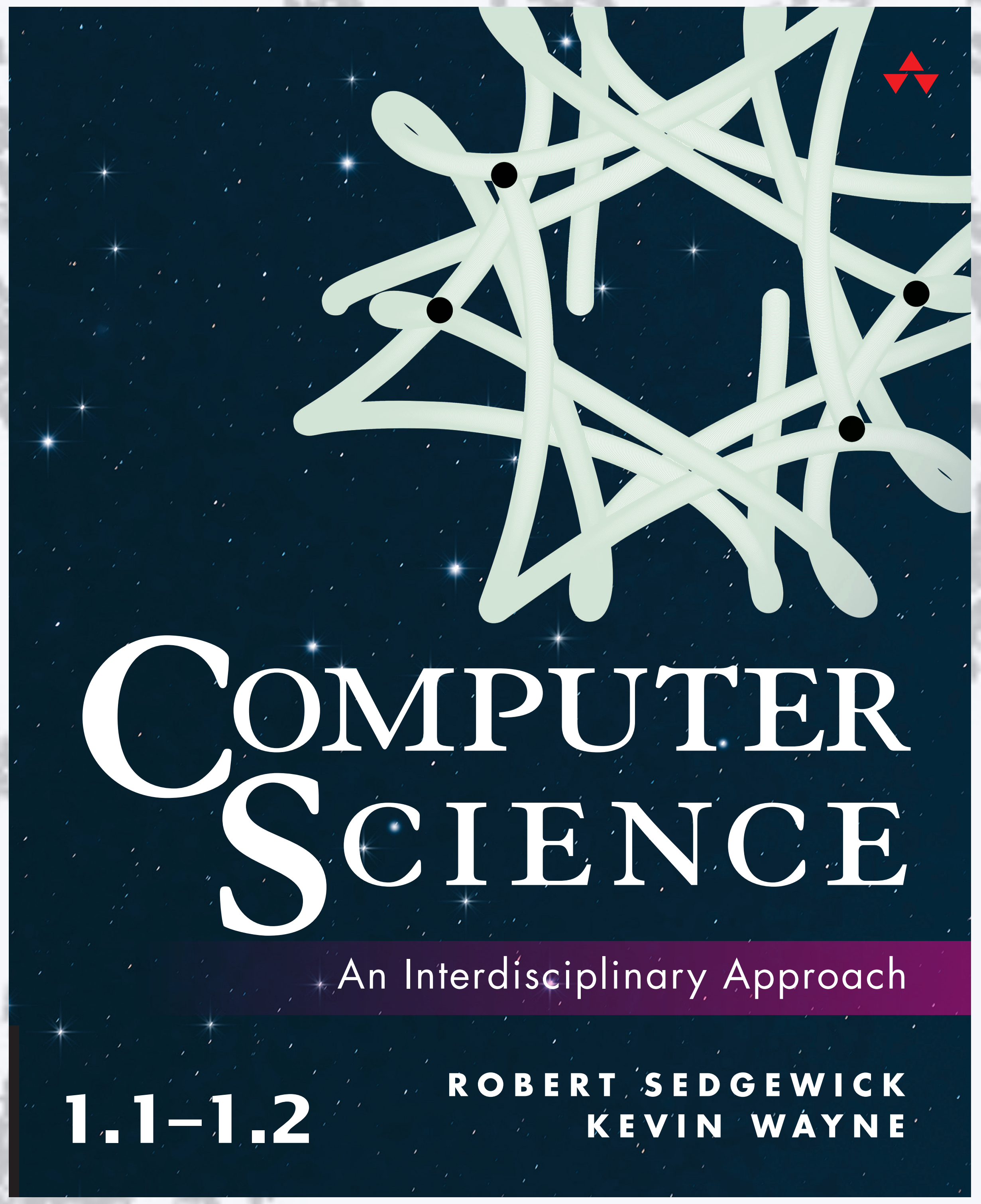
TIOBE Programming Community Index

Source: www.tiobe.com

TIOBE Programming Community Index

Source: www.tiobe.com

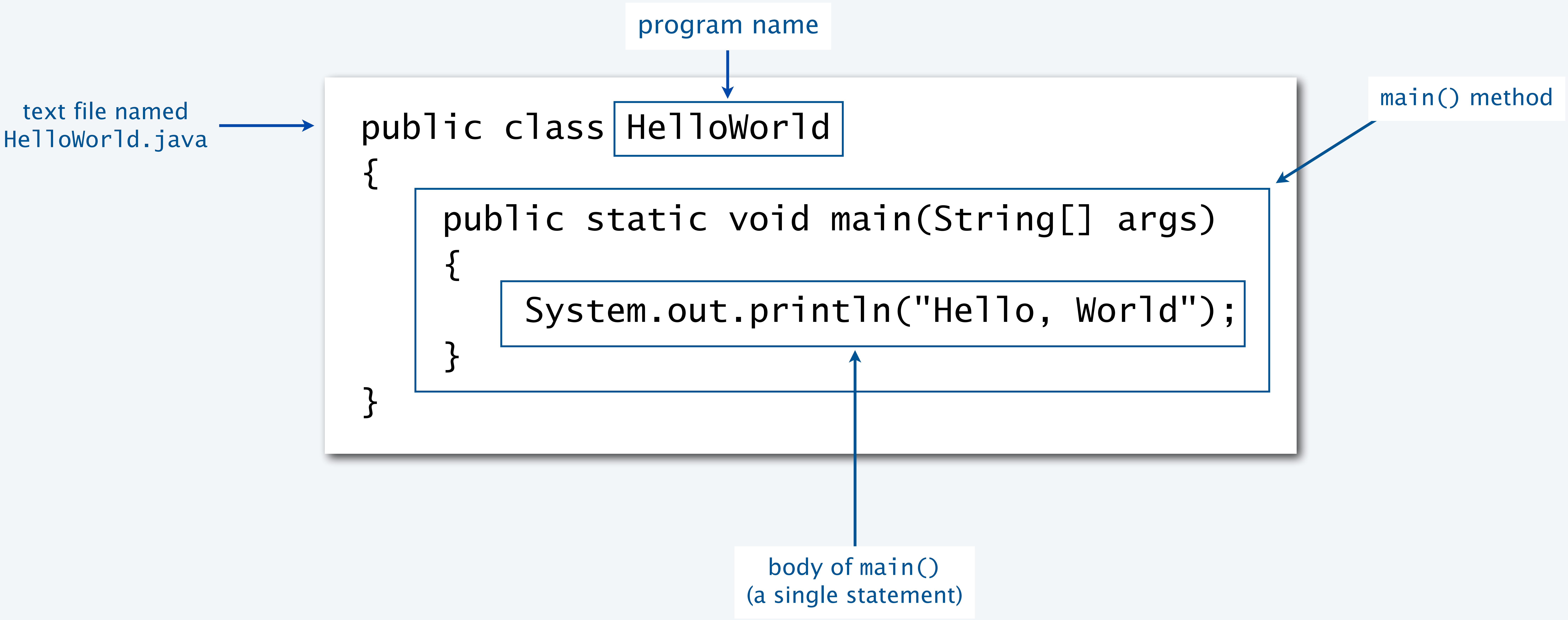
Ratings (%)
30
20
10
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
2020
2022
Python
C
C++
Java
C#
Visual Basic
JavaScript
SQL
Assembly language
PHP



http://introcs.cs.princeton.edu

1. Basic Programming Concepts

Anatomy of your first program



program name

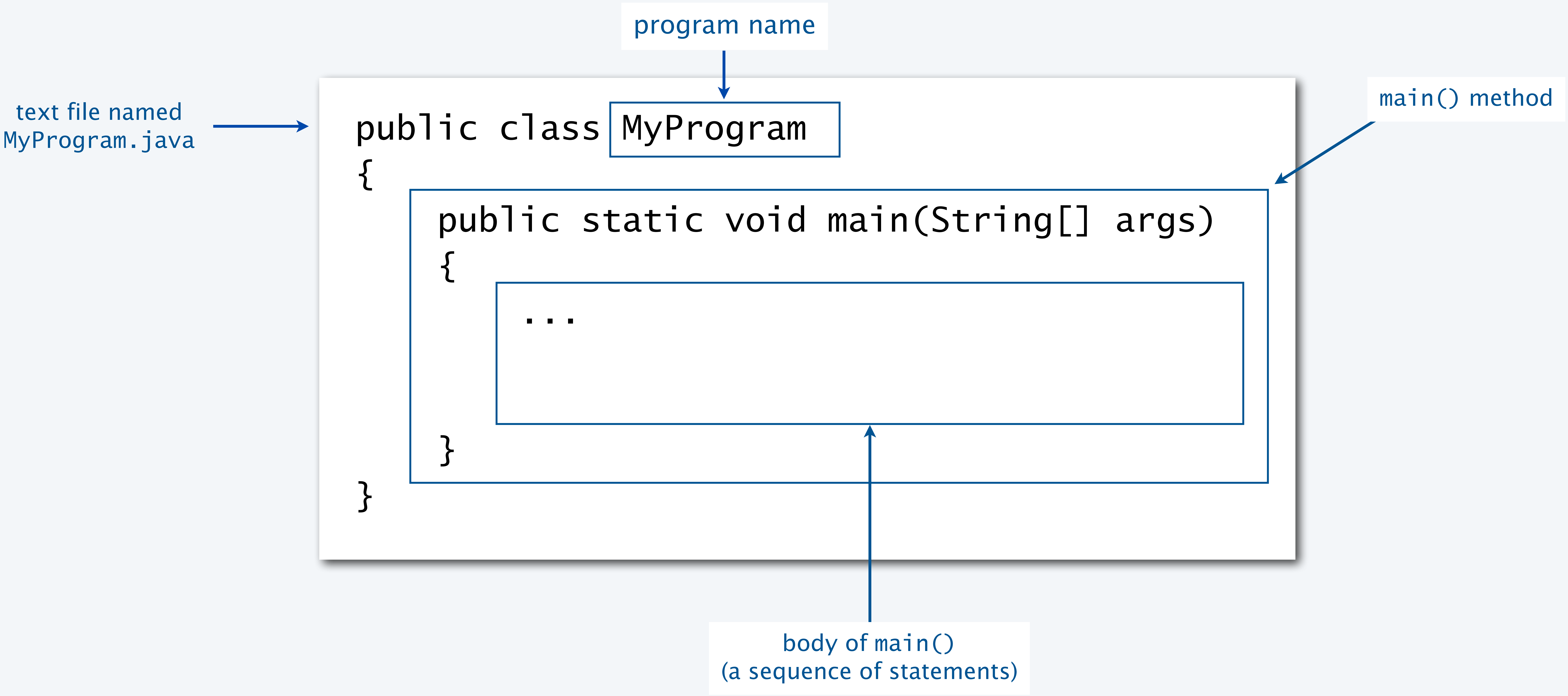
text file named
`HelloWorld.java`

main() method

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

body of main()
(a single statement)

Anatomy of your next several programs



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

    }
}
```

program name

text file named MyProgram.java

main() method

body of main()
(a sequence of statements)

Pop quiz on "your first program"

Q. Use common sense to cope with the following error messages.

```
% javac MyProgram.java
% java MyProgram
Main method not public.
```

```
% javac MyProgram.java
MyProgram.java:3: invalid method declaration; return type required
       public static main(String[] args)
                     ^
```

Pop quiz on "your first program"

Q. Use common sense to cope with the following error messages.

```
% javac MyProgram.java
% java MyProgram
Main method not public.
```

A. Must have forgotten “public”.

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

```
% javac MyProgram.java
MyProgram.java:3: invalid method declaration; return type required
       public static main(String[] args)
                     ^
```

A. Check HelloWorld. Aha! Forgot “void”.

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

Three versions of the same program.

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

```
/**************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 **************************************************************************/

public class HelloWorld {

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

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

Lesson: Fonts, color, comments, and extra space are not required by the Java language, **though extremely important for readability and maintainability.**

Note on program style

Different styles are appropriate in different contexts.

- Integrated development environment
- Booksite
- Book
- Your code

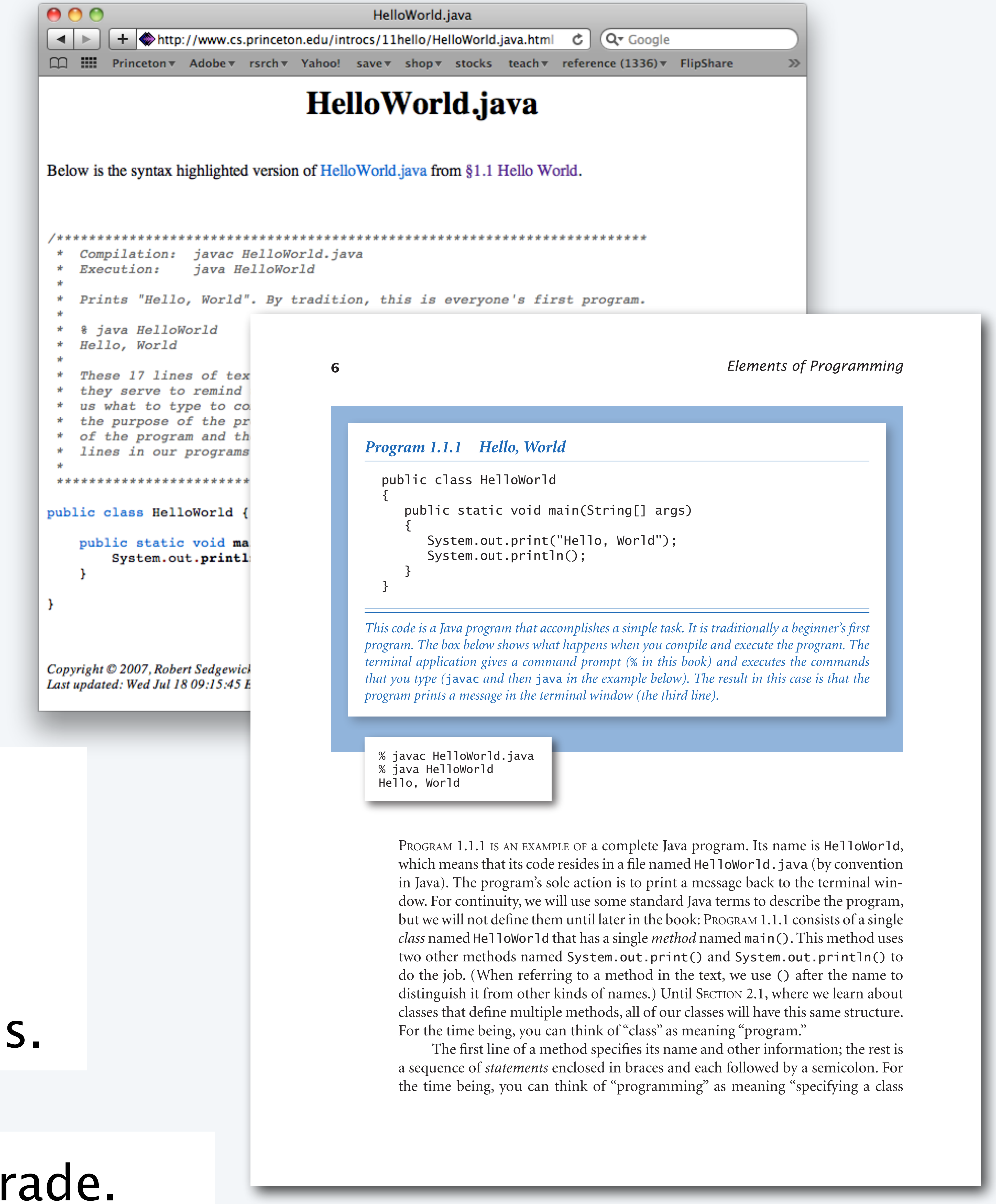
Enforcing consistent style can confuse style with language.

Emphasizing consistent style can

- Make it easier to spot errors.
- Make it easier for others to read and use code.
- Enable development environment to provide visual cues.

Bottom line for you: Listen to the person assigning your grade.

or your boss!



HelloWorld.java

Below is the syntax highlighted version of HelloWorld.java from §1.1 Hello World.

```
/*************************************************************************
 *  Compilation:  javac HelloWorld.java
 *  Execution:    java HelloWorld
 *
 *  Prints "Hello, World". By tradition, this is everyone's first program.
 *
 *  % java HelloWorld
 *  Hello, World
 *
 *  These 17 lines of tex
 *  they serve to remind
 *  us what to type to co
 *  the purpose of the pr
 *  of the program and th
 *  lines in our programs
 *
 *************************

public class HelloWorld {

    public static void ma
        System.out.printl
    }

}
```

Copyright © 2007, Robert Sedgewick
Last updated: Wed Jul 18 09:15:45 E

6 *Elements of Programming*

Program 1.1.1 Hello, World

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

This code is a Java program that accomplishes a simple task. It is traditionally a beginner's first program. The box below shows what happens when you compile and execute the program. The terminal application gives a command prompt (% in this book) and executes the commands that you type (`javac` and then `java` in the example below). The result in this case is that the program prints a message in the terminal window (the third line).

```
% javac HelloWorld.java
% java HelloWorld
Hello, World
```

PROGRAM 1.1.1 IS AN EXAMPLE OF a complete Java program. Its name is `HelloWorld`, which means that its code resides in a file named `HelloWorld.java` (by convention in Java). The program's sole action is to print a message back to the terminal window. For continuity, we will use some standard Java terms to describe the program, but we will not define them until later in the book: PROGRAM 1.1.1 consists of a single *class* named `HelloWorld` that has a single *method* named `main()`. This method uses two other methods named `System.out.print()` and `System.out.println()` to do the job. (When referring to a method in the text, we use `()` after the name to distinguish it from other kinds of names.) Until SECTION 2.1, where we learn about classes that define multiple methods, all of our classes will have this same structure. For the time being, you can think of "class" as meaning "program."

The first line of a method specifies its name and other information; the rest is a sequence of *statements* enclosed in braces and each followed by a semicolon. For the time being, you can think of "programming" as meaning "specifying a class

A rich subset of the Java language vocabulary

built-in types
int
long
double
char
String
boolean

punctuation
{
}
(
)
,
;

operations on numeric types
+
-
*
/
%
++
--

comparisons
<
<=
>
>=
==
!=

String *operations*
+
""
length()
charAt()
compareTo()
matches()

boolean operations
true
false
!
&&
||

assignment
=

flow control
if
else
for
while

arrays
a[]
length
new

object oriented
static
class
public
private
new
final
toString()
main()

Math *methods*
Math.sin()
Math.cos()
Math.log()
Math.exp()
Math.pow()
Math.sqrt()
Math.min()
Math.max()
Math.abs()
Math.PI

type conversion methods
Integer.parseInt()
Double.parseDouble()

System *methods*
System.print()
System.println()
System.printf()

our Std *methods*
StdIn.read*()
StdOut.print*()
StdDraw.*()
StdAudio.*()
StdRandom.*()

Your programs will primarily consist of these plus identifiers (names) that you make up.

Image sources

http://commons.wikimedia.org/wiki/File:KnuthAtOpenContentAlliance.jpg

http://commons.wikimedia.org/wiki/File:Ada_Lovelace.jpg

http://commons.wikimedia.org/wiki/File:Babbages_Analytical_Engine,_1834-1871._(9660574685).jpg

http://commons.wikimedia.org/wiki/File:James_Gosling_2005.jpg

http://commons.wikimedia.org/wiki/File:Bjarne-stroustrup.jpg

http://blog-images.muddymatches.co.uk.s3.amazonaws.com/dating-advice/wp-content/uploads/2013/01/Bad-guy.jpg

1. Basic Programming Concepts

- Why programming?
- **Program development**
- Built-in data types
- Type conversion

Program development in Java

is a three-step process, *with feedback*

1. EDIT your program

- Create it by typing on your computer's keyboard.
- Result: a text file such as `HelloWorld.java`.

2. COMPILE it to create an executable file

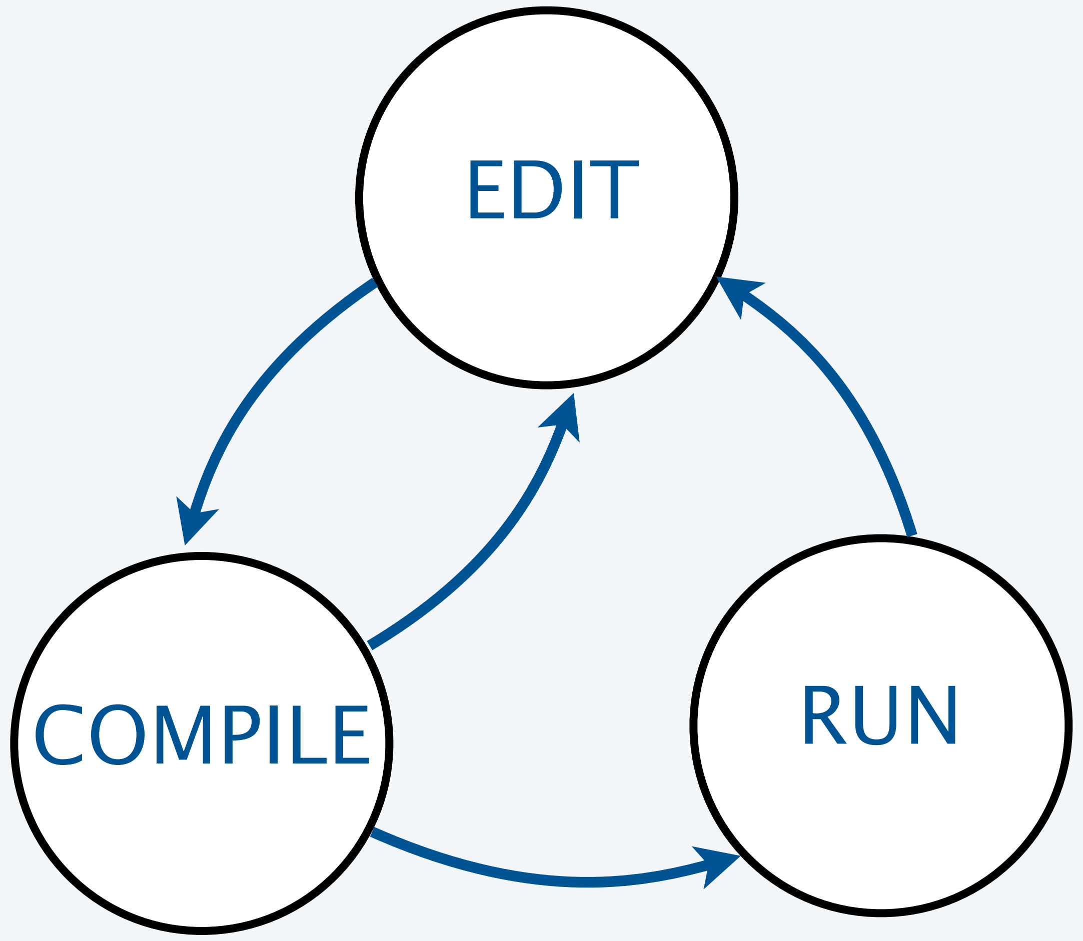
- Use the Java compiler
- Result: a Java bytecode file such as `HelloWorld.class`
- Mistake? Go back to 1. to fix and recompile.

not a legal Java program

3. RUN your program

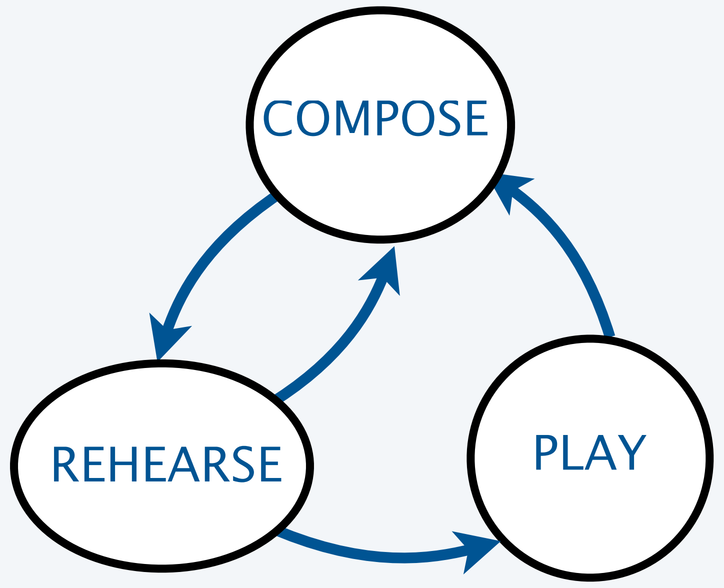
- Use the Java runtime.
- Result: your program’s output.
- Mistake? Go back to 1. to fix, recompile, and run.

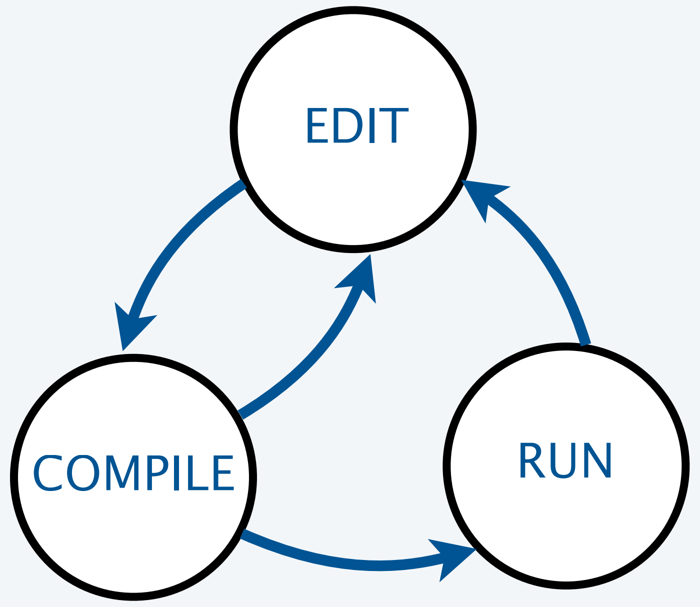
a legal Java program that does the wrong thing



Software for program development

Any creative process involves cyclic refinement/development.





A significant difference with programs: *We can use our computers to facilitate the process.*

Program development environment: Software for editing, compiling and running programs.

Two time-tested options: (Stay tuned for details).

Virtual terminals

- Same for many languages and systems.
- Effective even for beginners.

Bottom line: Extremely simple and concise.

Integrated development environment

- Often language- or system-specific.
- Can be helpful to beginners.

Bottom line: Variety of useful tools.

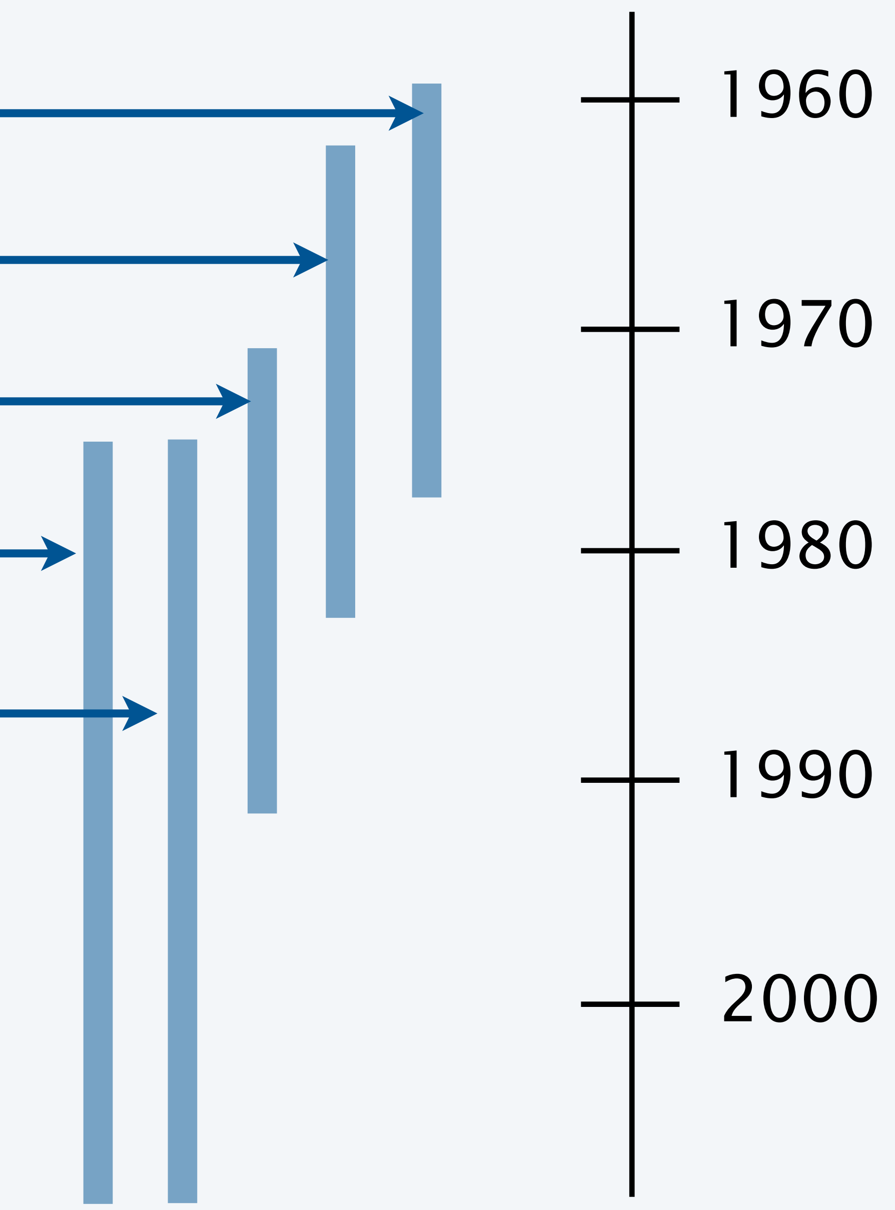
Program development environments: a very short history

Historical context is important in computer science.

- We regularly use old software.
- We regularly emulate old hardware.
- We depend upon old concepts and designs.

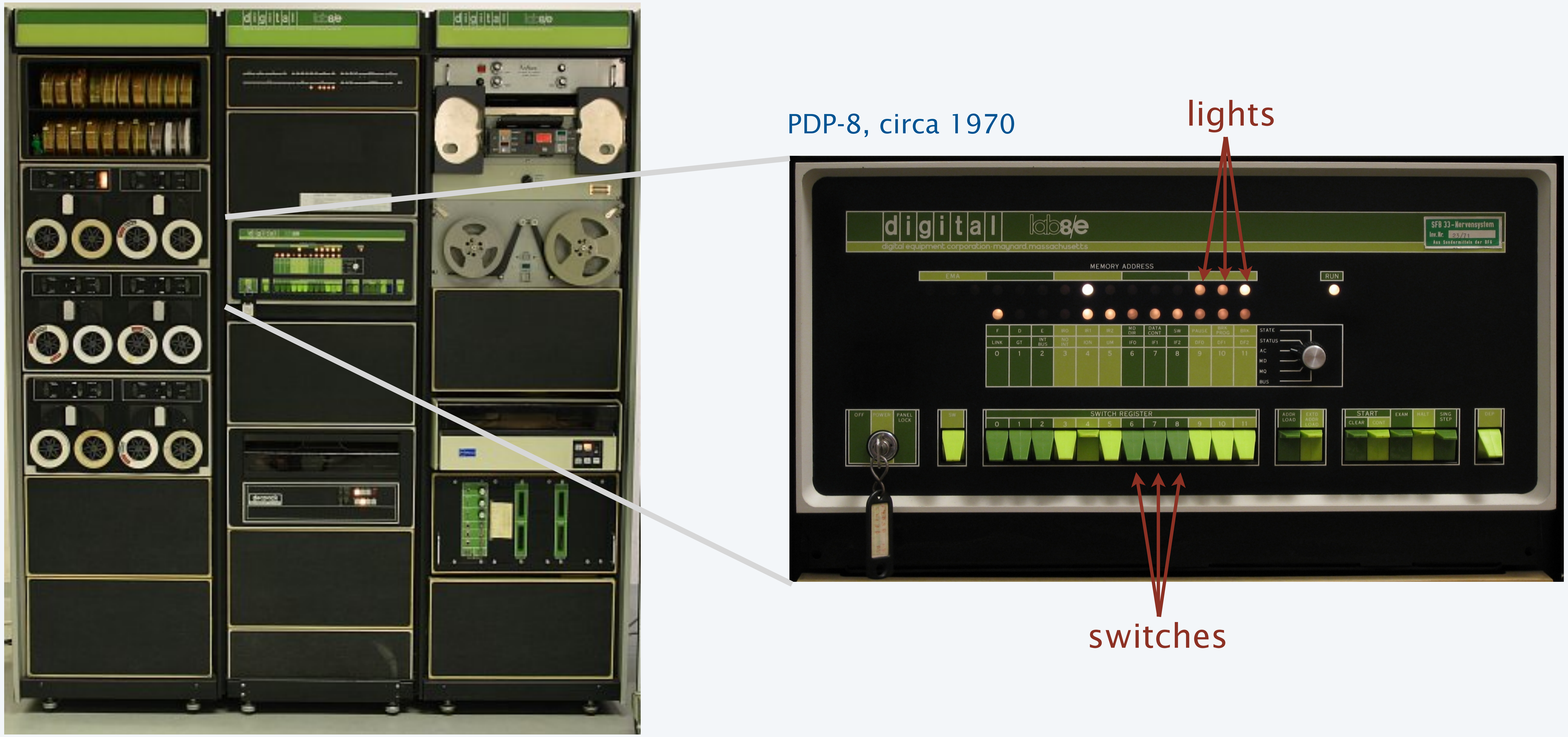
Widely-used methods for program development

- switches and lights
- punched cards/compiler/runtime
- editor/compiler/runtime/terminal
- editor/compiler/runtime/*virtual* terminal
- integrated development environment



Program development with switches and lights

Circa 1970: Use switches to input binary program code and data, lights to read output.



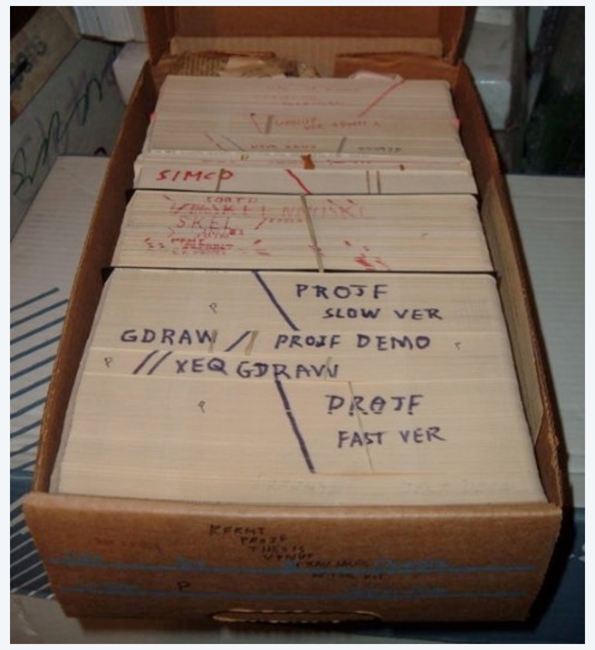
PDP-8, circa 1970

lights

switches

Program development with punched cards and line printers

Mid 1970s: Use punched cards to input program code and data, line printer for output.

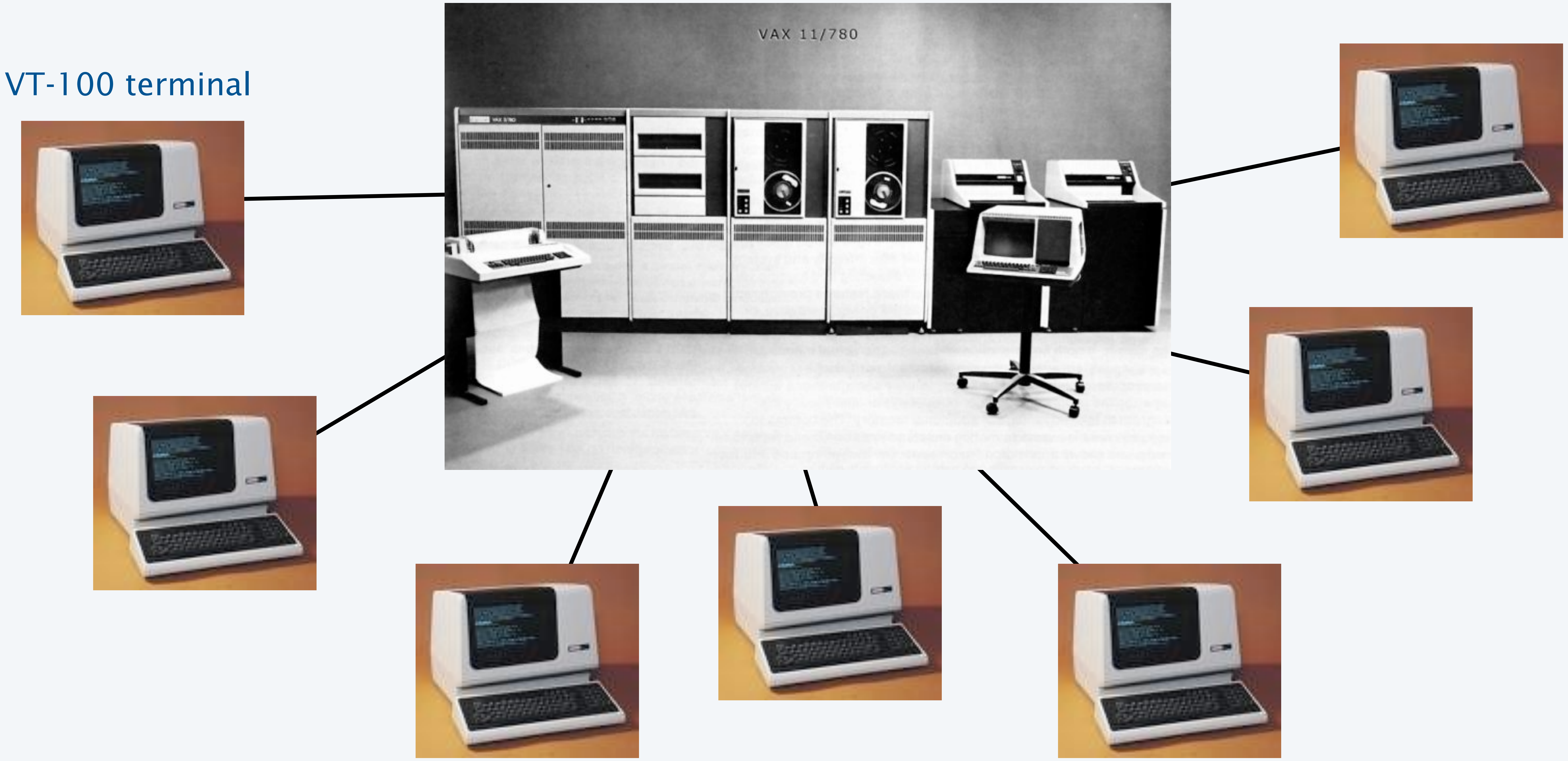


IBM System 360, circa 1975

Ask your parents about the "computer center" for details.

Program development with timesharing terminals

Late 1970s: Use terminal for editing program, reading output, and controlling computer.



VAX 11/780 circa 1977

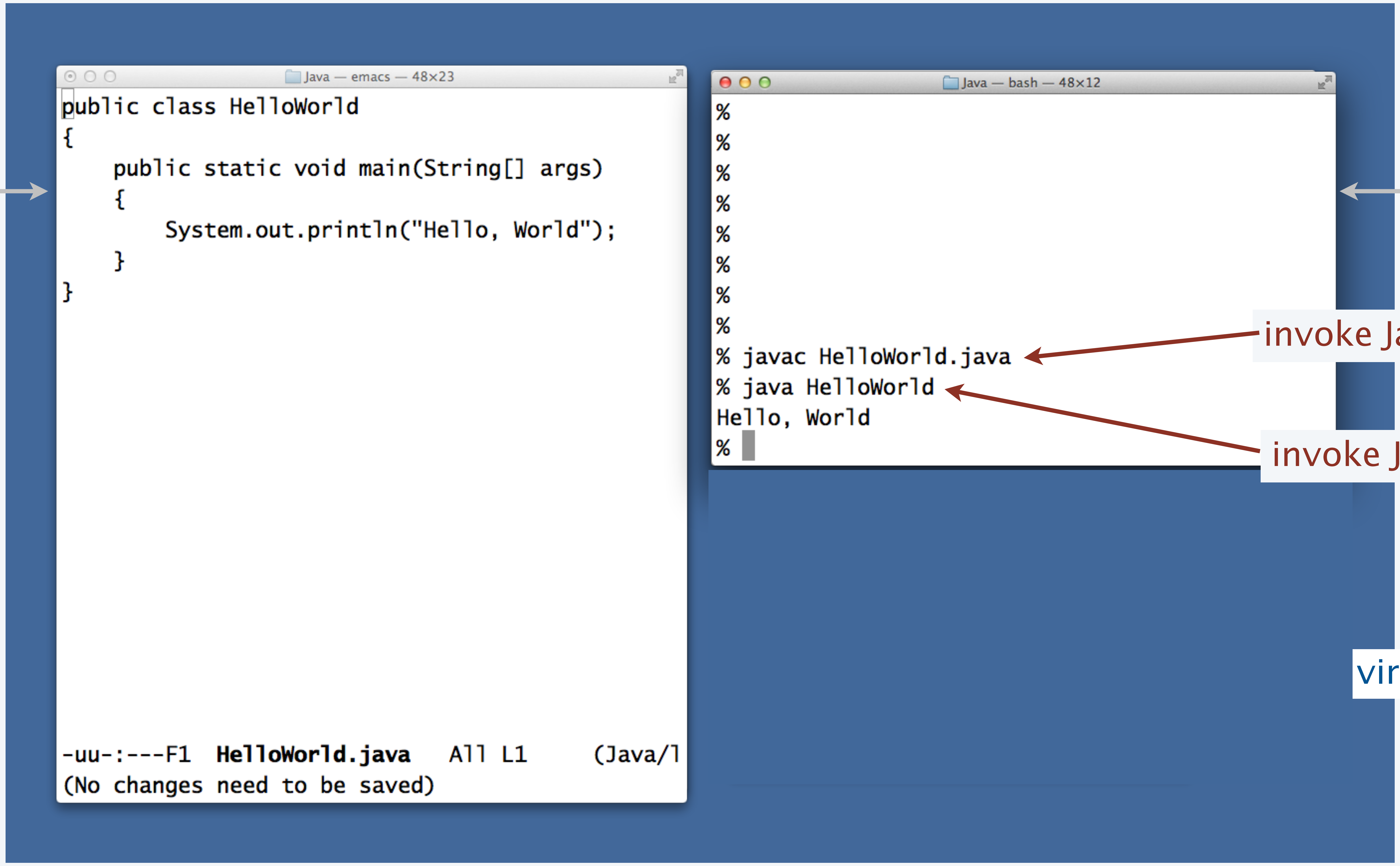
VT-100 terminal

Timesharing allowed many users to share the same computer.

Program development with personal computers (one approach)

1980s to present day: Use multiple *virtual terminals* to interact with computer.

- Edit your program using any text editor in a virtual terminal.
- Compile it by typing `javac HelloWorld.java` in another virtual terminal.
- Run it by typing `java HelloWorld`



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




-uu-:---F1  HelloWorld.java   All L1     (Java/l
(No changes need to be saved)
```

```
%
%
%
%
%
%
%
%
% javac HelloWorld.java
% java HelloWorld
Hello, World
%
```

virtual terminal for editor

virtual terminal to compile, run and examine output

invoke Java compiler at command line

invoke Java runtime at command line

virtual TV set

Program development with personal computers (another approach)

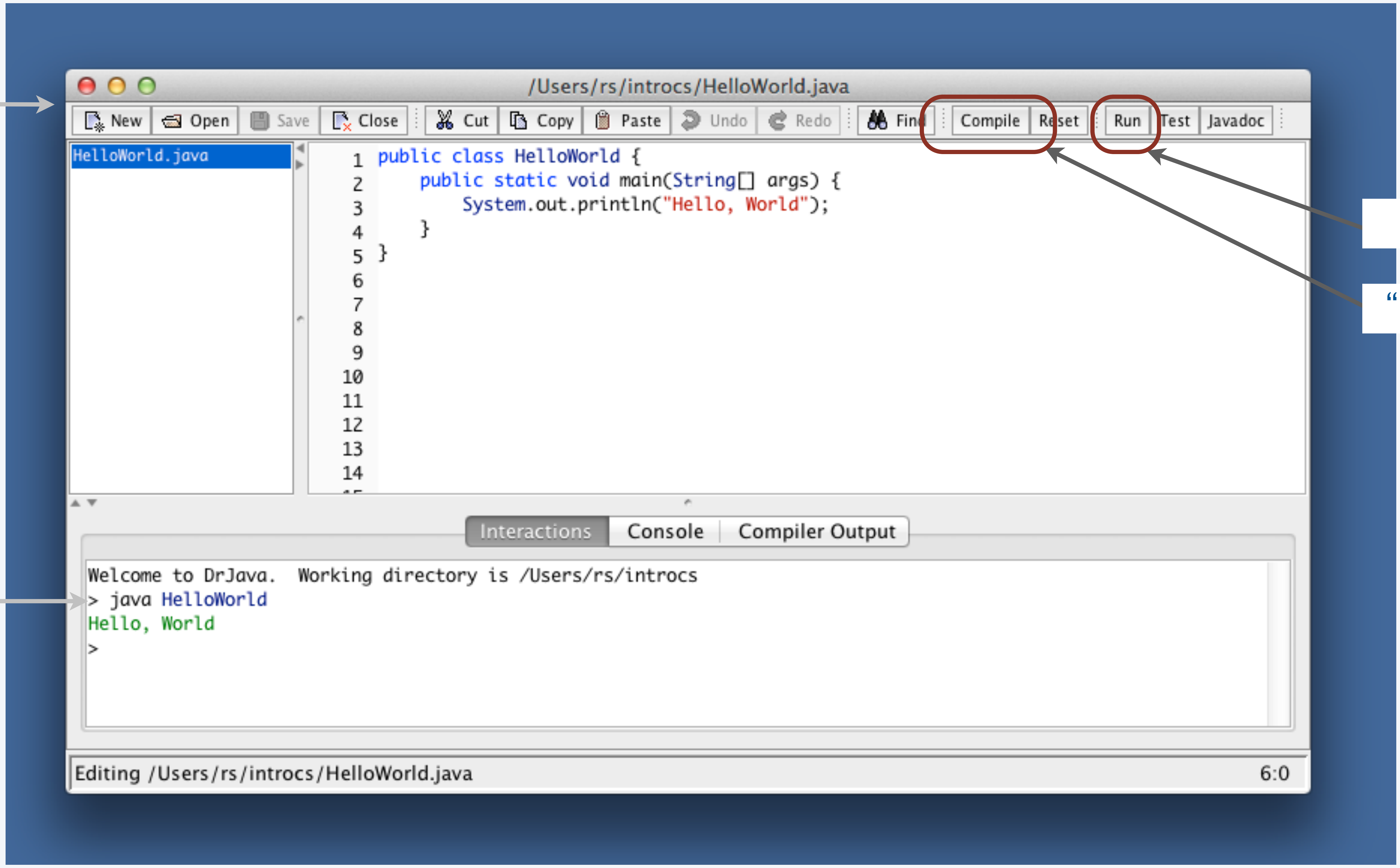
1980s to present day: Use a *customized application* for program development tasks.

- Edit your program using the built-in text editor.
- Compile it by clicking the "compile" button.
- Run it by clicking the "run" button or using the pseudo-command line.

"Integrated Development Environment" (IDE)

drjava

http://drjava.org



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

```
Welcome to DrJava.  Working directory is /Users/rs/introcs
> java HelloWorld
Hello, World
>
```

Editing /Users/rs/introcs/HelloWorld.java 6:0

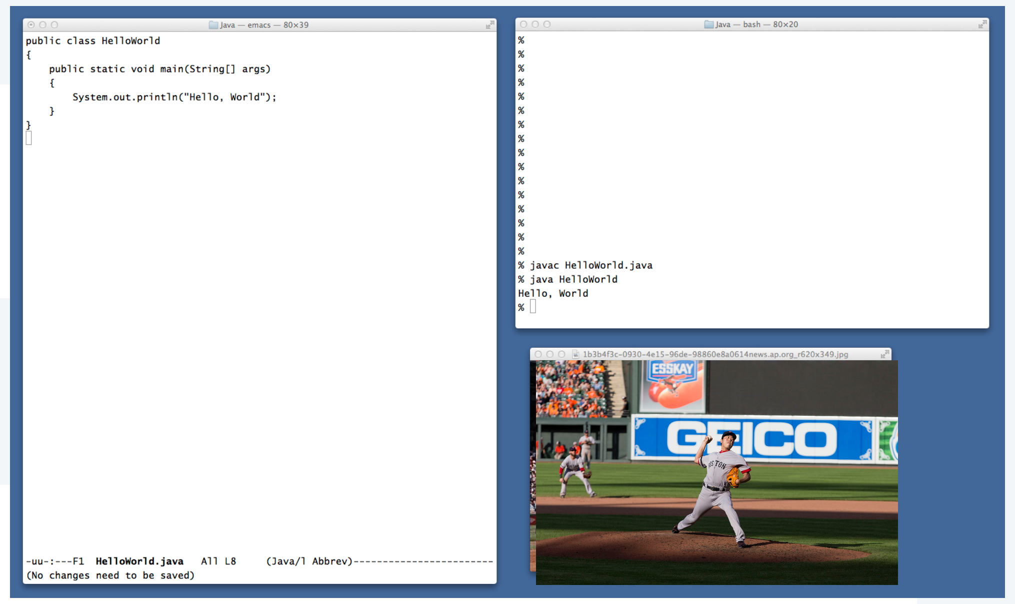
"run" button

"compile" button

pseudo-command line

Software for program development: tradeoffs

Virtual terminals


Java — emacs — 80×39
Java — bash — 80×20
1b3b4f3c-0930-4e15-96de-98860e8a0614news.ap.org_r620x349.jpg
ESSKAY
GEICO

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

-uu-:---F1  HelloWorld.java   All L8     (Java/l Abbrev)------------------------
(No changes need to be saved)
```

```
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
%
% javac HelloWorld.java
% java HelloWorld
Hello, World
%
```

Pros

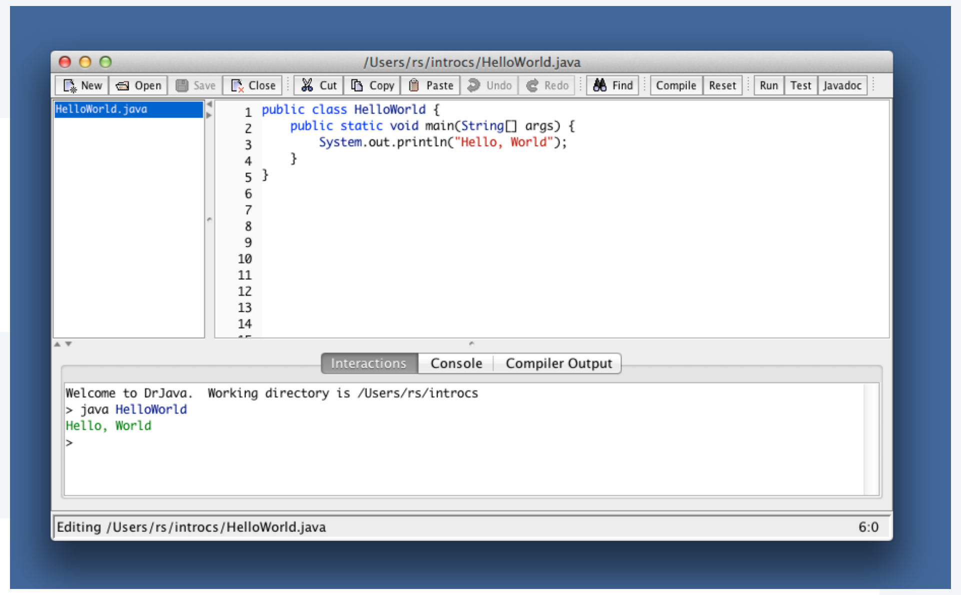
- Approach works with any language.
- Useful beyond programming.
- Used by professionals.
- Has withstood the test of time.

Cons

- Good enough for long programs?
- Dealing with independent applications.
- Working at too low a level?

This course: Used in lectures/book.

IDE



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

```
Welcome to DrJava.  Working directory is /Users/rs/introcs
> java HelloWorld
Hello, World
>
```

Editing /Users/rs/introcs/HelloWorld.java 6:0

Pros

- Easy-to-use language-specific tools.
- System-independent (in principle).
- Used by professionals.
- Can be helpful to beginners.

Cons

- Overkill for short programs?
- Big application to learn and maintain.
- Often language- or system-specific.

Recommended for assignments.

Lessons from short history

Every computer has a program development environment that allows us to

- EDIT programs.
- COMPILE them to create an executable file.
- RUN them and examine the output.

Two approaches that have served for decades and are still effective:

- multiple virtual terminals.
- integrated development environments.

Macbook Air 2014

Xerox Alto 1978

Apple Macintosh 1984

IBM PC 1990s

Wintel ultrabooks 2010s

Image sources

http://commons.wikimedia.org/wiki/Category:2013_Boston_Red_Sox_season#mediaviewer/
    File:Koji_Uehara_2_on_June_15,_2013.jpg

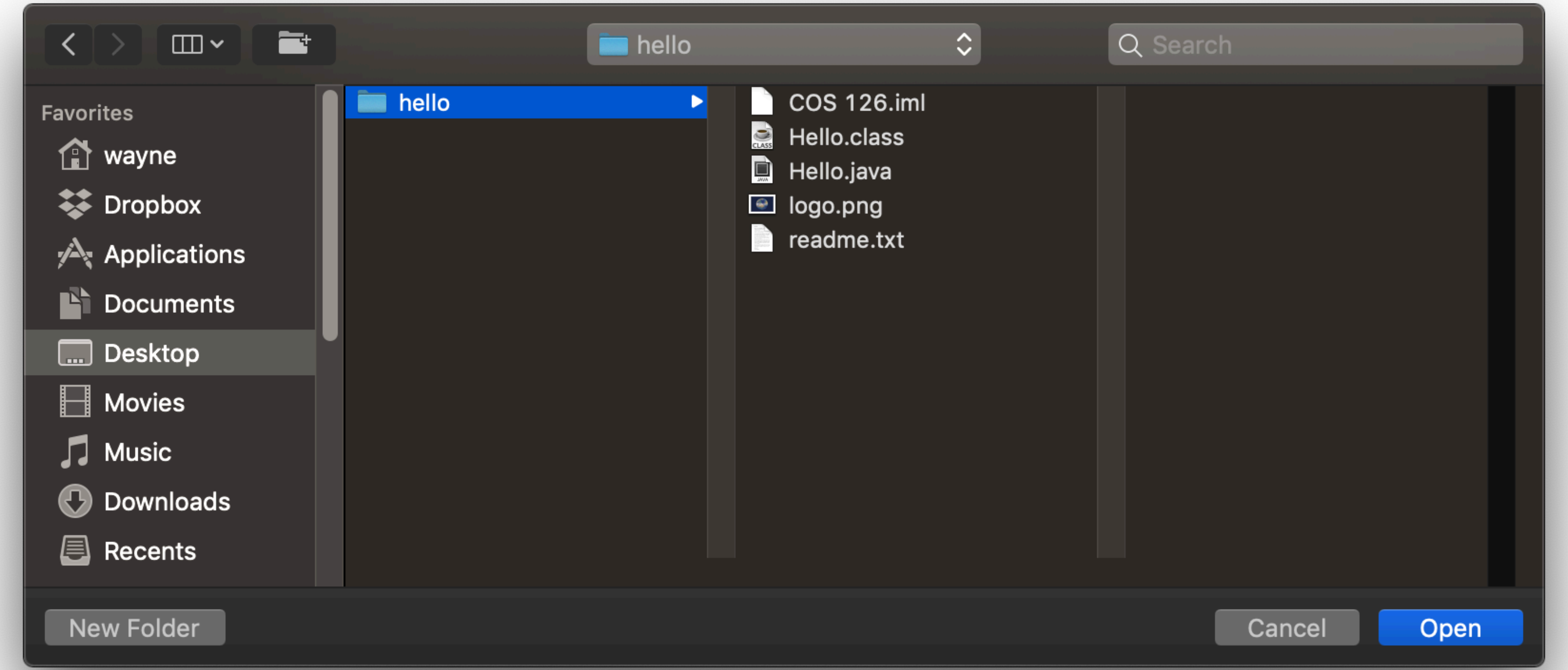
http://thenationalforum.org/wp-content/uploads/2011/03/Legendary-Musicians.png

http://pixabay.com/p-15812/?no_redirect

Programming with IntelliJ

Step 1: Open IntelliJ

Step 2: Open Starter Files



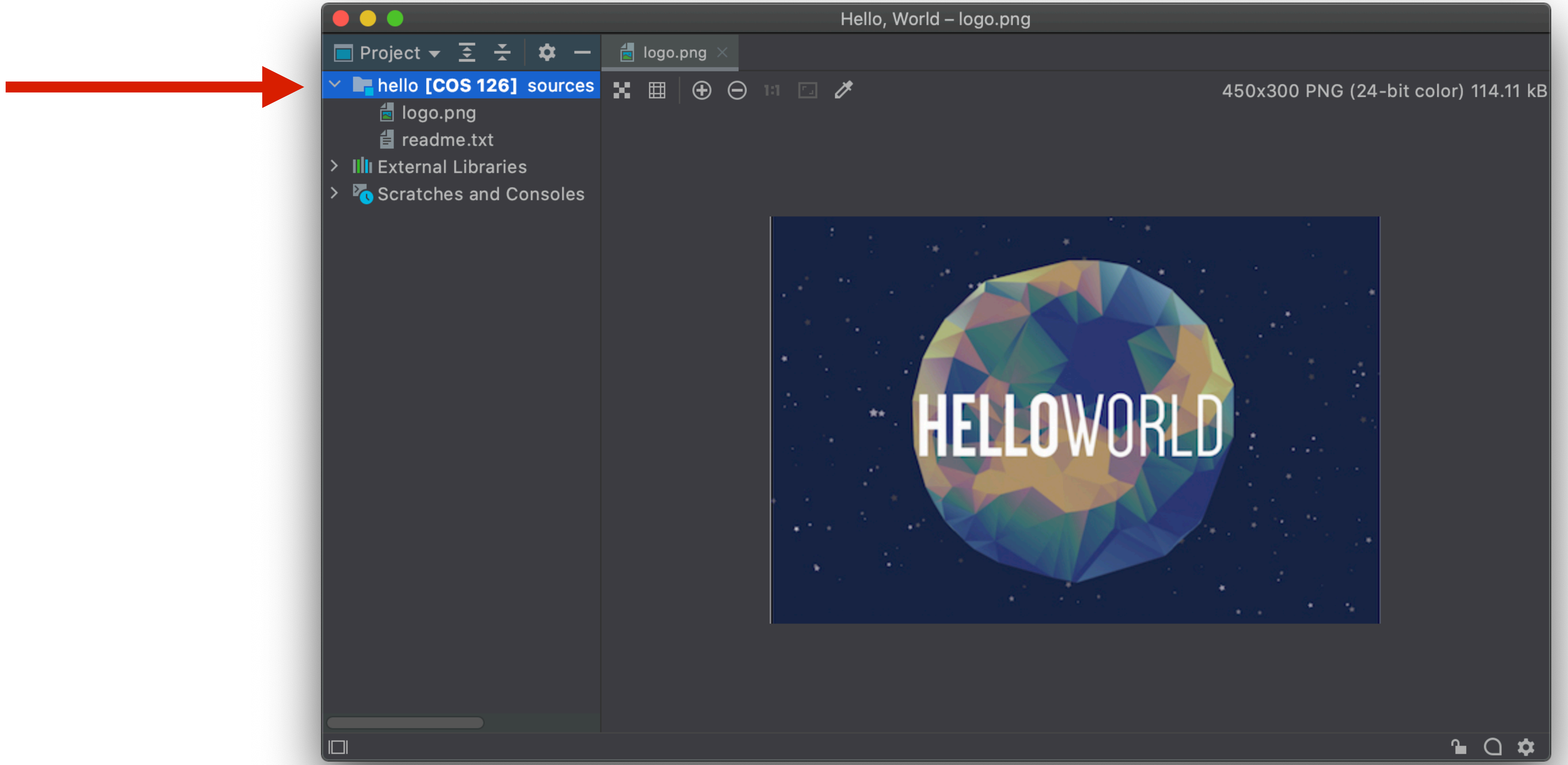
Credit to: https://lift.cs.princeton.edu/java/mac/

Easy: Edit Existing .java Files

More Complicated: Create Your Own .java Files

Credit to: https://lift.cs.princeton.edu/java/mac/

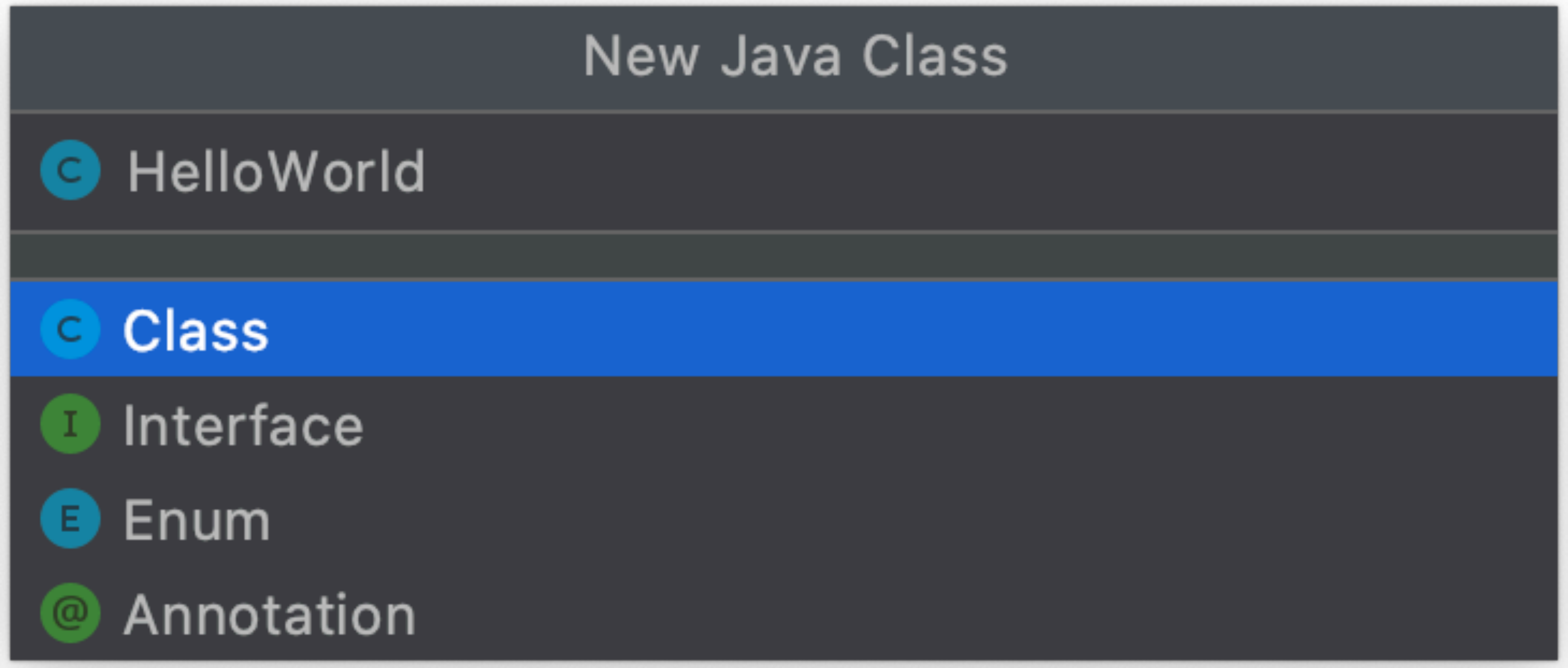
Step 3: Click Project Name



Credit to: https://lift.cs.princeton.edu/java/mac/

Step 4: Create .java File Using the LIFT Menu

- Select the “LIFT” → “New Java Class” menu
- When prompted, type the class name (e.g., HelloWorld)
    - **Don’t type .java!** IntelliJ automatically adds .java.
- Then, press Return

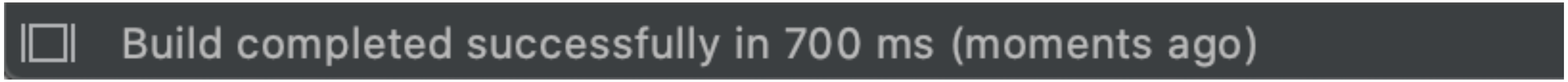

New Java Class

HelloWorld

Credit to: https://lift.cs.princeton.edu/java/mac/

Step 5: Compile Your Code

- Click on the file (e.g., HelloWorld)
- Select the “LIFT” → “Recompile ‘HelloWorld.java’” menu option
- If compilation succeeds, you will receive confirmation in the status bar (at bottom)
    - Otherwise, an error message will be displayed

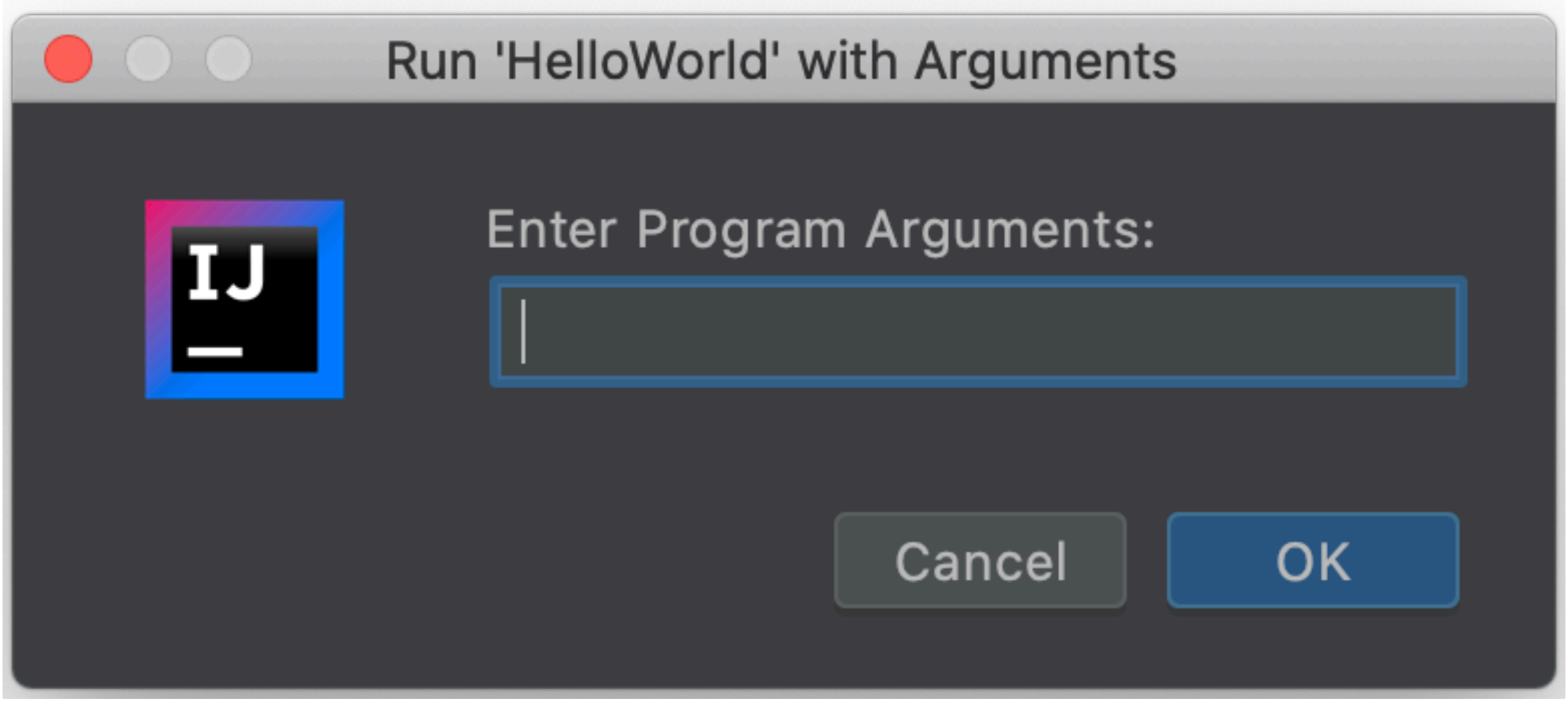


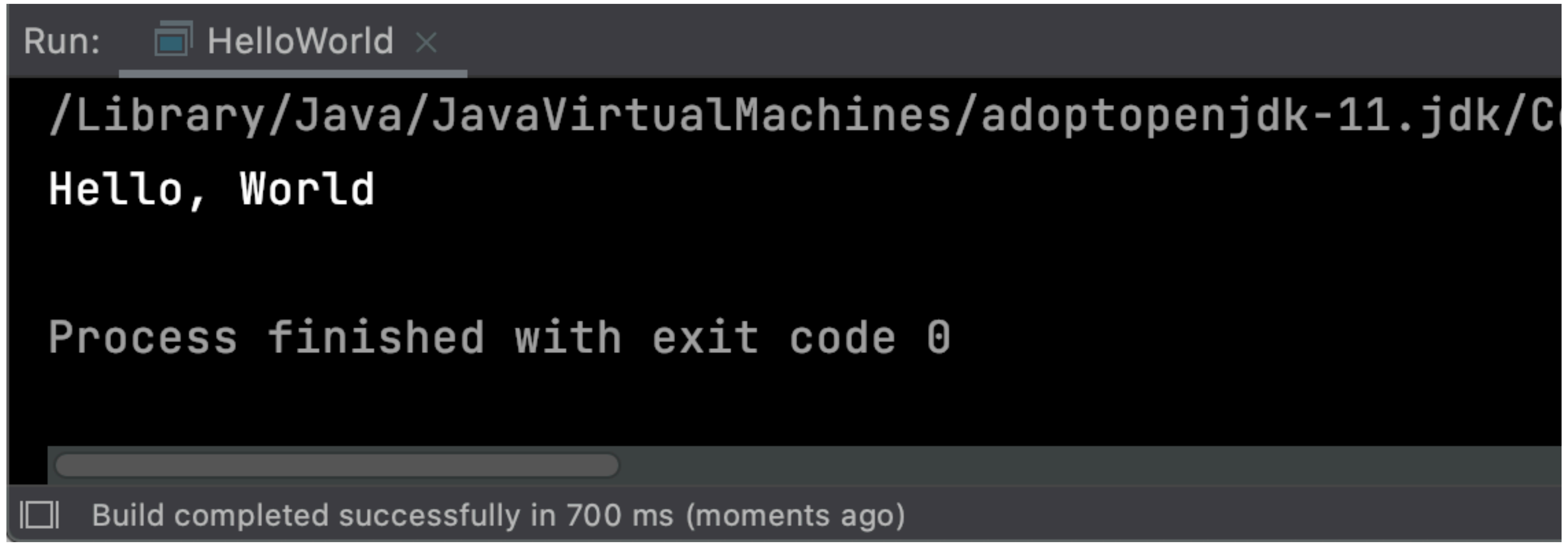
Build completed successfully in 700 ms (moments ago)

Credit to: https://lift.cs.princeton.edu/java/mac/

Step 6: Run Your Code

- Select the “LIFT” → “Run ‘HelloWorld' with Arguments” menu option
  - When prompted, you can optionally enter program arguments
- Program output will appear at the bottom of the window





```
/Library/Java/JavaVirtualMachines/adoptopenjdk-11.jdk/C
Hello, World

Process finished with exit code 0
```

Build completed successfully in 700 ms (moments ago)

Credit to: https://lift.cs.princeton.edu/java/mac/

Technical Details

- What is special about **the version of IntelliJ** you installed?
    - It includes the LIFT plugin
    - It disables more advanced IntelliJ menus (these can be re-enabled, if you’re adventurous)
    - It installs four command-line programs (`javac-introcs, java-introcs, javac-algs4, java-algs4`), and corresponding Java libraries from the textbook authors (`algs4.jar, introcs.jar, stdlib.jar`)

Technical Details

- What is special about **the project starter files**?
  - They contain a hidden .lift folder, which includes libraries from the textbook authors, and unit testing libraries
  - The IntelliJ project is configured to use these files

Before Next Class

- Get started on Homework 1
    - **At least try to install IntelliJ on your PC**
- My office hours are Monday 3pm-4pm and Friday 1:30pm-2:30pm
    - Come prepared with questions!

Start of Semester Survey

Either follow the link on Canvas, or scan this QR code:

I’ll use this survey to take attendance for today.

START RECORDING

Outline

- Attendance quiz
- Built-in data types
- Type conversion

Attendance Quiz:
Differences Between Java and Python

- On a sheet of paper:
    - Write your name and the date
    - Briefly describe **three** differences between Java and Python
- We'll discuss, then you can turn in the paper

Java vs JavaScript

- Java and JavaScript code is completely different!
  - The naming similarities are due to JavaScript trying to piggyback on Java's popularity in the 90s

Java Hello World

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

JavaScript Hello World

```
<html>
<head><title>Hello World</title></head>
<body>
  <script>alert("Hello World!");</script>
</body>
</html>
```

TA Hours

1. Basic Programming Concepts

- Why programming?
- Program development
- Built-in data types
- Type conversion

Built-in data types

A data type is a set of values and a set of operations on those values.

type	*set of values*	*examples of values*	*examples of operations*
`char`	characters	`'A'` `'@'`	compare

`String`	sequences of characters	`"Hello World"`
		`"CS is fun"`
			concatenate

`int`	integers	`17`
		`12345`
			add, subtract, multiply, divide

`double`	floating-point numbers	`3.1415`
		`6.022e23`
			add, subtract, multiply, divide

`boolean`	truth values	`true` `false`	and, or, not

Java's built-in data types

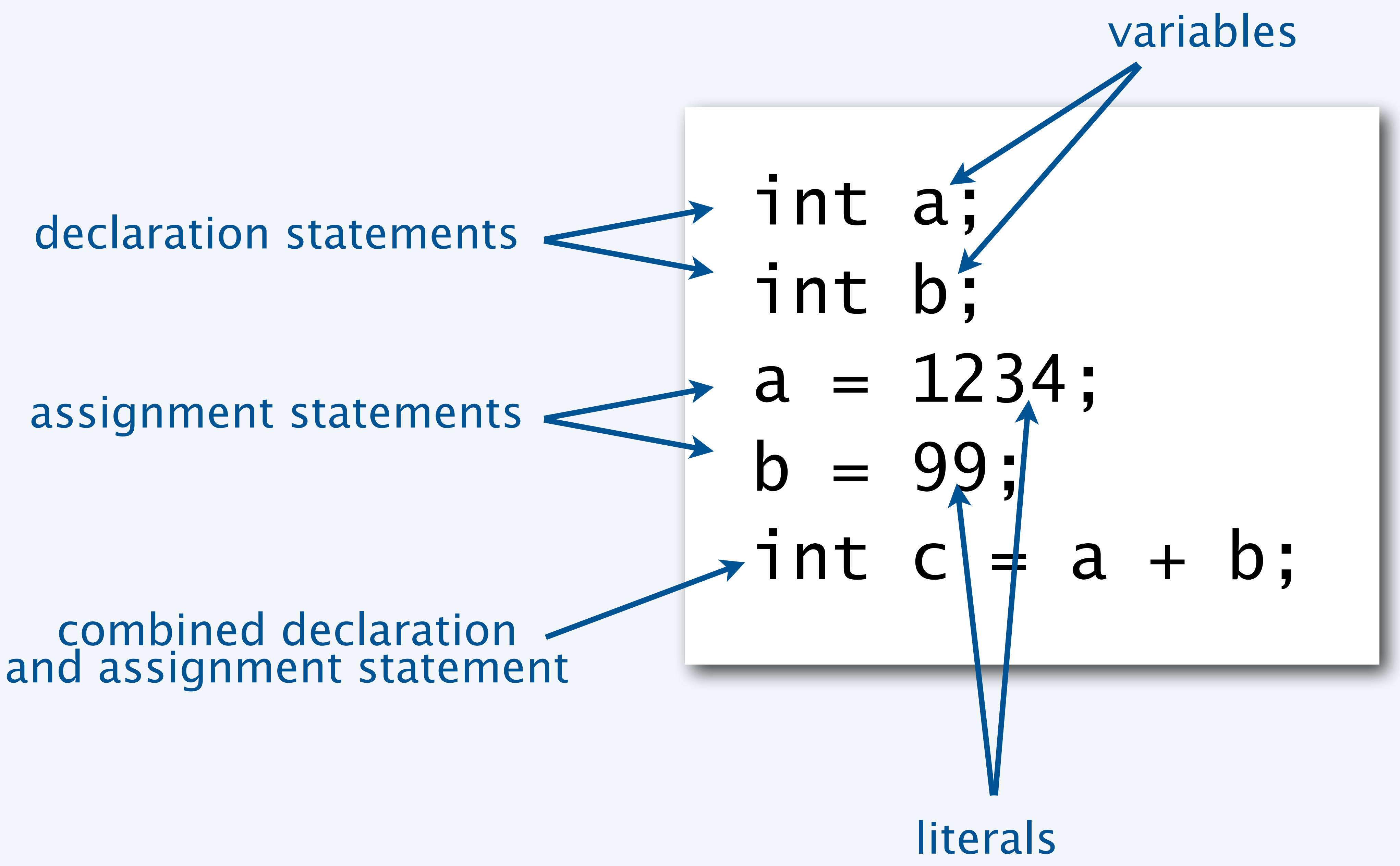
Basic Definitions

A variable is a name that refers to a value.

A literal is a programming-language representation of a value.

A declaration statement associates a variable with a type.

An assignment statement associates a value with a variable.



variables

```
int a;
int b;
a = 1234;
b = 99;
int c = a + b;
```

declaration statements

assignment statements

combined declaration
and assignment statement

literals

Variables, literals, declarations, and assignments example: exchange values

```
public class Exchange
{
    public static void main(String[] args)
    {
        int a = 1234;
        int b = 99;
        int t = a;
        a = b;
        b = t;
    }
}
```

This code *exchanges* the values of a and b.

A trace is a table of variable values after each statement.

	a	b	t
	undeclared	*undeclared*	*undeclared*
`int a = 1234;`	1234	*undeclared*	*undeclared*
`int b = 99;`	1234	99	*undeclared*
`int t = a;`	1234	99	1234
`a = b;`	99	99	1234
`b = t;`	99	1234	1234

Q. What does this program do?

A. No (easy) way for us to see the result of the exchange! (Need output, stay tuned).

Data type for computing with strings: `String`

`String` data type

values	sequences of characters
typical literals	`"Hello, "` `"1 "` `" * "`
operation	concatenate
operator	+

Examples of `String` operations (concatenation)

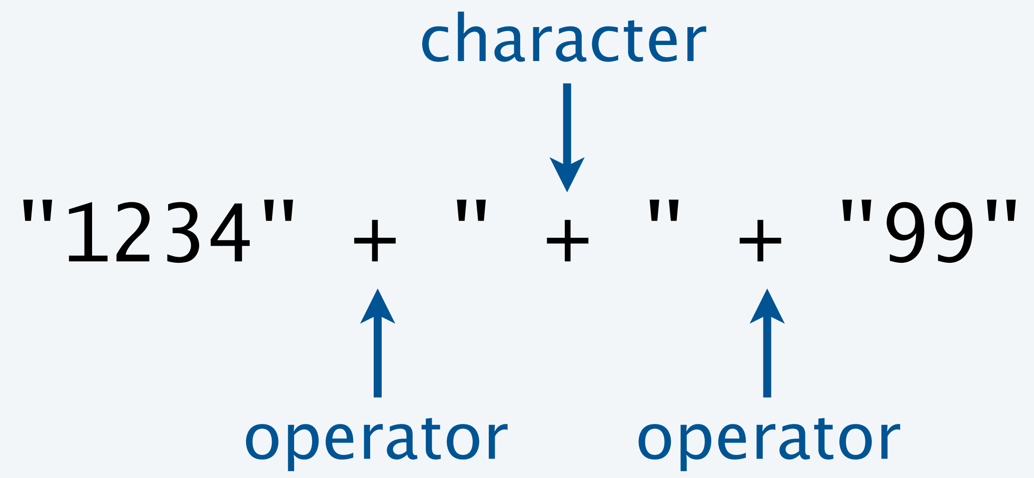
expression	*value*
`"Hi, " + "Bob"`	`"Hi, Bob"`
`"1" + " 2 " + "1"`	`"1 2 1"`
`"1234" + " + " + "99"`	`"1234 + 99"`
`"1234" + "99"`	`"123499"`

Typical use: Input and output.

Important note:

Character interpretation depends on context!

Ex 1: plus signs



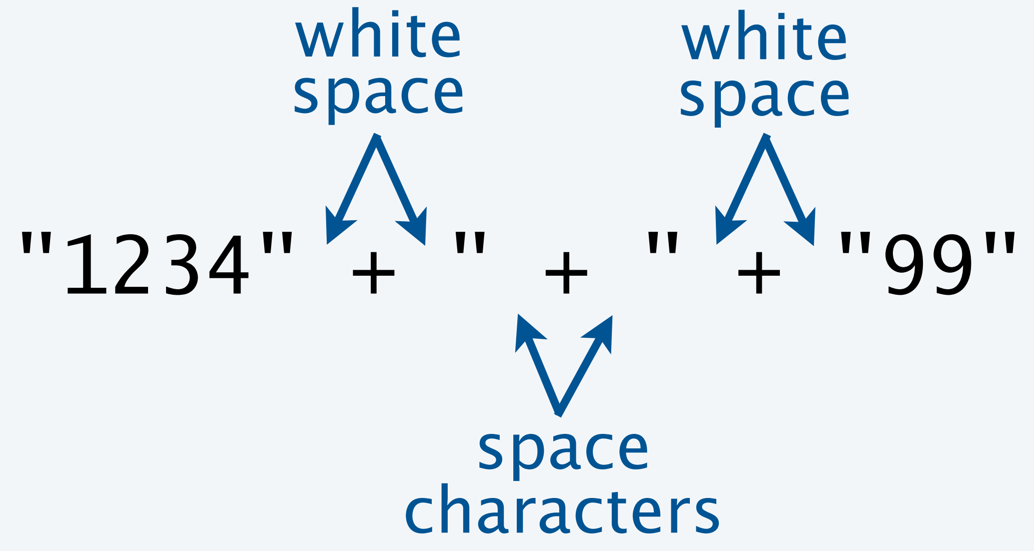
character

`"1234" + " + " + "99"`

operator

operator

Ex 2: spaces



white space

white space

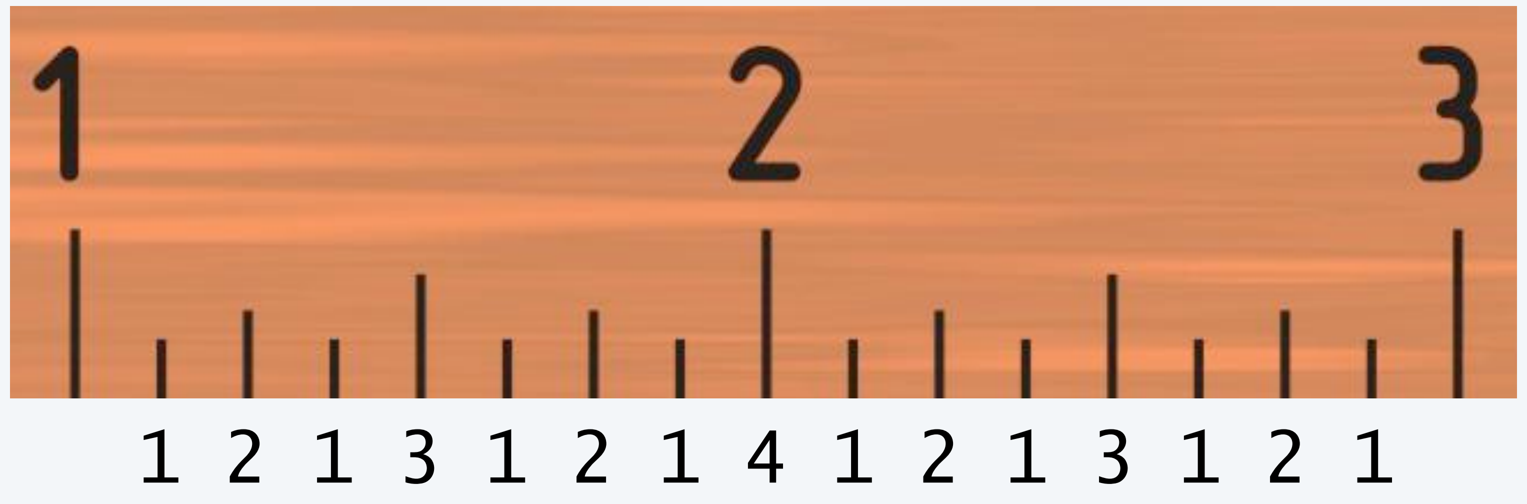
`"1234" + " + " + "99"`

space characters

Example of computing with strings: subdivisions of a ruler

```
public class Ruler
{
   public static void main(String[] args)
   {
      String ruler1 = "1";
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      String ruler4 = ruler3 + " 4 " + ruler3;
      System.out.println(ruler4);
   }
}
```

all + ops are concatenation



```
% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

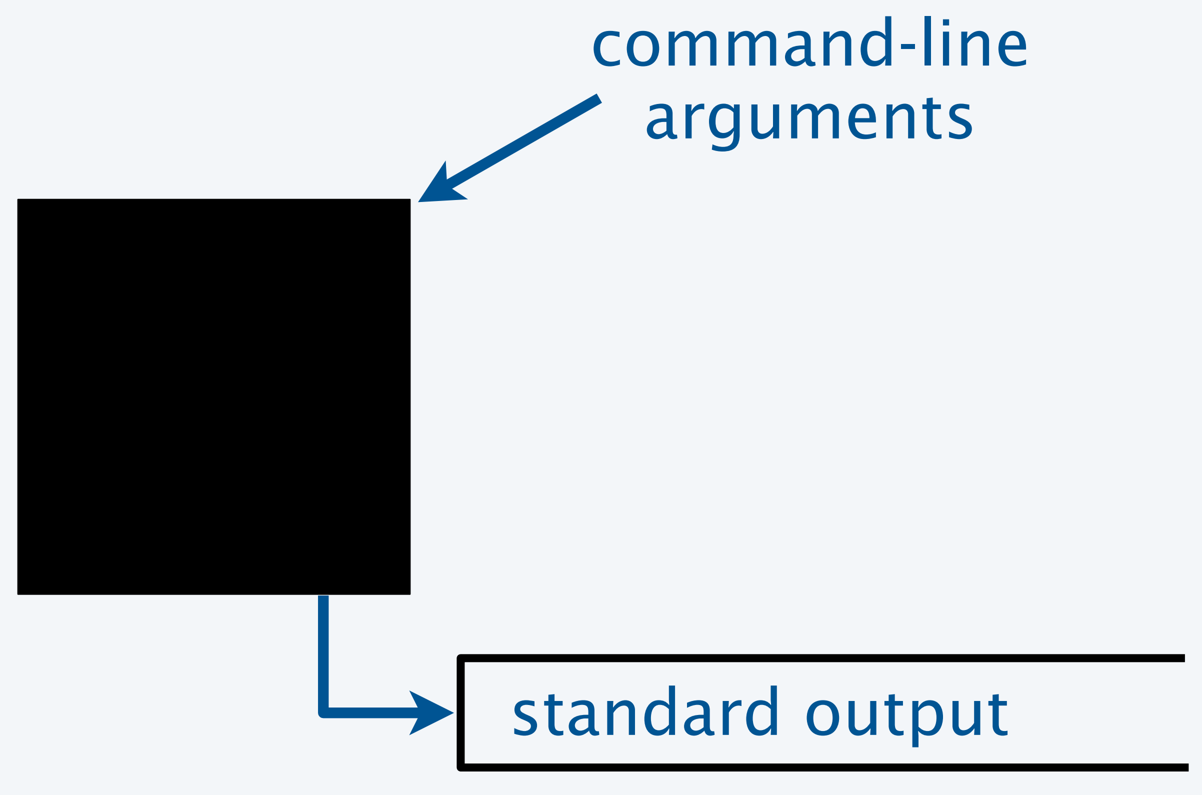
	ruler1	ruler2	ruler3	ruler4
	undeclared	*undeclared*	*undeclared*	*undeclared*
`ruler1 = "1";`	1	*undeclared*	*undeclared*	*undeclared*
`ruler2 = ruler1 + " 2 " + ruler1;`	1	1 2 1	*undeclared*	*undeclared*
`ruler3 = ruler2 + " 3 " + ruler2;`	1	1 2 1	1 2 1 3 1 2 1	*undeclared*
`ruler4 = ruler3 + " 4 " + ruler3;`				1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

Input and output

is necessary for us to provide data to our programs and to learn the result of computations.

Humans prefer to work with strings.

Programs work more efficiently with numbers.



Bird's eye view of a Java program

Output

- `System.out.println()` method prints the given string.
- Java automatically converts numbers to strings for output.

Command-line input

- Strings you type after the program name are available as `args[0]`, `args[1]`, ... at *run* time.
- Q. How do we give an *integer* as command-line input?
- A. Need to call system method `Integer.parseInt()` to convert the strings to integers.

Stay tuned for many more options for input and output, and more details on type conversion.

Input and output warmup: exchange values

```
public class Exchange
{
    public static void main(String[] args)
    {
        int a = Integer.parseInt(args[0]);
        int b = Integer.parseInt(args[1]);
        int t = a;
        a = b;
        b = t;
        System.out.println(a);
        System.out.println(b);
    }
}
```

Java automatically converts int values to String for output

```
% java Exchange 5 2
2
5

% java Exchange 1234 99
99
1234
```

Q. What does this program do?

A. Reads two integers from the command line, then prints them out in the opposite order.

Data type for computing with integers: `int`

`int` data type

values	integers between -2^{31} and $2^{31}-1$				
typical literals	1234 99 0 1000000				
operations	add	subtract	multiply	divide	remainder
operator	+	-	*	/	%

Important note:

Only 2^{32} different `int` values.

↑

not *quite* the same as integers

Examples of `int` operations

expression	*value*	*comment*
`5 + 3`	8	
`5 - 3`	2	
`5 * 3`	15	
`5 / 3`	1	*drop fractional part*
`5 % 3`	2	*remainder*
`1 / 0`		*runtime error*

Precedence

expression	*value*	*comment*
`3 * 5 - 2`	13	* *has precedence*
`3 + 5 / 2`	5	/ *has precedence*
`3 - 5 - 2`	-4	*left associative*
`( 3 - 5 ) - 2`	-4	*better style*

Typical usage: Math calculations; specifying programs (stay tuned).

Example of computing with integers and strings, with type conversion

```
public class IntOps
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }
}
```

Java automatically converts int values to String for concatenation

```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

?

Note: 1234 = 12*99 + 46

Data type for computing with floating point numbers: double

double data type

values	real numbers				
typical literals	3.14159	2.0	1.4142135623730951	6.022e23	
operations	add	subtract	multiply	divide	remainder
operator	+	-	*	/	%

6.022×10^{23}

Typical double values are *approximations*

Examples:

no double value for π.

no double value for $\sqrt{2}$

no double value for 1/3.

Examples of double operations

expression	*value*
3.141 + .03	3.171
3.141 - .03	3.111
6.02e23/2	3.01e23
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
Math.sqrt(2.0)	1.4142135623730951

Special values

expression	*value*
1.0 / 0.0	Infinity
Math.sqrt(-1.0)	NaN

"not a number"

Typical use: Scientific calculations.

Other built-in numeric types

short data type

values	integers between -2^{15} and $2^{15}-1$
operations	[same as int]

long data type

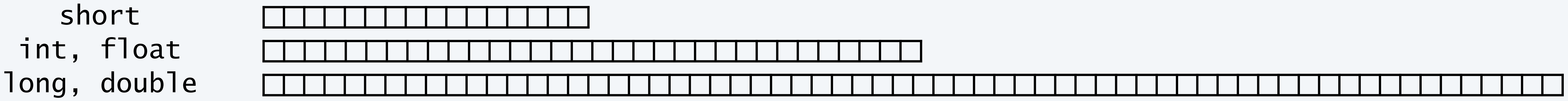
values	integers between -2^{63} and $2^{63}-1$
operations	[same as int]

float data type

values	real numbers
operations	[same as double]

Why different numeric types?

- Tradeoff between memory use and range for integers.
- Tradeoff between memory use and precision for real numbers.



Excerpts from Java’s Math Library

public class Math	
double abs(double a)	*absolute value of* a
double max(double a, double b)	*maximum of* a *and* b
double min(double a, double b)	*minimum of* a *and* b
double sin(double theta)	*sine function*
double cos(double theta)	*cosine function*
double tan(double theta)	*tangent function*
double exp(double a)	*exponential* (e^a)
double log(double a)	*natural log* ($\log_e$ a, *or* ln a)
double pow(double a, double b)	*raise a to the bth power* (a^b)
long round(double a)	*round to the nearest integer*
double random()	*random number in [0. 1)*
double sqrt(double a)	*square root of* a
double E	*approx. value of e (constant)*
double PI	*approx. value of* π *(constant)*

also defined for int, long, and float

inverse functions also available: asin(), acos(), and atan()

Degrees in radians. Use toDegrees() and toRadians()) to convert.

No need for calculators in this course!

Example of computing with floating point numbers: quadratic equation

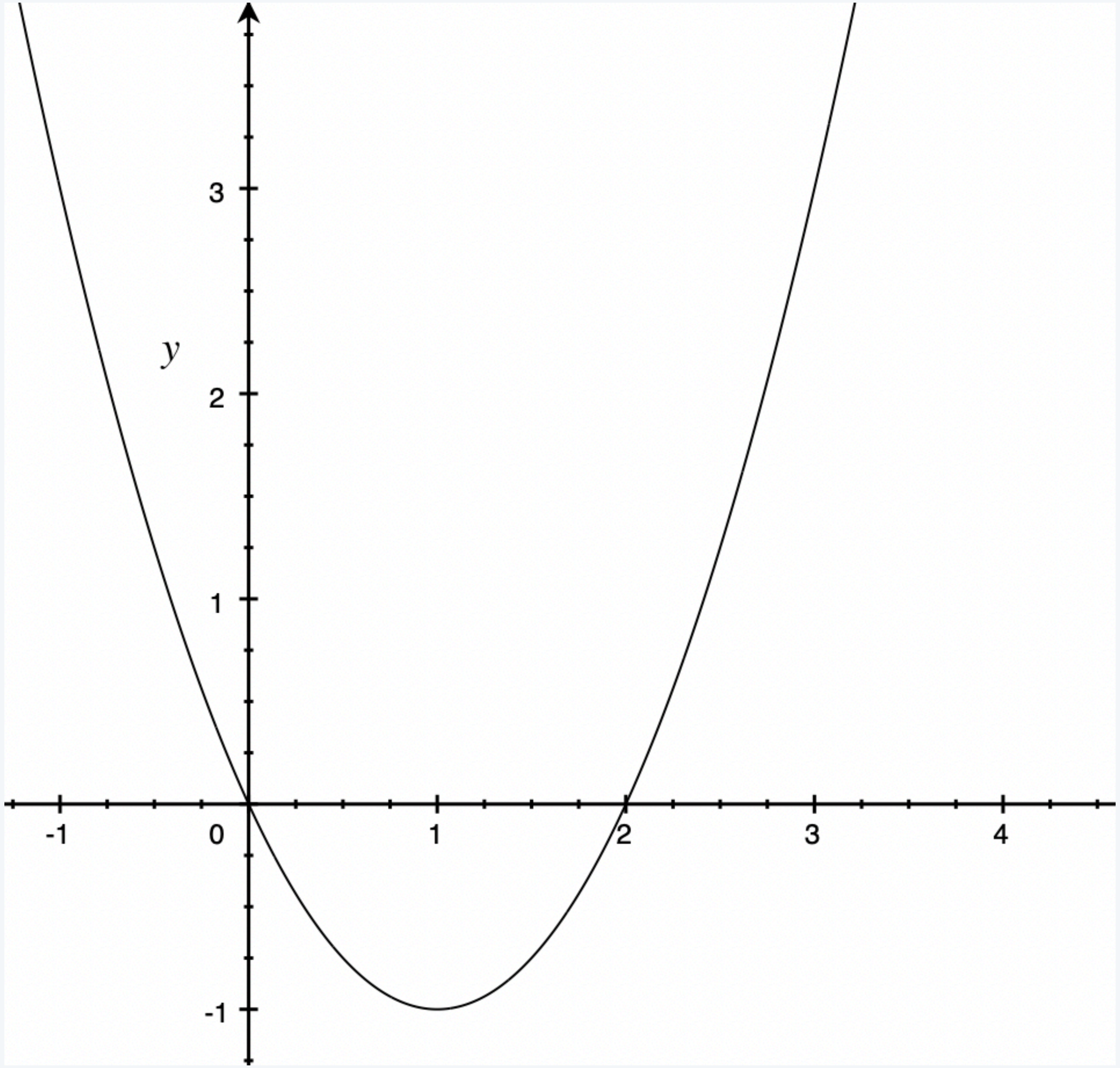
From algebra: the roots of $x^2 + bx + c$ are $\dfrac{-b \pm \sqrt{b^2 - 4c}}{2}$

What are roots?

Example

Roots of:

$$y = x^2 - 2x + 0$$



Example of computing with floating point numbers: quadratic equation

From algebra: the roots of $x^2 + bx + c$ are $\dfrac{-b \pm \sqrt{b^2 - 4c}}{2}$

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

      // Parse coefficients from command-line.
      double b = Double.parseDouble(args[0]);
      double c = Double.parseDouble(args[1]);

      // Calculate roots of x*x + b*x + c.
      double discriminant = b*b - 4.0*c;
      double d = Math.sqrt(discriminant);
      double root1 = (-b + d) / 2.0;
      double root2 = (-b - d) / 2.0;

      // Print them out.
      System.out.println(root1);
      System.out.println(root2);
   }
}
```

```
% java Quadratic -3.0 2.0
2.0
1.0
```

$x^2 - 3x + 2$

```
% java Quadratic -1.0 -1.0
1.618033988749895
-0.6180339887498949
```

$x^2 - x - 1$

```
% java Quadratic 1.0 1.0
NaN
NaN
```

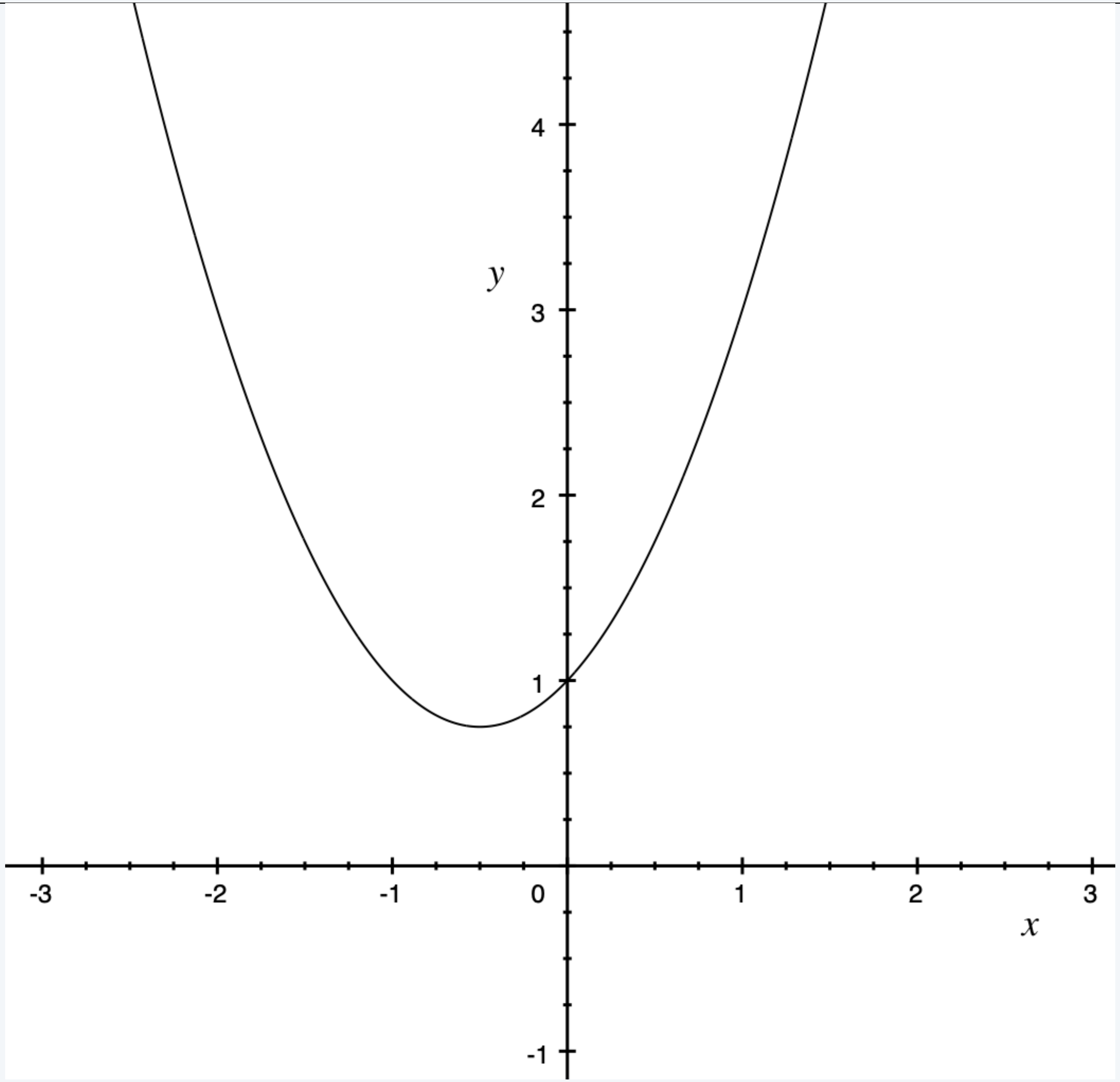
$x^2 + x + 1$

```
% java Quadratic 1.0 hello
java.lang.NumberFormatException: hello

% java Quadratic 1.0
java.lang.ArrayIndexOutOfBoundsException
```

Need two arguments.
(Fact of life: Not all error messages are crystal clear.)

Where are the roots?



Example

Roots of:

$$y = x^2 + x + 1$$

No roots!

Data type for computing with true and false: boolean

boolean data type

values	true	false	
literals	true	false	
operations	and	or	not
operator	&&	||	!

Truth-table definitions

a	!a		a	b	a && b	a || b
true	false		false	false	false	false
false	true		false	true	false	true
			true	false	false	true
			true	true	true	true

Q. a XOR b?

A. (!a && b) || (a && !b)

Proof

a	b	!a && b	a && !b	(!a && b) || (a && !b)
false	false	false	false	false
false	true	true	false	true
true	false	false	true	true
true	true	false	false	false

Typical usage: Control logic and flow of a program (stay tuned).

Comparison operators

Fundamental operations that are defined for each primitive type allow us to *compare* values.

- Operands: two expressions of the same type.
- Result: a value of type `boolean`.

operator	*meaning*	`true`	`false`
`==`	equal	`2 == 2`	`2 == 3`
`!=`	not equal	`3 != 2`	`2 != 2`
`<`	less than	`2 < 13`	`2 < 2`
`<=`	less than or equal	`2 <= 2`	`3 <= 2`
`>`	greater than	`13 > 2`	`2 < 13`
`>=`	greater than or equal	`3 >= 2`	`2 >= 3`

Examples

non-negative discriminant?	`( b*b - 4.0*a*c ) >= 0.0`
beginning of a century?	`( year % 100 ) == 0`
legal month?	`( month >= 1 ) && ( month <= 12 )`

Typical double values are *approximations* so beware of == comparisons

Example of computing with booleans: leap year test

Q. Is a given year a leap year?

A. Yes if either (i) divisible by 400 or (ii) divisible by 4 but not 100.

```
public class LeapYear
{
   public static void main(String[] args)
   {
      int year = Integer.parseInt(args[0]);
      boolean isLeapYear;

      // divisible by 4 but not 100
      isLeapYear = (year % 4 == 0) && (year % 100 != 0);

      // or divisible by 400
      isLeapYear = isLeapYear || (year % 400 == 0);

      System.out.println(isLeapYear);
   }
}
```

```
% java LeapYear 2016
true

% java LeapYear 1993
false

% java LeapYear 1900
false

% java LeapYear 2000
true
```

Image sources

http://commons.wikimedia.org/wiki/File:Calculator_casio.jpg

COMPUTER SCIENCE

SEDGEWICK/WAYNE

PART I: PROGRAMMING IN JAVA

1. Basic Programming Concepts

- Why programming?
- Program development
- Built-in data types
- Type conversion

Type checking

Types of variables involved in data-type operations always must match the definitions.

The Java compiler is your *friend*: it checks for type errors in your code.

```
public class BadCode
{
   public static void main(String[] args)
   {
      String s = "123" * 2;
   }
}
```

```
% javac BadCode.java
BadCode.java:5: operator * cannot be applied to java.lang.String,int
        String s = "123" * 2;
                         ^
1 error
```

When appropriate, we often *convert* a value from one type to another to make types match.

Type conversion with built-in types

Type conversion is an essential aspect of programming.

Automatic

- Convert number to string for "+".
- Make numeric types match if no loss of precision.

expression	*type*	*value*
"x: " + 99	String	"x: 99"
11 * 0.25	double	2.75

Explicitly defined for function call.

Integer.parseInt("123")	int	123
Math.round(2.71828)	long	3

Cast for values that belong to multiple types.

- Ex: small integers can be short, int or long.
- Ex: double values can be truncated to int values.

(int) 2.71828	int	2
(int) Math.round(2.71828)	int	3
11 * (int) 0.25	int	0

⚠ Pay attention to the type of your data. ← Type conversion can give counterintuitive results but gets easier to understand with practice

Pop quiz on type conversion

Q. Give the type and value of each of the following expressions.

a. `( 7 / 2 ) * 2.0`

b. `( 7 / 2.0 ) * 2`

c. `"2" + 2`

d. `2.0 + "2"`

Pop quiz on type conversion

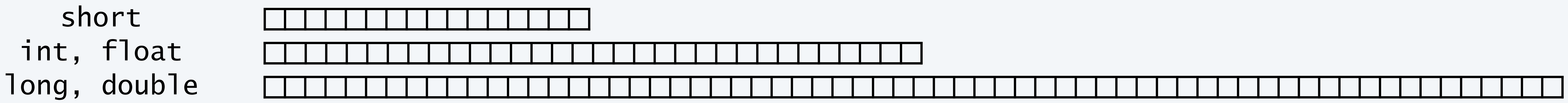
Q. Give the type and value of each of the following expressions.

a. `( 7 / 2 ) * 2.0`	6.0, a `double` (7/2 is 3, an `int`)
b. `( 7 / 2.0 ) * 2`	7.0, a `double`
c. `"2" + 2`	22, a `String`
d. `2.0 + "2"`	2.02, a `String`

An instructive story about type conversion

Why different numeric types?

- Tradeoff between memory use and range for integers.
- Tradeoff between memory use and precision for floating-point.



A conversion may be impossible.

- Example: `(short) 70000`.
- Short values must be between -2^{15} and $2^{15} - 1 = 32767$.

First launch of Ariane 5, 1996

https://www.bugsnag.com/blog/bug-day-ariane-5-disaster

What to do with an impossible conversion?

- Approach 1: Avoid doing it in the first place.
- Approach 2 (Java): Live with a well-defined result.
- Approach 3: Crash.

Example of type conversion put to good use: pseudo-random integers

System method Math.random() returns a pseudo-random double value in [0, 1).

Problem: Given *N*, generate a pseudo-random *integer* between 0 and *N* − 1.

```
public class RandomInt
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      double r = Math.random();
      int t = (int) (r * N);

      System.out.println(t);
   }
}
```

String to int (system method)

double to int (cast)

int to double (automatic)

```
% java RandomInt 6
3

% java RandomInt 6
0

% java RandomInt 10000
3184
```

Summary

A data type is a set of values and a set of operations on those values.

Commonly-used built-in data types in Java

- `String`, for computing with *sequence of characters*, for input and output.
- `int`, for computing with *integers*, for math calculations in programs.
- `double`, for computing with *floating point numbers*, typically for science and math apps.
- `boolean`, for computing with *true* and *false*, for decision making in programs.

In Java you must:

- Declare the types of your variables.
- Convert from one type to another when necessary.
- Identify and resolve type errors in order to *compile* your code.

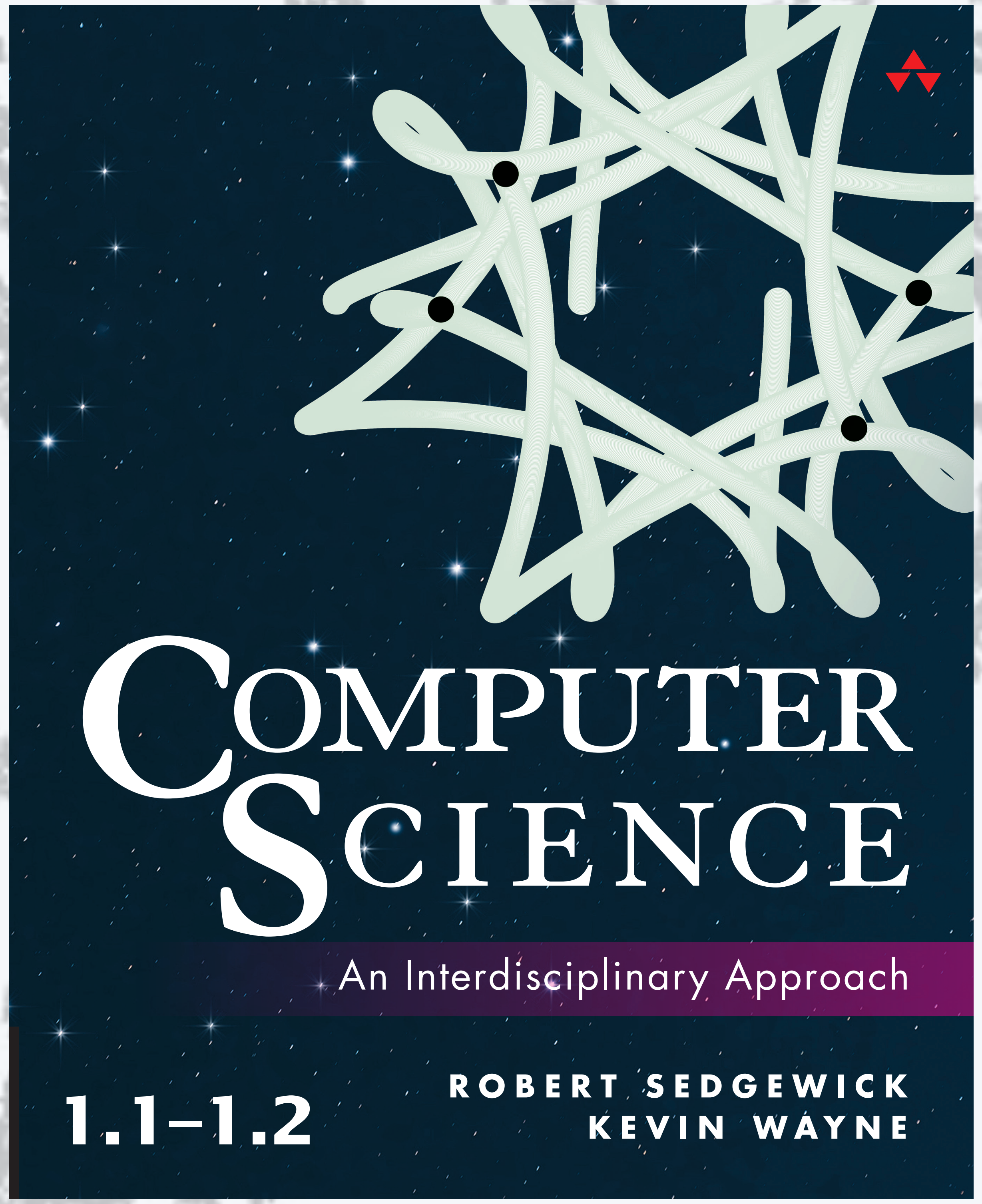
Pay attention to the type of your data.

The Java compiler is your *friend*: it will help you identify and fix type errors in your code.

COMPUTER SCIENCE

SEDGEWICK / WAYNE

PART I: PROGRAMMING IN JAVA



http://introcs.cs.princeton.edu

1. Basic Programming Concepts