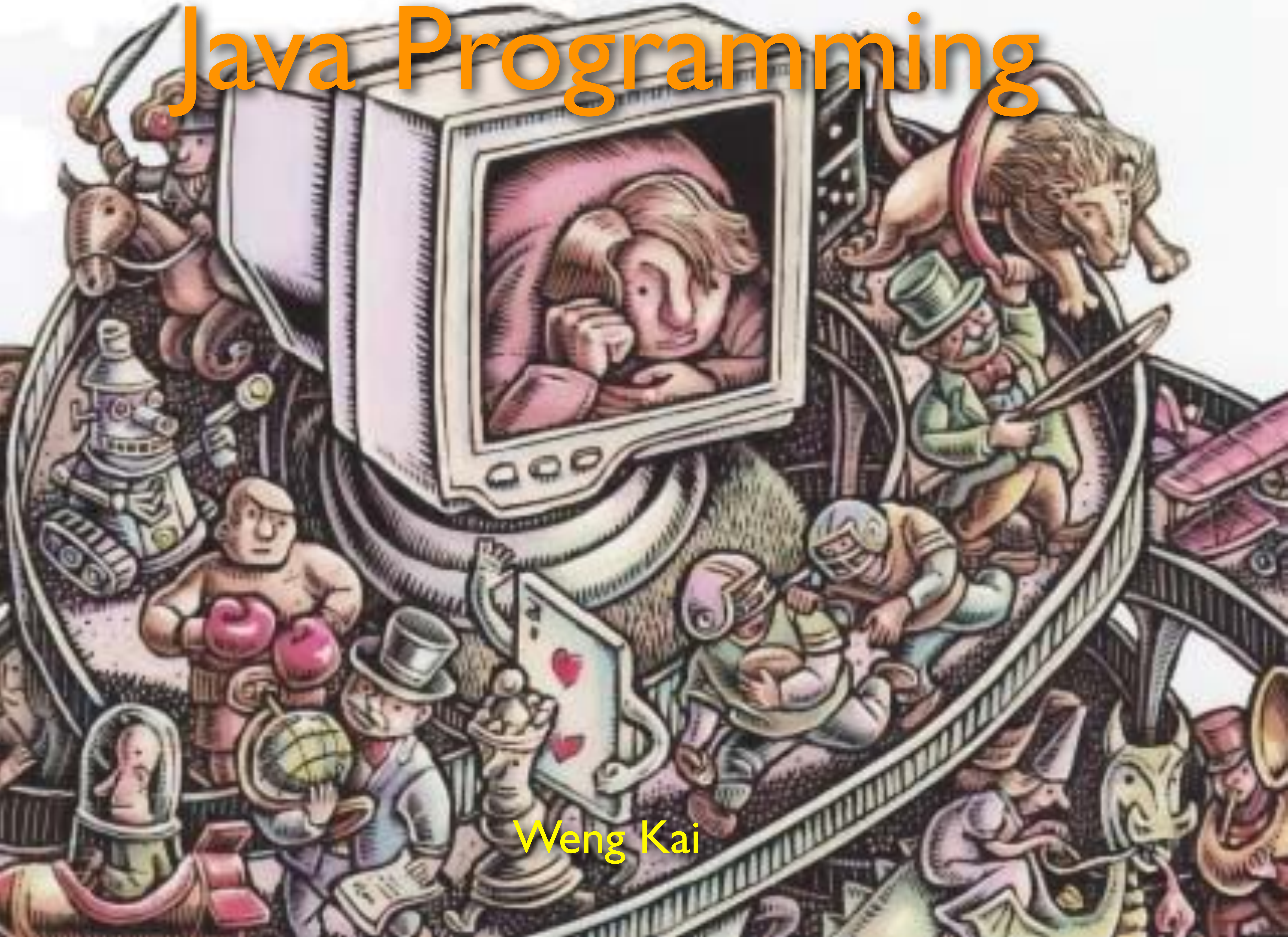
Java Programming



Weng Kai

Goals

- Focus on Java application programming
  - desktop or server
- Be able to implement a small software system in Java

内容

- Java as a language
- IO, exception, RTTI
- GUI, MVC
- Threads
- Unit test, CI
- JDBC, socket
- Design patterns
- Lambda, Stream
- parallel, concurrency, distributed

Assessment

1. 作业：10%
   1. 课后：PTA，截止下周上课前
   2. 课内：选择、是非、讨论
2. 实验：40%，4次，PTA，截止下次实验前，计归一分
3. 大程：20%，2次，学在浙大，有实验报告，人工批改
4. 期末考试：30%，PTA，3个班统一

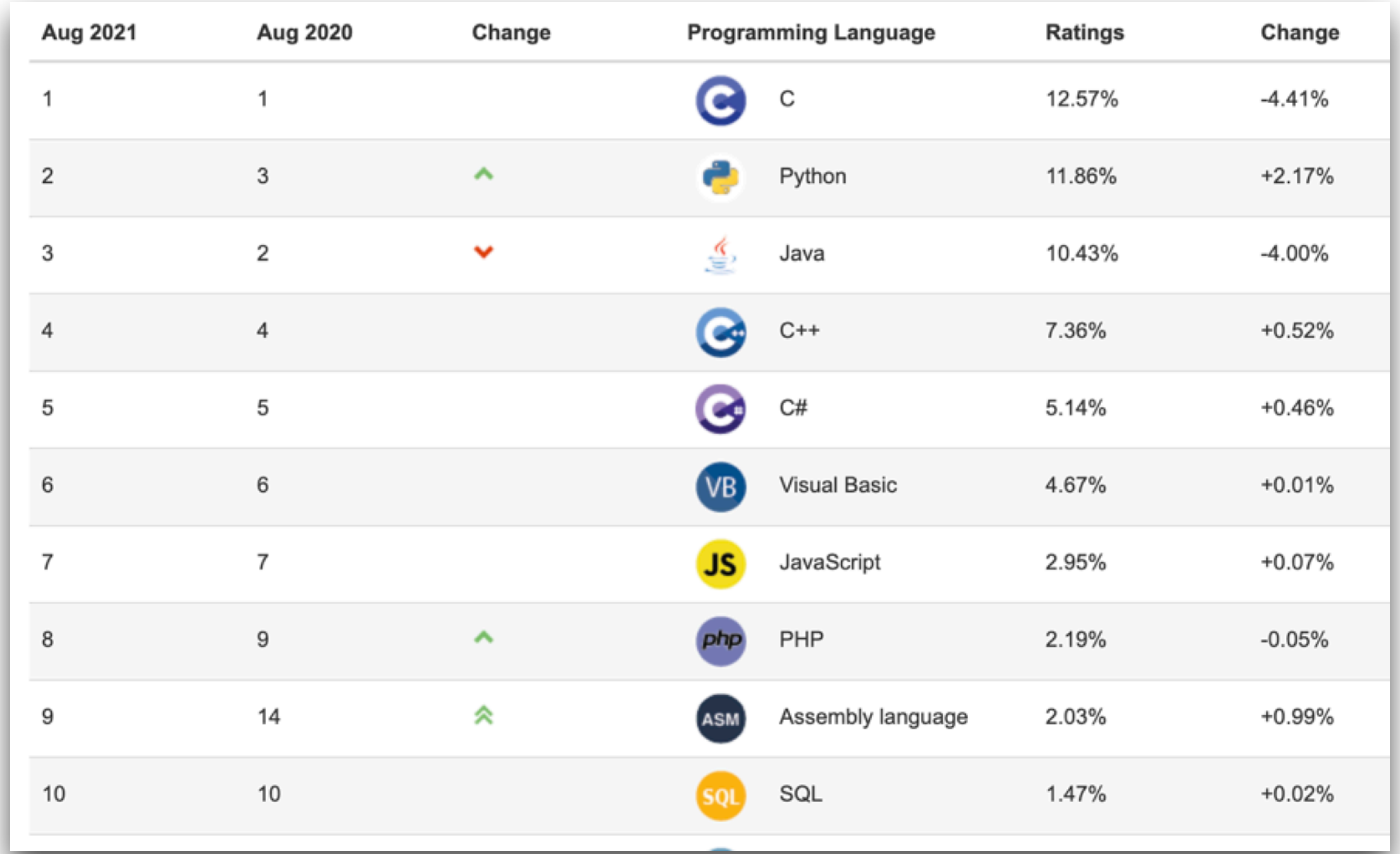
course web

- 课件：http://course.zju.edu.cn
- 实验、作业、大程、考试：https://pintia.cn

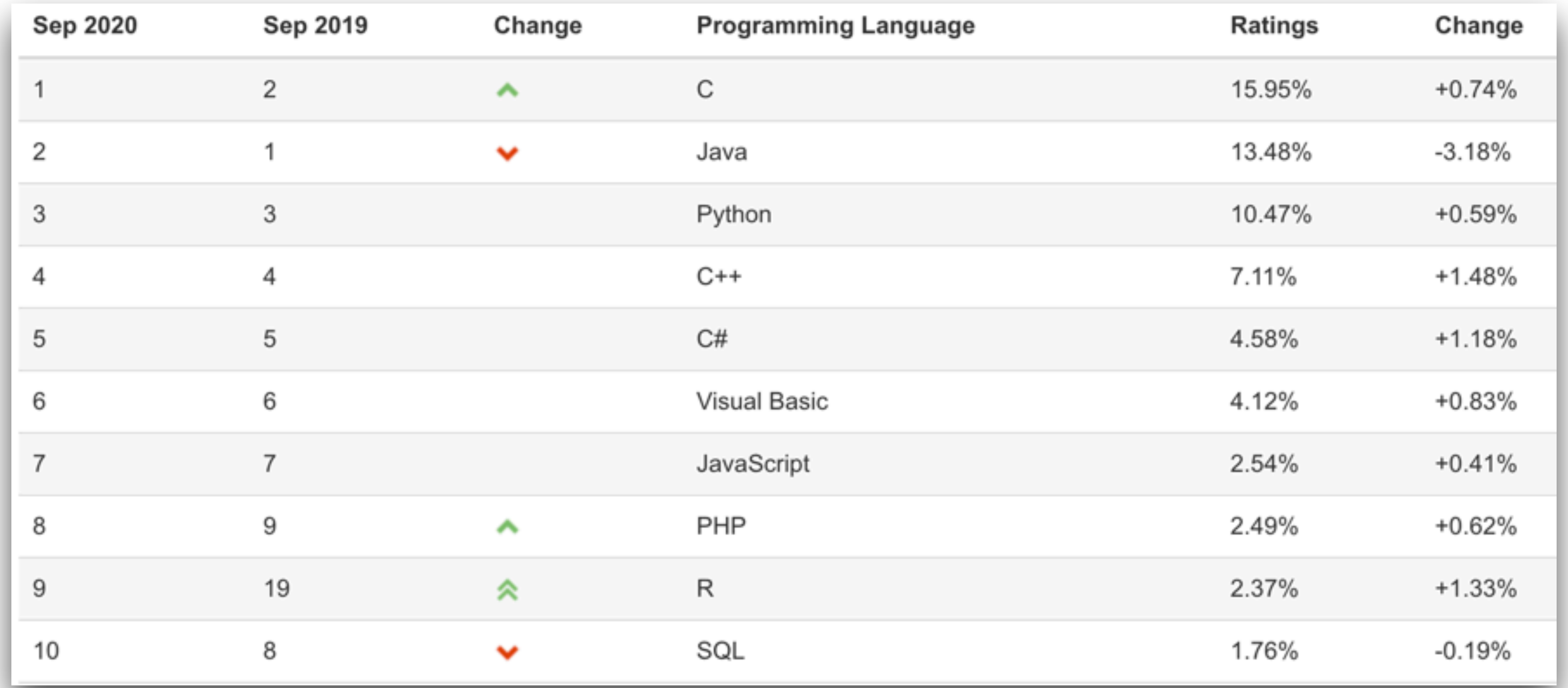
Software you need

- JDK
- Visual Studio Code
- Eclipse
- IDEA

学哪个语言

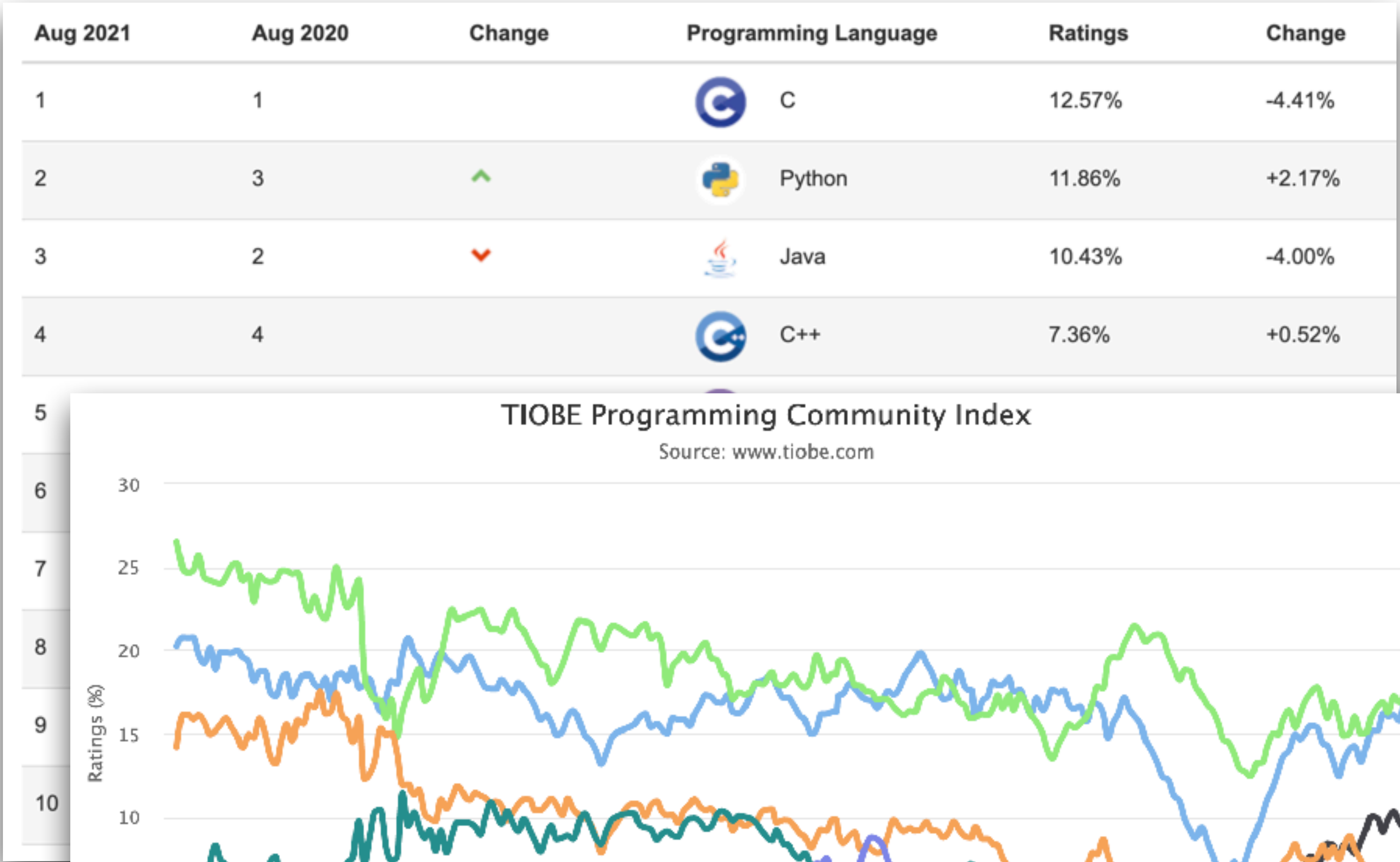


Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		C	12.57%	-4.41%
2	3	▲	Python	11.86%	+2.17%
3	2	▼	Java	10.43%	-4.00%
4	4		C++	7.36%	+0.52%
5	5		C#	5.14%	+0.46%
6	6		Visual Basic	4.67%	+0.01%
7	7		JavaScript	2.95%	+0.07%
8	9	▲	PHP	2.19%	-0.05%
9	14	▲▲	Assembly language	2.03%	+0.99%
10	10		SQL	1.47%	+0.02%

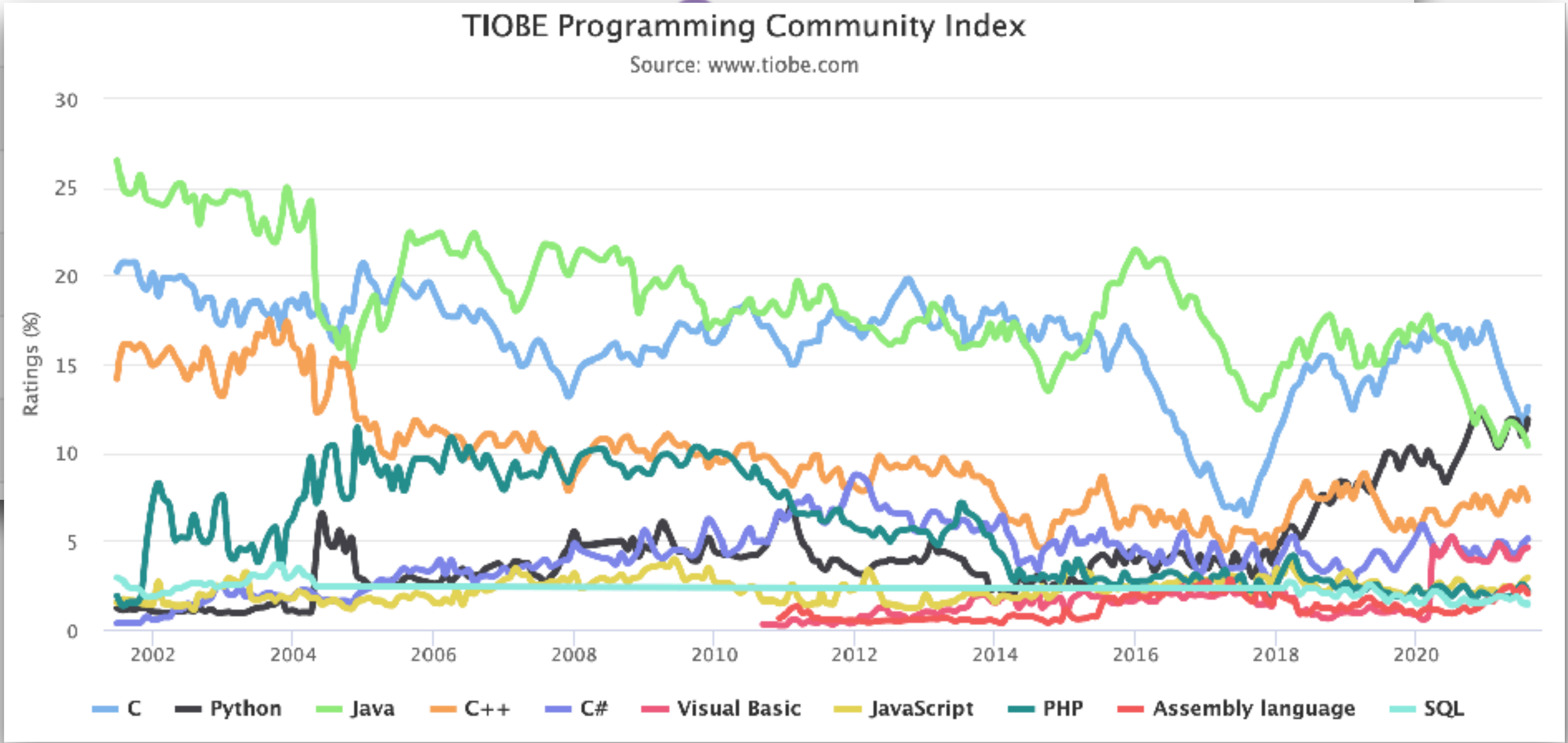


Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	15.95%	+0.74%
2	1	▼	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	▲	PHP	2.49%	+0.62%
9	19	▲▲	R	2.37%	+1.33%
10	8	▼	SQL	1.76%	-0.19%

图片来源：http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

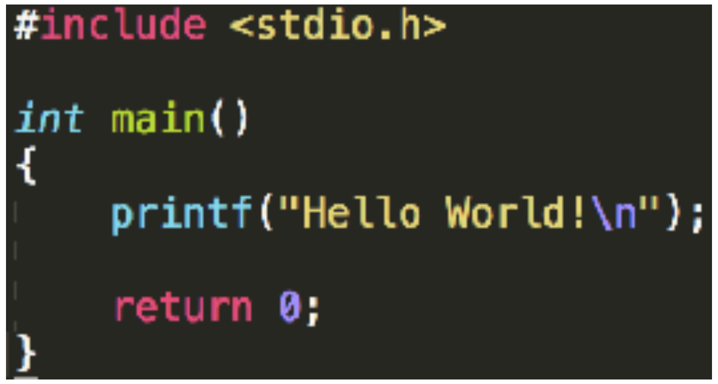


Aug 2021	Aug 2020	Change	Programming Language	Ratings	Change
1	1		C	12.57%	-4.41%
2	3	⌃	Python	11.86%	+2.17%
3	2	⌄	Java	10.43%	-4.00%
4	4		C++	7.36%	+0.52%
5					
6					
7					
8					
9					
10					



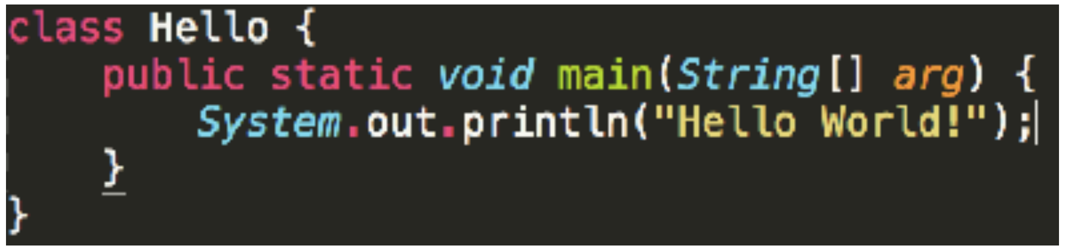
图片来源：<u>http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html</u>

其他语言?



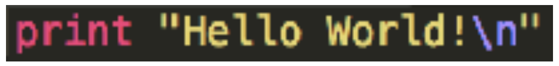
```
#include <stdio.h>

int main()
{
    printf("Hello World!\n");

    return 0;
}
```

- 现代的编程语言在语法上的差异很小
    - 几乎都是C-like语言



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

- 语言的能力/适用领域主要是由
    - 库, and



```
print "Hello World!\n"
```

- 传统所决定的

Java生逢其时

- 计算模式在演变
  - 主机—>单机—>C/S—>B/S—>移动与云
  - 计算在远端->计算在桌面->桌面计算远端存储->桌面交互远端计算和存储->前端交互远端计算和存储
- Java一直不太擅长桌面，但是桌面不那么重要了
- 语言上升是因为有库，有人写库是因为它流行

前端/后端模式

- CGI->MVC->服务化
- 从前：一次点击做一次提交换一个新页面下来
- 现在：一点动作做一次提交得到新数据做部分刷新
- 网站不再是提供网页的站，而是服务提供者
- 前端：HTML、CSS、JavaScript
- 后端：Java、PHP、.net、Python以及JavaScript

为什么Java受到挑战

- 一门语言独领风骚20年了，总有人想要挑战的
- Java还是带有明显的20年前的语言的特色：编译执行
  - Java源码—>字节码—>虚拟机执行
- 新的语言普遍采用解释执行
- Java起步的时候也是做网页中运行的程序的，但是Java太重了

为什么Java还是第一

- Java仍然是大多数情况下首先想到的
  - 程序是人写的，用惯的工具不会一下子扔了的
  - Java适合于很多场合，不仅仅是服务器后端
- 语言的生命周期是很长的
  - Java还在不断发展：IoC、泛型、Lamba和流计算
- 新语言的问题
  - 普遍在应对多人协同开发上存在弱点

结论建议

- 对计算机本身（体系结构、操作系统、编译）感兴趣—>C
- 想编程解决手头的问题（统计、AI、桌面小程序）—>Python
- 有明确的需求（求职）—>人家要什么学什么（PHP、JavaScript、C++）
- 还没想好 —>Java

The Java

Programming Languages

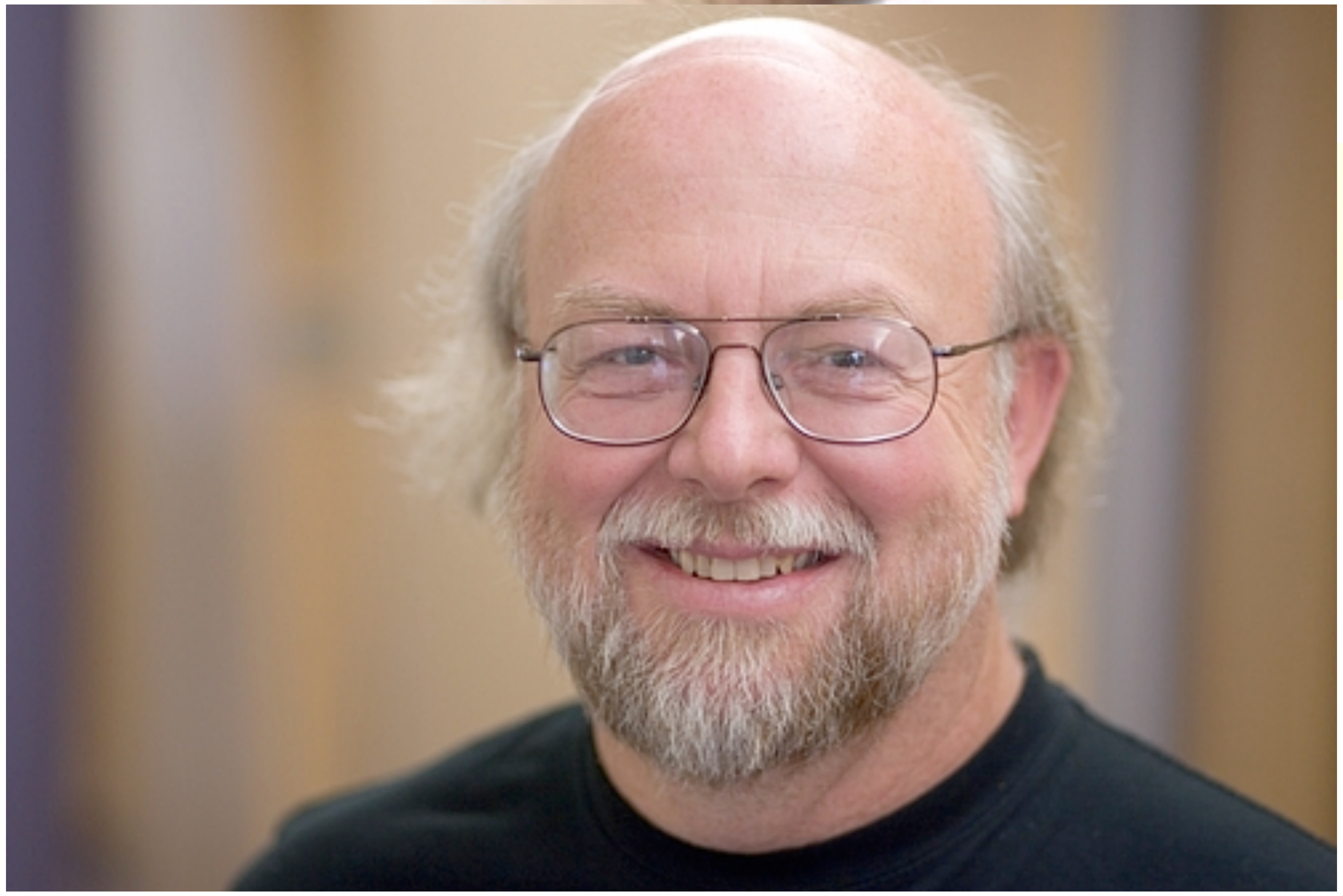
- is a way to describe how to do a stuff
- compiled vs. interpreted
  - C
  - Python

Java

- A compiled language
- A productive OOP language
- A member of UNIX world
- Widely used in variety kinds of applications

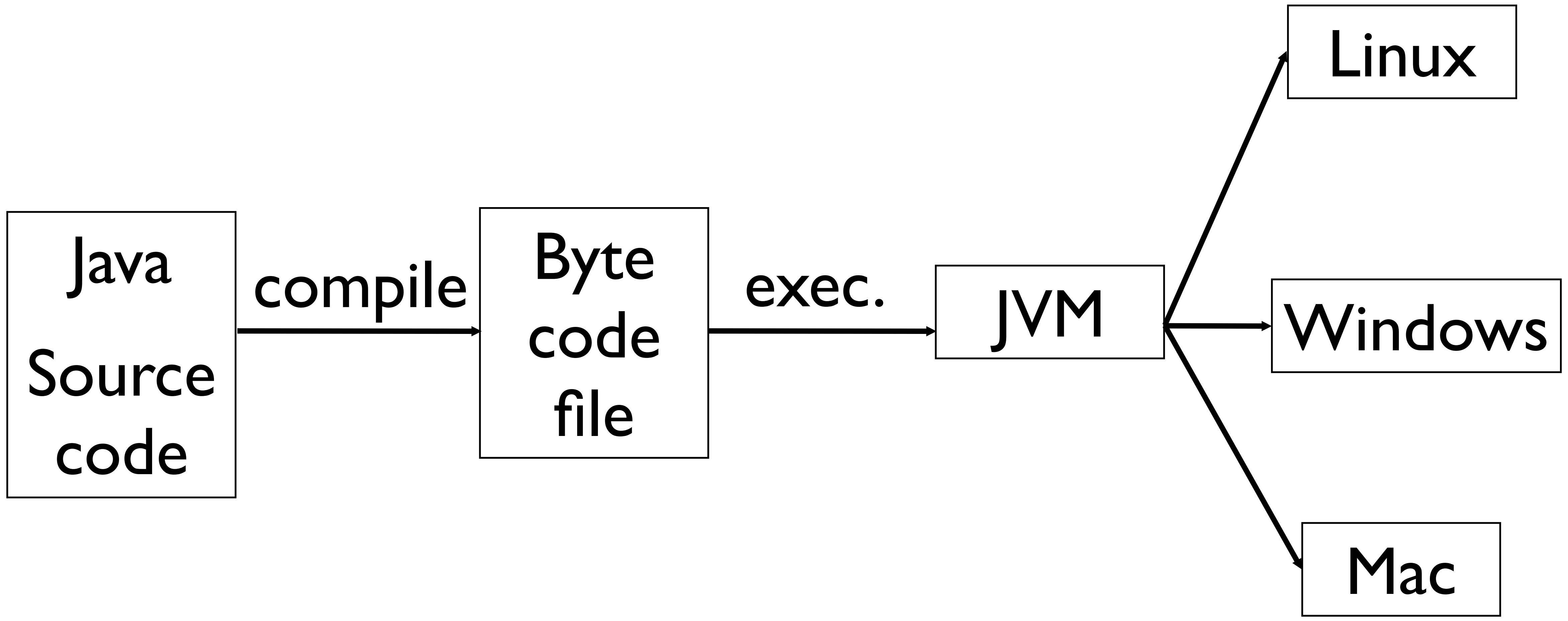
History of Java

- Birthday: May 23, 1995
- 1991: Set-Top box: James Gosling
- 1994: OAK and FirstPerson
- 1995: Java and HotJava – Internet play



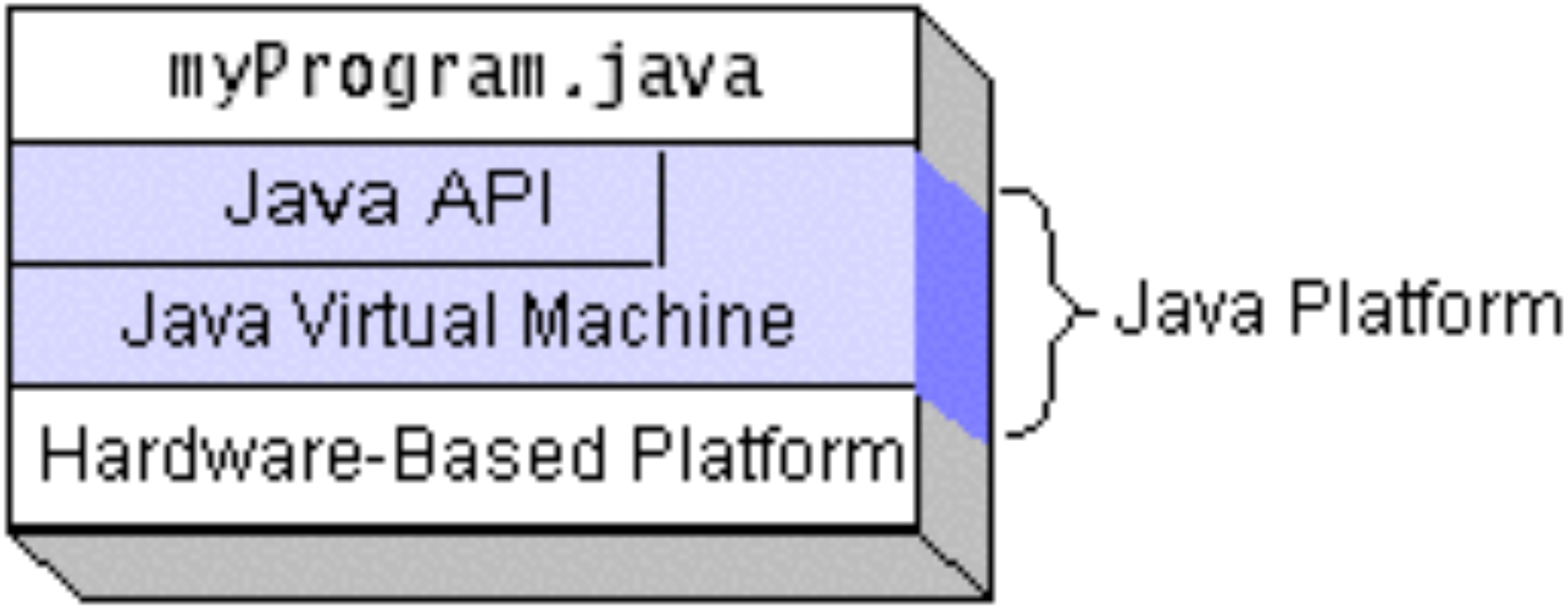
Scott McNealy
555-1212

How Java runs



Java platform

- The *Java Virtual Machine* (Java VM)
- The *Java Application Programming Interface* (Java API)



Cross platform/portable

- Portable: a dream from UNIX and C to Java
  - JVM
  - Unique data type
    - int: 32-bit
    - char: Unicode

Capacity: high or low?

- Low:
  - Interpreted
  - Code check
  - Stack machine
- High
  - JIT
  - Multi-threads
  - Byte code simple design

Stable and KISS

- No point
- Index check
- Auto memory management
- C++ like

Pure OOP

- Java = C++ --
  - Multi-inheritance
  - Virtual inheritance
  - Template
  - Operator overloading

Dynamic

- Java has no such idea of program
- Java believes all classes and objects in the world are in one space
- The compilation result of your program is one part of the space
- To "run a program" is to start a thread from one method of one class in the space

Storage of objects

- For C, efficiency is the most important, so programmers can make decision by their own.
- For Java, all objects should be constructed in runtime and be stored in ***heap***.

Create and destroy objects

- You must create your objects
- But can not destroy them by yourself
- There is a recycle mechanism to help clean up objects that will never be used again.

root

- Every class in Java is a descendant of one class: **Object**
- So all objects in Java is objects of the class Object.
- In this way, container is an object that can hold objects.

First Java Program

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

read in

```java
import java.util.Scanner;

public class A {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int i = in.nextInt();
        System.out.println("Hello World!");
        System.out.println(i);
        in.close();
    }
}
```

Read a String in

- in.next();读入一个单词，单词的标志是空格
  - 空格包括空格、tab和换行
- in.nextLine();读入一整行

handle

- You can not hold the object itself, instead, you can have a handle that *refer* to it.

```
String s;
```

- Here s is only a handle that refer to nothing.

new

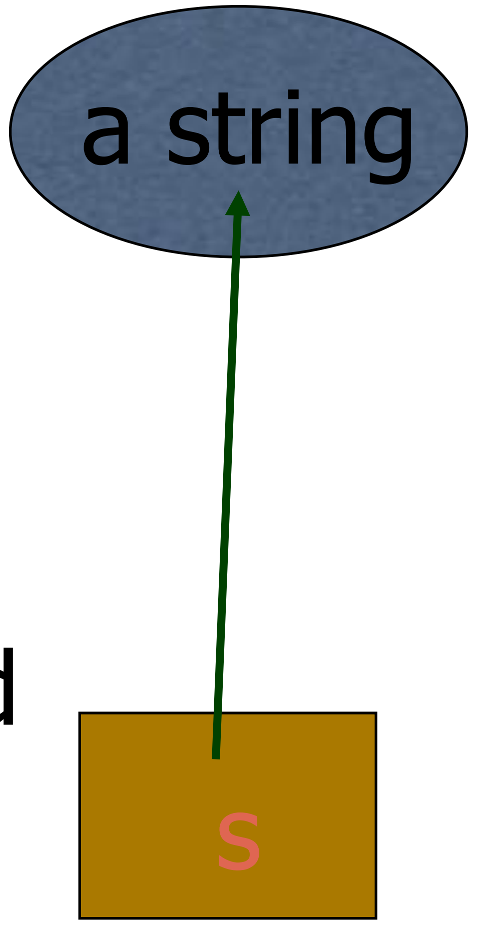
```
String s = new String("a string");
```

An object of String is created

The object is initialized w/ “a string”

A variable as handle to String is created

The value of the handle variable is assign to the object



String Handling

- String is sequence of characters
- In Java, String is an immutable object
- Immutable : Strings within objects of type **String** are unchangeable means that the contents of the **String** instance cannot be changed after it has been created.
- However, a variable declared as a **String** reference can be changed to point at some other **String** object at any time.

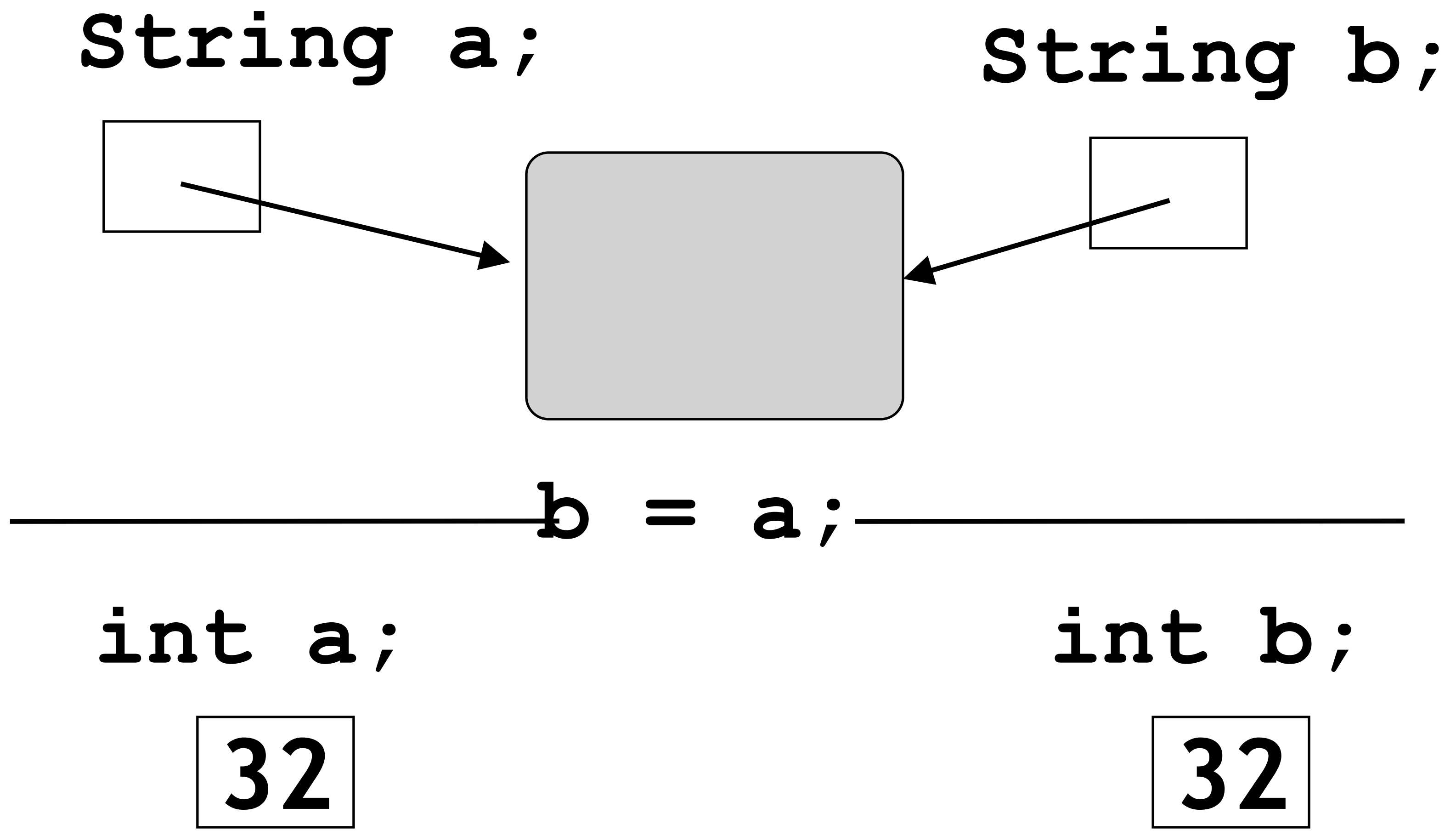
String APIs

```
String s1="Hello";
```

`s1.length();` gives the length of string

```
int age =25;

String s2="I am"+age+"years young”;


"I’m "+18—>"I’m 18"

1+2+"age"—>"3age"

"age"+1+2—>"age12"
```

对象变量的赋值



String equality

```
if(input == "bye") {
    ...
}
```

tests identity

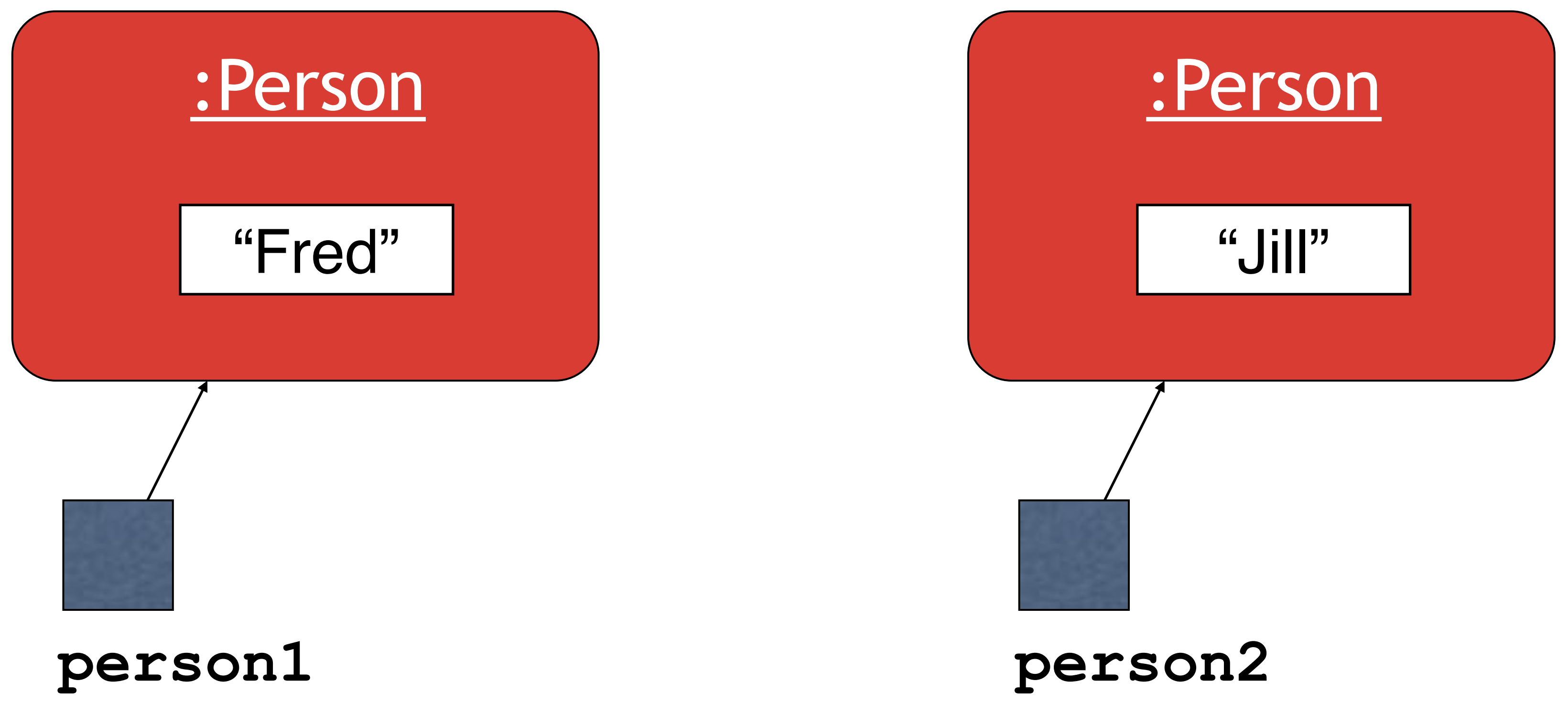
```
if(input.equals("bye")) {
    ...
}
```

tests equality

- Strings should always be compared with **`.equals`**

Identity vs equality 1

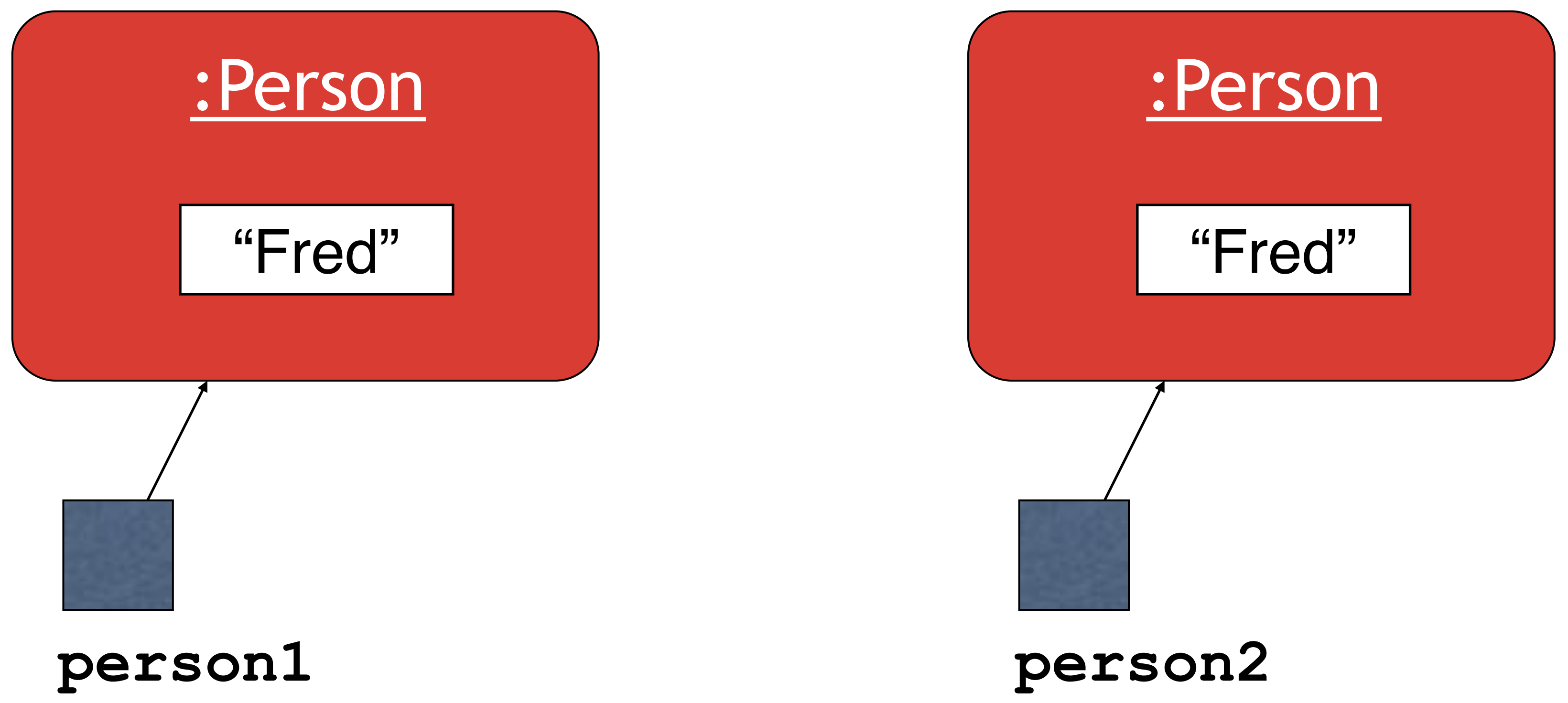
Other (non-String) objects:



```
person1 == person2 ?
```

Identity vs equality 2

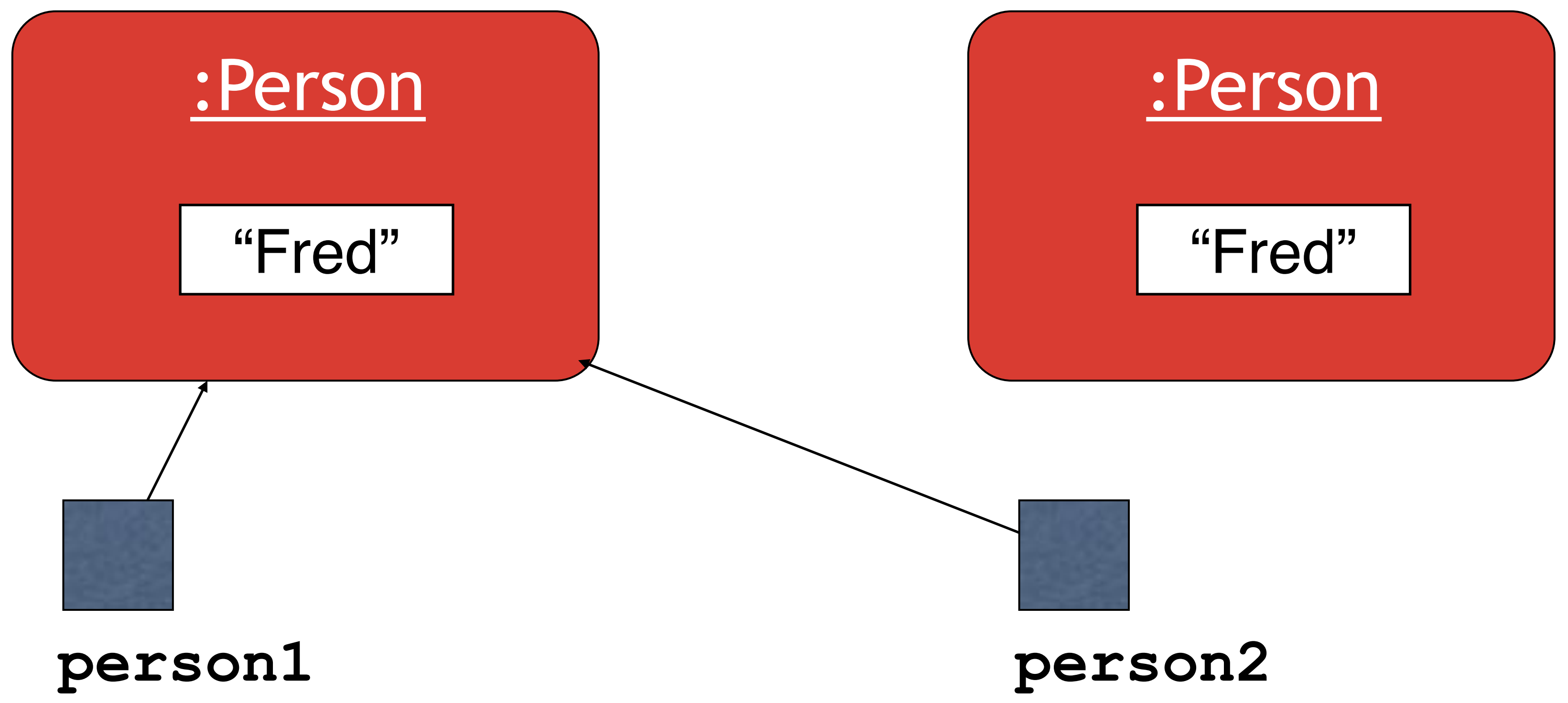
Other (non-String) objects:



`person1 == person2 ?`

Identity vs equality 3

Other (non-String) objects:

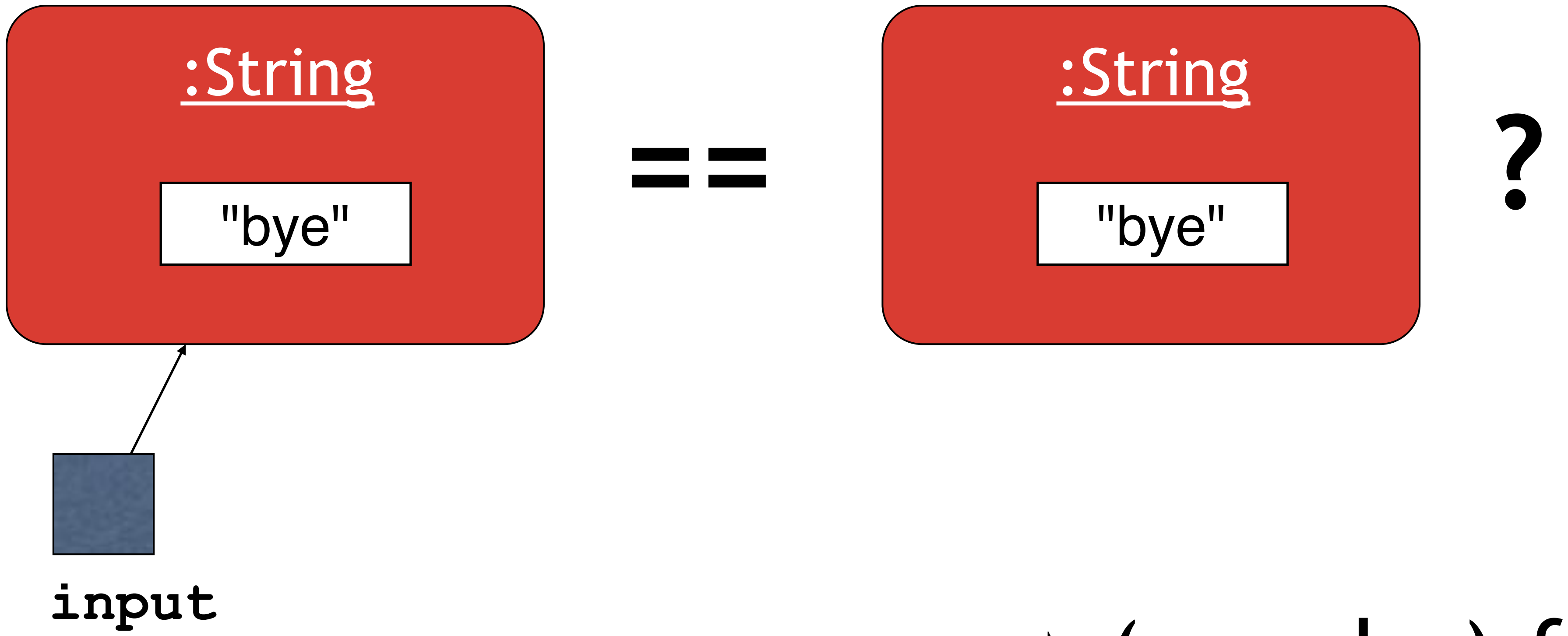


`person1 == person2 ?`

Identity vs equality (Strings)

```
String input = reader.getInput();
if(input == "bye") {
    ...
}
```

== tests identity

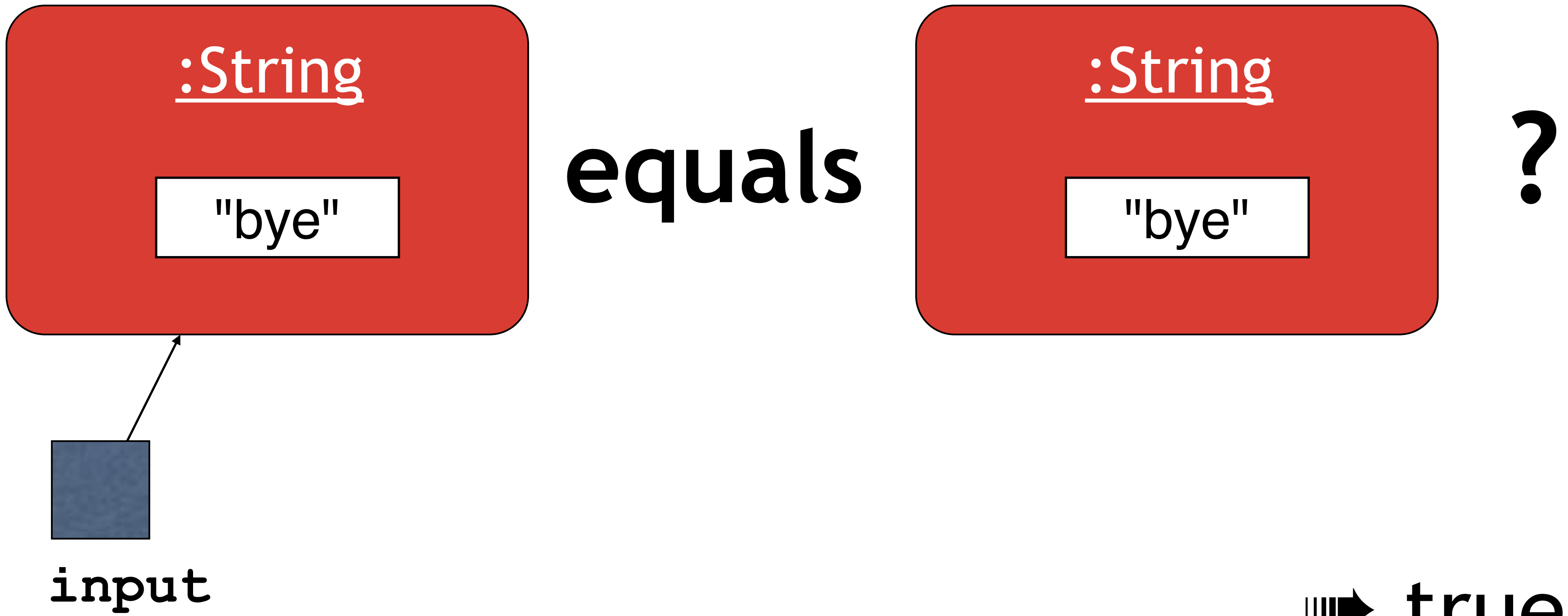


➠ (may be) false!

Identity vs equality (Strings)

```
String input = reader.getInput();
if(input.equals("bye")) {
    ...
}
```

equals tests equality



➠ true!

String Operations

- **equals() -** Compares the invoking string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as the invoking object.

  ```
  public boolean equals(Object anObject)
  ```

- **equalsIgnoreCase()-** Compares this String to another String, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length, and corresponding characters in the two strings are equal ignoring case.

  ```
  public boolean equalsIgnoreCase(String
  anotherString)
  ```

Comparing Strings

- The fact that characters are primitive types with a numeric internal form allows you to compare them using the relational operators. If **`c1`** and **`c2`** are characters, the expression

```
c1 < c2
```

  is **`true`** if the Unicode value of **`c1`** is less than that of **`c2`**.

- The **`String`** class allows you to compare two strings using the internal values of the characters, although you must use the **`compareTo`** method instead of the relational operators:

```
s1.compareTo(s2)
```

  This call returns an integer that is less than 0 if **`s1`** is less than **`s2`**, greater than 0 if **`s1`** is greater than **`s2`**, and 0 if the two strings are equal.

String Operations

- **compareTo()** - Compares two strings lexicographically.
  - The result is a negative integer if this String object lexicographically precedes the argument string.
  - The result is a positive integer if this String object lexicographically follows the argument string.
  - The result is zero if the strings are equal.
  - compareTo returns 0 exactly when the equals(Object) method would return true.

```
public int compareTo(String anotherString)

public int compareToIgnoreCase(String str)
```

Accessing char in a String

- **Individual characters in a String accessed with the charAt method.**

  - char charAt(int index)
    - *Returns the character at the specified index.*
    - *An index ranges from* 0 *to* length( ) - 1.
    - *The first character of the sequence is at index* 0, *the next at index* 1, *and so on, as for array indexing.*

substring in a String

- substring in a String accessed with the **substring** method.
    - String substring(int beginIndex)
        - *Returns a new string that is a substring of this string. The substring begins with the character at the specified* beginIndex *and extends to the end of this string.*
    - *Examples:* "unhappy".
        - substring(2) returns "happy"
        - substring(3) returns “appy"
- String substring(int beginIndex, int endIndex)
    - *Returns a new string that is a substring of this string. The substring begins at the specified* beginIndex *and extends to the character at index* endIndex - 1*. Thus the length of the substring is* endIndex-beginIndex*.*
- *Examples:* "hamburger".
    - substring(4, 8) returns "urge"
- "smiles".
    - substring(1, 5) returns "mile"

Determining the String Size

- We determine the number of characters in a String with the length method.

```
String name =
“Sumatra”,
       str1 = “one”,
       str2 = “”,
```

```
name.length( );
```
→ 7

```
str1.length( );
```
→ 3

```
str2.length( );
```
→ 0

```
str3.length( );
```
→ **Error!**

Error because no object is created for str3, so it is a null.

More String Methods

- *int* ***indexOf****(int* ***character****)*

- *int* ***lastIndexOf****(int* ***character****)*
  - Return the index of the first (last) occurrence of the specified character.

- *int* ***indexOf****(int* ***character****, int* ***from****)*

- *int* ***lastIndexOf****(int* ***character****, int* ***from****)*
  - Return the index of the first (last) occurrence of the specified character, searching forward (backward) from the specified index.

- void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
  - Copies characters from this string into the destination character array.

Other Methods in the `String` Class

`int lastIndexOf(char ch)` *or* **`lastIndexOf(String str)`**	Returns the index of the last match of the argument, or −1 if none exists.
`boolean equalsIgnoreCase(String str)`	Returns **`true`** if this string and **`str`** are the same, ignoring differences in case.
`boolean startsWith(String str)`	Returns **`true`** if this string starts with **`str`**.
`boolean endsWith(String str)`	Returns **`true`** if this string starts with **`str`**.
`String replace(char c1, char c2)`	Returns a copy of this string with all instances of **`c1`** replaced by **`c2`**.
`String trim()`	Returns a copy of this string with leading and trailing whitespace removed.
`String toLowerCase()`	Returns a copy of this string with all uppercase characters changed to lowercase.
`String toUpperCase()`	Returns a copy of this string with all lowercase characters changed to uppercase

Immutable String

```
String s = “abc”;
s = “def ”;
s = s.trim();
```

- When Java converts data into its string representation during concatenation, it does so by calling one of the overloaded versions of the string conversion method `valueOf()` defined by `String`.
- `valueOf()` is overloaded for all the simple types and for type Object.
- For the simple types, valueOf( ) returns a string that contains the human-readable equivalent of the value with which it is called.
- Every class implements `toString()` because it is defined by `Object`.
- For the classes we create we want to override `toString()` and provide our own string representations.
- The `toString()` method has this general form:

```
String toString()
```

- To implement `toString()`, simply return a `String` object that contains the human-readable string that appropriately describes an object of your class.

String.format()

- use %x like in printf()

Converting Strings to Numbers

- The "type wrapper" classes (`Integer`, `Long`, `Float`, and `Double`) provide a class method named `valueOf` that converts a `String` to an object of that type.

```
String piStr = "3.14159";
Float pi = Float.valueOf(piStr);

float pi2 =
Float.parseFLOAT(piStr);
```

Object