Objectives

- Basics of Java Syntax
- Java fundamentals
  - Print statements
  - Data types
  - Assignments
  - Arithmetic operators

Review

- Version Control
  - What are the benefits of version control?
  - What are some of the common Git commands and what do they do?
  - How did the git lab go? Make sense? Have questions?
- Java
  - What are the benefits of Java?
  - How do you compile and run Java programs?

Review: Common Git Commands

Command	What it does
clone	Clones a repository – sets up your repository so that you can coordinate
add <*file*>	Adds the *file* to the staging area
commit	Commits all the staged files (locally)
push	Push all your changes to the remote → You need your code to be pushed so that I can see it.
branch	List all local branches
branch <*name*>	Creates a new branch named *name*
checkout <*name*>	Switches to the branch named *name*

THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

https://xkcd.com/1597/

3

Typical Git Workflow

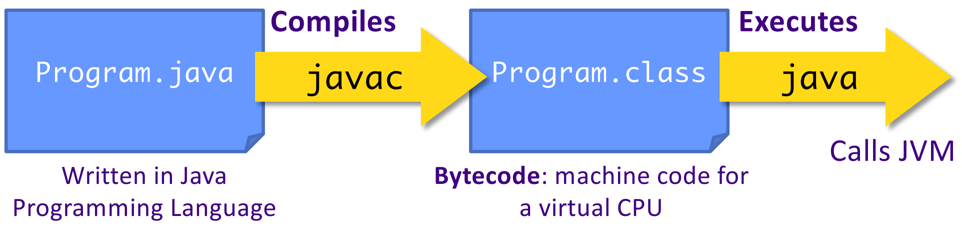
1. Clone repository
2. Branch from `main` to a work-in-progress branch
   - Work on feature/next step/…
3. When complete, merge branch back into `main`
   - Optionally, push `main`
4. Switch back to and continue in work-in-progress branch (either same branch or new one)
5. Repeat

4

Review: Benefits of Java

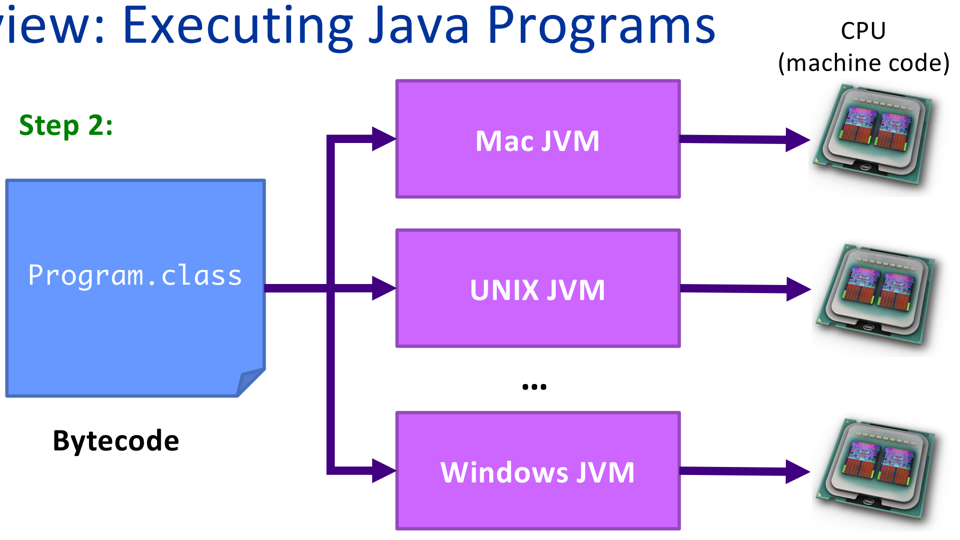
- Rapid development of programs
  - ➢ Large library of classes, including GUIs, Enterprise-level applications, Web applications
- Portability
  - ➢ Run program on multiple platforms without recompiling
- Compiled
  - ➢ Find some errors before execution!
    - Statically typed
  - ➢ Can give performance boost through optimizations

Review: Compiling, Executing Java Programs


Program.java
Compiles
javac
Program.class
Executes
java
Calls JVM
Written in Java Programming Language
Bytecode: machine code for a virtual CPU

```
javac Program.java
java Program
```

Review: Executing Java Programs



- Same **bytecode** is executed on each platform
- Don't need to provide the source code

Sept 19, 2022 Sprenkle - CSCI209 7

7

Reminder: Reloading Assignments

- Reload assignment pages whenever you return to them
  - Get most recent updates
  - I may have addressed issues that students alerted me to

Sept 19, 2022 Sprenkle - CSCI209 8

8

LET’S PROGRAM!

9

Example Java Program: Hello.java

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

What are your observations about this program?
What can you figure out?

10

Example Java Program

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

- Everything in Java is inside a `class`
  - Java is *entirely* object-oriented*
  - This class is named **`Hello`**

11

Java: Files and Class Definitions

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

***Defines* the class `Hello`**

`Hello.java`

- Name of the file **must** match the name of the class
  - E.g., `Hello.java`

- In general, each Java program file contains **one** class definition

12

Java: Blocks of Code

Blocks of code marked with { }

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

Defines the class Hello

Hello.java

Java: Access Modifiers

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

Access Modifier:
controls if other classes can use code in this class

Java: Method Definitions

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

- This class contains one *method* definition: **`main`**

The **`main`** Method

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

- Similar to `main` in Python
  - But *must be* associated with a *class*
- Must take one parameter: an *array* of Strings
  - For command-line arguments
- Must be `public static`
- Must be `void`: data type of what method returns (nothing)
- `main` is *automatically* called when program is executed

Example Java Program

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

- Method contains one line of code
  - What do you think it does?

Java: Print Statements

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

- Calls the **`println`** method on the **`System.out`** object
- **`println`** takes one parameter, a `String`
- Displays string on terminal, terminates the line with new line (\n) character

Java: Comments

```
/**
 * Our first Java class: displays Hello!
 * @author Sara Sprenkle
 */
public class Hello {
   public static void main(String[] args) {
      //print a message
      System.out.println("Hello!");
   }
}
```

- Comments: `/*  */` or `//`
  - `/**  */` are special JavaDoc comments

Java Code Style

```
/**
 * Displays "Hello!"
 * @author Sara Sprenkle
 */
```

- **Comments** describing class
  - Sprenkle CSCI209 requirements:
    - **Must** include high-level description of program
    - **Must** include your name as author
- Proper **indentation**
  - Similar to Python
  - Everything within pairs of {} is indented the same
  - Not required by compiler but for readability

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

A Note About Examples' Comments

- The example code that I provide is often "over" commented
- I'm providing information for you that isn't needed in your submissions
  - However, if it's helpful for you, you can keep "over"commenting

Demo: Compiling and Running Programs

- Compiler errors:
  - Errors in the program's syntax
- Logic errors
  - Errors in your logic/coding
  - Found at runtime
  - After fixing program, need to go back and recompile

Unix Output Redirection: >

- We can redirect output to a file
  - For example

```
ls *.java > java_files.out
```

  - Above command saves the output from the `ls` command into the file named `java_files.out`
- This is how you will save output from your Java programs initially
  - For example

```
java Intro > out
```

Please follow instructions on names in assignments

Compare Python and Java

```
# a Python program
def main():
      print("Hello")

main()
```

```
/**
 * Our first Java class
 * @author Sara Sprenkle
 */
public class Hello {
   public static void main(String[] args) {
      //print a message
      System.out.println("Hello");
   }
}
```

Java vs. Python, so far…

- ***Semantics*** the same, ***syntax*** different
  - Blocks of code
  - End statements
- Access modifiers
- Data type declarations
- Class-based programs
- Compiled

We'll see more differences as we go…

Translate to Python Program?

```
/**
 * Our first Java class
 * @author Sara Sprenkle
 */
public class Hello {
   public static void main(String[] args) {
      //print a message
      System.out.println("Hello");
   }
}
```

Translation to Python Program

```
print("Hello")
```

Literal translation:

```
class Hello:
   """Our first Python class"""

   @staticmethod
   def main(self):
       print("Hello")
```

Aside: JavaScript vs Java

- JavaScript is **not** Java
  - JavaScript is a *scripting* language, primarily embedded in HTML, executed by Web browsers*

```
<script type="text/javascript">
function myFunction() {
    return ("Hello, have a nice day!")
}
</script>
</head>
<body>
<script type="text/javascript">
    document.write(myFunction())
</script>
```

JAVA FUNDAMENTALS

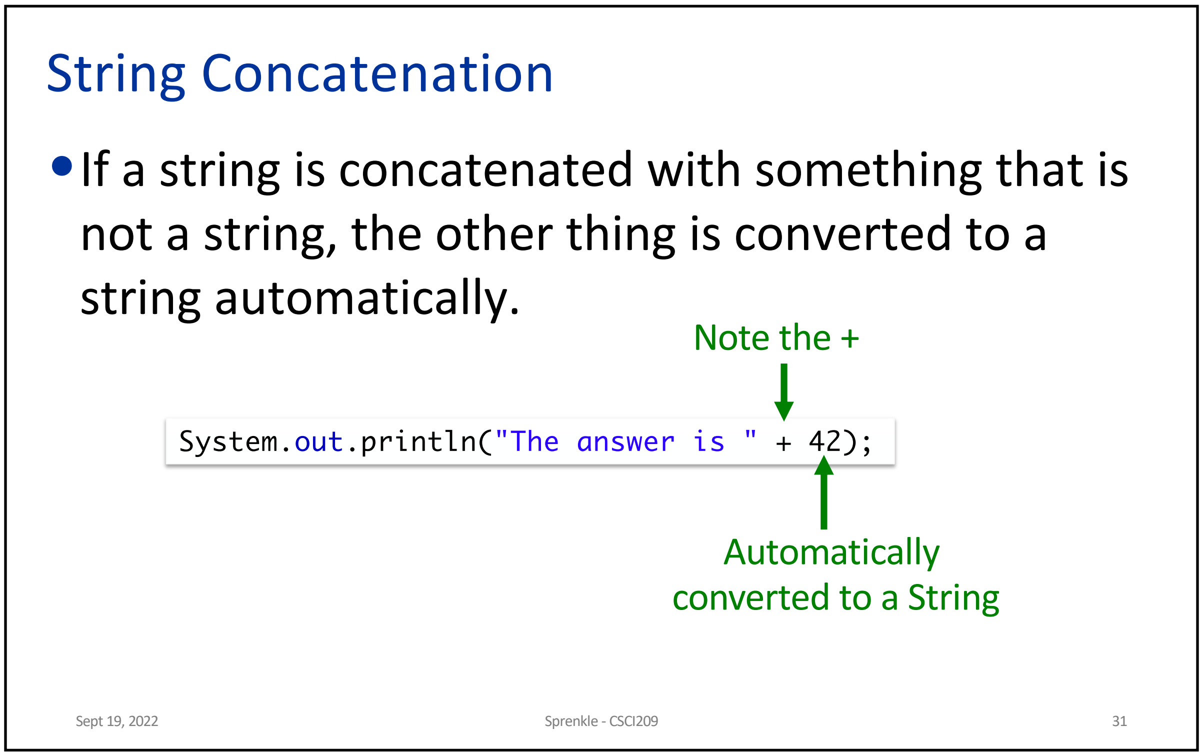
Print Statement

- Syntax:

```
System.out.println(<String>);
System.out.print(<String>);
```

No newline at end

- Closer to how you use Python's `file.write()` method
  - Need to combine parameter into one `String` using `+`'s
    - Python's `print` used *commas*
  - More on `String` operations later



String Concatenation

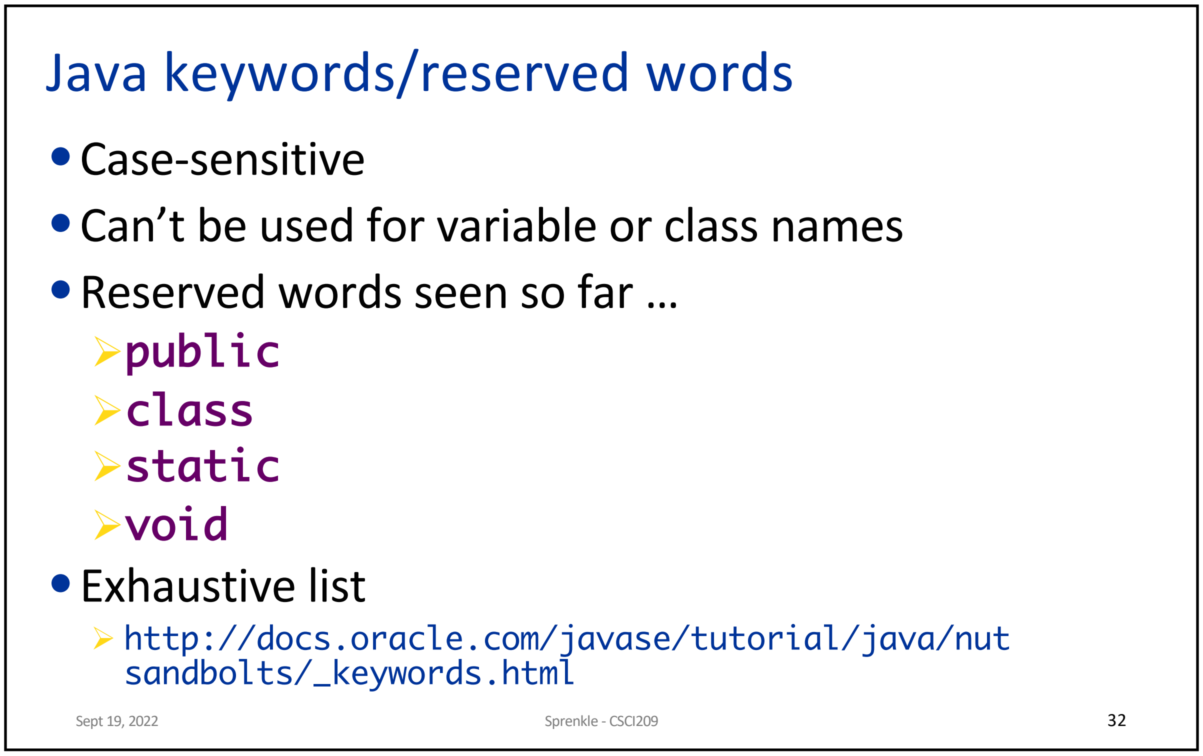
- If a string is concatenated with something that is not a string, the other thing is converted to a string automatically.

Note the +

```
System.out.println("The answer is " + 42);
```

Automatically converted to a String

31



Java keywords/reserved words

- Case-sensitive
- Can't be used for variable or class names
- Reserved words seen so far …
  - `public`
  - `class`
  - `static`
  - `void`
- Exhaustive list
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

32

Data Types

- Java is ***strongly-typed***
  - Every variable must have a ***declared* type**
- All data in Java is an ***object*** – except for the ***primitive data types***:

`int`	4 bytes (-2,147,483,648 -> 2,147,483,647)
`short`	2 bytes (-32,768 -> 32,767)
`long`	8 bytes (really big integers)
`byte`	1 byte (-128 -> 127)
`float`	4 bytes (floating point)
`double`	8 bytes (floating point)
`char`	2 bytes (Unicode representation), **single** quotes
`boolean`	`true` or `false`

Variables

- Must be ***declared*** before used
  - **Syntax**: `<datatype> <name> [= value];`

Optional assignment

- Naming conventions:
  - Variable names (identifiers) typically start with *lowercase* letter
    - _ (underscore) also a valid first character
  - Subsequent words are capitalized
    - Examples: myFile, firstCousinOnceRemoved
    - Called "Camel Casing"

Variable Examples

- Must be ***declared*** before used
  - **Syntax**: `<datatype> <name> [= value];`
- Examples:
  - `int x;`
  - `double pi = 3.14;`
  - `char exit = 'q';`
  - `boolean isValid = false;`

Note ***must*** use *single* quotes for `chars`

Camel Casing

35

Python Transition **Warning**

You can**not** redeclare a variable name in the same scope

- OK:

```
int x = 3;      ← Declaration
x = -3;         ← Definition
… // more code
x = 7;
```

- Not OK:

```
int x = 3;
int x = -3;        ← Compiler errors

boolean x = true;
```

36

More Data Type-Related Information

- Result of integer division is an `int`
  - Same as Python **2**, **not** Python 3
  - Example: `4/3 = ??`
- Casting
  - Similar to Python for primitive types
  - Example: `4/(double) 3`

`TestScore.java`

37

Floats in Java

- Decimal literals are considered *doubles*
- This code won't compile:

```
float f = 3.14;
```

Compiler reads `3.14` as a *double*

- Compiler error message:

```
Float.java:15: error: incompatible types: possible lossy
conversion from double to float
       float f = 3.14;
                 ^
1 error
```

- To fix code, add an `f` to specification of number or declare as `double`

`Float.java`

38

Policy: Using the Web and Others

- I provide a lot of online resources
- Most of what I ask you to do is similar to my slides or examples
  - Exception: machine/software configuration
- Use my resources first
  - Example programs are on the course web site
- Search online/ask someone else as a last resort
  - Need more experience to sort through the results you get in search engine
    - How do you get experience? More practice in CSCI209!

If it's taking more than ~3 minutes to get an answer, check in with me

Looking Ahead

- Start reading Chapter 1 through 1.4: Lets look at a Java Program
- Complete Assignment 0 before next class