A large, spreading tree with many branches and a thick trunk, set against a bright sky.

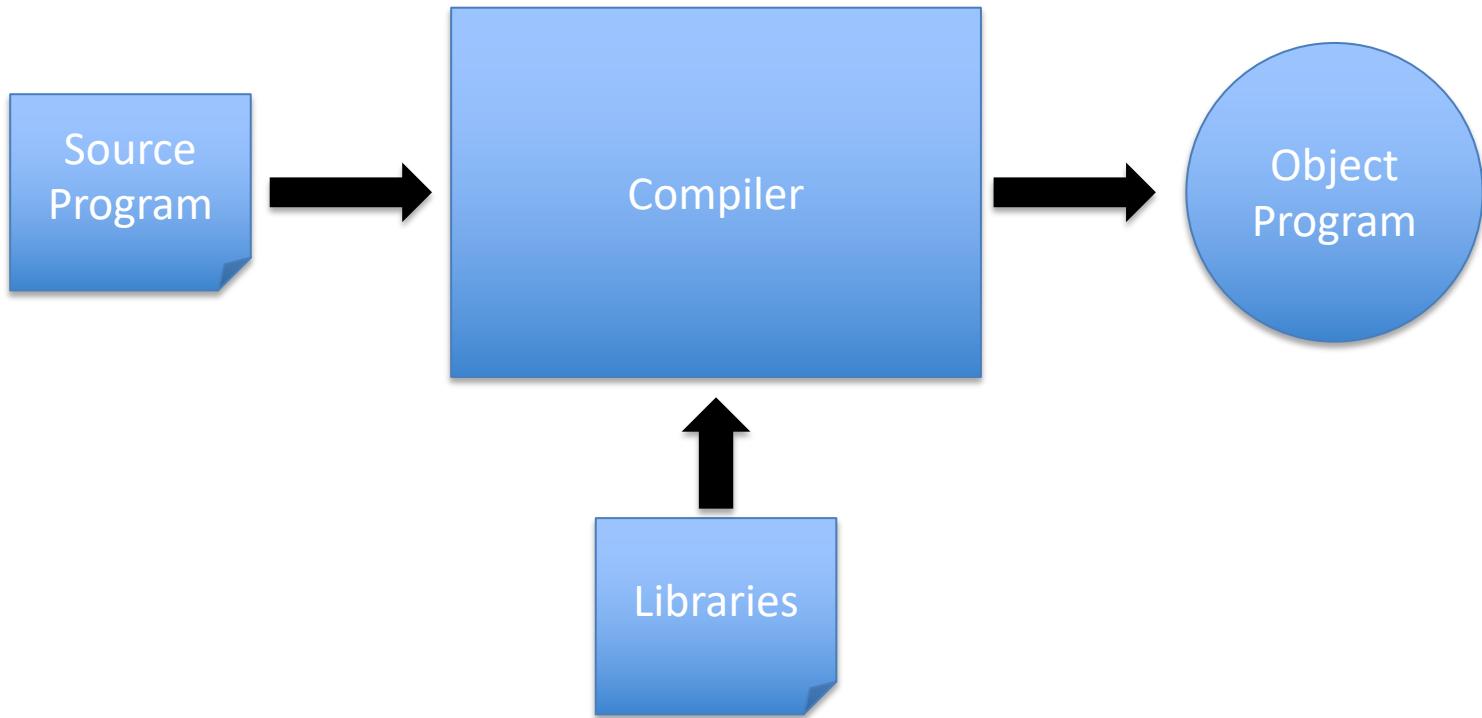
CSCI 1102 Computer Science 2

Meeting 2: Tuesday 2/2/2021
Getting Started with Java

Today

- Compilation, Compile-time & run-time
- The basic structure of a Java program
- Static & Dynamic functions
- Memory & Storage
- A Simple Example

Compilation



Object Program

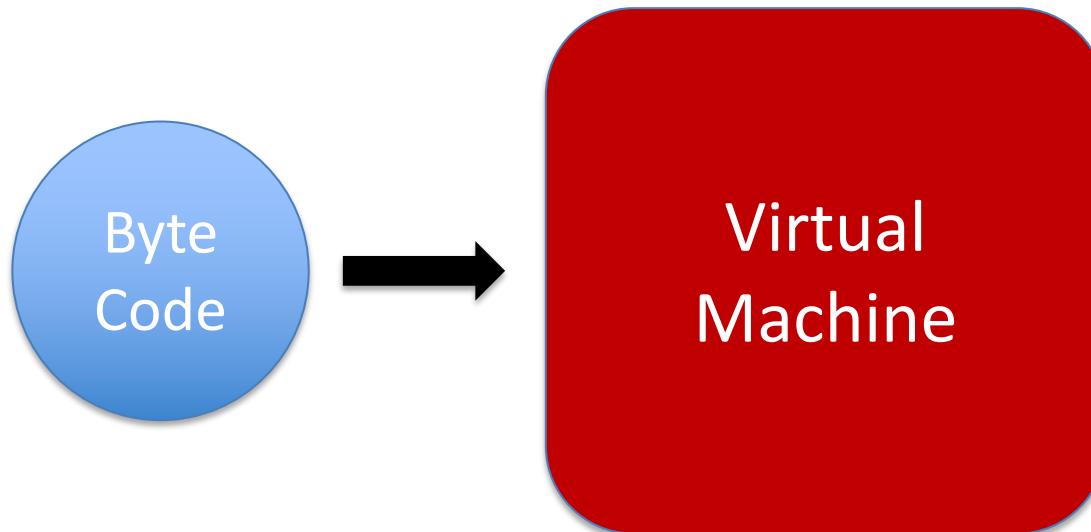
1. **Native code** – a sequence of simple instructions in the language of a particular physical machine (processor)
2. **Byte code** – a sequence of simple (byte-sized) instructions in the language of a **virtual computer** -- a software program that can be implemented on any physical processor.

Object Program

1. Native code – fast but not portable;
2. Byte code – portable but slow.

Virtual Machine

Typically uses a **stack** as its primary tool to interpret/execute the byte-code program



Dis-compiling byte-codes: Python

```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

```
|>>> dis.dis(fact)
2           0 LOAD_FAST               0  (n)
            3 LOAD_CONST               1  (0)
            6 COMPARE_OP              2  (==)
            9 POP_JUMP_IF_FALSE       16

3           12 LOAD_CONST              2  (1)
            15 RETURN_VALUE

5           >>  16 LOAD_FAST               0  (n)
            19 LOAD_GLOBAL              0  (fact)
            22 LOAD_FAST               0  (n)
            25 LOAD_CONST              2  (1)
            28 BINARY_SUBTRACT
            29 CALL_FUNCTION           1
            32 BINARY_MULTIPLY
            33 RETURN_VALUE
            34 LOAD_CONST              0  (None)
            37 RETURN_VALUE
```

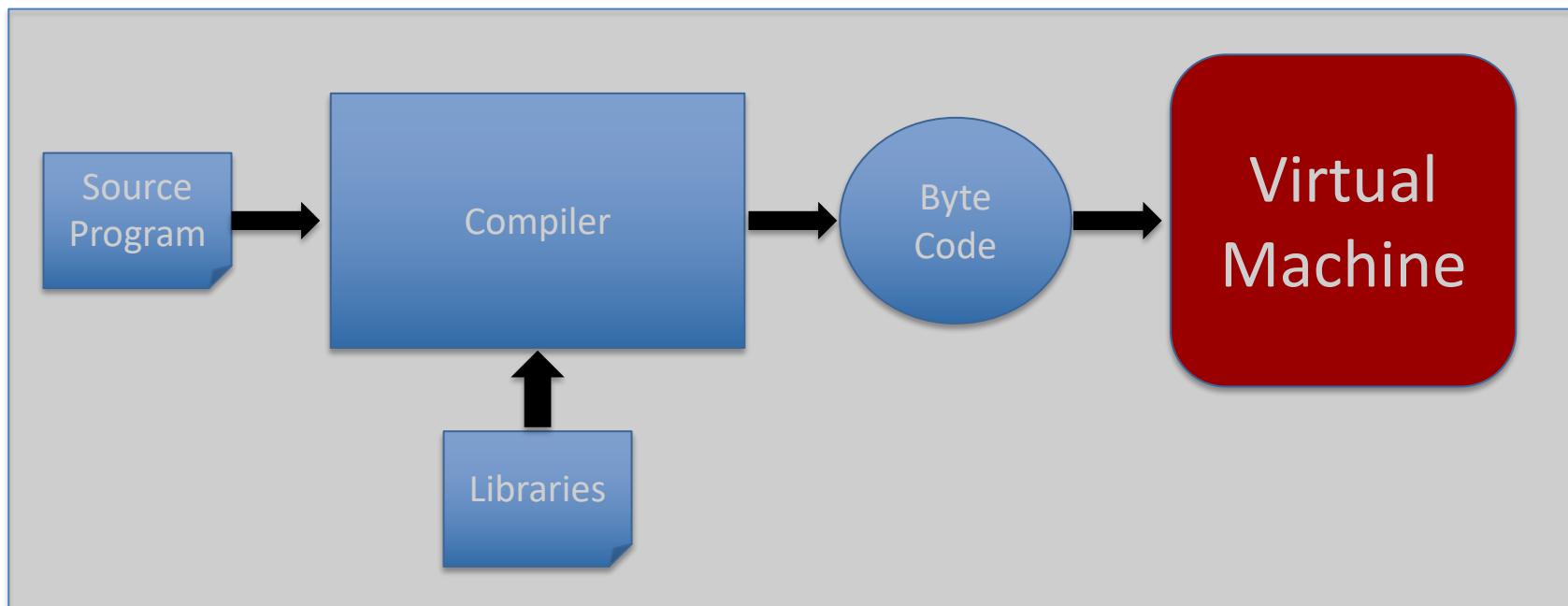
Dis-compiling byte-codes: Java

```
public class Fact {  
  
    static int fact(int n) {  
        if (n == 0)  
            return 1;  
        else  
            return n * fact(n - 1);  
    }  
}
```

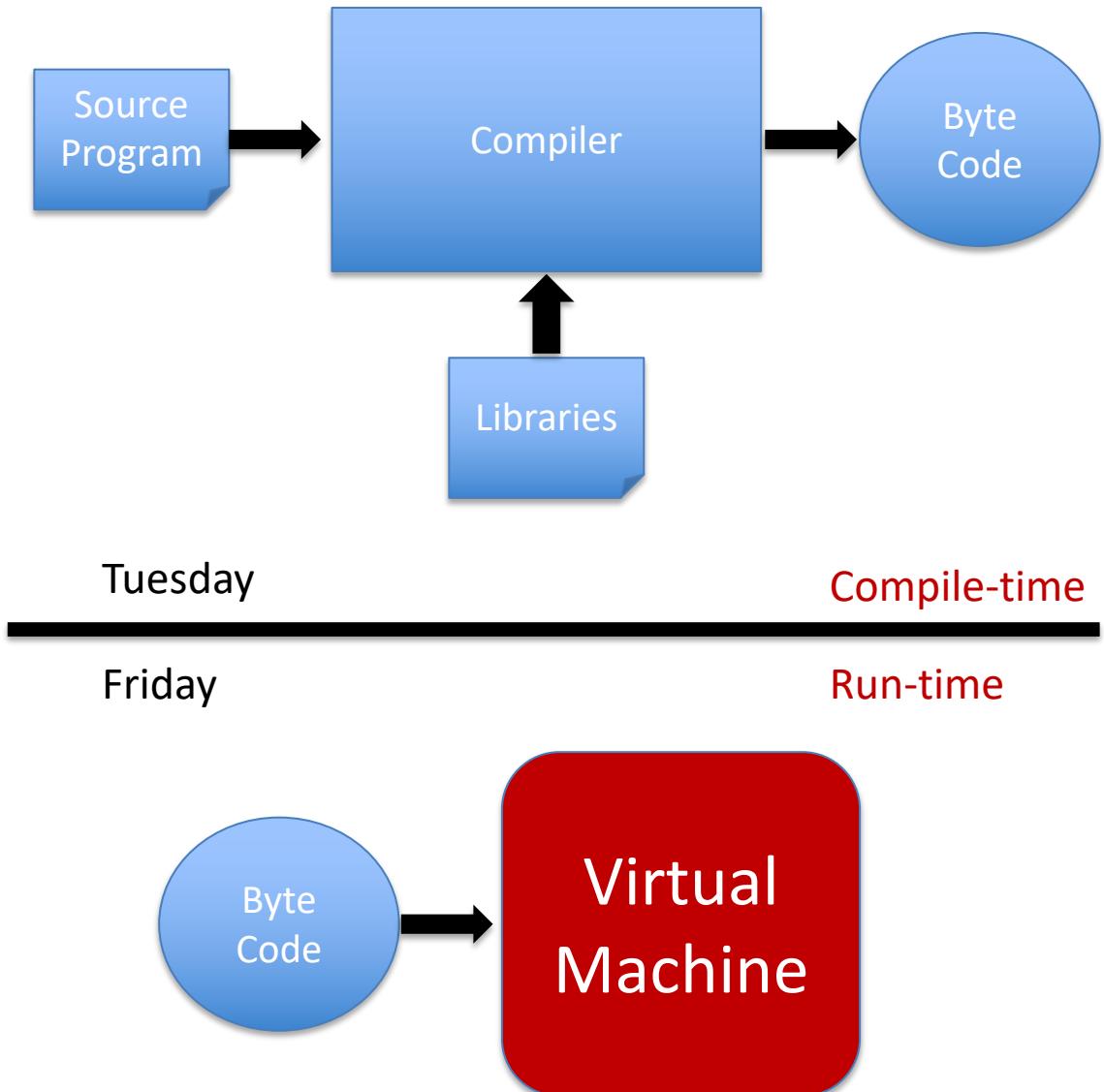
```
> javac Fact.java  
> javap -c Fact  
Compiled from "Fact.java"  
public class Fact {  
    public Fact();  
    Code:  
        0: aload_0  
        1: invokespecial #1  
        4: return  
  
    static int fact(int);  
    Code:  
        0: iload_0  
        1: ifne           6  
        4: iconst_1  
        5: ireturn  
        6: iload_0  
        7: iload_0  
        8: iconst_1  
        9: isub  
       10: invokestatic #2  
       13: imul  
       14: ireturn  
    }  
}
```

Compiler + Virtual Machine

A compiler and VM are sometimes combined into one program, often called an **interpreter** or **REPL**. Python uses this model.



- In some language implementations, the compiler & VM are separate.
- The byte-code can be run on the VM any time after compilation.



Advantages of the Phase Separation

- The compiler can find and report many (but not all) errors before the program is running;
- The compiler can sort out the type information at compile time so this work isn't required at run-time, this leads to more performant code.
- NB: compile-time information about a program is often said to be *static*

Python is dynamically-typed – the types of operands must be determined as the program is running

No type information

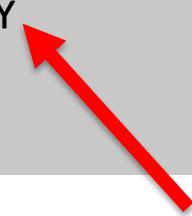
```
def fact(n):
    if n == 0:
        return 1
    else:
        return n * fact(n - 1)
```

What kind of multiply?

```
>>> dis.dis(fact)
 2      0  LOAD_FAST              0  (n)
           3  LOAD_CONST             1  (0)
           6  COMPARE_OP             2  (==)
           9  POP_JUMP_IF_FALSE       16

 3      12 LOAD_CONST            2  (1)
           15 RETURN_VALUE

 5      >> 16 LOAD_FAST              0  (n)
           19 LOAD_GLOBAL            0  (fact)
           22 LOAD_FAST              0  (n)
           25 LOAD_CONST             2  (1)
           28 BINARY_SUBTRACT
           29 CALL_FUNCTION          1
           32 BINARY_MULTIPLY
           33 RETURN_VALUE
           34 LOAD_CONST             0  (None)
           37 RETURN_VALUE
```



```
>>> dis.dis(fact)
 2      0 LOAD_FAST              0 (n)
 3      3 LOAD_CONST             1 (0)
 6      6 COMPARE_OP             2 (==)
 9      9 POP_JUMP_IF_FALSE       16

 3     12 LOAD_CONST             2 (1)
 15    15 RETURN_VALUE

 5     >> 16 LOAD_FAST              0 (n)
 19    19 LOAD_GLOBAL             0 (fact)
 22    22 LOAD_FAST              0 (n)
 25    25 LOAD_CONST             2 (1)
 28    28 BINARY_SUBTRACT
 29    29 CALL_FUNCTION           1
 32    32 BINARY_MULTIPLY
 33    33 RETURN_VALUE
 34    34 LOAD_CONST             0 (None)
 37    37 RETURN_VALUE
```



In order for `BINARY_MULTIPLY` to choose between integer multiply and floating point multiply, the operands (i.e., numbers) must be accompanied by type information **at run-time**. This is costly.

Java and Ocaml are **statically typed** – type information is known at compile time.

```
public class Fact {  
    static int fact(int n) {  
        if (n == 0)  
            return 1;  
        else  
            return n * fact(n - 1);  
    }  
}
```

Inferred to be an integer
multiply on Tuesday.

```
> javac Fact.java  
> javap -c Fact  
Compiled from "Fact.java"  
public class Fact {  
    public Fact();  
    Code:  
        0: aload_0  
        1: invokespecial #1  
        4: return  
  
    static int fact(int);  
    Code:  
        0: iload_0  
        1: ifne           6  
        4: iconst_1  
        5: ireturn  
        6: iload_0  
        7: iload_0  
        8: iconst_1  
        9: isub  
       10: invokestatic #2  
       13: imul          ←  
       14: ireturn
```

No need for run-time
type information

```
static int fib(int n) {  
    if (n < 2)  
        return 1;  
    else  
        return fib(n - 1) + fib(n - 2);  
}
```

```
def fib(n):  
    if (n < 2):  
        return 1  
    else:  
        return fib(n - 1) + fib(n - 2)
```

Two wildly inefficient Fibonacci functions.

Run-times on VMs for fib(38)

```
> javac Fib.java  
> time java Fib 38  
fib(38) = 63245986
```

```
real      0m0.327s ←  
user      0m0.322s  
sys       0m0.035s  
> time python fib.py 38  
63245986
```

```
real      0m15.247s ←  
user      0m14.767s  
sys       0m0.125s  
>
```

The Python time includes time for translation.

| | REPL | Byte-Code Compiler | Native-Code Compiler |
|--------|------|--------------------|----------------------|
| Python | YES | | NO |
| Java | NO | YES | NO |
| OCaml | YES | YES | YES |

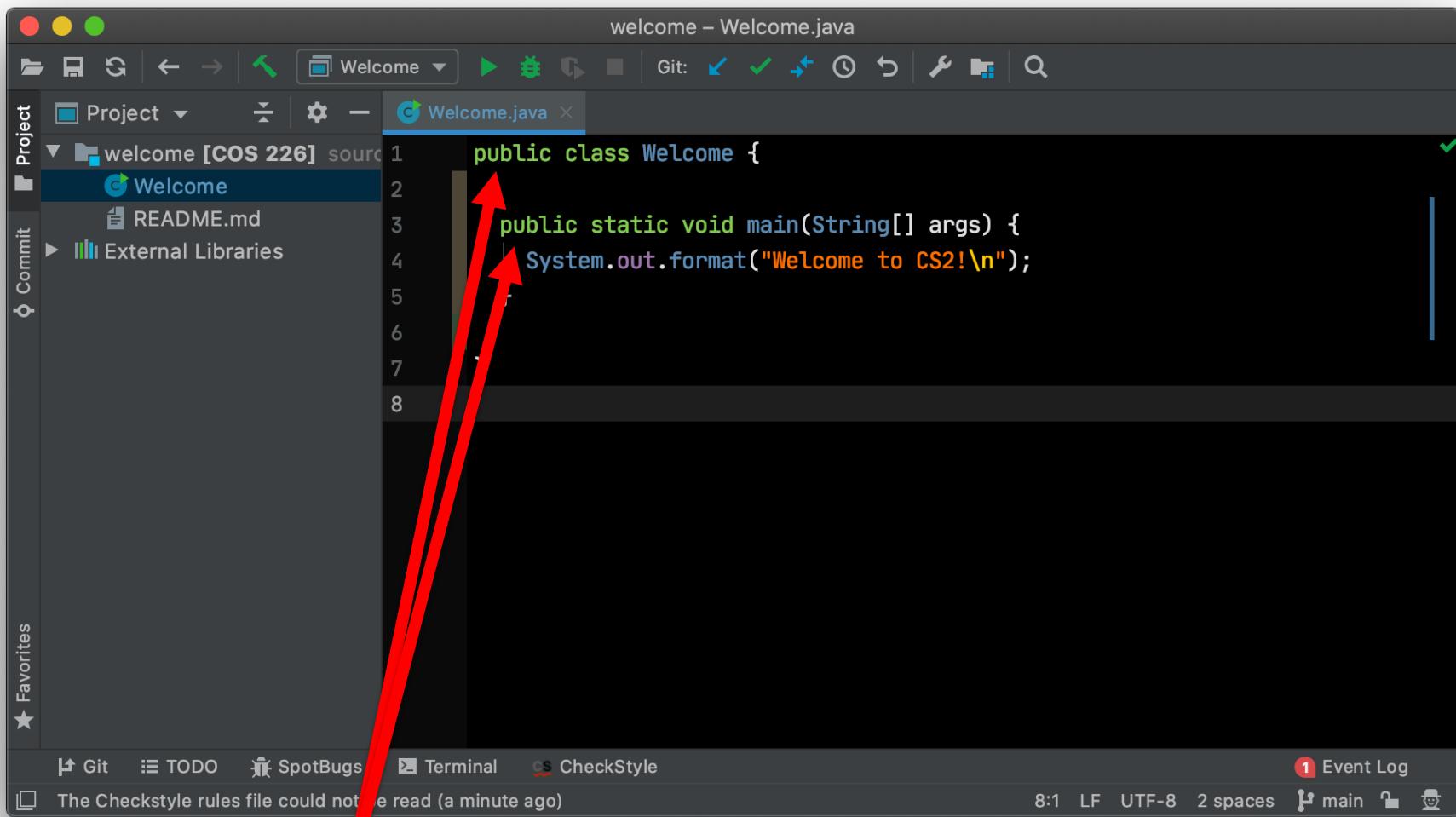
2. The basic structure of a Java program

- What Java programs look like.
- How they work.

A screenshot of a Java development environment, likely IntelliJ IDEA, showing a project named "welcome". The "Welcome.java" file is open in the editor, displaying the following code:

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.format("Welcome to CS2!\n");  
    }  
}
```

The code prints "Welcome to CS2!" to the console. The IDE interface includes a Project sidebar on the left with "welcome [COS 226] source", "Welcome", and "README.md". The bottom status bar shows "Event Log" with one entry, "The Checkstyle rules file could not be read (a minute ago)".



```
welcome – Welcome.java
Project Welcome.java
welcome [COS 226] sources
  Welcome
  README.md
External Libraries

public class Welcome {
    public static void main(String[] args) {
        System.out.format("Welcome to CS2!\n");
    }
}

Favorites

Git TODO SpotBugs Terminal CheckStyle
The Checkstyle rules file could not be read (a minute ago)
Event Log
8:1 LF UTF-8 2 spaces main
```

The keyword symbol **public** is a *visibility* attribute. The class **Welcome** and function **main** are visible anywhere. Another common visibility attribute for a declaration is **private** – the name is visible only within the class.

welcome – Welcome.java

Project

Welcome [COS 226] source

Welcome

README.md

External Libraries

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.format("Welcome to CS2!\n");  
    }  
}
```

Favorites

Git TODO SpotBugs Terminal CheckStyle

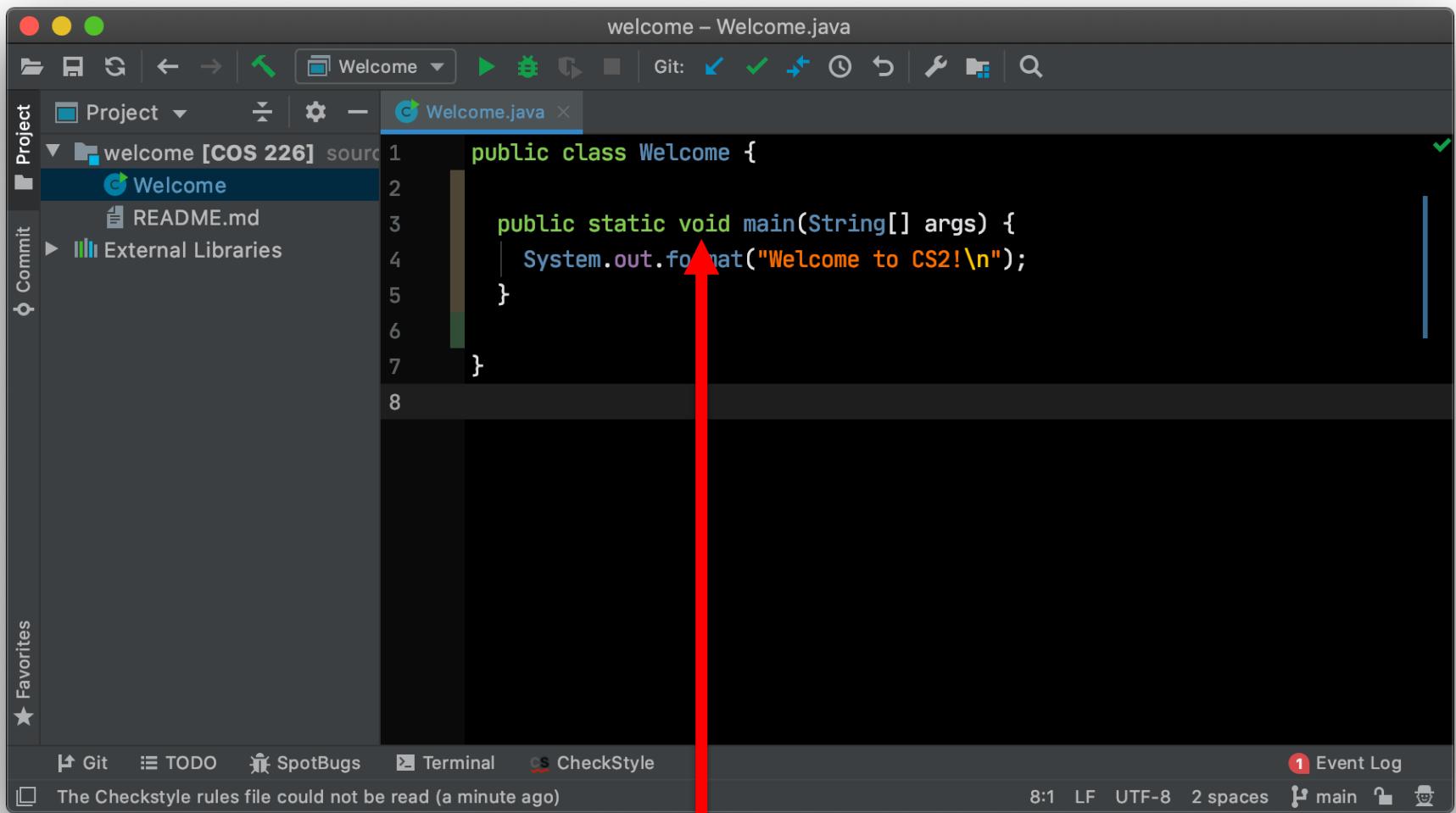
The Checkstyle rules file could not be read (a minute ago)

Event Log

Class names begin with a capital letter. A file can have at most one public class. The name must match the file name.

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.format("Welcome to CS2!\n");  
    }  
}
```

If you compile a Java source file containing a public class and want to execute the code in it, it must have a **public** and **static** function called **main**.



welcome – Welcome.java

Project Welcome.java

commit

Favorites

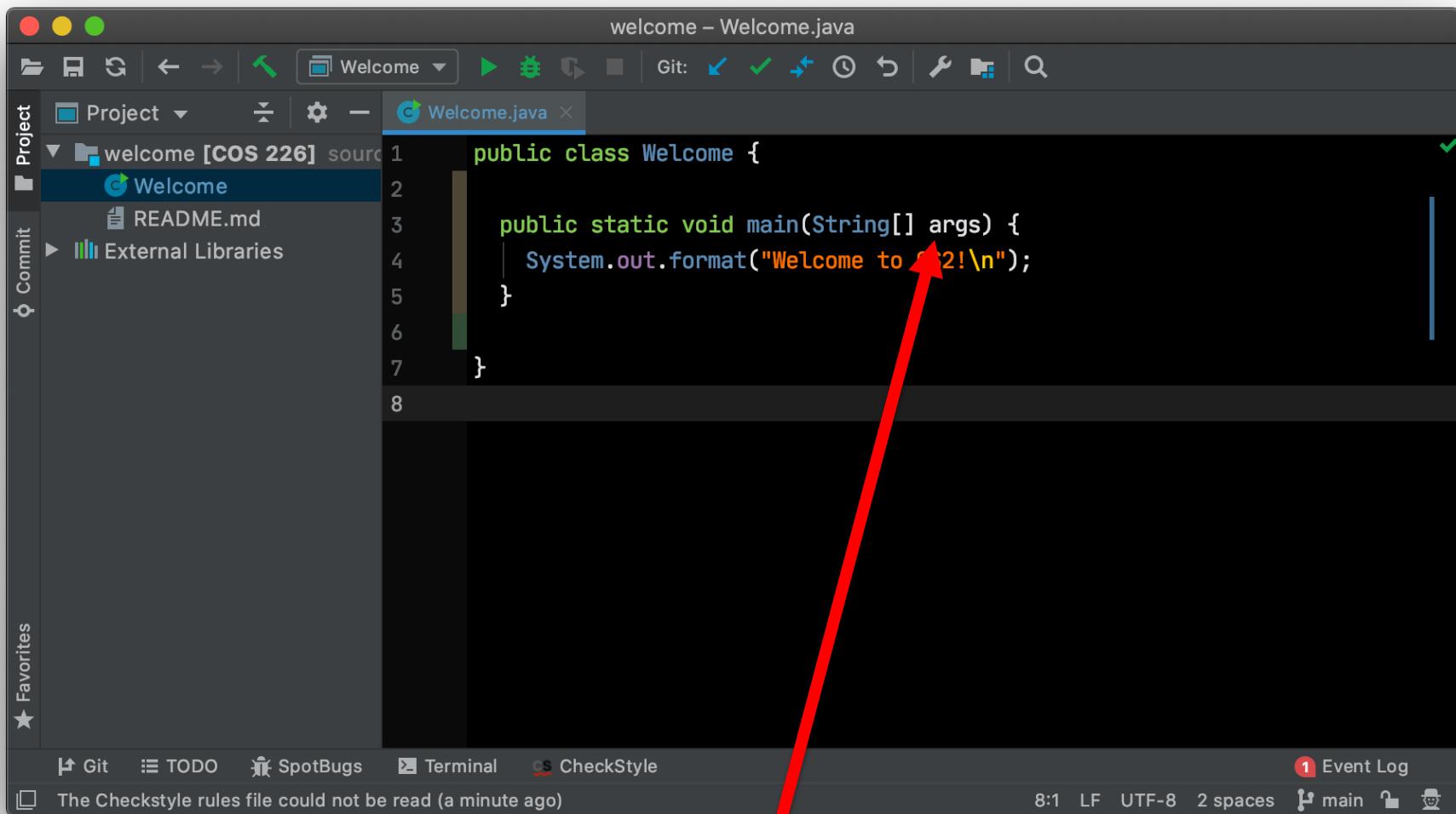
```
1 public class Welcome {  
2     public static void main(String[] args) {  
3         System.out.format("Welcome to CS2!\n");  
4     }  
5 }  
6  
7 }  
8
```

Git TODO SpotBugs Terminal CheckStyle

The Checkstyle rules file could not be read (a minute ago)

Event Log

The symbol **void** is a *type*, in this position it is the *return type* of function **main**. Function **main** doesn't return a value.



```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.format("Welcome to CSC210!\n");  
    }  
}
```

The symbol **args** is a *formal parameter* of the function **main**.
The programmer declares that it will hold a value of type **array of Strings**. These are the *command line arguments*.

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.format("Welcome to CS2!\n");  
    }  
}
```

Git TODO SpotBugs Terminal CheckStyle

The Checkstyle rules file could not be read (a minute ago)

Event Log

The body of the function, enclosed in braces {}, simply calls the **format** function contained in the **System.out** class.

Example: Who is taking CS2?

| S | School | Major | Grad Year | | | |
|-------------|----------------------|---------------------|-----------|------|------|------|
| omas@bc.edu | Morrissey College, A | computer science b. | 2023 | MCAS | CSBA | 2024 |
| i@bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | PHIL | 2022 |
| ne@bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBA | 2024 |
| c.edu | Morrissey College, A | computer science b. | 2022 | MCAS | CSBA | 2022 |
| @bc.edu | Morrissey College, A | philosophy | 2022 | MCAS | CSBS | 2024 |
| @bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBA | 2023 |
| y@bc.edu | Morrissey College, A | computer science b. | 2023 | CSOM | MATH | 2022 |
| dia@bc.edu | Morrissey College, A | mathematics bs | 2022 | MCAS | ISYS | 2023 |
| an@bc.edu | Carroll School of Ma | information systems | 2023 | CSOM | BIOL | 2021 |
| is.5@bc.edu | Morrissey College, A | biology | 2021 | MCAS | UNDC | 2023 |
| c.edu | Carroll School of Ma | undecided | 2023 | MCAS | CSBA | 2024 |
| @bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBA | 2024 |
| bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBS | 2024 |
| za@bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBS | 2024 |
| .edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBA | 2024 |
| .2@bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | MATH | 2024 |
| @bc.edu | Morrissey College, A | computer science b. | 2024 | MCAS | CSBS | 2024 |
| es@bc.edu | Morrissey College, A | computer science b. | 2024 | MCAG | BIOL | 2026 |
| | | | | MCAS | ECON | 2023 |

Count Enrolled Students

The screenshot shows a Java application running in an IDE. The code in `Count.java` reads student names from a CSV file and counts them. The output shows 98 students.

```
public class Count {
    public static void main(String[] args) {
        int count = 0;
        String student;
        In in = new In(args[0]);

        student = in.readLine();
        while (student != null) {
            count++;
            System.out.format("%s\n", student);
            student = in.readLine();
        }
        in.close();
        System.out.format("There are %d students.\n", count);
    }
}
```

Run: Count

```
MCAS      MATH    2021
There are 98 students.

Process finished with exit code 0
```

Event Log

Build completed successfully in 1 s 116 ms (moments ago)

12:20 LF UTF-8 2 spaces master