Introduction to Java

Introduction to Programming and Computational Problem Solving: Accelerated Pace

CSE 11

Lecture 2

Announcements

- Assignment 1 will be released today
  - Due Apr 9, 11:59 PM

Programs

- Computer programs (i.e., software) are instructions to the computer
- You tell a computer what to do through programs
- Computers do not understand human languages, so you need to use computer languages to communicate with them
- Programs are written using programming languages

Programming languages

- Machine language
- Assembly language
- High-level language

Programming languages

- Machine language
  - Machine language is a set of primitive instructions built into every computer
  - The instructions are in the form of binary code, so you must enter binary codes for various instructions
  - Programming with native machine language is a tedious process, and the programs are highly difficult to read and modify
  - For example, to add two numbers, you might write an instruction in binary like this:

    ```
    1101101010011010
    ```

Programming languages

- Assembly language
  - Assembly languages were developed to make programming easier than machine languages
  - Since the computer cannot understand assembly language, a program called assembler is used to convert assembly language programs into machine code
  - For example, to add two numbers, you might write an instruction in assembly code like this:

```
ADDF3 R1, R2, R3
```

Programming languages

- High-level language
  - High-level languages are English-like and easier to learn and program than assembly languages
    - For example, the following is a high-level language statement that computes the area of a circle with radius 5:

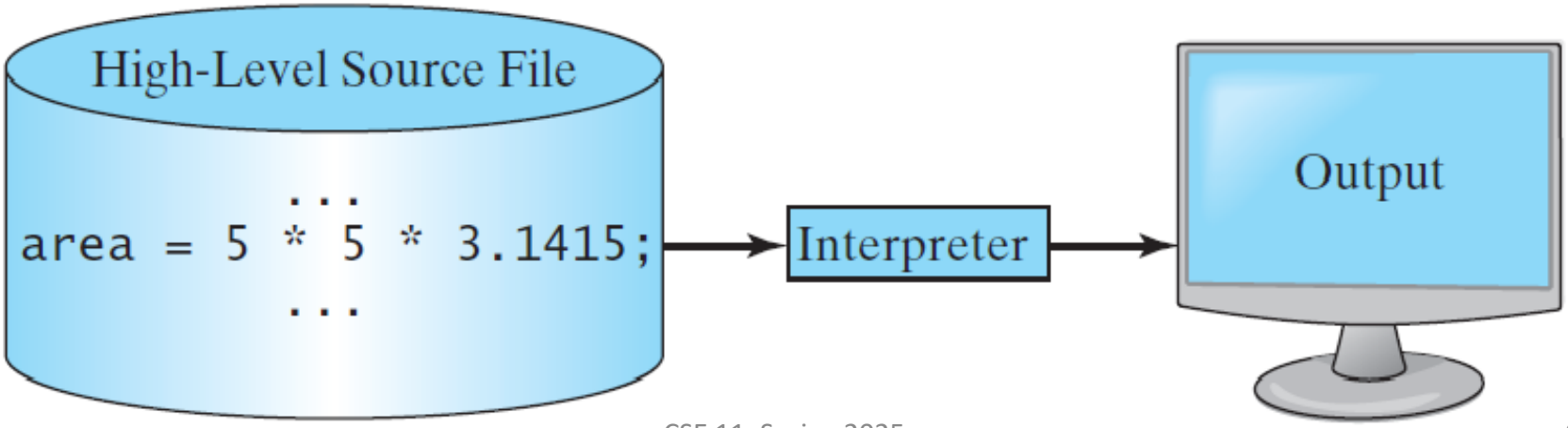
```
area = 5 * 5 * 3.1415;
```

Interpreting/Compiling source code

- A program written in a high-level language is called a source program or source code
- Because a computer cannot understand a source program, a source program must be translated into machine code for execution
- The translation can be done using another programming tool called an interpreter or a compiler

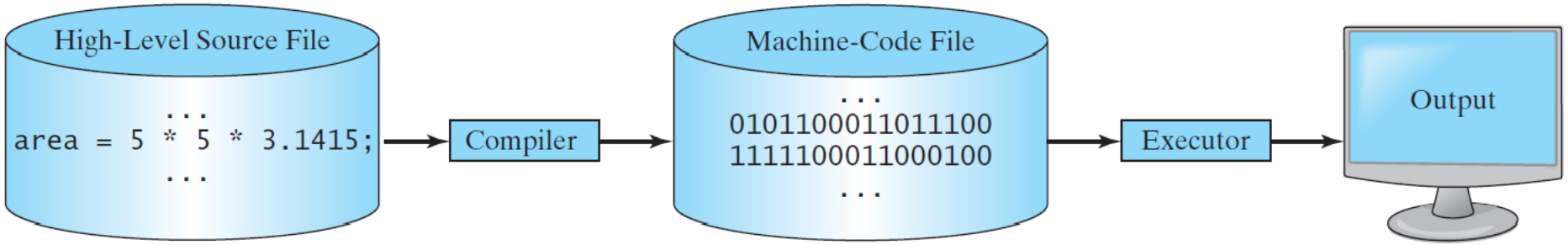
Interpreting source code

- An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away
- A statement from the source code may be translated into several machine instructions



Compiling source code

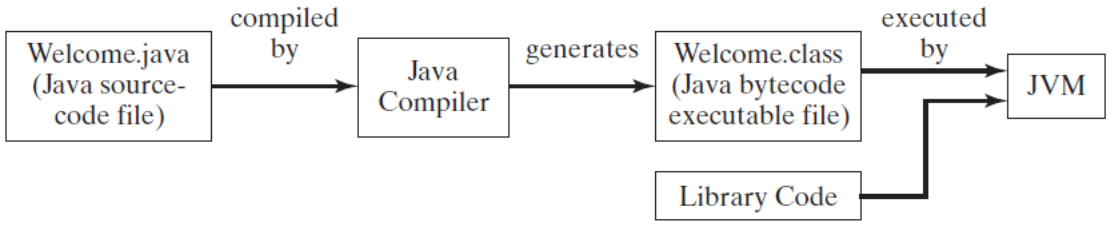
- A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed



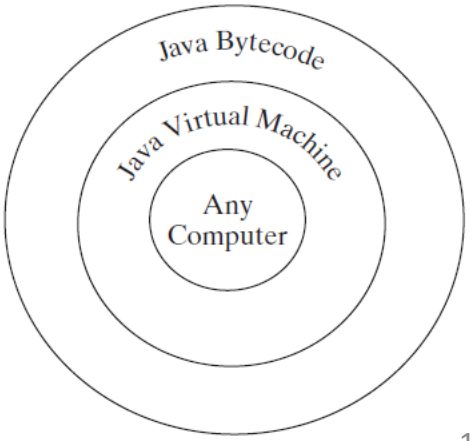
Java

- The **compiler** of Java is called `javac`
  - Java source code is compiled into the Java Virtual Machine (JVM) code called bytecode
- The **interpreter** of Java is called `java`
  - The bytecode is machine-independent and can run on any machine that has a Java interpreter, which is part of the JVM (write once, run anywhere)

Compile source code, interpret bytecode



(a)



(b)

Introduction to Java

- Java is:
  - a high-level programming language
    - Computer-specific details are abstracted
  - an object-oriented programming language
    - Based on classes
  - a strongly typed language
    - **Programmers must explicitly identify the type of every variable, method, and object**
  - a general-purpose programming language
    - Not specialized to a particular application domain
  - platform independent
    - Write a program once and run it on any computer

Anatomy of a Java program

- Class name
- Main method
- Statements
- Statement terminator
- Reserved words
- Comments
- Blocks

Class name

- Every Java program must have at least one class
- Each class has a name corresponding to filename
- Naming convention: capitalize the first letter of each word in the name class (e.g., `VideoWindow`)
- This class name is `Welcome` (stored in `Welcome.java`)

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

`main` method

- In order to run a class, the class must contain a method named `main`
- The program is executed from the `main` method
- This line defines the `main` method

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Statement

- A statement represents an action or a sequence of actions
- This is a statement to display the greeting "Welcome to Java!"

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Java application programming interface (API) documentation

- Documentation for all Java built-in classes and methods
- Java 8 API Documentation

  https://docs.oracle.com/javase/8/docs/api/
- Java 11 API Documentation

  https://docs.oracle.com/en/java/javase/11/docs/api/
- **Use the documentation!**

println and print

println

```
public void println(String x)
```

Prints a String and then terminate the line. This method behaves as though it invokes `print(String)` and then `println()`.

Parameters:

`x` - The `String` to be printed.

print

```
public void print(String s)
```

Prints a string. If the argument is `null` then the string `"null"` is printed. Otherwise, the string's characters are converted into bytes according to the platform's default character encoding, and these bytes are written in exactly the manner of the `write(int)` method.

Parameters:

`s` - The `String` to be printed

println

```
public void println()
```

Terminates the current line by writing the line separator string. The line separator string is defined by the system property `line.separator`, and is not necessarily a single newline character (`'\n'`).

Statement terminator

- Every statement in Java ends with a semicolon

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Reserved words

- Reserved words or keywords are words that have a specific meaning to the compiler and **cannot be used for other purposes in the program**
- For example, when the compiler sees the word `class`, it understands that the word after `class` is the name for the class

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

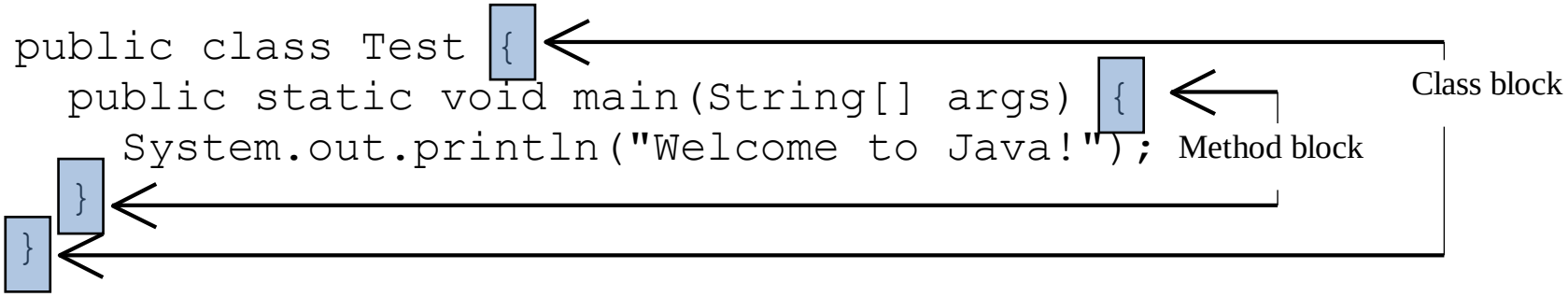
Comments

- Comments make the code more readable by adding details
- Implementation comments are meant for commenting out code or for comments about the particular implementation
- // comments out everything after it on the line
- The comment delimiters /*...*/ comments out everything between /* and */

```
// This program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

Blocks

- A pair of braces in a program forms a block that groups components of a program



Blocks

- Two different block styles

Next-line style

Corresponding braces are column-aligned

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

End-of-line style

First and last lines of block are column-align

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

Special symbols

Character	Name	Description
{}	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string (i.e., sequence of characters).
;	Semicolon	Marks the end of a statement.

Identifiers

- Identifiers are the **names** that identify the elements such as **classes, methods, and variables** in a program
- An identifier is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($)
- An identifier must start with a letter, an underscore (_), or a dollar sign ($)
- An identifier cannot start with a digit
- An identifier cannot be a reserved word
  - List of reserved words
    - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html
    - https://docs.oracle.com/javase/specs/jls/se11/html/jls-3.html#jls-3.9
- An identifier cannot be `true`, `false`, or `null`
- An identifier can be of any length

Variable and method names

- Naming convention: Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name
  - For example, the variables `radius` and `area`, and the method `computeArea`.

Variables

- Variables are used to represent values that may be changed in the program

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " +
   radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " +
   radius);
```

Declaring variables

```
int x;          // Declare x to be an
                // integer variable
double radius; // Declare radius to
                // be a double variable
char a;         // Declare a to be a
                // character variable
```

Assignment statements

```
x = 1;              // Assign 1 to x

radius = 1.0;       // Assign 1.0 to radius

a = 'A';            // Assign 'A' to a
```

Declaring and initializing in one step

```
int x = 1;

double radius = 1.0;

char a = 'A';
```

Named constants

- Naming convention: capitalize all letters in constants, and use underscores to connect words

```
final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int MAX_VALUE = 3;
final char FIRST_UPPER_CASE = 'A';
```

Trace a program execution

```
public class ComputeArea {
 /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

Documentation (or doc or Javadoc) comments

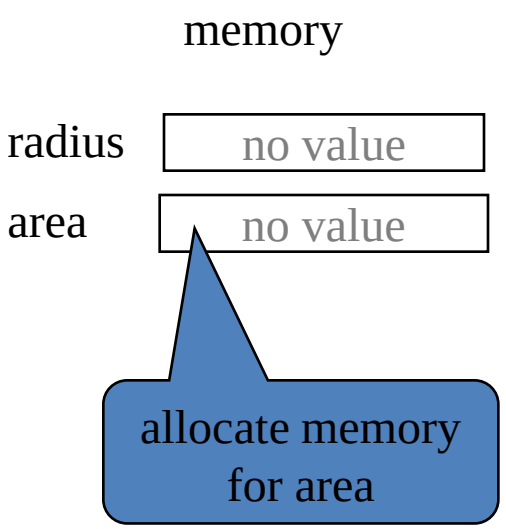
- The comment delimiters /*...*/ comments out everything between /* and */, including the * following the begin delimiter /*
- /** indicates this is beginning of a doc comment

Trace a program execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

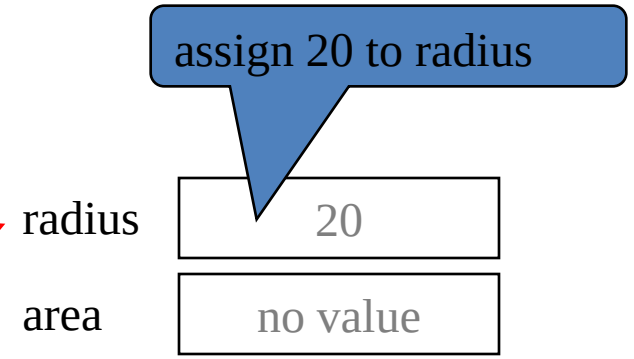
Trace a program execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



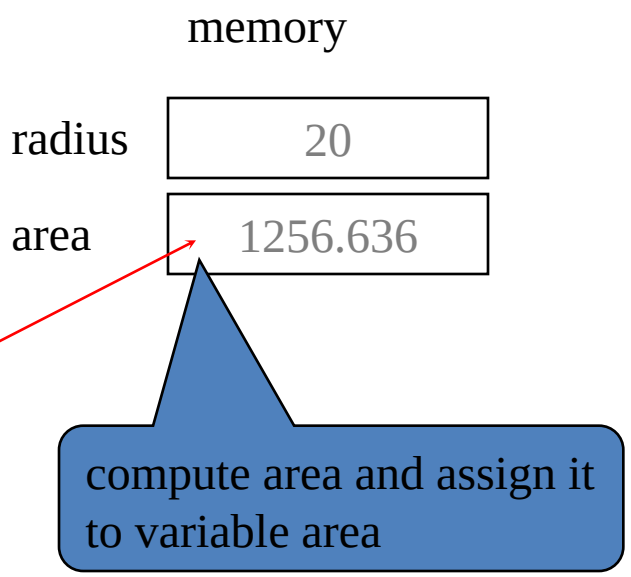
Trace a program execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



Trace a program execution

```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```



Trace a program execution

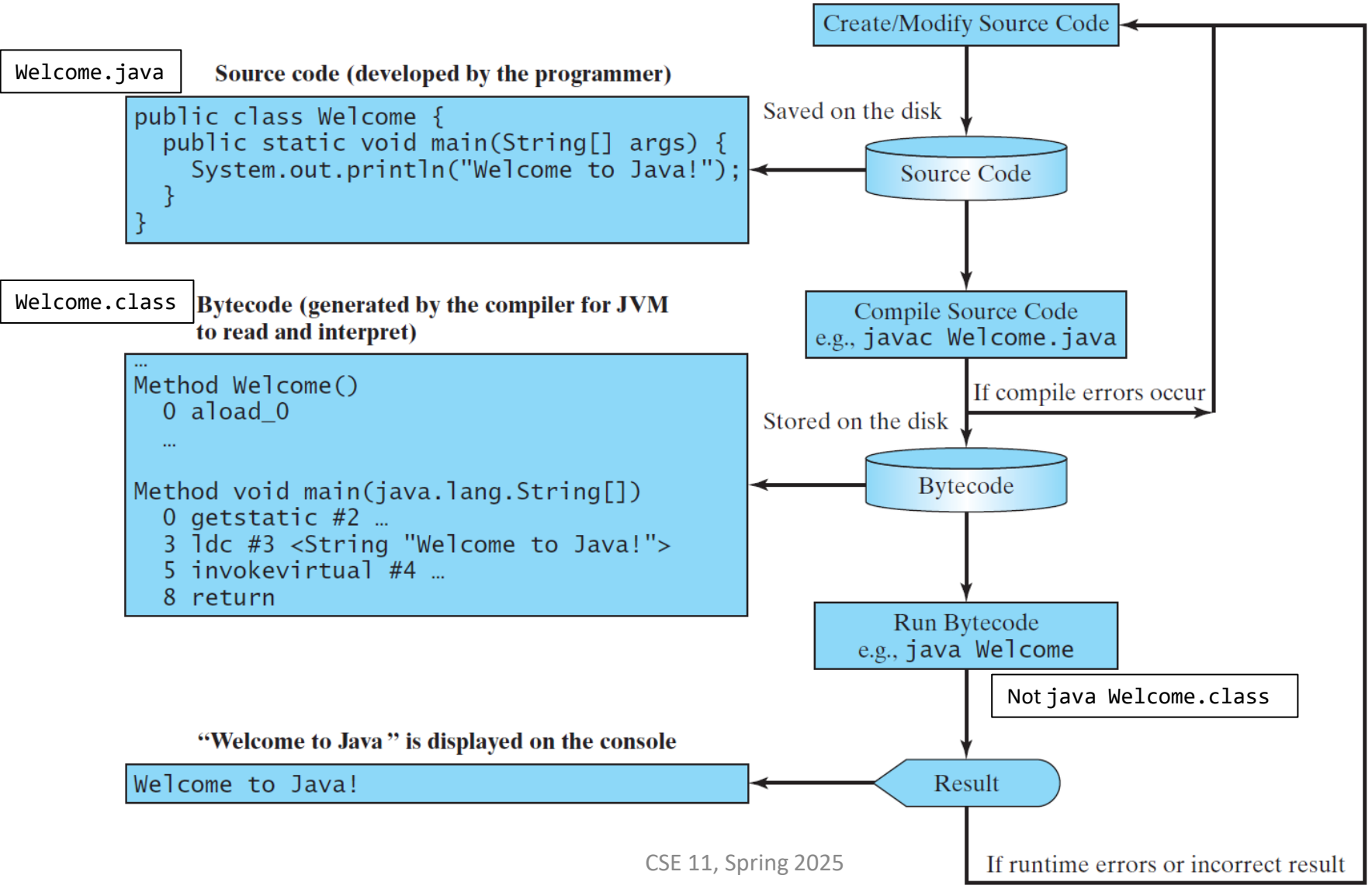
```
public class ComputeArea {
  /** Main method */
  public static void main(String[] args) {
    double radius;
    double area;

    // Assign a radius
    radius = 20;

    // Compute area
    area = radius * radius * 3.14159;

    // Display results
    System.out.println("The area for the circle of radius " +
      radius + " is " + area);
  }
}
```

memory

radius	20
area	1256.636

print a message to the console

Developing, compiling, and running Java programs



Programming errors

- Syntax errors
  - Detected by the compiler
    - The **compiler** of Java is called `javac`
- Runtime errors
  - Causes the program to abort
    - The **interpreter** of Java is called `java`
- Logic errors
  - Produces incorrect result

Syntax errors

- If you mistype part of a program, the compiler may issue a syntax error. The message usually displays the type of the error, the line number where the error was detected, the code on that line, and the position of the error within the code.
- For example, following is an error caused by omitting a semicolon at the end of a statement

Filename:line number Type of error

```
Testing.java:8: error: ';' expected
                count++
                      ^
1 error
```

Position of the error within the code

- If you see any compiler errors, then your program did not successfully compile, and the compiler did not create a `.class` file. Carefully verify the program, fix any errors that you detect, and try again.

Syntax errors

- Semantic Errors: In addition to verifying that your program is syntactically correct, the compiler checks for other basic correctness. For example, the compiler warns you each time you use a variable that has not been initialized.

```
Testing.java:8: error: variable count might not have been initialized
            count++;
            ^ Position of the error within the code
Testing.java:9: error: variable count might not have been initialized
        System.out.println("Input has " + count + " chars.");
                                          ^ Position of the error within the code
2 errors
```

- Again, your program did not successfully compile, and the compiler did not create a `.class` file. Fix the error and try again.

Runtime errors

`Exception in thread "main"`

- If you encounter this, see https://docs.oracle.com/javase/tutorial/getStarted/problems/index.html#interpreter

Java Software Development

Java versions

- The Java roadmap includes long-term support (LTS) versions, with non-LTS versions along the way
- Non-LTS versions are unsupported feature releases, allowing developers to explore features that may be in the next LTS version
- Best practice is to only use LTS versions
- LTS versions
  - Java 8 (released Mar 2014, supported until Dec 2030)
  - Java 11 (released Sep 2018, supported until Jan 2032)
  - Java 17 (released Sep 2021, supported until Sep 2029)
  - Java 21 (released Sep 2023, supported until Sep 2031)

Covered in Assignment 1

Java Development Kit (JDK)

1. Develop on a UCSD lab computer (ieng6.ucsd.edu) or UCSD Linux Cloud (https://linuxcloud.ucsd.edu)
   - The correct version of Java is already installed and configured on your account
2. Develop on your personal computer
   - Download, install, and configure the Java SE Development Kit that is the same version installed and configured on your UCSD Linux Cloud account
   - Important: issues with your personal computer will not excuse late assignment submissions (or missed prelecture quizzes, etc.)
     - Best practice is to save your source code to your UCSD Google Drive, so you can finish on UCSD Linux Cloud and/or a UCSD computer

- **Your source code must compile and run from the command-line on your UCSD account**

Covered in Assignment 1

Text editor

- Use whatever text editor you want
  - Visual Studio Code (**highly recommended and used in lecture**), Notepad, Notepad++, vi (or Vim), Emacs, etc.
  - Important note: Visual Studio Code will offer to install extensions, enabling it to be an integrated development environment (IDE)
    - Should you choose to do this, the instructional team will not assist you in configuring it
    - **On UCSD Linux Cloud, do not install Visual Studio Code Extensions! Linux Cloud will go down.**
- **Compile and run from the command line**
- Beware of integrated development environments (IDEs) that generate source code!
  - **Your source code must compile and run from the command-line on your UCSD account**
  - Typing all code yourself will best prepare you for the midterm and final assessments/exams

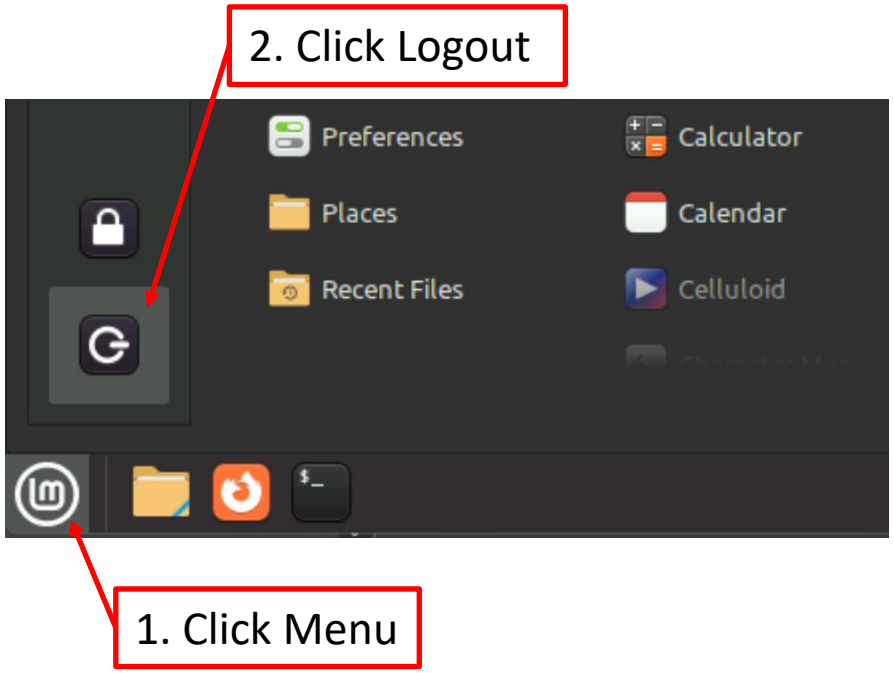
Covered in Assignment 1

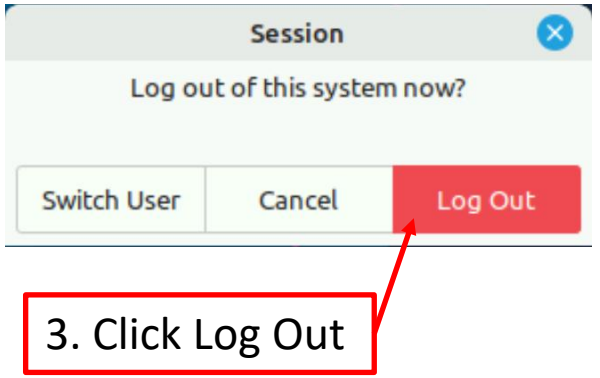
UCSD Linux Cloud

Covered in Assignment 1

UCSD Linux Cloud

- Log in using your UCSD account
- **Do not forget to logout before closing browser tab/window! Linux Cloud will go down.**





Next lecture

- Numbers and mathematics