

# COMP 110/L Lecture 3

Mahdi Ebrahimi

Some slides adapted from Dr. Kyle Dewey

# Outline

- Introduction
- Types (`int` and `String`)
- String concatenation
- Variables
- User input

# Introduction

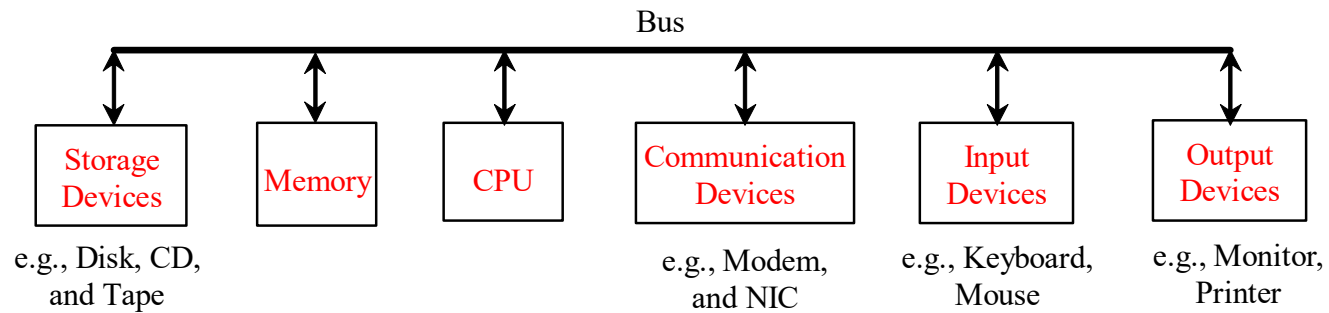
- Java is one of the world's most widely used computer programming languages.
- You'll learn to write instructions in the Java programming language
- Computer Programs are step by step instruction that tell computers to perform tasks.
- *Software: Set of programs that tell computer how to solve specific problem*

# Computer System

- A computer is a digital machine for storing, processing, and retrieving information
- Computers can perform calculations and make logical decisions phenomenally faster than human beings *can*.
- Computer System
  - Hardware and Software Computers can perform calculations and make logical decisions phenomenally faster than human beings *can*.

# Hardware

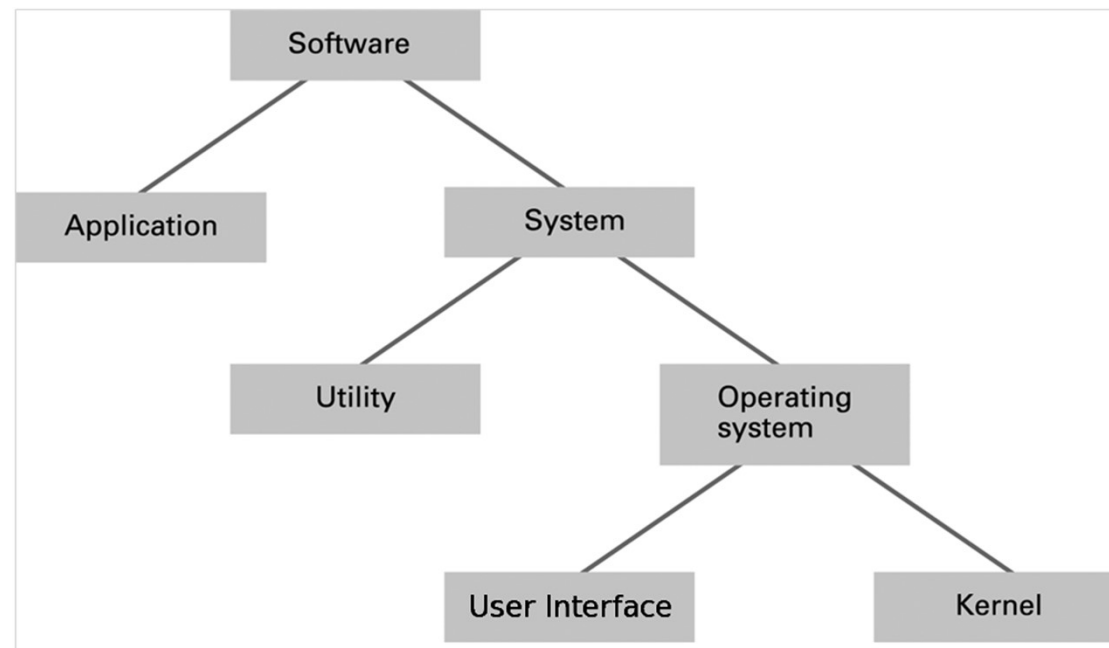
- Physical Components
  - A computer consists of a CPU, memory, hard disk, floppy disk, monitor, printer, and communication devices.



# Software

- Computers process data under the control of sequences of instructions called **computer programs**.
- **Computer Programs**: Set of step-by-step instructions that tell computer how to solve a specific problem,
- **Software**: Collection of computer programs

# Software classification



# Operating System Components

- **User Interface:** Communicates with users
  - Text based (Shell)
  - Graphical user interface (GUI)
- **Kernel:** Performs basic required functions
  - File manager
  - Device drivers
  - Memory manager
  - Scheduler and dispatcher



# Check Point

- What are hardware and software?
- List the five major hardware components of a computer.
- What does the acronym CPU stand for? What unit is used to measure CPU speed?
- What is a bit? What is a byte?
- What is memory for? What does RAM stand for? Why is memory called RAM?
- What is the primary difference between memory and a storage device

# Programming Languages

Machine Language    Assembly Language    High-Level Language

- Any computer can directly understand only its own **machine language**, *defined by its hardware design*.
  - Generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time.
  - *Machine dependent—a particular machine language can be used on only one type of computer.*
  - For example, to add two numbers, you might write an instruction in binary like this:

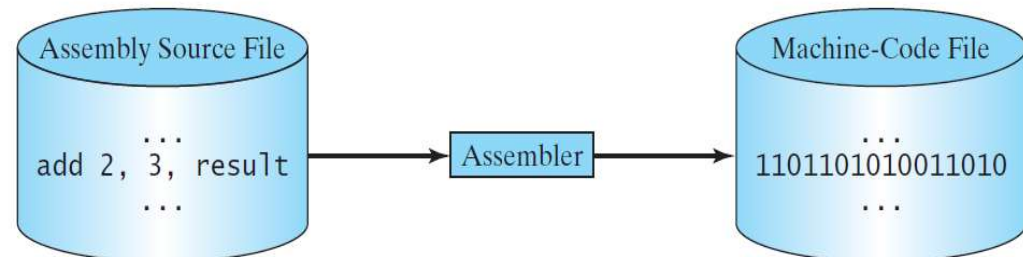
```
1101101010011010
```

# Programming Languages

Machine Language   **Assembly Language**   High-Level Language

- Assembly languages were developed to make programming easy.
- English-like abbreviations that represent elementary operations formed the basis of **assembly languages**.
- *Translator programs* called **assemblers** convert early assembly-language programs to machine language.
- For example, to add two numbers, you might write an instruction in assembly code like this:

ADDF3 R1, R2, R3



# Programming Languages

Machine Language    Assembly Language    **High-Level Language**

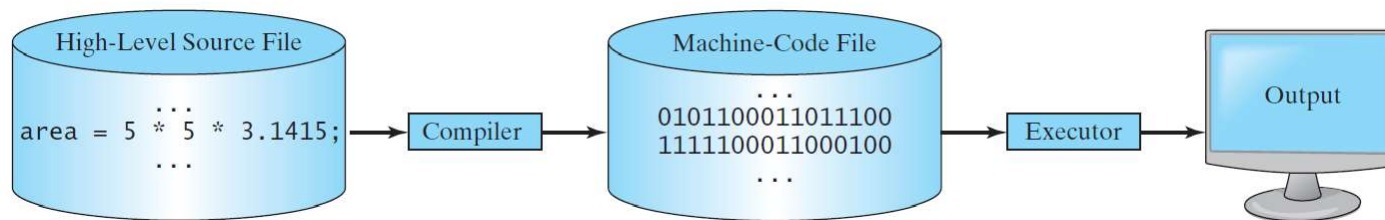
- The high-level languages are English-like and easy to learn and program.
- Single statements accomplish substantial tasks.
- A Java program to calculate the area of a circle might contain a single statement such as  
$$\text{area} = \text{radius} * \text{radius} * \text{Math.PI};$$
- High-Level programs must be converted to Machine Language.

## 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.

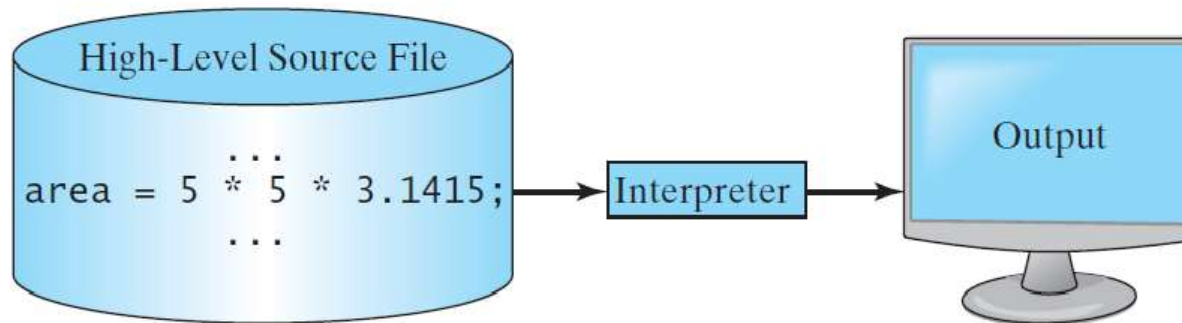
# Compiling Source Code

- A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed, as shown in the following figure.
- Compiling a high-level language program into machine language can take considerable computer time.



# Interpreting Source Code

- Interpreter programs, developer to execute high-level language programs directly, avoid the delay or compilation, although they run slower than compiled programs.



# Popular High-Level Languages

<b>Language</b>	<b>Description</b>
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	C++ is an object-oriented language, based on C.
C#	Pronounced "C Sharp." It is a hybrid of Java and C++ and was developed by Microsoft.
COBOL	COmmon Business Oriented Language. Used for business applications.
FORTRAN	FORmula TRANslation. Popular for scientific and mathematical applications.
Java	Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications.
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming.
Python	A simple general-purpose scripting language good for writing short programs.
Visual Basic	Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop graphical user interfaces.



# Check Point

- What language does the CPU understand?
- What is an assembly language? What is an assembler?
- What is a high-level programming language? What is a source program?
- What is an interpreter? What is a compiler?
- What is the difference between an interpreted language and a compiled language?

# Java

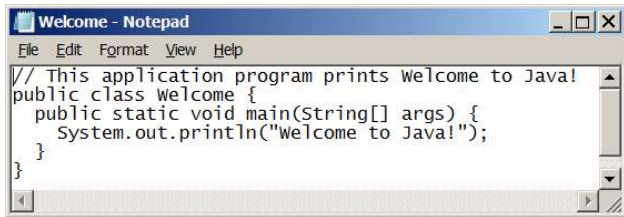
- Java was developed by a team at Sun Microsystem led by James Gosling
- Based on C++ object-oriented programming language
- Java can be used to develop standalone applications.
- Java can be used to develop applications running from a browser.
- Java can also be used to develop applications for hand-held devices.
- Java can be used to develop applications for Web servers.
- Using Java, you can write programs that will run on a great variety of computer systems and computer-controlled devices. This is sometimes called “write once, run anywhere.”

## Java (Cont.)

### ***Java Class Libraries***

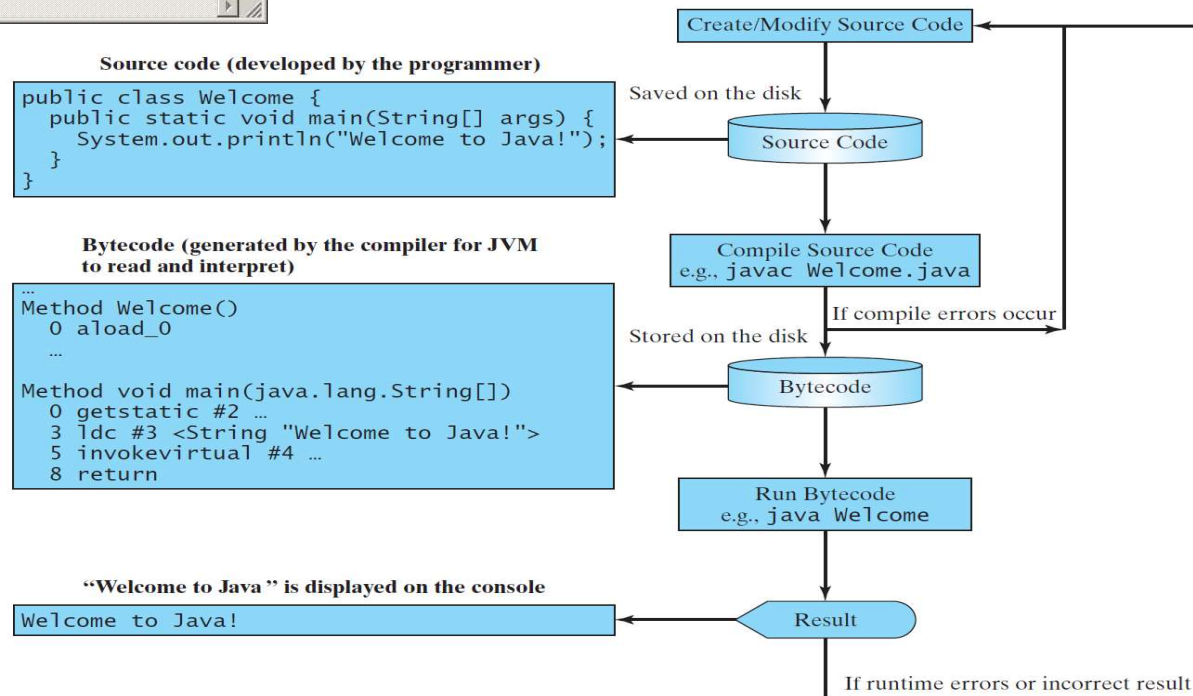
- Rich collections of existing classes and methods
- Also known as the **Java APIs (Application Programming Interfaces)**.

# Java Development Environment



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

Edit → Compile → Load → Verify → Execute



## Java Development Environment (Cont.)

- Linux systems - vi and emacs
- Windows - Notepad.
- macOS - TextEdit.
- Many freeware and shareware editors are also available online
  - Notepad++ (<http://notepad-plus-plus.org>)
  - EditPlus (<http://www.editplus.com>)
  - TextPad (<http://www.textpad.com>)
  - jEdit (<http://www.jedit.org>) and more.

## Java Development Environment (Cont.)

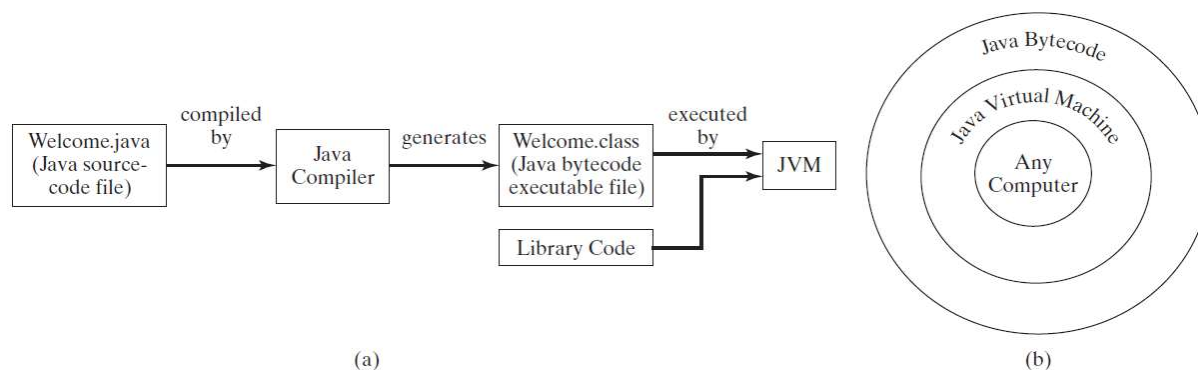
- **Integrated development environments (IDEs)** provide tools that support the software development process, such as editing, compiling, debugging, and executing.
- Some Java IDEs are:
  - JGrasp (<http://www.jgrasp.org>)
  - Eclipse (<http://www.eclipse.org>)
  - NetBeans (<http://www.netbeans.org>)

## Java Development Environment (Cont.)

- Compiling a Java Program into Bytecodes
  - Install jdk (Java Development Kit)
    - Java SE Development Kit 8 or 9  
(<http://www.oracle.com/technetwork/java/javase/overview/index.html>)
  - Use the command `javac` (the **Java compiler**) to **compile** a program.
    - `javac Welcome.java`
  - Java compiler translates Java source code into bytecodes that represent the tasks to execute. (`Welcome.class`)
  - The **Java Virtual Machine (JVM)**—a part of the JDK and the foundation of the Java platform—executes bytecodes.

## Java Development Environment (Cont.)

- Bytecodes are **portable** (platform independent)
- The JVM is invoked by the **java** command.
  - `java Welcome`
- The JVM places the program in memory to execute it. (loading **.class** file)
  - As the classes are loaded, the **bytecode verifier** examines their bytecodes
  - Ensures that they're valid and do not violate Java's security restrictions.
- The JVM **executes** the program's bytecodes.





# Check Point

- What is the Java language specification?
- What does JDK stand for? What does JRE stand for?
- What does IDE stand for?
- Are tools like NetBeans and Eclipse different languages from Java, or are they dialects or extensions of Java?

# Your First Program in Java: Printing a Line of Text

- Java **application**

- A computer program that executes when you use the **java command** to launch the Java Virtual Machine (JVM).

Block Comments

```
1. /* File: Welcome.java
2. This program prints Welcome to Java!
3. */
4. public class Welcome {
5.     public static void main(String[] args) {
6.         System.out.println( "Welcome to Java!" );
7.     } // end main method
8. } // end class Welcome
```

Line Comments

Java Keywords (blue text)

# Your First Program in Java: Printing a Line of Text

```
1. /* File: Welcome.java
2. This program prints Welcome to Java!
3. */
4. public class Welcome {
5.     public static void main(String[] args) {
6.         System.out.println( "Welcome to Java!" );
7.     } // end main method
8. } // end class Welcome
```

Class Name

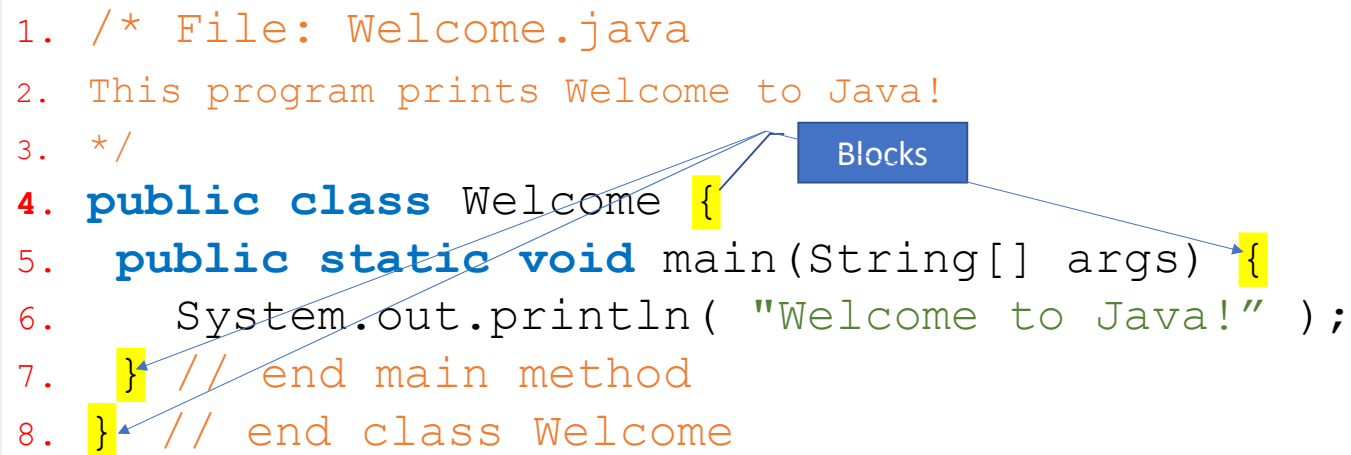
## *Class Names and Identifiers*

### ***Class Names and Identifiers***

- A class name is an **identifier**—a series of characters consisting of letters, digits, underscores (`_`) and dollar signs (`$`) that does not begin with a digit and does not contain spaces.
- Java is **case sensitive**—uppercase and lowercase letters are distinct—so `a1` and `A1` are different (but both valid) identifiers.
- By convention, begin with a capital letter and capitalize the first letter of each word they include (e.g., `SampleClassName`).
- Use meaningful names.

# Your First Program in Java: Printing a Line of Text

```
1. /* File: Welcome.java
2. This program prints Welcome to Java!
3. */
4. public class Welcome {
5.     public static void main(String[] args) {
6.         System.out.println( "Welcome to Java!" );
7.     } // end main method
8. } // end class Welcome
```



The diagram illustrates the concept of code blocks in Java. A blue box labeled "Blocks" has three arrows pointing to the closing curly braces of the class, the main method, and the println statement. These braces are highlighted in yellow in the original image.

## Declaring main Method

- Starting point of every Java application.
- **Parentheses** after the identifier `main` indicate that it's a program building block called a **method**.
- Java class declarations normally contain one or more methods.
- Keyword **void** indicates that this method will not return any information.

```
1. /* File: Welcome.java
2. This program prints Welcome to Java!
3. */
4. public class Welcome {
5.     public static void main(String[] args) {
6.         System.out.println( "Welcome to Java!" );
7.     } // end main method
8. } // end class Welcome
```

# System.out.println() Statement

- Instructs the computer to perform an action
  - Display the characters contained between the double quotation marks.
- Together, the quotation marks and the characters between them are a **string**—also known as a **character string** or a **string literal**.
- White-space characters in strings are *not* ignored by the compiler.
- Strings *cannot* span multiple lines of code.
- Most Java statements end with a semicolon.

```
1. /* File: Welcome.java
2. This program prints Welcome to Java!
3. */
4. public class Welcome {
5.     public static void main(String[] args) {
6.         System.out.println("Welcome to Java!");
7.     } // end main method
8. } // end class Welcome
```

Standard output object

The string in the parentheses the **argument** to the method.

Method: Displays (or prints) a line of text in the command window

# Special Symbols

<b>Character Name</b>	<b>Description</b>
{ }	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.



## Programming Style and Documentation

- Appropriate Comments
  - Naming Conventions
  - Proper Indentation and Spacing Lines
  - Block Styles
- 
- Make sure to read and apply the following recommendations:  
[https://mebrahimii.github.io/comp110-fall2020/resources/java\\_guide/](https://mebrahimii.github.io/comp110-fall2020/resources/java_guide/)

# Programming Errors

- Syntax Errors
  - Detected by the compiler
- Runtime Errors
  - Causes the program to abort
- Logic Errors
  - Produces incorrect result

## Syntax Errors

```
public class Welcome {  
    public static main(String[] args) {  
        System.out.println("Welcome to Java);  
    }  
}
```

## Runtime Errors

```
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println(1 / 0);  
    }  
}
```

# Logic Errors

```
public class Circle {  
    public static void main(String[] args) {  
        System.out.print("Area of a circle with radius 5 is ");  
        System.out.println( 5 * Math.PI );  
    }  
}
```

# Check Point

- What is a keyword? List some Java keywords.
- Is Java case sensitive? What is the case for Java keywords?
- What is a comment? Is the comment ignored by the compiler? How do you denote a comment line and a comment paragraph?
- What is the statement to display a string on the console?
- Show the output of the following code:

```
public class Test {  
    public static void main(String[] args) {  
        System.out.println("3.5 * 4 / 2 - 2.5 is ");  
        System.out.println(3.5 * 4 / 2 - 2.5);  
    }  
}
```

**Types**

# Expressions

- From the last lab, you wrote code like:
  - `"Hello, world!"`
  - `2 * (1 + 4)`
- Each of these is an expression (produces a value)



# Types

- All values are of a particular type
  - `"Hello, world!"`: `String`
  - `2 * (1 + 4)`: `int` (**integers**)
- Transitively, all expressions are of a particular type

# String Concatenation

# String Concatenation

Strings can be combined together with the + operator.

# String Concatenation

Strings can be combined together with the + operator.

---

```
"foo" + "bar"
```

# String Concatenation

Strings can be combined together with the + operator.

---

```
"foo" + "bar"
```

```
"foobar"
```

# String Concatenation

Strings can be combined together with the + operator.

---

```
"foo" + "bar"  
"foobar"
```

---

```
"foo" + "bar" + "baz"
```

# String Concatenation

Strings can be combined together with the + operator.

---

```
"foo" + "bar"  
"foobar"
```

---

```
"foo" + "bar" + "baz"  
"foobarbaz"
```

**Demo:**

`StringConcat.java`



# Concatenation with `int`

String concatenation also works with  
Strings and integers (`int`).

# Concatenation with `int`

String concatenation also works with  
Strings and integers (`int`).

---

```
"foo" + 7
```

# Concatenation with `int`

String concatenation also works with  
Strings and integers (`int`).

---

```
"foo" + 7
```

```
"foo7"
```

# Concatenation with `int`

String concatenation also works with  
Strings and integers (`int`).

---

```
"foo" + 7
```

```
"foo7"
```

---

```
"bar" + 28
```

# Concatenation with `int`

String concatenation also works with  
Strings and integers (`int`).

---

```
"foo" + 7
```

```
"foo7"
```

---

```
"bar" + 28
```

```
"bar28"
```

**Demo:**

`IntStringConcat.java`

**2 vs. "2" vs. '2'**

# Variables



# Variables

- Related to variables in math
- A named “box” you can put a value in

# Variables

**A variable is a container** which holds values that are used in a Java program.

Do you remember the basic math you learned in school?

$$y = x + 1$$

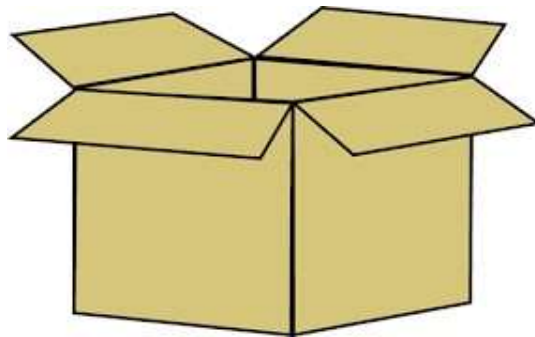
Here, as you can see, **the y variable changes when the x variable is different**. For example:

- if  $x = 1$ , then  $x + 1 = 2$
- if  $x = 2$ , then  $x + 1 = 3$
- if  $x = 1.5$ , then  $x + 1 = 2.5$

In Java, variables play the same role as in the above math example:  $y = x + 1$ . So, variables are containers that hold values.

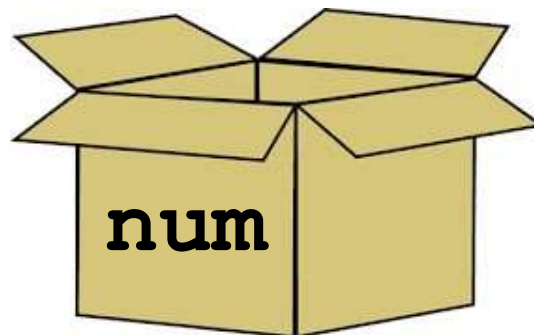
# Variables

- Related to variables in math
- A named “box” you can put a value in



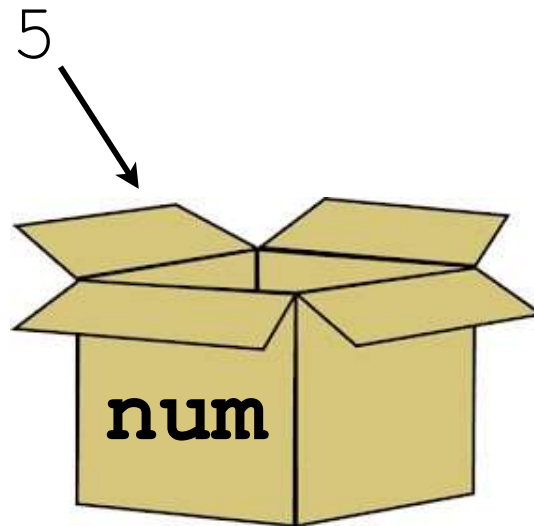
# Variables

- Related to variables in math
- A named “box” you can put a value in



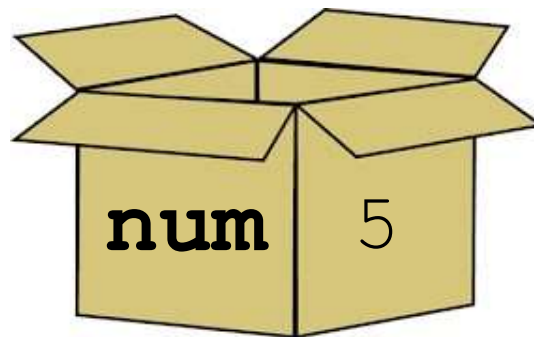
# Variables

- Related to variables in math
- A named “box” you can put a value in



# Variables

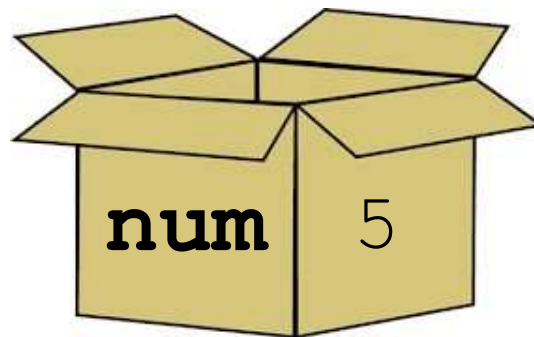
- Related to variables in math
- A named “box” you can put a value in



# Variables

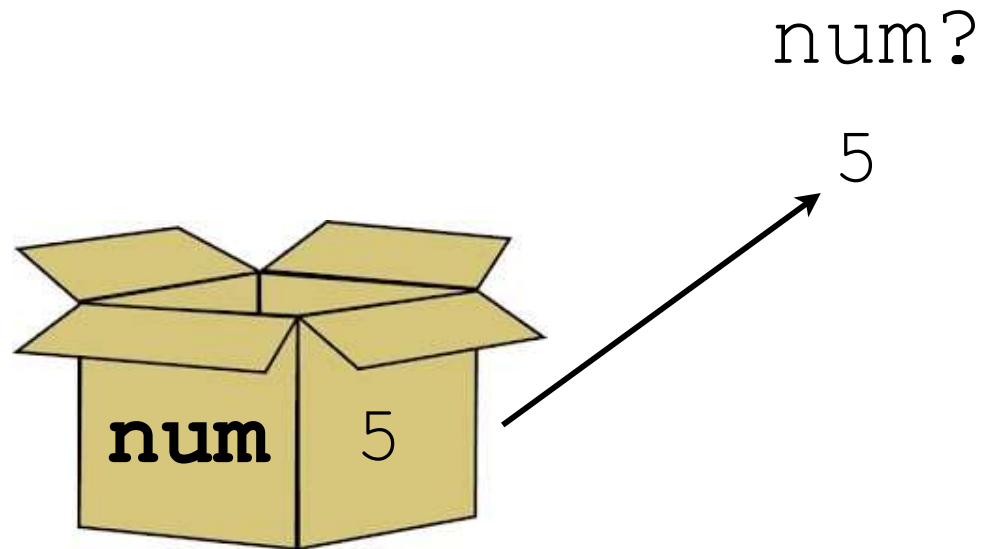
- Related to variables in math
- A named “box” you can put a value in

num?



# Variables

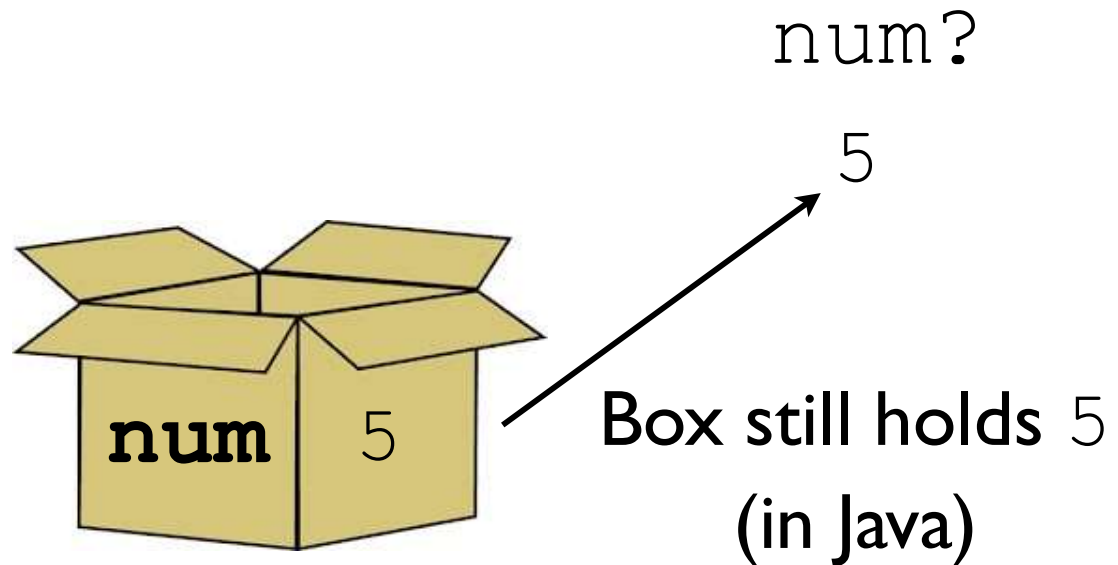
- Related to variables in math
- A named “box” you can put a value in





# Variables

- Related to variables in math
- A named “box” you can put a value in



# Getting a Box

In Java, we must ***declare a variable*** to get a new box. Part of this declaration includes the ***type*** of the thing we want to put into the box.

# Getting a Box

In Java, we must *declare a variable* to get a new box.

Part of this declaration includes the *type* of the thing we want to put into the box.

```
int num;
```

# Getting a Box

In Java, we must *declare a variable* to get a new box.

Part of this declaration includes the *type* of the thing we want to put into the box.

```
int num;
```

Variable named `num`, holds values of type `int`

# Getting a Box

In Java, we must *declare a variable* to get a new box. Part of this declaration includes the *type* of the thing we want to put into the box.

```
int num;
```

Variable named `num`, holds values of type `int`

```
String str;
```

Variable named `str`, holds values of type `String`

# Example:

`VariableDeclarations.java`

# Putting Values in the Box

- To put values into variables, we *assign into* them
- Assignment is performed with =

# Putting Values in the Box

- To put values into variables, we *assign into* them
- Assignment is performed with =

```
int num;  
num = 7;
```



# Putting Values in the Box

- To put values into variables, we *assign into* them
- Assignment is performed with =

```
int num;  
num = 7;
```

```
int num = 7;
```

# Retrieving Values from the Box

- To get a value out of a variable, we need to *access* it
- Variable access is done by referencing a variable in an expression context

# Retrieving Values from the Box

- To get a value out of a variable, we need to *access it*
- Variable access is done by referencing a variable in an expression context

```
int num = 7;  
int otherNum = num;  
int thirdNum = num + otherNum;
```

**Example:**

`VariableUsage.java`

# Question

- Variables can have their values *reassigned*
- Question: what might this code snippet print?

```
int num = 9;  
num = 12;  
System.out.println(num);
```

# Question

- Variables can have their values *reassigned*
- Question: what might this code snippet print?

```
int num = 9;  
num = 12;  
System.out.println(num);
```

Answer:12

# User Input

# Program Input

- Programs without input can't do much
  - Can only produce predetermined values
- We'll look at one kind of input: user input from the console/terminal



# Reading in Input

New bit of magic: Scanner

# Reading in Input

New bit of magic: Scanner

```
import java.util.Scanner;  
  
public class Test {  
    public static void  
    main(String[] args) {  
        Scanner in =  
            new Scanner(System.in);  
        ...  
    }  
}
```

# Reading in Integers (int)

```
Scanner in = new Scanner(System.in);  
int first = in.nextInt();  
int second = in.nextInt();  
int third = in.nextInt();
```

```
// above code reads in  
// three integers from the user
```

**Demo:**

AddTwo.java

# Reading in Text(String)

```
Scanner in = new Scanner(System.in);  
String firstLine = in.nextLine();  
String secondLine = in.nextLine();  
  
// above code reads in two lines  
// of text
```

**Demo:**  
Parrot.java

**Demo:**

`DoubleParrot.java`

# Data Types in Java

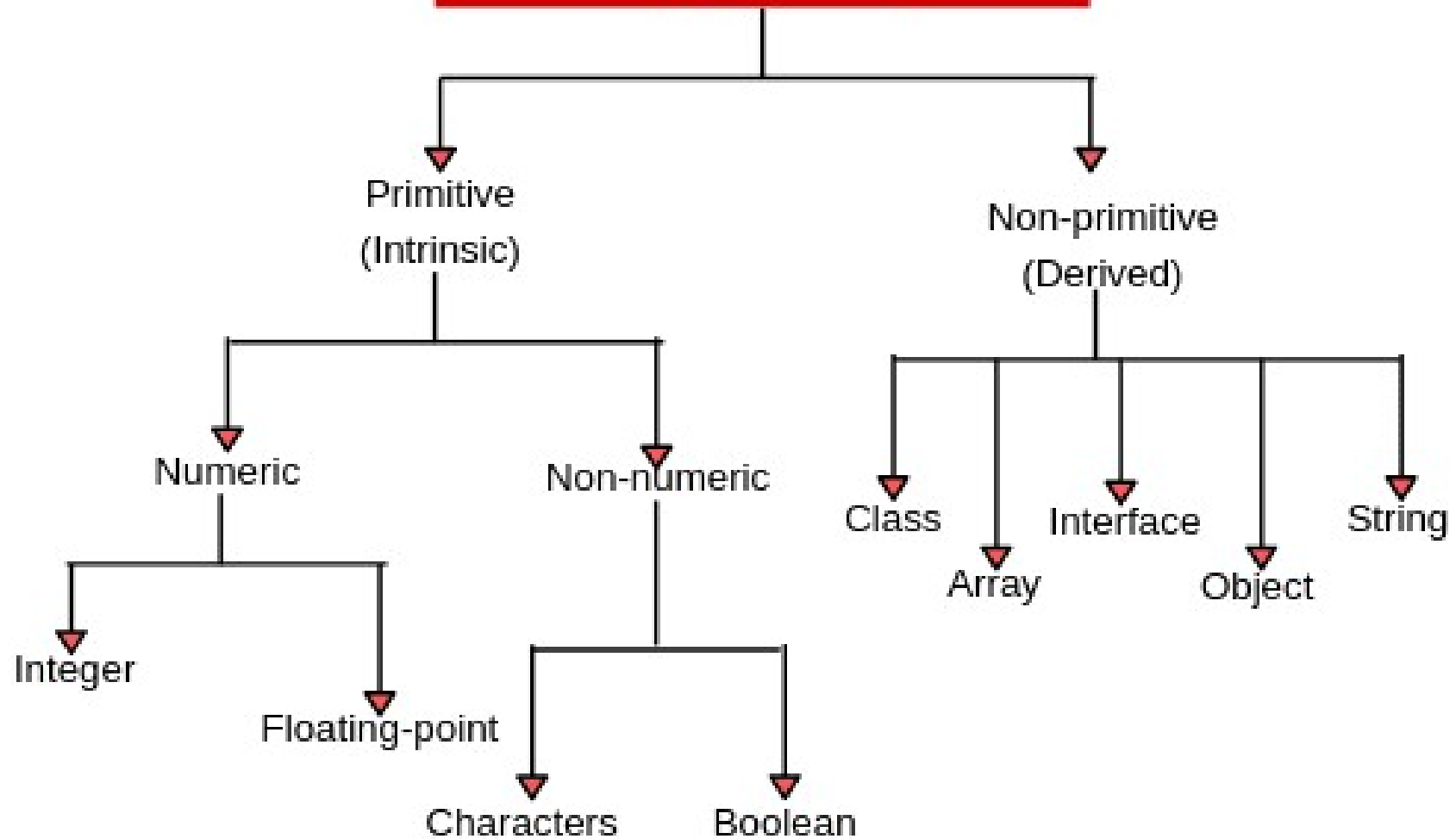


Fig: Classification of data types in java



```
// Java program to read data of various types using Scanner class.
import java.util.Scanner;
public class ScannerDemo1
{
    public static void main(String[] args)
    {
        // Declare the object and initialize with
        // predefined standard input object
        Scanner sc = new Scanner(System.in);

        // String input
        String name = sc.nextLine();

        // Character input
        char gender = sc.next().charAt(0);

        // Numerical data input
        // byte, short and float can be read
        // using similar-named functions.
        int age = sc.nextInt();
        long mobileNo = sc.nextLong();
        double cgpa = sc.nextDouble();

        // Print the values to check if the input was correctly obtained.
        System.out.println("Name: "+name);
        System.out.println("Gender: "+gender);
        System.out.println("Age: "+age);
        System.out.println("Mobile Number: "+mobileNo);
        System.out.println("CGPA: "+cgpa);
    }
}
```