

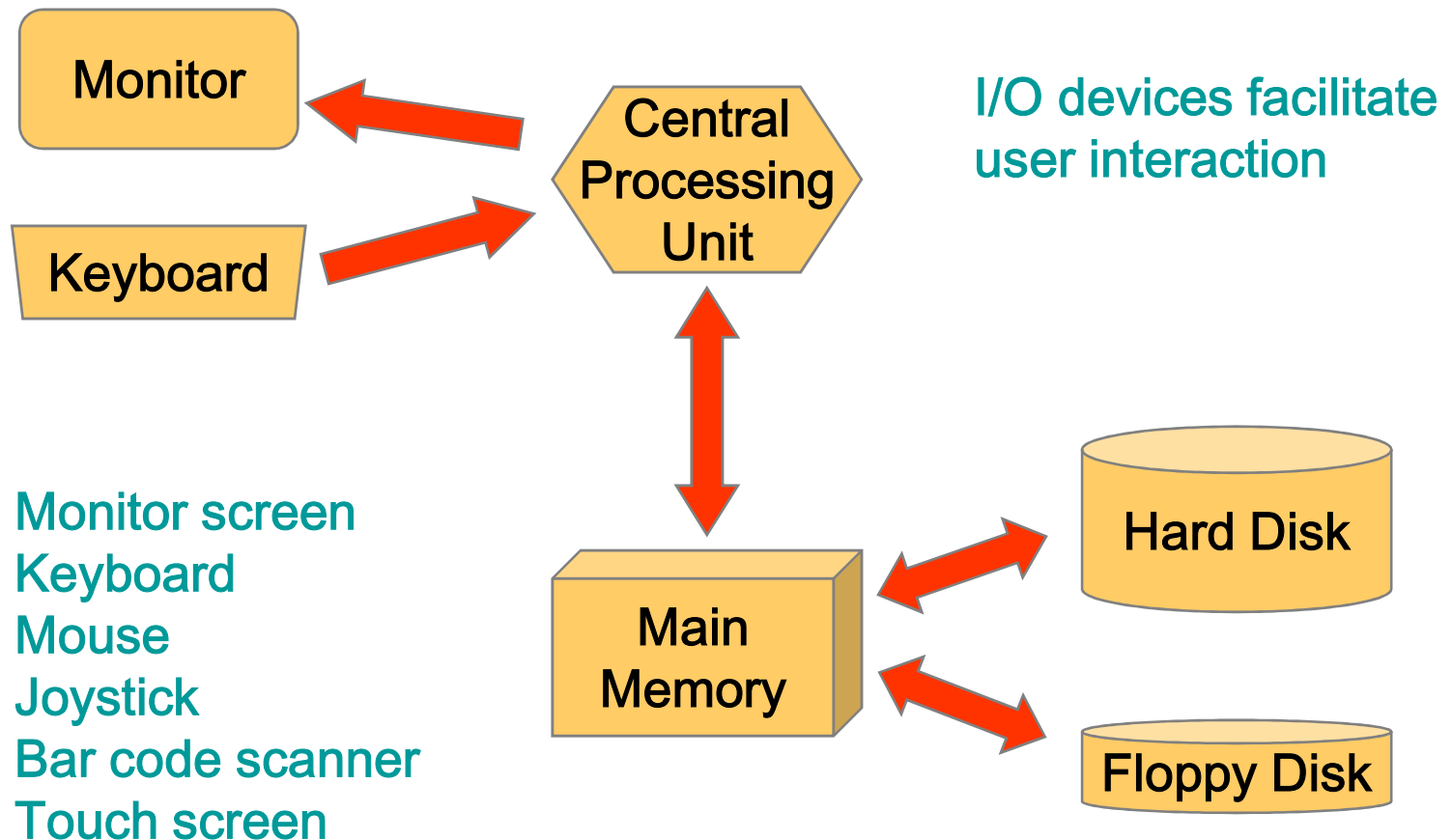
Getting Started

# CHAPTER 1

---

Slides prepared by Rose Williams, *Binghamton University*

# Computer Organization



# Computer Language Levels

- *High-level language*: A language that people can read, write, and understand
  - A program written in a high-level language must be translated into a language that can be understood by a computer before it can be run
- *Machine language*: A language that a computer can understand
- *Low-level language*: Machine language or any language similar to machine language
- *Compiler*: A program that translates a high-level language program into an equivalent low-level language program
  - This translation process is called *compiling*

# Introduction To Java

- Most people are familiar with Java as a language for Internet applications
- We will study Java as a general purpose programming language
  - The syntax of expressions and assignments will be similar to that of other high-level languages
  - Details concerning the handling of strings and console output will probably be new

# Origins of the Java Language

- Created by Sun Microsystems team led by James Gosling (1991)
- Originally designed for programming home appliances
  - Difficult task because appliances are controlled by a wide variety of computer processors
  - Team developed a two-step translation process to simplify the task of compiler writing for each class of appliances

# Origins of the Java Language

- Significance of Java translation process
  - Writing a compiler (translation program) for each type of appliance processor would have been very costly
  - Instead, developed intermediate language that is the same for all types of processors : Java *byte-code*
  - Therefore, only a small, easy to write program was needed to translate byte-code into the machine code for each processor

# Origins of the Java Language

- Patrick Naughton and Jonathan Payne at Sun Microsystems developed a Web browser that could run programs over the Internet (1994)
  - Beginning of Java's connection to the Internet
  - Original browser evolves into *HotJava*
- Netscape Incorporated made its Web browser capable of running Java programs (1995)
  - Other companies follow suit



# Byte-Code and the Java Virtual Machine

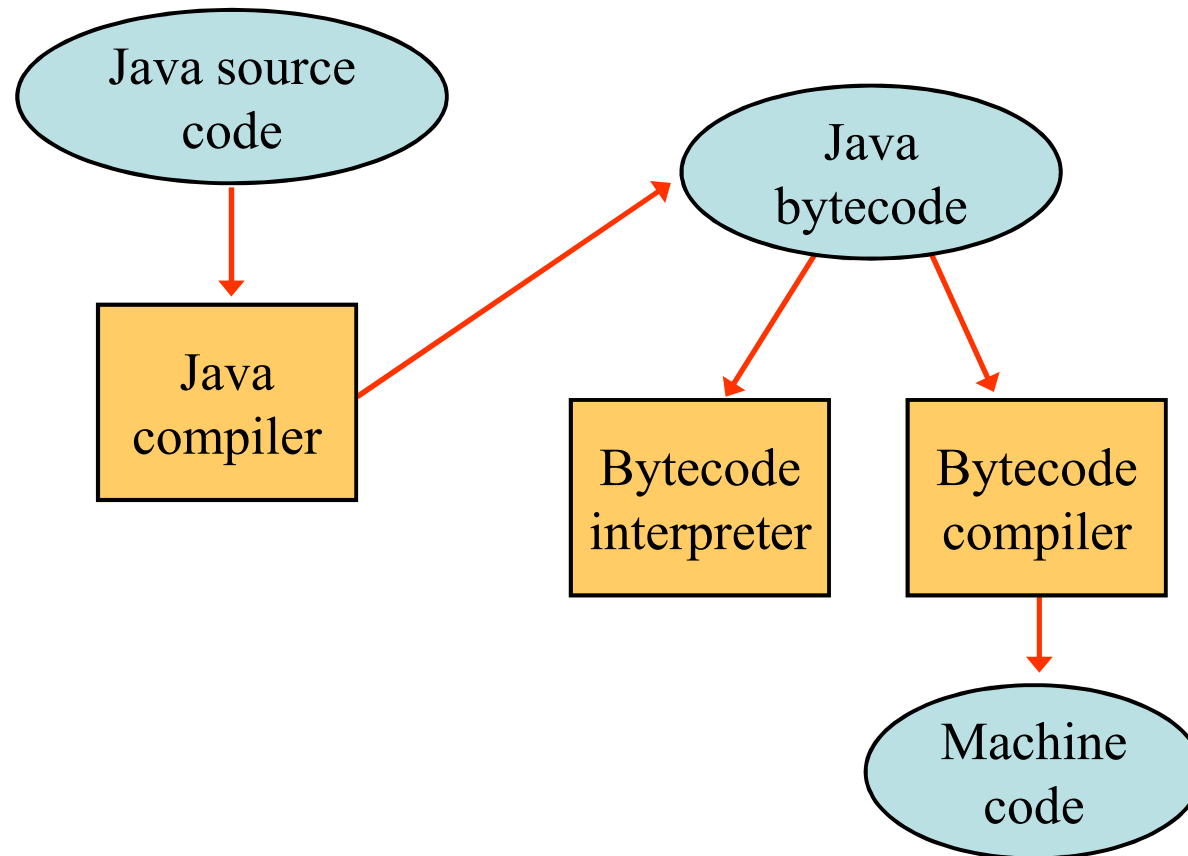
- The compilers for most programming languages translate high-level programs directly into the machine language for a particular computer
  - Since different computers have different machine languages, a different compiler is needed for each one
- In contrast, the Java compiler translates Java programs into *byte-code*, a machine language for a fictitious computer called the *Java Virtual Machine*
  - Once compiled to *byte-code*, a Java program can be used on any computer, making it very portable



# Byte-Code and the Java Virtual Machine

- *Interpreter:* The program that translates a program written in Java byte-code into the machine language for a particular computer when a Java program is executed
  - The interpreter translates and immediately executes each byte-code instruction, one after another
  - Translating byte-code into machine code is relatively easy compared to the initial compilation step

# Java Translation



# Programming terminology

- *Code*: A program or a part of a program
- *Source code* (or *source program*): A program written in a high-level language such as Java
  - The input to the compiler program
- *Object code*: The translated low-level program
  - The output from the compiler program, e.g., Java byte-code
  - In the case of Java byte-code, the input to the Java byte-code interpreter

# Class Loader

- Java programs are divided into smaller parts called *classes*
  - Each class definition is normally in a separate file and compiled separately
- *Class Loader*: A program that connects the byte-code of the classes needed to run a Java program
  - In other programming languages, the corresponding program is called a *linker*

# Objects and Methods

- Java is an *object-oriented programming (OOP)* language
  - Programming methodology that views a program as consisting of *objects* that interact with one another by means of actions (called *methods*)
  - Objects of the same kind are said to have the same *type* or be in the same *class*

# Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*
- These terms will be explored in detail throughout the course
- A Java application always contains a method called `main`

# Java Program Structure

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

class header



class body


Comments can be placed almost anywhere

```
}
```



# Java Program Structure

```
// comments about the class
public class MyProgram
{
    // comments about the method
    public static void main (String[] args)
    {
    }
}
```



method header

method body

# Terminology Comparisons

- Other high-level languages have constructs called procedures, methods, functions, and/or subprograms
  - These types of constructs are called *methods* in Java
  - All programming constructs in Java, including *methods*, are part of a *class*

# Java Application Programs

- There are two types of Java programs: *applications* and *applets*
- A Java *application program* or "regular" Java program is a class with a method named **main**
  - When a Java application program is run, the *run-time system* automatically invokes the method named **main**
  - All Java application programs start with the **main** method

# Applets

- A Java *applet* (*little Java application*) is a Java program that is meant to be run from a Web browser
  - Can be run from a location on the Internet
  - Can also be run with an applet viewer program for debugging
  - Applets always use a windowing interface
- In contrast, application programs may use a windowing interface or console (i.e., text) I/O

# A Sample Java Application Program

**Display 1.1 A Sample Java Program**

```
1 public class FirstProgram
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hello reader.");
6         System.out.println("Welcome to Java.");
7
8         System.out.println("Let's demonstrate a simple calculation.");
9         int answer;
10        answer = 2 + 2;
11        System.out.println("2 plus 2 is " + answer);
12    }
13 }
```

Annotations:

- ← Name of class (program) (points to `FirstProgram`)
- ← The main method (points to `main`)

## **SAMPLE DIALOGUE I**

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

# System.out.println

- Java programs work by having things called *objects* perform actions
  - **System.out**: an object used for sending output to the screen
- The actions performed by an object are called *methods*
  - **println**: the method or action that the **System.out** object performs

# System.out.println

- *Invoking or calling* a method: When an object performs an action using a method
  - Also called *sending a message* to the object
  - Method invocation syntax (in order): an object, a dot (period), the method name, and a pair of parentheses
  - Arguments: Zero or more pieces of information needed by the method that are placed inside the parentheses

```
System.out.println("This is an argument");
```



# Variable declarations

- Variable declarations in Java are similar to those in other programming languages
  - Simply give the type of the variable followed by its name and a semicolon

```
int answer;
```

# Using = and +

- In Java, the equal sign (=) is used as the *assignment operator*
  - The variable on the left side of the assignment operator is *assigned the value* of the expression on the right side of the assignment operator

```
answer = 2 + 2;
```

- In Java, the plus sign (+) can be used to denote addition (as above) or *concatenation*
  - Using +, two strings can be connected together

```
System.out.println("2 plus 2 is " + answer);
```

# Compiling a Java Program or Class

- Each class definition must be in a file whose name is the same as the class name followed by **.java**
  - The class **FirstProgram** must be in a file named **FirstProgram.java**
- Each class is compiled with the command **javac** followed by the name of the file in which the class resides

**javac FirstProgram.java**

- The result is a byte-code program whose filename is the same as the class name followed by **.class**

**FirstProgram.class**

# Running a Java Program

- A Java program can be given the *run command* (**java**) after all its classes have been compiled
  - Only run the class that contains the **main** method (the system will automatically load and run the other classes, if any)
  - The **main** method begins with the line:  
**public static void main(String[ ] args)**
  - Follow the run command by the name of the class only (no **.java** or **.class** extension)  
**java FirstProgram**

# Syntax and Semantics

- *Syntax*: The arrangement of words and punctuations that are legal in a language, the *grammar rules* of a language
- *Semantics*: The meaning of things written while following the syntax rules of a language

## Tip: Error Messages

- *Bug*: A mistake in a program
  - The process of eliminating bugs is called *debugging*
- *Syntax error*: A grammatical mistake in a program
  - The compiler can detect these errors, and will output an error message saying what it thinks the error is, and where it thinks the error is

# Tip: Error Messages

- *Run-time error:* An error that is not detected until a program is run
  - The compiler cannot detect these errors: an error message is not generated after compilation, but after execution
- *Logic error:* A mistake in the underlying algorithm for a program
  - *The compiler cannot detect these errors, and no error message is generated after compilation or execution, but the program does not do what it is supposed to do*



# Identifiers

- *Identifier*: The name of a variable or other item (class, method, object, etc.) defined in a program
  - A Java identifier must not start with a digit, and all the characters must be letters, digits, or the underscore symbol
  - Java identifiers can theoretically be of any length
  - Java is a case-sensitive language: **Rate**, **rate**, and **RATE** are the names of three different variables

# Identifiers

- Keywords and Reserved words: Identifiers that have a predefined meaning in Java
  - Do not use them to name anything else

`public`      `class`      `void`      `static`

- Predefined identifiers: Identifiers that are defined in libraries required by the Java language standard
  - Although they can be redefined, this could be confusing and dangerous if doing so would change their standard meaning

`System`      `String`      `println`

# Naming Conventions

- Start the names of variables, classes, methods, and objects with a lowercase letter, indicate "word" boundaries with an uppercase letter, and restrict the remaining characters to digits and lowercase letters

`topSpeed`    `bankRate1`    `timeOfArrival`

- Start the names of classes with an uppercase letter and, otherwise, adhere to the rules above

`FirstProgram`    `MyClass`    `String`

# Variable Declarations

- Every variable in a Java program must be *declared* before it is used
  - A variable declaration tells the compiler what kind of data (type) will be stored in the variable
  - The type of the variable is followed by one or more variable names separated by commas, and terminated with a semicolon
  - Variables are typically declared just before they are used or at the start of a block (indicated by an opening brace { )
  - Basic types in Java are called *primitive types*

```
int numberOfBeans;
```

```
double oneWeight, totalWeight;
```

# Primitive Types

**Display 1.2    Primitive Types**

TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
<code>boolean</code>	<code>true</code> or <code>false</code>	1 byte	not applicable
<code>char</code>	single character (Unicode)	2 bytes	all Unicode characters
<code>byte</code>	integer	1 byte	−128 to 127
<code>short</code>	integer	2 bytes	−32768 to 32767
<code>int</code>	integer	4 bytes	−2147483648 to 2147483647
<code>long</code>	integer	8 bytes	−9223372036854775808 to 9223372036854775807
<code>float</code>	floating-point number	4 bytes	$-3.40282347 \times 10^{+38}$ to $-1.40239846 \times 10^{-45}$
<code>double</code>	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

# Assignment Statements With Primitive Types

- In Java, the assignment statement is used to change the value of a variable
  - The equal sign (=) is used as the assignment operator
  - An assignment statement consists of a variable on the left side of the operator, and an *expression* on the right side of the operator
- An *expression* consists of a variable, number, or mix of variables, numbers, operators, and/or method invocations

`Variable = Expression;`

`temperature = 98.6;`  
`count = numberOfBeans;`

# Assignment Statements With Primitive Types

- When an assignment statement is executed, the expression is first evaluated, and then the variable on the left-hand side of the equal sign is set equal to the value of the expression

`distance = rate * time;`

- Note that a variable can occur on both sides of the assignment operator

`count = count + 2;`

- The assignment operator is automatically executed from right-to-left, so assignment statements can be chained

`number2 = number1 = 3;`



## Tip: Initialize Variables

- A variable that has been declared but that has not yet been given a value by some means is said to be *uninitialized*
- In certain cases an uninitialized variable is given a default value
  - It is best not to rely on this
  - Explicitly initialized variables have the added benefit of improving program clarity

## Tip: Initialize Variables

- The declaration of a variable can be combined with its initialization via an assignment statement

```
int count = 0;
```

```
double distance = 55 * .5;
```

```
char grade = 'A';
```

- Note that some variables can be initialized and others can remain uninitialized in the same declaration

```
int initialCount = 50, finalCount;
```