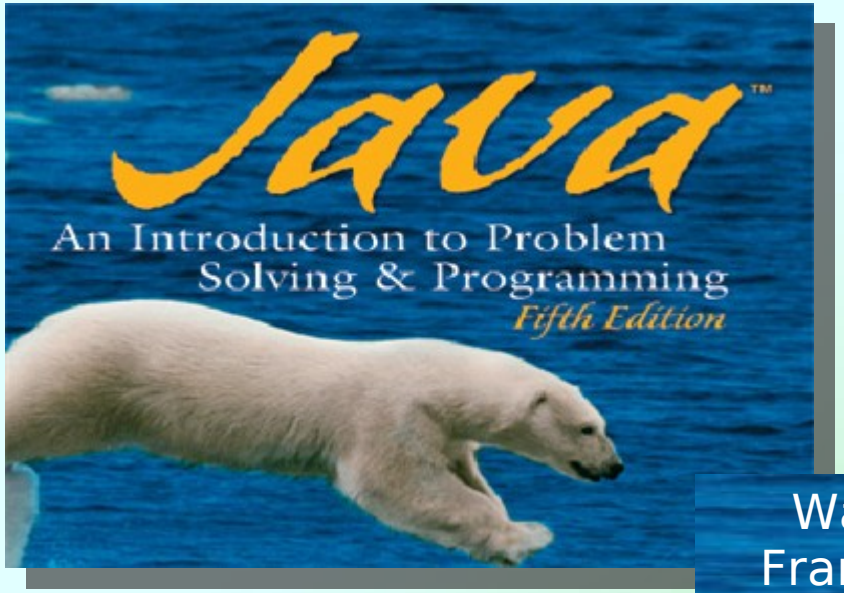



Walter Savitch
Frank M. Carrano

# Introduction to Computers and Java

Chapter 1

*JAVA: An Introduction to Problem Solving & Programming, 5*th Ed. By Walter Savitch and Frank M. Carrano.
ISBN 0136130887 

# Objectives

- Overview computer hardware and software
- Introduce program design and object-oriented programming
- Overview the java programming language

# Computer Basics: Outline

- Hardware and Memory
- Programs
- Programming Languages and Compilers
- Java Byte-Code

# Hardware and Software

- Computer systems consist of *hardware* and *software.*
  - Hardware includes the *tangible* parts of computer systems.
  - Software includes *programs* - sets of instructions for the computer to follow.
- Familiarity with hardware basics helps us understand software.

# Hardware and Memory

- Most modern computers have similar components including
  - Input devices (keyboard, mouse, etc.)
  - Output devices (display screen, printer, etc.)
  - A processor
  - Two kinds of memory (main memory and auxiliary memory).

# The Processor

- Also called the *CPU* (central processing unit) or the *chip* (e.g. Pentium processor)
- The processor **processes** a program's instructions.
- It can process only very simple instructions.
- The power of computing comes from speed and program intricacy.

# Memory

- Memory holds
  - programs
  - data for the computer to process
  - the results of intermediate processing.
- Two kinds of memory
  - main memory
  - auxiliary memory
- Measured in bits, bytes, megabytes, gigabytes...

# Main memory

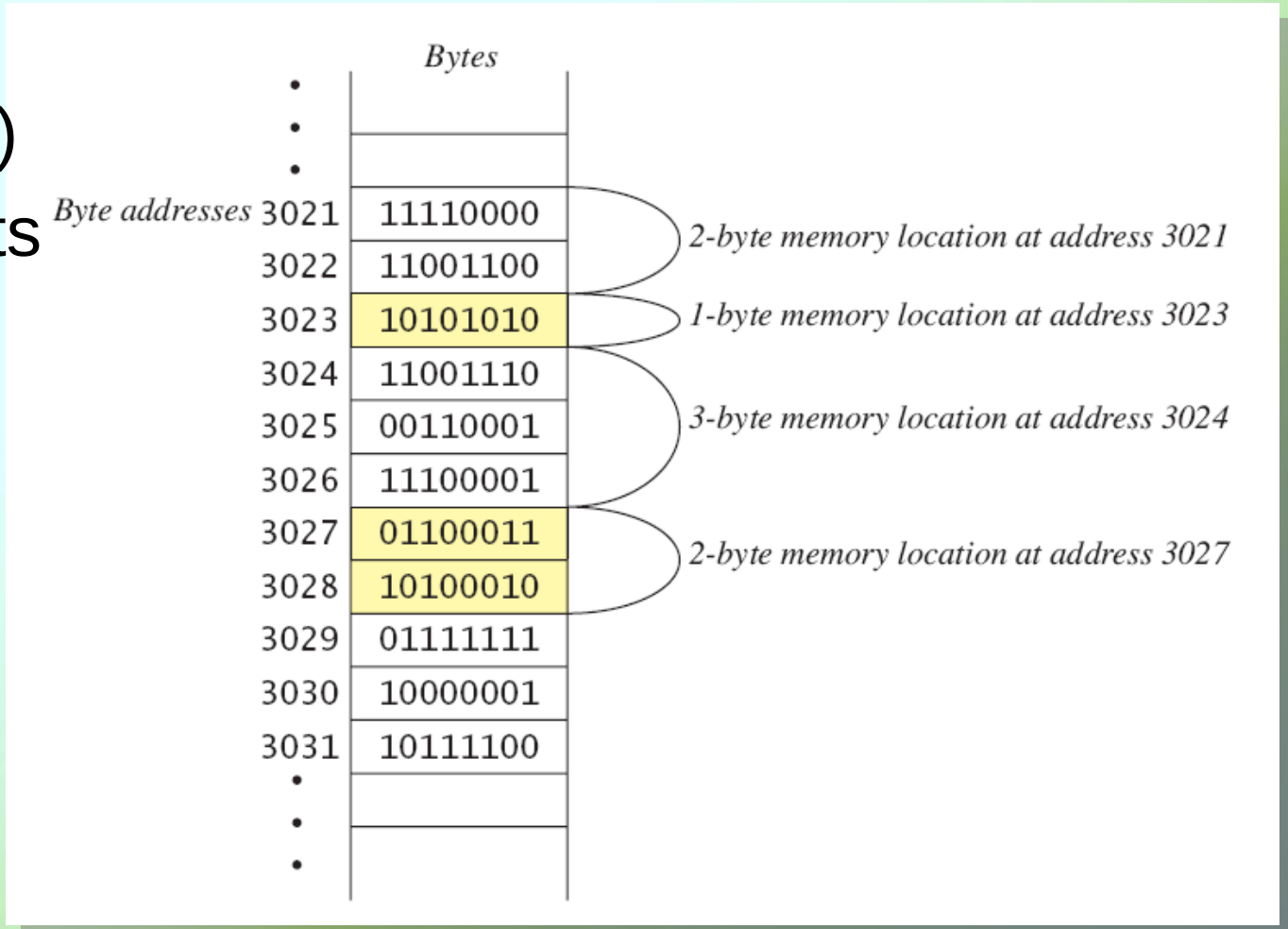
- Used to store
  - The current running program
  - The data the program is using
  - The results of intermediate calculations
- Also called RAM (Random Access Memory)
- Volatile
  - not permanent
  - Is overwritten when a program stops running

# Auxiliary Memory

- Used to store:
  - Program source code
  - Data files
- Disk drives, CDs, DVDs etc.
- Nonvolatile

# Main Memory

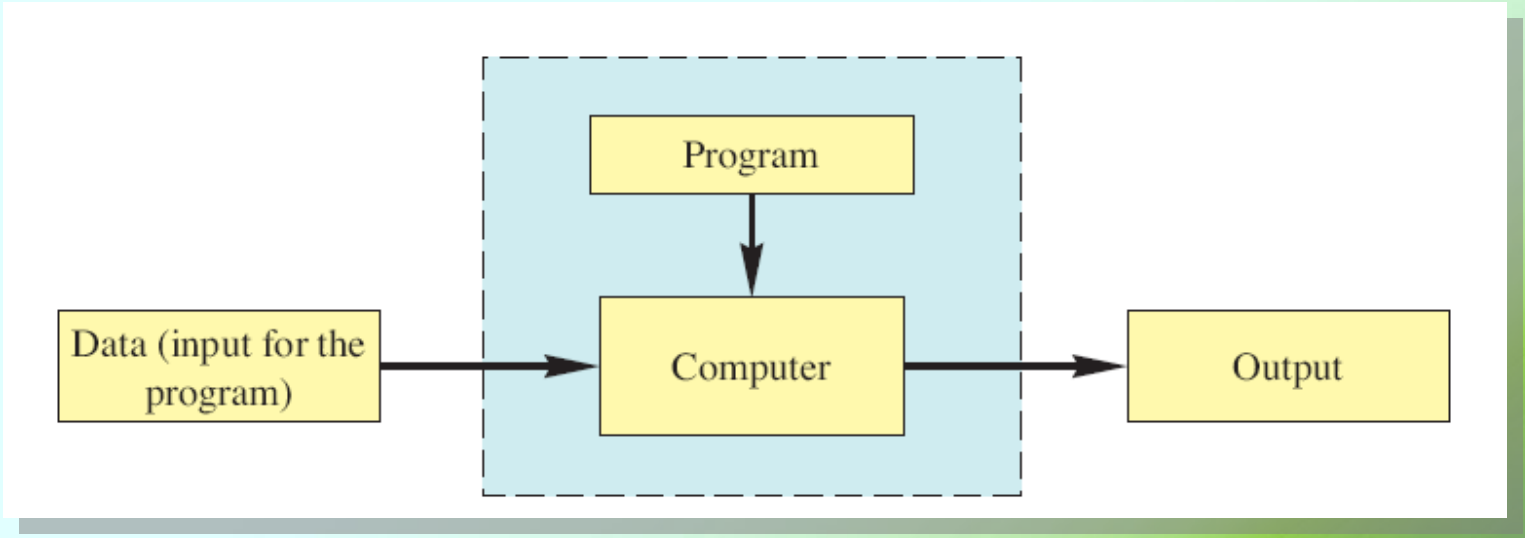
- **Bit** – single digit (0 or 1)
- **Byte** – 8 bits
- Each byte has a location called its **address**



# Files

- Large groups of bytes in auxiliary memory are called *files.*
- Files have names and are organized into groups called *directories* or *folders.*
- Java programs are stored in files.
- Programs files are copied from auxiliary memory to main memory in order to be run.

# Running a Program




- Program files are copied into main memory when a program is run
- The OS (Operating System – Windows, Linux, MAC OS) loads and starts a program

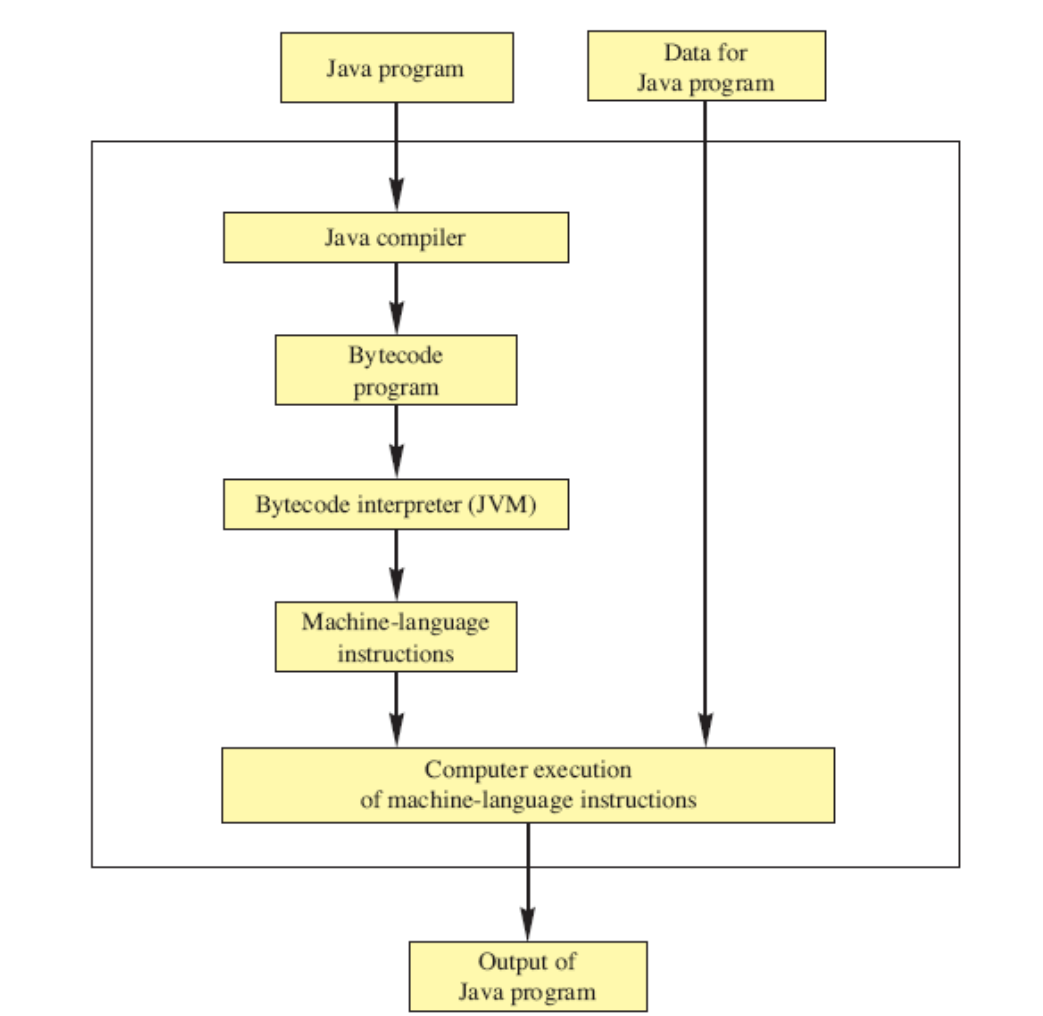
# Programming Languages

- *High-level languages* are relatively easy to use
  - Java, C#, C++, Visual Basic, Python, Ruby.
- Unfortunately, computer hardware does not understand high-level languages.
- A high-level language program must be translated into a *low-level language.*

# Compilers

- A *compiler* translates a program from a high-level language to a low-level language the computer can run.
- You *compile* a program by running the compiler on the high-level-language version of the program called the *source program.*
- Compilers produce *machine-* or *assembly-language* programs called *object programs.*
- The Java compiler produces byte-code, which is interpreted when a program is executed

# Compiling and Running a Program



# Applications and Applets

- Two kinds of java programs: *applications* and *applets*
- Applications
  - Regular programs
  - Meant to be run on your computer
- Applets
  - Little applications
  - Meant to be sent to another location on the internet and run there
- We will be writing applications

# First Application

- Create a directory for this course
  - mkdir java1
  - cd java1
- Create a directory for examples from the book
  - mkdir savitch
  - cd savitch
- Create a subdirectory for chapter1
  - mkdir ch01
  - cd ch01

# First Application

- Click the SavitchSrc link on the course webpage:

**http://www.sfs.uni-tuebingen.de/~fr/teaching/ws10-11/java/**

- Download ch01/FirstProgam.java to java1/savitch/ch01

# First Application

- DrJava is an IDE (integrated development environment)
  - Combines text editor with commands to compile and run java programs
- Start drjava
- Open FirstProgram.java
- Compile and run the program

# Some Terminology

- *programmer:* person who writes programs
- *user:* person who interacts with the program
- *package:* library of classes that have been defined already.
  - `import java.util.Scanner;`
- *argument(s)*: item(s) inside parenthesis
- variable: place to store data
- *statement*: instruction – ends with ;
- *syntax*: grammar rules for a programming language

# Printing to the Screen

```
System.out.println (“Whatever you want to print”);
```

- `System.out` is an object for sending output to the screen.
- `println` is a method to print whatever is in parentheses to the screen.
- The object performs an action when you *invoke* or *call* one of its methods

```
objectName.methodName(argumentsTheMethodNeeds);
```

# Compiling Programs or Classes

- A Java program consists of one or more classes, which must be compiled before running the program.
- Each class should be in a separate file.
- The name of the file should be the same as the name of the class, with the extension **.java**
  - Class FirstProgram is stored in file FirstProgram.java
- The compiler generates a file with the extension **.class** (FirstProgram.class)

# Compiling and Running

- A Java program can involve any number of classes.
- The class to run will contain the words

```
public static void main(String[] args)
```

somewhere in the file

# Object-Oriented Programming

- Our world consists of *objects* (people, trees, cars, cities, airline reservations, etc.).
- Objects can perform *actions* which affect themselves and other objects in the world.
- Object-oriented programming (*OOP*) treats a program as a collection of objects that interact by means of actions.

# OOP Terminology

- Objects, appropriately, are called *objects.*
- Actions are called *methods.*
- Objects of the same kind have the same *type* and belong to the same *class.*
  - Objects within a class have a common set of methods and the same kinds of data
  - but each object can have it's own data values.

# OOP Design Principles

- OOP adheres to three primary design principles:
  - Encapsulation
  - Polymorphism
  - Inheritance

# Introduction to Encapsulation

- The data and methods of class are encapsulated (“put in a capsule”)
- Only part of the capsule is accessible.
  - Details of how the class works are hidden.
  - Encapsulation often is called *information hiding.*

# Accessibility Example

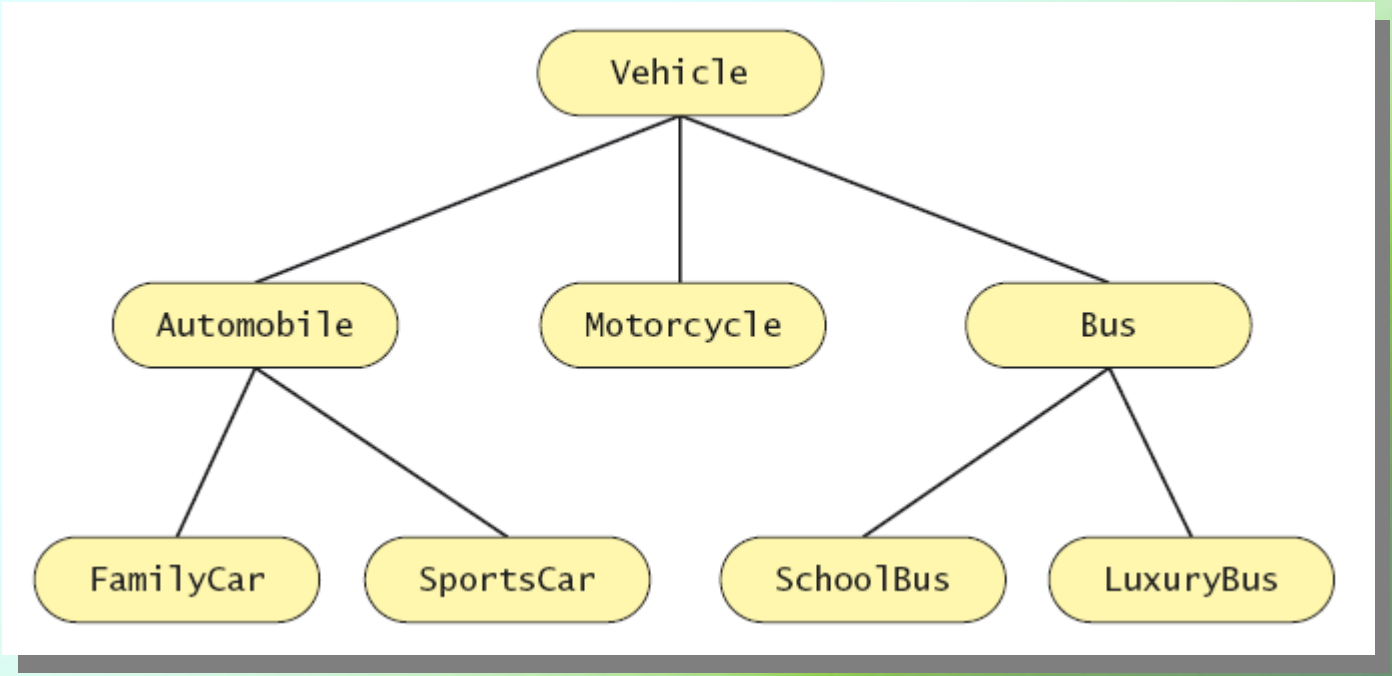
- An automobile consists of several parts and pieces and is capable of doing many useful things.
  - Awareness of the accelerator pedal, the brake pedal, and the steering wheel is important to the driver.
  - Awareness of the fuel injectors, the automatic braking control system, and the power steering pump is not important to the driver.

# Introduction to Polymorphism

- From the Greek meaning “many forms”
- The same program instruction adapts to mean different things in different contexts.
  - A method name, used as an instruction, produces results that depend on the class of the object that used the method.
  - Analogy: “take time to recreate” causes different people to do different activities
- More about polymorphism in a later chapter

# Introduction to Inheritance

- Inheritance is a way of organizing classes




- A class inherits all the characteristics of classes above it in the hierarchy
- At each level, classes become more specialized by adding more characteristics

# Algorithms

- By designing methods, programmers provide actions for objects to perform.
- An *algorithm* describes a means of performing an action.
- Once an algorithm is defined, expressing it in Java (or in another programming language) usually is easy.

# Algorithms

- An algorithm is a set of instructions for solving a problem.
- An algorithm must be expressed completely and precisely.
- Algorithms usually are expressed in English or in *pseudocode.*

# Example: Total Cost of All Items

- Write the number 0 on the whiteboard.
- For each item on the list
  - Add the cost of the item to the number on the whiteboard
  - Replace the number on the whiteboard with the result of this addition.
- Announce that the answer is the number written on the whiteboard.

# Reusable Components

- Most programs are created by combining components that exist already.
- Reusing components saves time and money.
- Reused components are likely to be better developed, and more reliable.
- New components should designed to be reusable by other applications.

# Testing and Debugging

- Eliminate errors by avoiding them in the first place.
  - Carefully design classes, algorithms and methods.
  - Carefully code everything into Java.
- Test your program with appropriate test cases (some where the answer is known), discover and fix any errors, then retest.

# Errors

- An error in a program is called a *bug.*
- Eliminating errors is called *debugging.*
- Three kinds or errors
  - Syntax errors
  - Runtime errors
  - Logic errors

# Syntax Errors

- Grammatical mistakes in a program
  - The grammatical rules for writing a program are very strict
- The compiler catches syntax errors and prints an error message.
- Example: using a period where a program expects a comma

# Runtime Errors

- Errors that are detected when your program is running, but not during compilation
- When the computer detects an error, it terminates the program and prints an error message.
- Example: attempting to divide by 0

# Logic Errors

- Errors that are not detected during compilation or while running, but which cause the program to produce incorrect results
- Example: using subtraction where addition is required

# Software Reuse

- Programs not usually created entirely from scratch
- Most contain components which already exist
- Reusable classes are used
  - Design class objects which are general
  - Java provides many classes
  - Note documentation on following slide

# Software Reuse