Chapter 1

Introduction to Computers and Java Objects

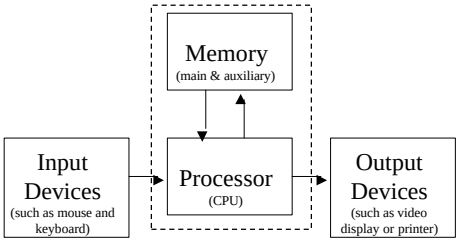
- Background information
  - » important regardless of programming language
- Introduction to Java

Computer Basics

- Computer system: hardware + software
- Hardware: the physical components
- Software: the instructions that tell the hardware what to do

Common Hardware Components

Standard Hardware Organization



- Processor (CPU)
  - » Central Processing Unit
  - » Interprets and executes the instructions
- Memory
  - » main & auxiliary
  - » holds data and instructions
- Input device(s)
  - » mouse, keyboard, etc.
- Output device(s)
  - » video display, printer, etc.
- CPU and memory are physically housed together

Physical Organization

- Keyboard
- Monitor
- Chassis
  - » CPU
  - » memory
  - » disk drives
  - » I/O connectors
  - » etc.

Two Kinds of Memory

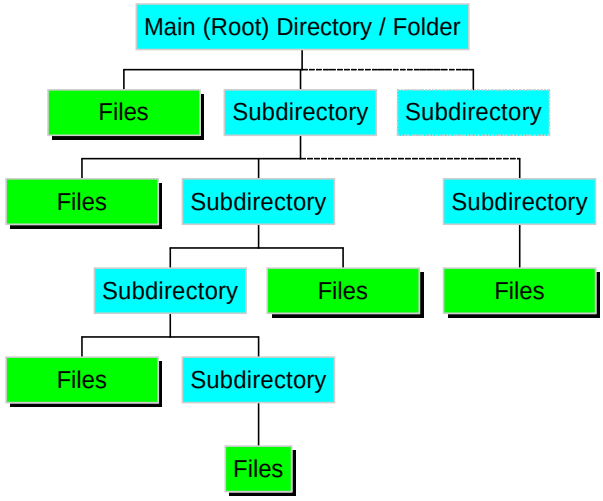
- Main
  - » working area
  - » temporarily stores program and data (while program is executing)
- Auxiliary
  - » permanent (more or less)
  - » saves program and results
  - » includes floppy & hard disk drives, CDs, tape, etc.

Main Memory Organization

- Bit = one binary digit
  - » Binary digit can have only one of two values, 0 or 1
- Byte = 8 bits
- "Byte Addressable"
  - » Main memory is a list of numbered locations that contain one byte of data in each location
- Number of bytes per data item may vary

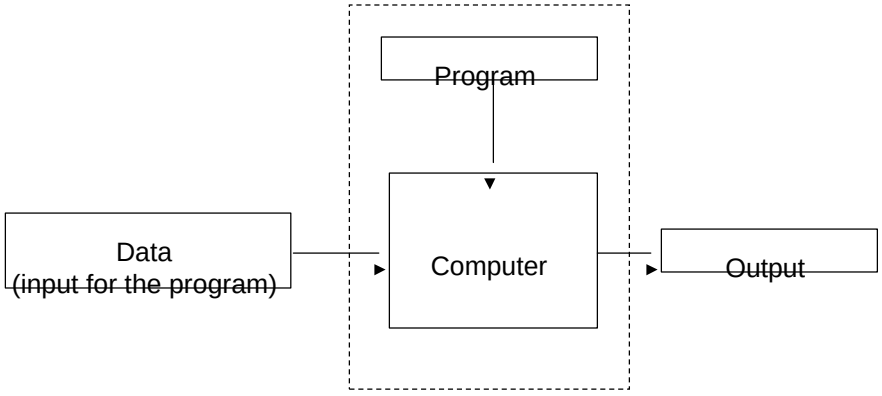
Address	Data Byte	
3021	1111 0000	Item 1: 2 bytes stored
3022	1100 1100	
3023	1010 1010	Item 2: 1 byte stored
3024	1100 1110	Item 3: 3 bytes stored
3025	0011 0001	
3026	1110 0001	
3027	0110 0011	Item 4: 2 bytes stored
3028	1010 0010	
3029	…	Next Item, etc.

Auxiliary Memory Organization



Running a Program

Program—a set of instructions for a computer to follow



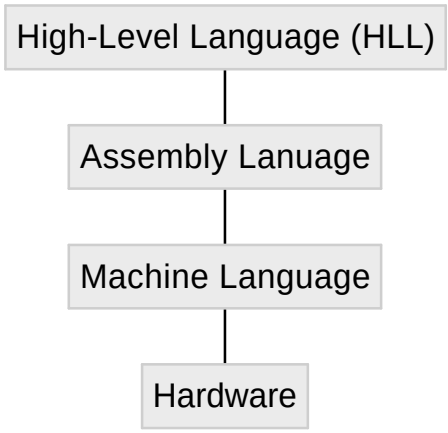
Many Types of Programs

- User-created applications
- Existing applications
  - » word-processor/editor
  - » web browser
  - » compiler or assembler
  - » etc.
- Operating System
  - » DOS, MS- Windows(3.x, 95, 98, NT), MacOS, UNIX, etc.

Various Types of User Interfaces

- Command-line
  - » type in key words and letters
  - » DOS and UNIX
- Menu
  - » parts of DOS and Windows
- GUI (Graphical User Interface)
  - » click on icon
  - » also called “event-driven”
  - » MacOS, Windows

Programming Language Hierarchy



The highs and lows of programming languages ...

- High-Level Language (HLL)
  - » closest to natural language
  - » words, numbers, and math symbols
  - » not directly understood by hardware
  - » “portable” source code (hardware independent)
  - » Java, C, C++, COBOL, FORTRAN, BASIC, Lisp, Ada, etc.
- Machine Language (lowest level)
  - » least natural language for humans, most natural language for hardware
  - » just 0s and 1s
  - » directly understood by hardware
  - » not portable (hardware *de*pendent)

Assembly Language (middle level)

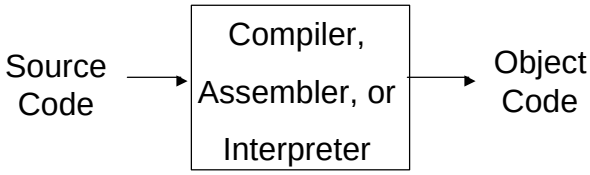
- a more or less human readable version of machine language
- words, abbreviations, letters and numbers replace 0s and 1s
- easily translated from human readable to machine executable code
- like machine code, not portable (hardware *de*pendent)

Getting from Source to Machine Code

- *"Compiling a program"*
  translating from a high-level language source code to machine (object, or executable) code.
- *"Compiler"*
  a program that translates HLL source code to machine (object, or executable) code.
- *"Assembly"*
  translating from assemble language source code to machine (object, or executable) code.
- *"Assembler"*
  a program that translates assembly source code to machine (object, or executable) code.
- Compilers need to know the specific target hardware

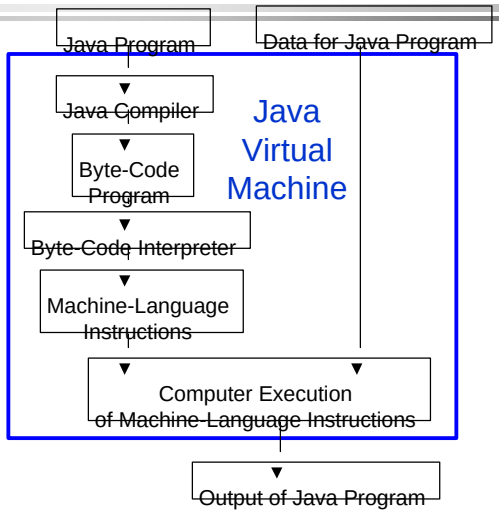
Compilers vs. Assemblers vs. Interpreters

- Compilers and Assemblers
  - » translation is a separate user step
  - » translation is “off-line,” i.e. not at run time
- Interpreters - another way to translate source to object code
  - » interpretation (from source to object code) is not a separate user step
  - » translation is “on-line,” i.e. at run time

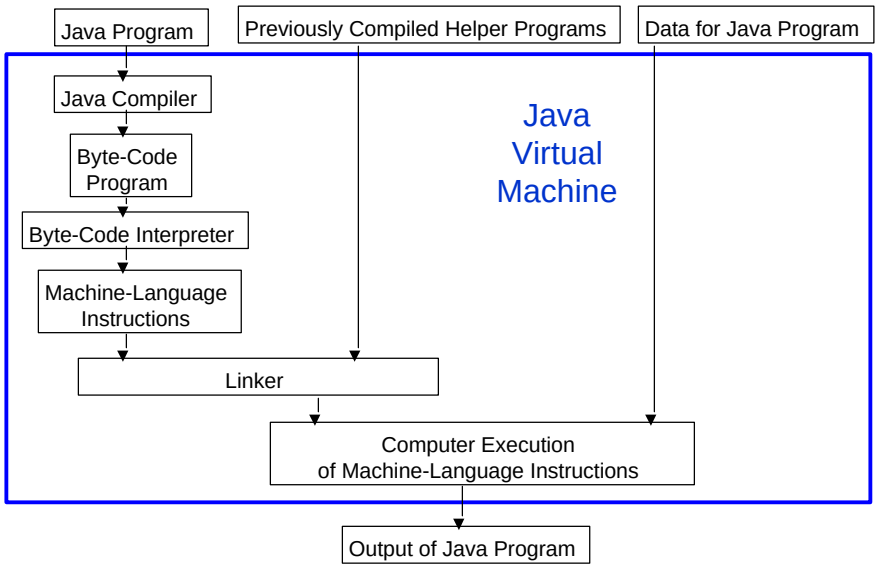


Java Program Translation

- Both Compilation and Interpretation
- Intermediate Code: *“Byte Code”*
  - » portable low-level code
  - » similar to assembly code, but hardware independent
  - » invisible to Java programmer
- Interpreter translates from generic byte code to hardware-specific machine code



Java Program Translation Including Linker



Object-Oriented Programming

- OOP
- A design and programming technique
- Some terminology:
  - » *object* - usually a person, place or thing (a noun)
  - » *method* - an action performed by an object (a verb)
  - » *type* or *class* - a category of similar objects (such as *automobiles)*
- Objects have both data and methods
- Objects of the same class have the same data elements and methods
- Objects send and receive *messages* to invoke actions

Example of an Object Class

Class: automobile

Data Items:

- » manufacturer's name
- » model name
- » year made
- » color
- » number of doors
- » size of engine
- » etc.

Methods:

- » Define data items (specify manufacturer's name, model, year, etc.)
- » Change a data item (color, engine, etc.)
- » Display data items
- » Calculate cost
- » etc.

Why OOP?

- Save development time (and cost) by reusing code
  - » once an object class is created it can be used in other applications
- Easier debugging
  - » classes can be tested independently
  - » reused objects have already been tested

Design Principles of OOP

Three main design principles of Object-Oriented Programming(OOP):

- Encapsulation
- Polymorphism
- Inheritance

Encapsulation

- *Encapsulation* means to design, produce, and describe software so that it can be easily used without knowing the details of how it works.
- Also known as *data hiding*

An analogy:

- When you drive a car, you don't have know the details of how many cylinders the engine has or how the gasoline and air are mixed and ignited.
- Instead you only have to know how to use the controls.

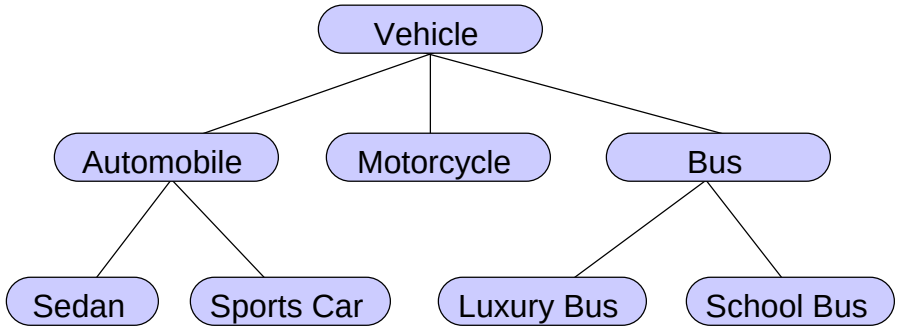
Polymorphism

- *Polymorphism*—the same word or phrase can be mean different things in different contexts
- Analogy: in English, **bank** can mean side of a river or a place to put money
- In Java, two or more classes could each have a method called `output`
- Each `output` method would do the right thing for the class that it was in.
- One `output` might display a number whereas a different one might display a name.

Inheritance

- *Inheritance*—a way of organizing classes
- Term comes from inheritance of traits like eye color, hair color, and so on.
- Classes with properties in common can be grouped so that their common properties are only defined once.

An Inheritance Hierarchy



What properties does each vehicle inherit from the types of vehicles above it in the diagram?

Algorithms

- *Algorithm* - a set of instructions (steps) for solving a problem.
  - » must be precise
  - » must be complete
- May be in a number of different formats
  - » natural language (such as English)
  - » a specific programming language
  - » a diagram, such as a flow chart
  - » pseudocode - a mix of natural and programming language

Example of an Algorithm

Algorithm that determines the total cost of a list of items:

1. Write the number 0 on the blackboard.

2. Do the following for each item on the list:
   Add the cost of the item to the number on the blackboard.
   Replace the old number on the board by this sum.

3. Announce that the answer is the number written on the board

Program Design Process

- Design, then code
- Design process
  - » define the problem clearly
  - » design objects your program needs
  - » develop algorithms for the methods of objects
  - » describe the algorithms, usually in pseudocode
  - » write the code
  - » test the code
  - » fix any errors and retest

Types of Errors

- *Syntax*
- *Run-Time*
- *Logic*

Syntax Errors

- a “grammatical” error
- caught by compiler (“compiler-time error”)
- automatically found, usually the easiest to fix
- cannot run code until all syntax errors are fixed
- error message may be misleading

Example:

Misspelling a command, for example “rturn” instead of “return”

Run-Time Errors

- An execution error (during run-time)
- Not always so easy to fix
- Error message may or may not be helpful

Example:

Division by zero - if your program attempts to divide by zero it automatically terminates and prints an error message.

Logic Errors

Just because it compiles and runs without getting an error message does not mean the code is correct!

- An error in the design (the algorithm) or its implementation
  - » code compiles without errors
  - » no run-time error messages
  - » but incorrect action or data occurs during execution
- Generally the most difficult to find and fix
- Need to be alert and test thoroughly
  - » think about test cases and predict results ***before*** executing the code

Logic Error Examples

- Algorithm Error:
  - » `averageOfFiveScores = SumOfScores/2`
    (should divide by 5)
- Implementation Error:
  - » typed in wrong symbol in source code -
    `sum = a - b;`
    (should be `sum = a + b;`)

Finally! Now, a taste of Java!

History

- 1991 - James Gosling, Sun Microsystems, Inc.Originally
  - » originally a language for programming home appliances
- later (1994) used for World Wide Web applications (since byte code can be downloaded and run without compiling it)
- eventually used as a general-purpose programming language (for the same reason as above plus it is object-oriented)
- Why the name “Java”? Not sure - it may just be a name that came during a coffee break and it had not been copyrighted, yet.

Applets vs. Java Applications

- Applets
  - » Java programs intended to be downloaded via the WWW and run immediately
  - » “little applications”
  - » requires a web browser
- Applications
  - » Java programs intended to be installed then run
  - » often larger applications
- Slightly different programming for each, but both are easy to do

A Sample Java Program

```
public class FirstProgram
{
   public static void main(String[] args)
   {
      System.out.println("Hello out there.");
      System.out.println("Want to talk some more?");
      System.out.println("Answer y for yes or n for no.");
      char answerLetter;
      answerLetter = SavitchIn.readLineNonwhiteChar();
      if (answerLetter == 'y')
         System.out.println("Nice weather we are having.");
      System.out.println("Good-bye.");
      System.out.println("Press enter key to end...");
      String junk;
      junk = SavitchIn.readLine();
    }
}
```

Explanation of Code ...

- Code to begin the program (to be explained later):

```
public class FirstProgram
{
    public static void main(String[ ] args)
    {
```

- Java applications all have similar code at the beginning
  - » The name of the class differs from one program to another.
  - » Other information about the class might also be included on the first line.

Explanation of Code ...

- Code to display a text string:

```
System.out.println("Hello out there.");
System.out.println("Want to talk some more?");
System.out.println("Answer y for yes or n for no.");
```

  - » Note the “dot” operator
  - » `System.out` is an object
  - » `println` is a method that it carries out
  - » double-quoted text inside the parentheses is an *argument* to the method
  - » general syntax: *`Object_Name.Method_Name(Arguments)`*

… Explanation of Code ...

- Code to create a variable named `answerLetter` to contain a single character of data:

```
char answerLetter;
```

- This variable is used to store the user's response.

… Explanation of Code ...

- Read a character typed in from the keyboard and store it in the variable `answerLetter`:

```
answerLetter =
  SavitchIn.readLineNonwhiteChar();
```

  - » `SavitchIn` is a class used for obtaining input from the keyboard
  - » `readLineNonwhiteChar()` is a method that reads a single, non-blank character from the keyboard *and discards any remaining characters on the line.*
  - » the equal sign is *not* the same as in math; it means "assign the value on the right to the variable on the left;" in this case, store the value read from the keyboard into the variable `answerLetter`

… Explanation of Code ...

Question: If "=" means "assign the value of the expression on the right to the variable on the left," how do we indicate "equals"?

Answer: use a double equals ("==")

Example: check to see if the character entered is 'y':

```
if (answerLetter == 'y')
```

» the value inside the parentheses will be True if the letter 'y' was typed in, otherwise it will be False (if any other letter was typed in)

… Explanation of Code ...

- Code to display the line "Nice weather we are having." if the user entered the character 'y':

```
if (answerLetter == 'y')
   System.out.println("Nice weather we are
   having.");
```

» Note that the line will not be printed if any letter other than 'y' is entered.

- Unconditionally display the line "Good-bye.":

```
System.out.println("Good-bye.");
```

» only the previous `System.out.println` is conditionally printed, depending on the value entered; the next instruction is executed regardless of the value entered.

… Explanation of Code

- Code to prevent the display from scrolling off the screen before you can read it:

```
System.out.println("Press enter key to end
  program.");
String junk;
junk = SavitchIn.readLine();
```

  - » `junk` is a variable that can contain a string of characters.
  - » `readLine()` is a method to read in an entire line of text.
  - » The program halts until a character is entered.
  - » Any character entered will make the program continue.
  - » The character entered is assigned to the variable `junk`, but is ignored (it is not used).
  - » There are no more lines of code, so the program terminates.

Syntax Rules for Identifiers

Identifier - the name of something (e.g. a variable, object, or method) used in a Java program.

Identifiers:

- » cannot use reserved words (e.g. “if,” “for”, etc.) (see App. 1)
- » must contain only letters, digits, and the underscore character, _.
- » cannot have a digit for the first character.
  - – $ is allowed but has special meaning, so do not use it.
- » have no official length limit (there is always a finite limit, but it is very large and big enough for reasonable names) .
- » ***are case sensitive!***
  - – junk, JUNK, and Junk are three valid and *different* identifiers, so be sure to be careful in your typing!
- » Note that no spaces or dots are allowed.

Good Programming Practice: Identifier Naming Conventions

- Always use meaningful names, e.g. `finalExamScore`, instead of something like `x`, or even just `score`.
- Use only letters and digits.
- Capitalize interior words in multi-word names, e.g. `answerLetter`.
- Names of classes start with an uppercase letter.
  - » ***every program in Java is a class as well as a program***.
- Names of variables, objects, and methods start with a lowercase letter.

Compiling a Java Program

Assuming the java compiler is already set up and all the files are in the same folder (subdirectory):

- Each class used in a program should be in a separate file
- The name of the file should be the same as the class except with ".java" added to it
- First compile each class definition used in the program
  - » e.g. `SavitchIn` in the sample program (Display 1.4, page 18)
  - » for Sun Microsystems' JDK (Java Development Kit), type `javac SavitchIn.java`
  - » a byte-code file is created with the name `SavitchIn.class`
- Next compile the program file:
  - » `javac <file>.java` (which creates `<file>.class`)

Running a Java Program

- Only the class with `public static void main(String[] args)` can be run
  - » the critical word to look for is `main`
- For Sun Microsystems' JDK (Java Development Kit), type `java <file>`
  - » `<file>` is the same name used in the original source file `<file>.java`
  - » use just `<file>`; do *not* use `<file>.java` or `<file>.class`
- Note that you compile in a separate step and invoke the Java interpreter and linker when you run the program.

Summary
Part 1

- A computer's main memory holds both the program that is currently running and its data.
- Main memory is a series of numbered locations, each one containing a single byte.
- Auxiliary memory is for more or less permanent storage.
- A compiler is a program that translates a high-level language, like java, into a lower level format ("byte-code" for java).
- Actual translation of Java byte-code to the hardware's specific machine code occurs at run time (it is interpreted).

Summary
Part 2

- An algorithm is a set of instructions for solving a problem (it must be complete and precise).
- An object is something that has both data and actions (methods) associated with it.
- A *class* defines a type of object; all objects of the same class have the same methods.
- Three OOP design principles are encapsulation, polymorphism, and inheritance.
- In a java program, a method invocation has the general form `Object_Name.Method_Name(Arguments)`