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

<table>
<thead>
<tr>
<th>Address</th>
<th>Data Byte</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3021</td>
<td>1111 0000</td>
<td>Item 1: 2 bytes stored</td>
</tr>
<tr>
<td>3022</td>
<td>1100 1100</td>
<td>Item 2: 1 byte stored</td>
</tr>
<tr>
<td>3023</td>
<td>1010 1010</td>
<td>Item 3: 3 bytes stored</td>
</tr>
<tr>
<td>3024</td>
<td>1100 1110</td>
<td>Item 3: 3 bytes stored</td>
</tr>
<tr>
<td>3025</td>
<td>0011 0001</td>
<td></td>
</tr>
<tr>
<td>3026</td>
<td>1110 0001</td>
<td></td>
</tr>
<tr>
<td>3027</td>
<td>0110 0011</td>
<td>Item 4: 2 bytes stored</td>
</tr>
<tr>
<td>3028</td>
<td>1010 0010</td>
<td></td>
</tr>
<tr>
<td>3029</td>
<td>…</td>
<td>Next Item, etc.</td>
</tr>
</tbody>
</table>
Auxiliary Memory Organization

Main (Root) Directory / Folder

- Files
- Subdirectory
- Subdirectory
  - Files
  - Subdirectory
    - Files
    - Subdirectory
      - Files
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, Microsoft Windows, MacOS, Linux, 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

High-Level Language (HLL)

Assembly Language

Machine Language

Hardware
The highs and lows of programming languages ...

<table>
<thead>
<tr>
<th>High-Level Language (HLL)</th>
<th>Machine Language (lowest level)</th>
</tr>
</thead>
<tbody>
<tr>
<td>» closest to natural language</td>
<td>» least natural language for humans, most natural language for hardware</td>
</tr>
<tr>
<td>» words, numbers, and math symbols</td>
<td>» just 0s and 1s</td>
</tr>
<tr>
<td>» not directly understood by hardware</td>
<td>» directly understood by hardware</td>
</tr>
<tr>
<td>» “portable” source code (hardware independent)</td>
<td>» not portable (hardware dependent)</td>
</tr>
<tr>
<td>» Java, C, C++, COBOL, FORTRAN, BASIC, Lisp, Ada, etc.</td>
<td></td>
</tr>
</tbody>
</table>
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 dependent)
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
  - similar to assembly code, but hardware independent
- Interpreter translates from generic byte code to hardware-specific machine code
Java Byte Code

- generated by Java compiler
  - Instead of generating machine language as most compilers do, the Java compiler generates byte code.
- easily translated to machine language of various kinds of computers
- executed by Java interpreter
- invisible to programmer
  - You don't have to know anything about how byte code works to write a Java program.
Why Use Byte Code?

Disadvantages:
• requires both compiler and interpreter
• slower program execution

Advantages:
• portability
  » very important
  » same program can run on computers of different types (useful with the Internet)
  » Java interpreter for new types of computers can be made quickly
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 *information 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.
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.
Reusable Components

Advantages of using reusable components:

- saves time and money
- components that have been used before are often better tested and more reliable than new software

Make your classes reusable:

- encapsulation
- general classes have a better chance of being reused than ad hoc classes
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
Testing and Debugging

- Even with careful programming, your code could still contain errors and must be thoroughly tested.

- Bug—a mistake in a program
- Debugging—fixing mistakes in a program
Types of Errors

- Syntax
- Run-Time
- Logic
The set of grammar rules for a programming language is called the syntax.

The compiler checks your program to make sure it is a valid Java program.

If your program is not a valid Java program, then the compiler outputs a message indicating a syntax error.
Syntax Errors

- 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 "rtturn" 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
- Not detected by the compiler.

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

- 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
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();
    }
}

A Sample Java Program
Code to begin the program (to be explained later):

```java
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:

```java
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:

  ```java
  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`:

```java
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`
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’:

```java
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’:

```java
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.”:

```java
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:

```java
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.
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)`