Chapter 1
Introduction
Chapter Scope

• Introduce the Java programming language
• Program compilation and execution
• Problem solving in general
• The software development process
• Overview of object-oriented principles
Java

- A computer is made up of hardware and software
- *hardware* – the physical, tangible pieces that support the computing effort
- *program* – a series of instructions that the hardware executes one after another
- Programs are sometimes called *applications*
- *software* – consists of programs and the data those programs use
Java

- A *programming language* specifies the words and symbols that we can use to write a program.

- A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements*.

- The Java programming language was created by Sun Microsystems, Inc.

- It was introduced in 1995 and its popularity grew quickly.
Java

• 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
public class Lincoln {
    // Prints a presidential quote.
    public static void main(String[] args) {
        System.out.println("A quote by Abraham Lincoln:");
        System.out.println("Whatever you are, be a good one.");
    }
}
A Java Program

```java
public class MyProgram {
    // class body
}
```

Comments can be placed almost anywhere
public class MyProgram {

    public static void main(String[] args) {

    }

}
Comments

• Comments should be included to explain the purpose of the program and describe processing

• They do not affect how a program works

• Java comments can take three forms:

  // this comment runs to the end of the line

  /*  this comment runs to the terminating symbol, even across line breaks        */

  /** this is a javadoc comment   */
Identifiers

• *Identifiers* are the words a programmer uses in a program
  – can be made up of letters, digits, the underscore character ( _ ), and the dollar sign
  – cannot begin with a digit

• Java is *case sensitive*
  – Total, total, and TOTAL are different identifiers

• By convention, programmers use different case styles for different types of identifiers, such as
  – *title case* for class names - *Lincoln*
  – *upper case* for constants - *MAXIMUM*
Identifiers

• Sometimes we choose identifiers ourselves when writing a program (such as \texttt{Lincoln})

• Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as \texttt{println})

• Often we use special identifiers called \textit{reserved words} that already have a predefined meaning in the language

• A reserved word cannot be used in any other way
Reserved Words

- Java reserved words:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>abstract</td>
<td>default</td>
<td>goto*</td>
<td>package</td>
<td>this</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>assert</td>
<td>do</td>
<td>if</td>
<td>private</td>
<td>throw</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>boolean</td>
<td>double</td>
<td>import</td>
<td>protected</td>
<td>throws</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>break</td>
<td>else</td>
<td>instanceof</td>
<td>public</td>
<td>transient</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>byte</td>
<td>enum</td>
<td>int</td>
<td>return</td>
<td>true</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>case</td>
<td>extends</td>
<td>interface</td>
<td>short</td>
<td>try</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>catch</td>
<td>false</td>
<td>long</td>
<td>static</td>
<td>void</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>final</td>
<td>native</td>
<td>strictfp</td>
<td>volatile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>finally</td>
<td>new</td>
<td>super</td>
<td>while</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>const*</td>
<td>float</td>
<td>null</td>
<td>synchronized</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>for</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

White Space

• Spaces, blank lines, and tabs are called *white space*
• White space is used to separate words and symbols in a program
• Extra white space is ignored
• A valid Java program can be formatted many ways
• Programs should be formatted to enhance readability, using consistent indentation
public class Lincoln2{public static void main(String[] args){
    System.out.println("A quote by Abraham Lincoln:");
    System.out.println("Whatever you are, be a good one.");}}
public class Lincoln3 {
    public static void main(String[] args) {
        System.out.println("A quote by Abraham Lincoln:");
        System.out.println("Whatever you are, be a good one.");
    }
}
Program Development

• The mechanics of developing a program include several activities
  
  – writing the program in a specific programming language (such as Java)
  
  – translating the program into a form that the computer can execute
  
  – investigating and fixing various types of errors that can occur

• Software tools can be used to help with all parts of this process
Language Levels

• There are four programming language levels
  – machine language
  – assembly language
  – high-level language
  – fourth-generation language

• Each type of CPU has its own specific *machine language*

• The other levels were created to make it easier for a human being to read and write programs
Language Levels

- A high-level expression and its lower level equivalents:

<table>
<thead>
<tr>
<th>High-Level Language</th>
<th>Assembly Language</th>
<th>Machine Language</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;a + b&gt;</code></td>
<td><code>1d [%fp-20], %o0</code></td>
<td><code>...</code></td>
</tr>
<tr>
<td></td>
<td><code>1d [%fp-24], %o1</code></td>
<td><code>1101 0000 0000 0111</code></td>
</tr>
<tr>
<td></td>
<td><code>add %o0, %o1, %o0</code></td>
<td><code>1011 1111 1110 1000</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>1101 0010 0000 0111</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>1011 1111 1110 1000</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>1001 0000 0000 0000</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>...</code></td>
</tr>
</tbody>
</table>
Compilation

• Each type of CPU executes only a particular *machine language*

• A program must be translated into machine language before it can be executed

• *A compiler* is a software tool which translates *source code* into a specific target language

• Often, that target language is the machine language for a particular CPU type

• The Java approach is somewhat different
Basic Programming Steps

• A program is written in an editor, compiled into an executable form, and then executed
• If errors occur during compilation, an executable version is not created
Java Translation

• The Java compiler translates Java source code into a special representation called *bytecode*.

• Java bytecode is not the machine language for any traditional CPU.

• Another software tool, called an *interpreter*, translates bytecode into machine language and executes it.

• Therefore the Java compiler is not tied to any particular machine.

• Java is considered to be *architecture-neutral*.
Java Translation
Development Environments

• *A development environment* is the set of tools used to create, test, and modify a program

• *An integrated development environment* (IDE) combine the tools into one software program

• All development environments contain key tools, such as a compiler and interpreter

• Others include additional tools, such as a debugger, which helps you find errors
Development Environments

• There are many environments that support the development of Java software, including:
  – Sun Java Development Kit (JDK)
  – Eclipse
  – NetBeans
  – BlueJ
  – jGRASP

• Though the details of these environments differ, the basic compilation and execution process is essentially the same
Syntax and Semantics

• The *syntax rules* of a language define how we can put together symbols, reserved words, and identifiers to make a valid program.

• The *semantics* of a program statement define what that statement means (its purpose or role in a program).

• A program that is syntactically correct is not necessarily logically (semantically) correct.

• A program will always do what we tell it to do, not what we *meant* to tell it to do.
Errors

• A program can have three types of errors:
  
  – The compiler will find syntax errors and other basic problems (*compile-time errors*)
  
  – A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (*run-time errors*)
  
  – A program may run, but produce incorrect results, perhaps using an incorrect formula (*logical errors*)
Problem Solving

• The purpose of writing a program is to solve a problem

• Solving a problem consists of multiple activities
  – understand the problem
  – design a solution
  – consider alternatives and refine the solution
  – implement the solution
  – test the solution

• These activities are not purely linear – they overlap and interact
Problem Solving

• The key to designing a solution is breaking it down into manageable pieces

• When writing software, we design separate pieces that are responsible for certain parts of the solution

• An object-oriented approach lends itself to this kind of solution decomposition

• We will dissect our solutions into pieces called objects and classes
Development Activities

• Any proper software development effort consists of four basic *development activities*
  – establishing the requirements
  – creating a design
  – implementing the design
  – testing

• These steps also are never purely linear and often overlap and interact
Development Activities

• *Software requirements* specify *what* a program must accomplish
• Requirements are expressed in a document called a *functional specification*
• A *software design* indicates how a program will accomplish its requirements
• *Implementation* is the process of writing the source code that will solve the problem
• *Testing* is the act of ensuring that a program will solve the intended problem given all of the constraints under which it must perform
Object-Oriented Programming

• Java is an *object-oriented* programming language

• As the term implies, an object is a fundamental entity in a Java program

• Objects can be used effectively to represent real-world entities

• For instance, an object might represent a particular employee in a company

• Each employee object handles the processing and data management related to that employee
Objects

• An object has
  – \textit{state} - descriptive characteristics
  – \textit{behaviors} - what it can do (or what can be done to it)

• The state of a bank account includes its account number and its current balance

• The behaviors associated with a bank account include the ability to make deposits and withdrawals

• Note that the behavior of an object might change its state
Classes

• An object is defined by a *class*

• A class is the blueprint of an object

• The class uses methods to define the behaviors of the object

• The class that contains the main method of a Java program represents the entire program

• A class represents a concept, and an object represents the embodiment of that concept

• Multiple objects can be created from the same class
Classes and Objects

• A class is like a blueprint from which you can create many of the "same" house with different characteristics
Classes and Objects

• An object is *encapsulated*, protecting the data it manages

• One class can be used to derive another via *inheritance*

• Classes can be organized into hierarchies
Classes and Objects

A class defines a concept

Bank Account

Multiple encapsulated objects can be created from one class

John's Bank Account
Balance: $5,257

Jason's Bank Account
Balance: $1,245,069

Mary's Bank Account
Balance: $16,833

Classes can be organized into inheritance hierarchies

Account

Charge Account

Bank Account

Savings Account

Checking Account