COIN 325: Java I and Elementary Data Structures

Dr. Paul E. West

Department of Computer Science
Charleston Southern University

January 16, 2015

CHARLESTON SOUTHERN UNIVERSITY

About the Professor

- PhD from Florida State University in Computer Science
- Faculty Experience:
  - College of Charleston: Adjunct 2013-2014
- Work Experience:
  - Google (2014): Android Bluetooth/Wi-Fi/Telephony
  - SPAWAR (2009-2014): Communication systems
  - DenimGroup (2004-2005): Start-up; web design and network security

Syllabus

Lets go over the syllabus...

Overview

- We begin by examining a very basic Java program and using it to explore some initial programming concepts
- Chapter 1 focuses on
  - Introducing the Java programming language
  - Describing the steps involved in program compilation and execution
  - Exploring the issues related to problem solving in general
  - Discussing the activities involved in the software development process
  - Presenting an overview of object-oriented principles

Java Programming

- 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 Programming

- 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 `http://www.tiobe.com/`

Java Programming

- 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

Example Java Program

```java
//**********************************************************************
//  Lincoln.java       Java Foundations
//
//  Demonstrates the basic structure of a Java application.
//**********************************************************************

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.");
   }
}
```

Structure

```
// comments about the class

public class MyProgram { // <-- class heading

    // class body

}

/*
 * Note that comments can be place almost anywhere and you
 * may have single line and multi-line comments like in C.
 */
```

Structure

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

Comments

- Comments in a program are called inline documentation
- They should be included to explain the purpose of the program and describe processing steps
- 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 Lincoln)
- Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as println)
- Often we use special identifiers called reserved words that already have a predefined meaning in the language
- A reserved word cannot be used in any other way

Reserved Words

- The Java reserved words

abstract
assert
boolean
break
byte
case
catch
char
class
const
continue
default
do
double

else
enum
extends
false
final
finally
float
for
goto
if
implements
import
instanceof
int

interface
long
native
new
null
package
private
protected
public
return
short
static
strictfp
super

switch
synchronized
this
throw
throws
transient
true
try
void
volatile
while

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

Readability

```
//********************************************************************
//  Lincoln2.java       Java Foundations
//
//  Demonstrates a poorly formatted, though valid, program.
//********************************************************************

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.");}}
```

Readability

```
//********************************************************************
//  Lincoln3.java       Java Foundations
//
//  Demonstrates another valid program that is poorly formatted.
//********************************************************************

          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."
      )
  ;
}
          }
```

- 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 main programming language levels
  1. machine language
  2. assembly language
  3. high-level language
  4. 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

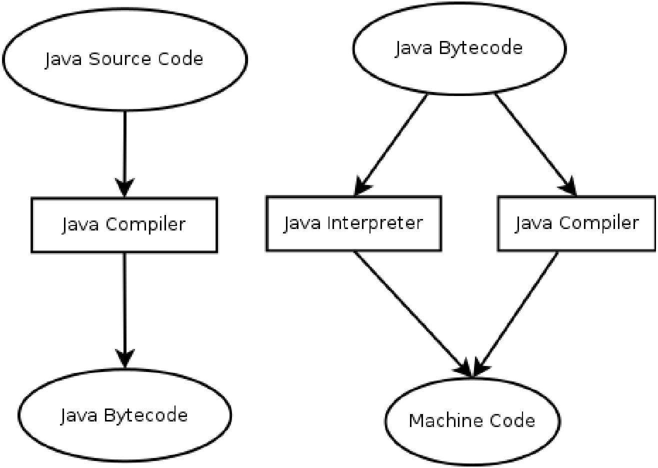
Programming Languages

- 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

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

- There are many programs that support the development of Java software, including
  - Sun Java Development Kit (JDK)
  - Eclipse
  - NetBeans
  - BlueJ
  - Your favorite text editor (notepad, notepad++, vi, emacs, nano, etc...)
- Though the details of these environments differ, the basic compilation and execution process is essentially the same

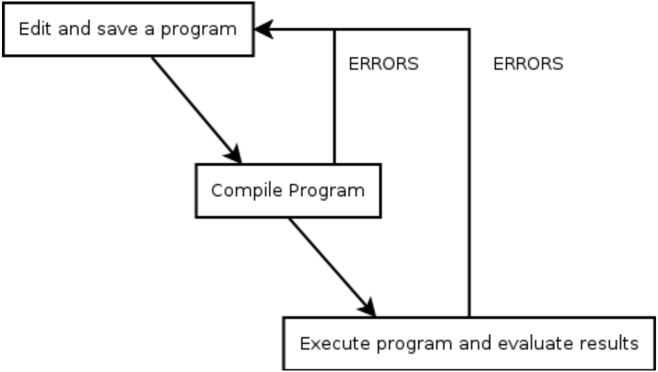
Syntax and Semantics

- 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)
    - If compile-time errors exist, an executable version of the program is not created
  - 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)

Basic Program Development



- When is it easiest to catch an error?
- Software Engineering/process.

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 ← this is VITALLY important!!!
- 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

Software Development Activities

- Any proper software development effort consists of (at least) 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
- Examples:
  - the test cases *should* be considered during design
  - the quality of the code (which determines how well we test) *should* be considered during requirements

Software 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

OOP

- Java is an object-oriented programming (OOP) 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

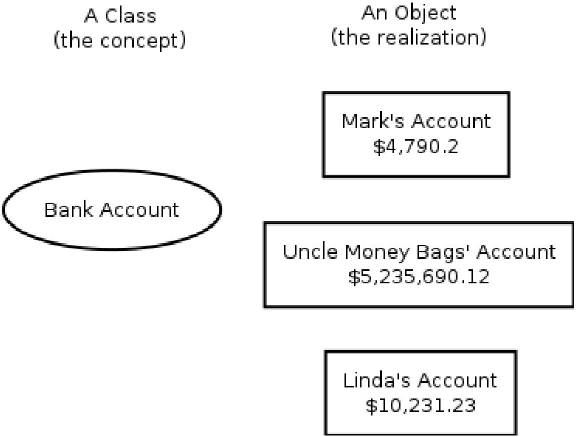
Objects

- An object has
  - *state* - descriptive characteristics
  - *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
- Also Note: for those coming from C/C++, basically Java allows you to group your functions (behaviors) with a struct (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 (instantiated) from the same class

Objects and Classes



A Class
(the concept)

An Object
(the realization)

Bank Account

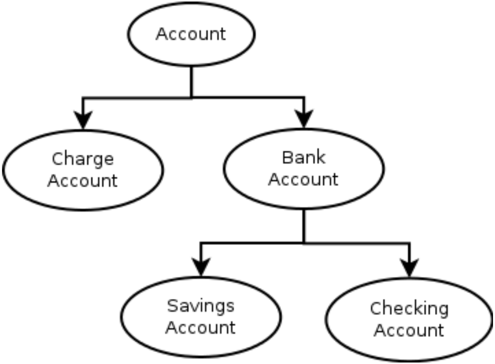
Mark's Account
$4,790.2

Uncle Money Bags' Account
$5,235,690.12

Linda's Account
$10,231.23

Inheritance

- One class can be used to derive another via inheritance
- Classes can be organized into hierarchies



Why would we do this?

- Login into your machine
- Write and TEST a simple Hello World application
- Submit via Blackboard
- This is to test building a program on your machine and submitting via blackboard.
- This is not graded.

Possible Future Working tools

- git
- make
- Android Development Tools