Java Language and SW Dev’t

- Programming Languages
- Java Program Structure
- Problem Solving
- Object-Oriented Programming
- Reading for this class: L&L, 1.4-1.6

Programming Languages

- Computer programmers write programs for computers using one or more programming languages
- Some languages are better for one type of program or one style of user interface than for others
- You may have heard of some programming languages: COBOL, Basic, Pascal, C/C++, Java, Assembly Language, and Others

Programming Languages

- 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*
- A programming language has both *syntax* and *semantics*

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 <u>meant</u> to tell it to do

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

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 type of CPU
- The Java approach is somewhat different

Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode* in the .class file
- Java bytecode is not the machine language for any specific CPU
- Another software tool, called an *interpreter (in our case the Java Virtual Machine)*, executes the bytecode
- Java is considered to be *architecture-neutral*
- The Java compiler is not tied to any particular machine
- The JVM can be implemented on any machine

Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains zero or more *attributes*
  - 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 starts with a class containing a method called `main`
- See `Lincoln.java` (page 29)

Java Program Structure

```
//  comments about the class
public class MyProgram
{
```

class header

class body

Comments can be placed almost anywhere

```
}
```

Java Program Structure

```
//  comments about the class
public class MyProgram
{
   //  comments about the attributes

                                       attribute definitions


   //  comments about the 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
- An identifier can be made up of letters, digits, the underscore character (_), and the dollar sign
- Identifiers 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`
  - *lower* case for object or other variable names – `current`
  - *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    else         int          strictfp
boolean     enum         interface    super
break       extends      long         switch
byte        false        native       synchronized
case        final        new          this
catch       finally      null         throw
char        float        package      throws
class       for          private      transient
const       goto         protected    true
continue    if           public       try
default     implements   return       void
do          import       short        volatile
double      instanceof   static       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
- See `Lincoln2.java` (page 34)
- See `Lincoln3.java` (page 35)

"Always code as if the person who ends up maintaining your code will be a violent psychopath who knows where you live."

-- Martin Golding

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

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 bank account
- Each bank account object handles the processing and data management related to that bank account

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 balance
- The behaviors associated with a bank account include the ability to get the balance, make deposits, and make withdrawals
- Note that the behavior of an object might change its state, e.g. making a deposit will increase the balance

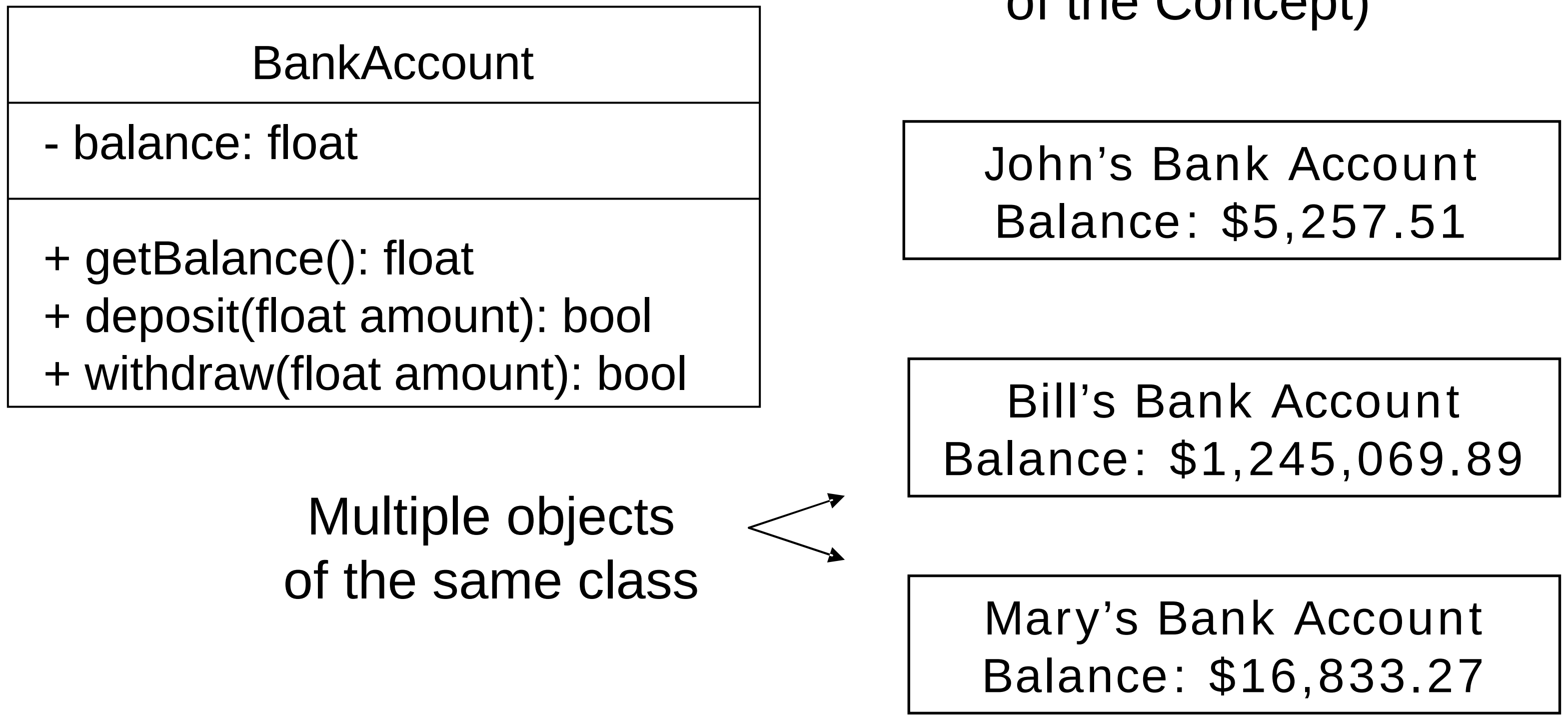
Classes

- An object is defined by a *class* representing a *concept*
- A class is the blueprint for each *instance* of an object
- Multiple objects can be created from the same class
- A class has *attributes* that define the state of each object
- A class has *methods* that define the behavior of the object
- The class that contains the `main` method represents the starting point for a Java program
- The program can and usually does contain more classes than just the one that contains the `main` method

Objects and Classes

A Class
(The Concept)

Three objects
(Three Instances
of the Concept)



Multiple objects
of the same class

Java Program Structure

```
public class BankAccount
{
   private float balance;      attribute definition

   public float getBalance()
   {
              method body
   }
   public boolean deposit(float amount)
   {
              method body
   }
   public boolean withdraw(float amount)
   {
              method body
   }
}
```