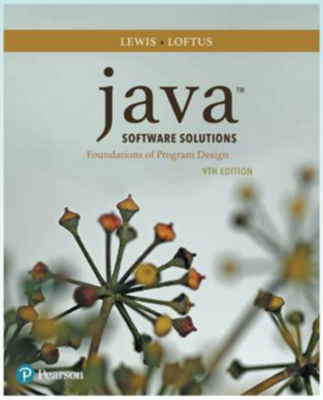
Chapter 1
Introduction

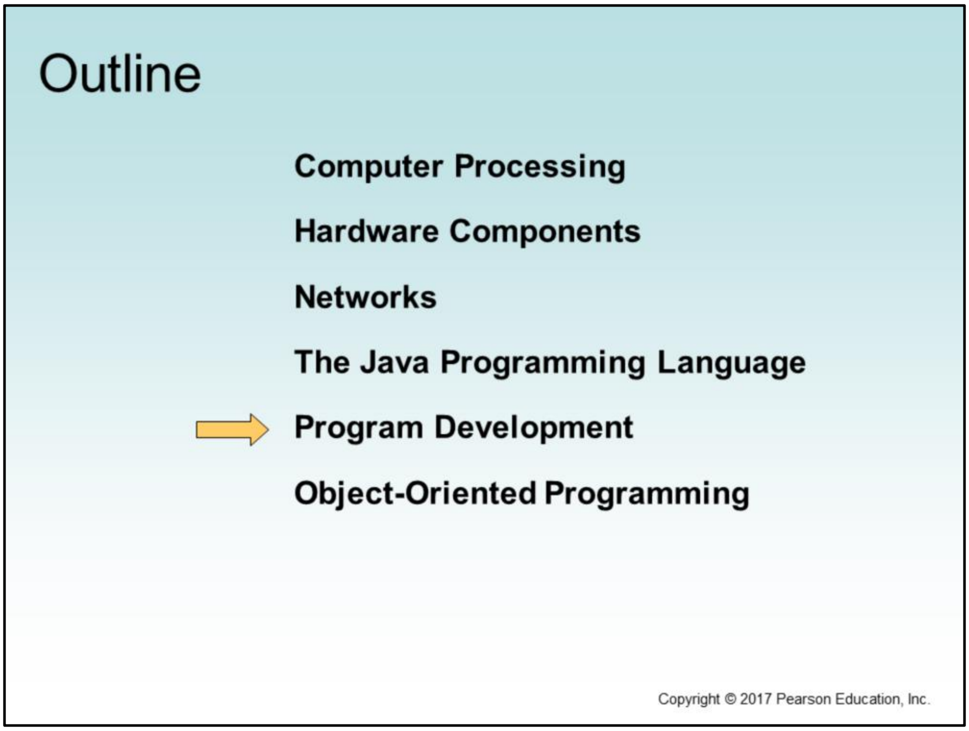


Java Software Solutions

Foundations of Program Design

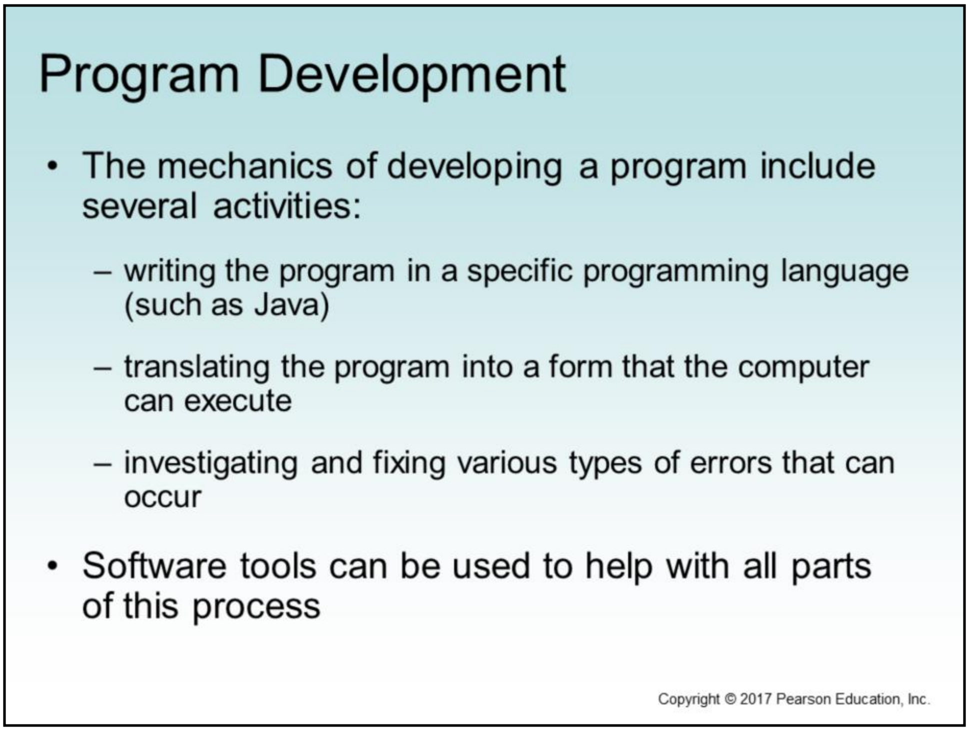
9th Edition

John Lewis
William Loftus



Outline

- Computer Processing
- Hardware Components
- Networks
- The Java Programming Language
- ➡ Program Development
- Object-Oriented Programming



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

-Problems can be solved using a computer in different languages

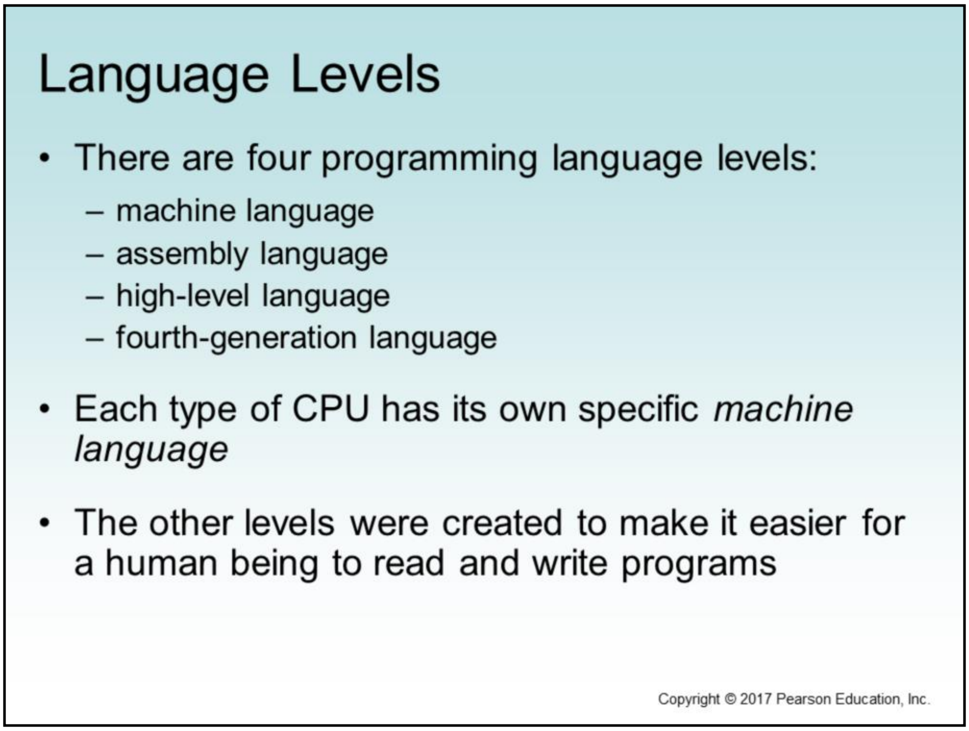
-Regardless of the language; the goal is to **solve problems**

-One of our goals is to learn how to instruct a machine to solve problems

-Involved in this goal is how to organize (design) the instructions in our program

-This **design** process involves thinking about problems in an **object-oriented** manner

-Remember, learning Java is secondary – goal is designing software to solve problems!



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

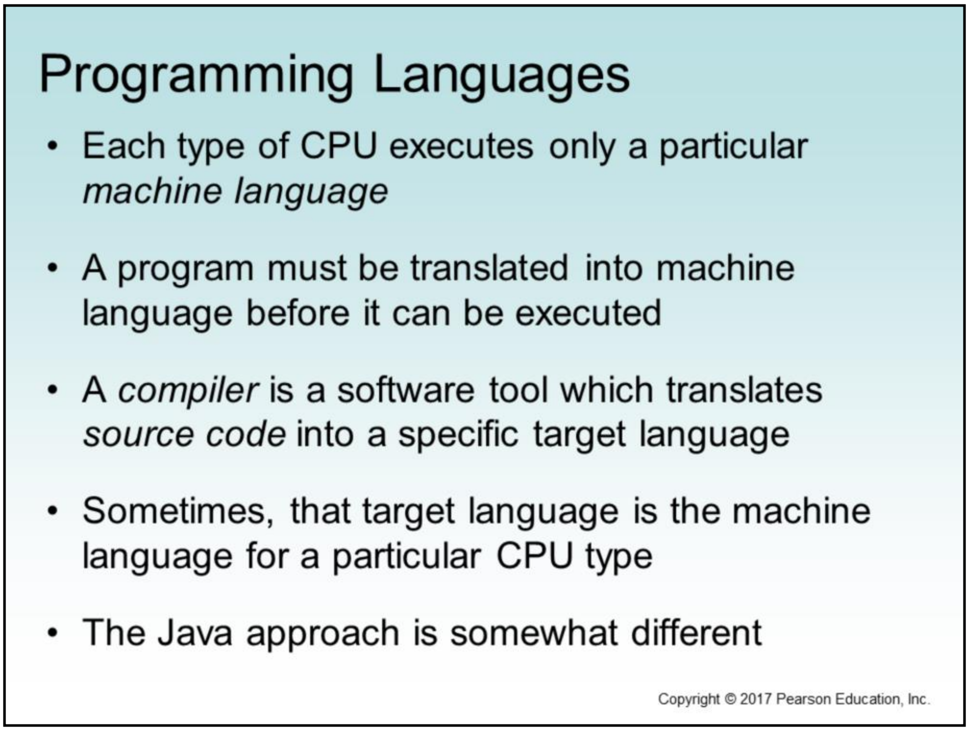
-Programs in a machine language for one type of CPU won't execute on a different CPU

-We'll see that Java technology solves this problem using a **Virtual Machine**

-Assembly language uses short words to represent instructions

-High-level languages can represent more instructions (note. Fig. 1.19)

-4th generation examples: SQL, Mathematica



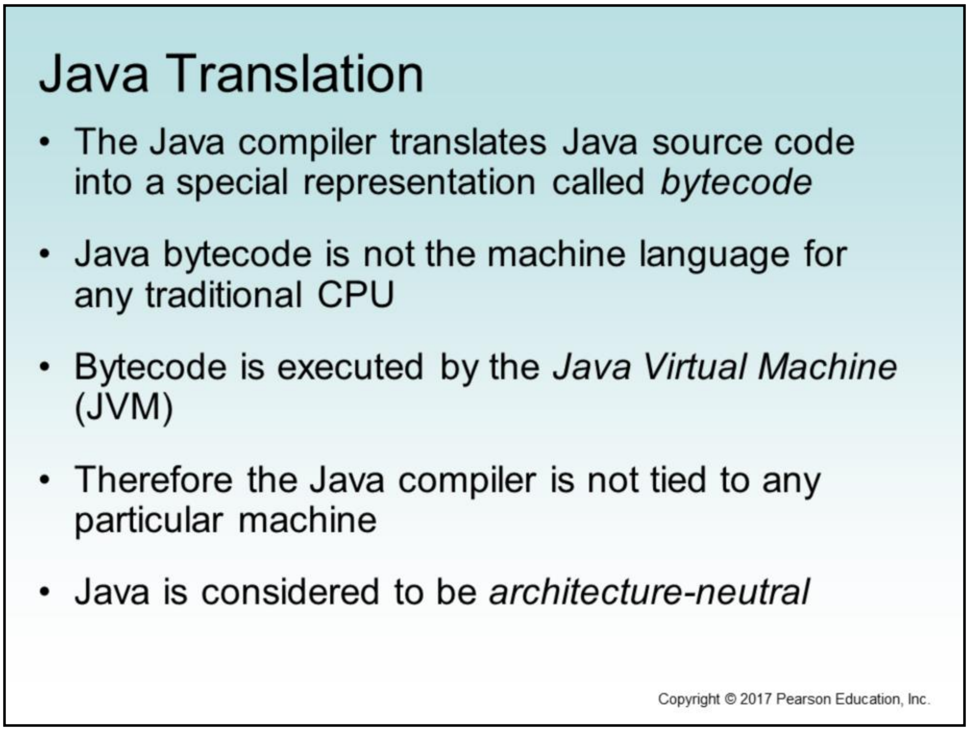
Programming Languages

- 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
- Sometimes, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

-**Source code**: Language instructions originating (i.e. “source”) from programmer

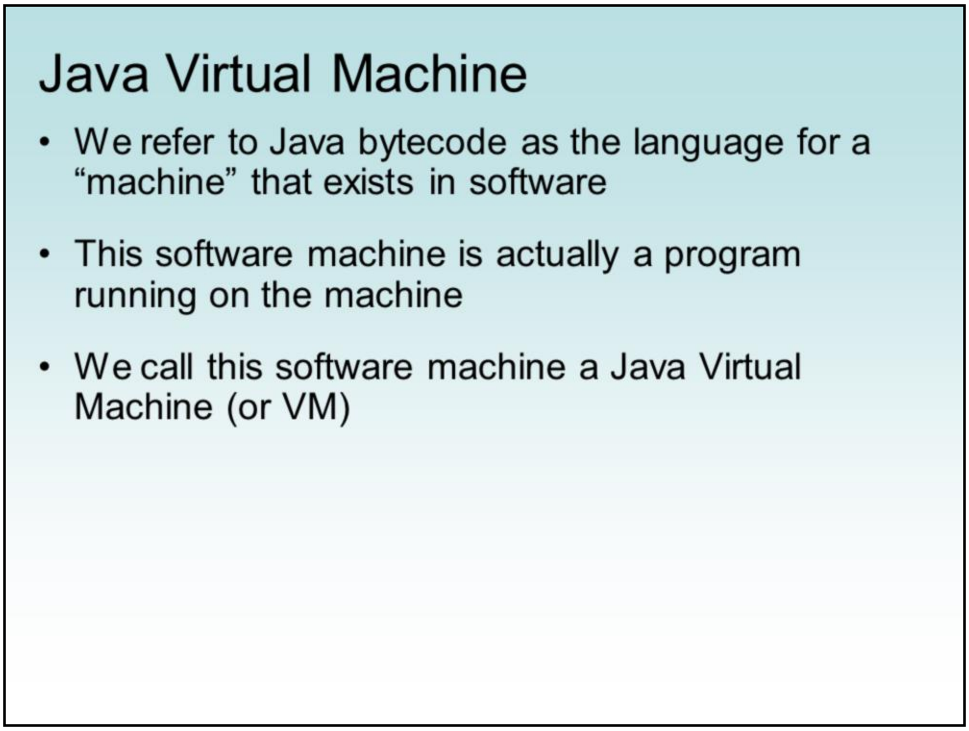
-**Compilers** translate the **entire** program into machine language

-**Interpreters** (typically) translate **one statement** into machine language, execute it, then repeat



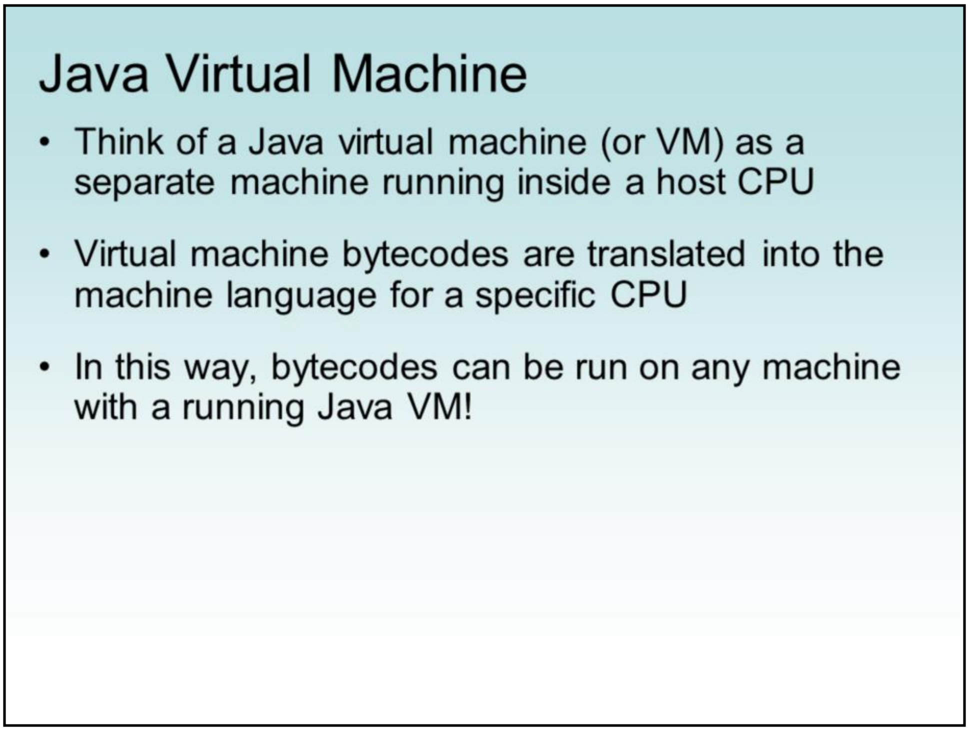
Java Translation

- The Java compiler translates Java source code into a special representation called *bytecode*
- Java bytecode is not the machine language for any traditional CPU
- Bytecode is executed by the *Java Virtual Machine* (JVM)
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be *architecture-neutral*



Java Virtual Machine

- We refer to Java bytecode as the language for a "machine" that exists in software
- This software machine is actually a program running on the machine
- We call this software machine a Java Virtual Machine (or VM)

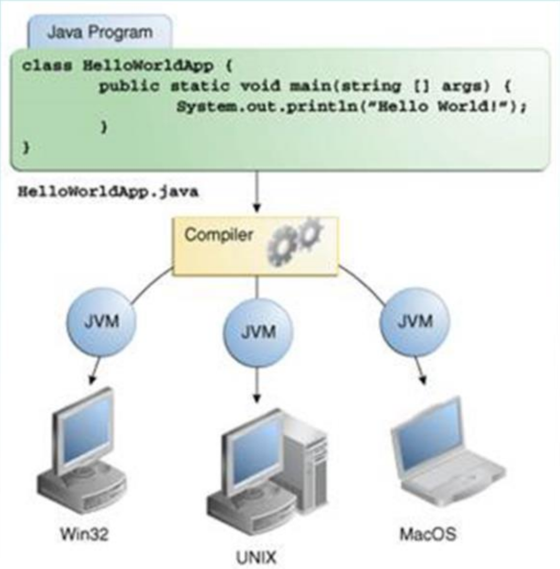


Java Virtual Machine

- Think of a Java virtual machine (or VM) as a separate machine running inside a host CPU
- Virtual machine bytecodes are translated into the machine language for a specific CPU
- In this way, bytecodes can be run on any machine with a running Java VM!

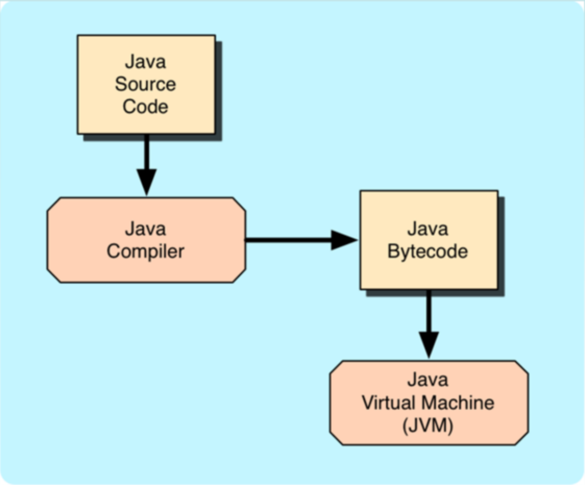
Java Virtual Machine

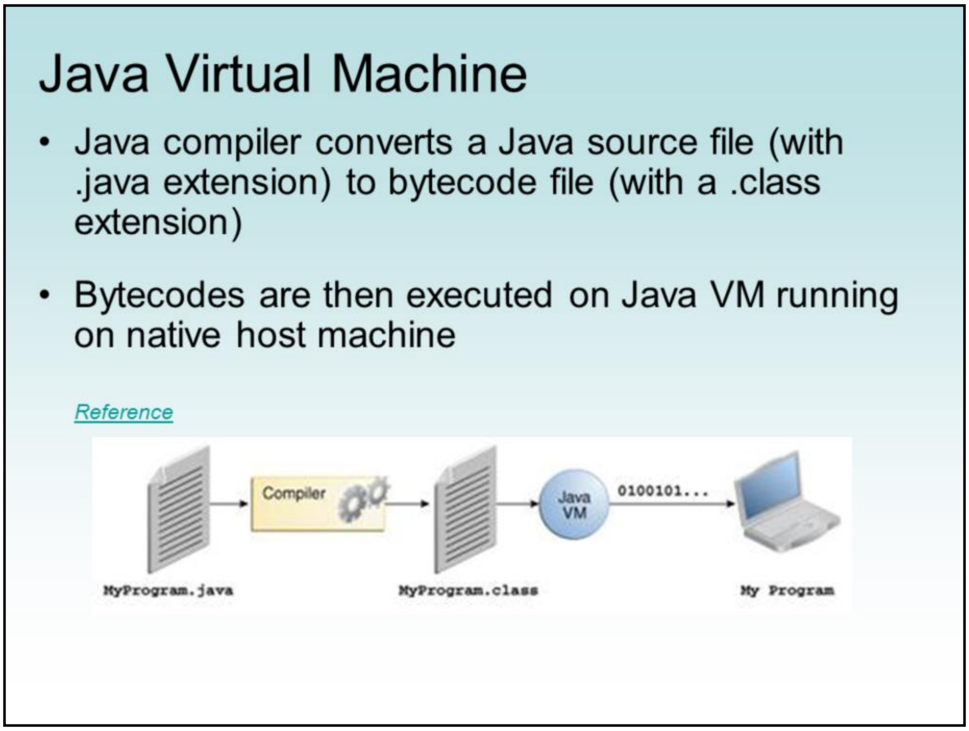
Reference



```
class HelloWorldApp {
        public static void main(string [] args) {
                System.out.println(“Hello World!”);
        }
}
```

Java Translation





Java Virtual Machine

- Java compiler converts a Java source file (with .java extension) to bytecode file (with a .class extension)
- Bytecodes are then executed on Java VM running on native host machine

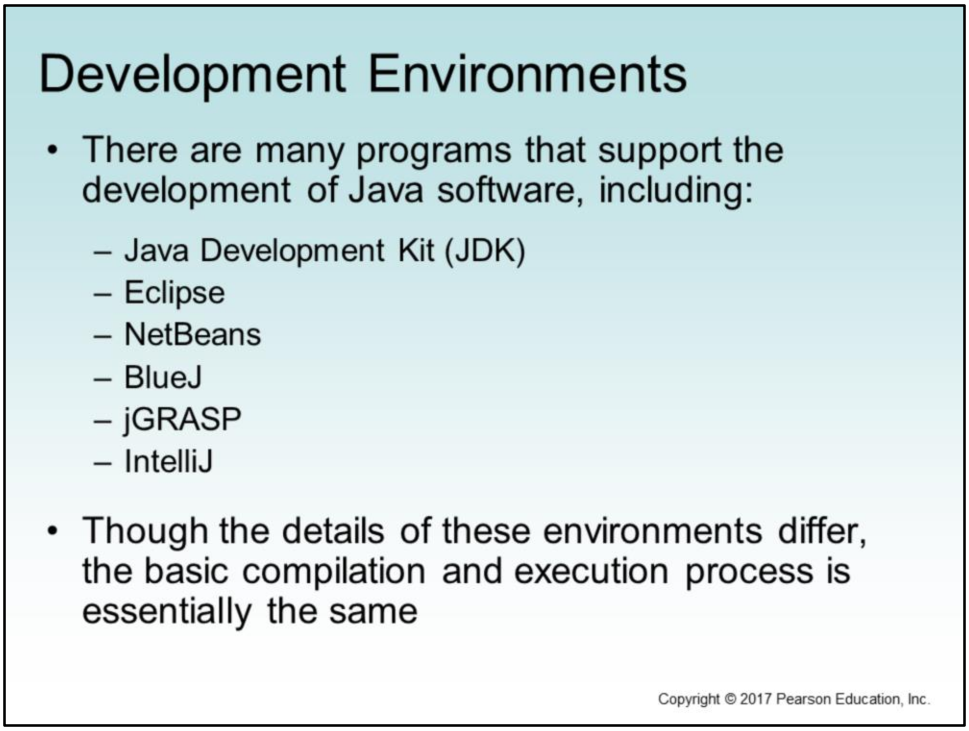
Reference

-Look in your Eclipse workspace directory for your first sample project or lab

-Note that Java source code is in a .java file in your **src** directory

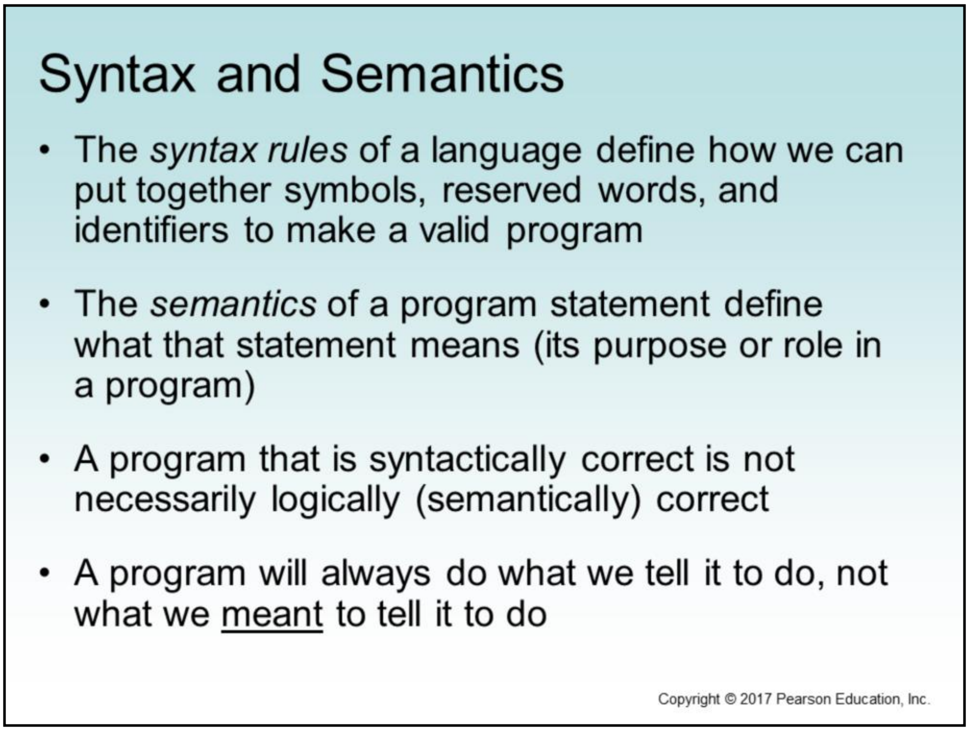
-Note that the compiled Java bytecode is in a .class file in your **bin** directory

-Java bytecodes can be ported to any machine running a Java virtual machine



Development Environments

- There are many programs that support the development of Java software, including:
  - Java Development Kit (JDK)
  - Eclipse
  - NetBeans
  - BlueJ
  - jGRASP
  - IntelliJ
- Though the details of these environments differ, the basic compilation and execution process is essentially the same



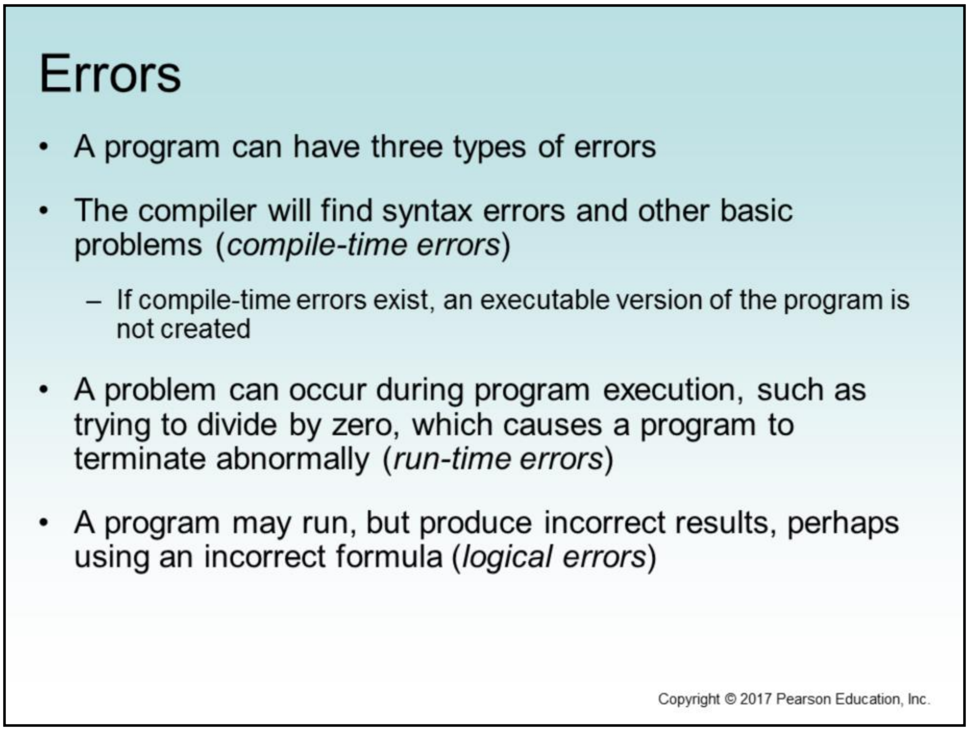
Syntax and Semantics

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

-Appendix I in textbook contains formal description of rules of language (syntax)

-When we write statements, we need to “think like the compiler”

-Try to interpret what the compiler will do when it sees a statement



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*)

-Debugging: finding and correcting defects in a program

-As programmers, we must all learn to work hard at debugging our programs

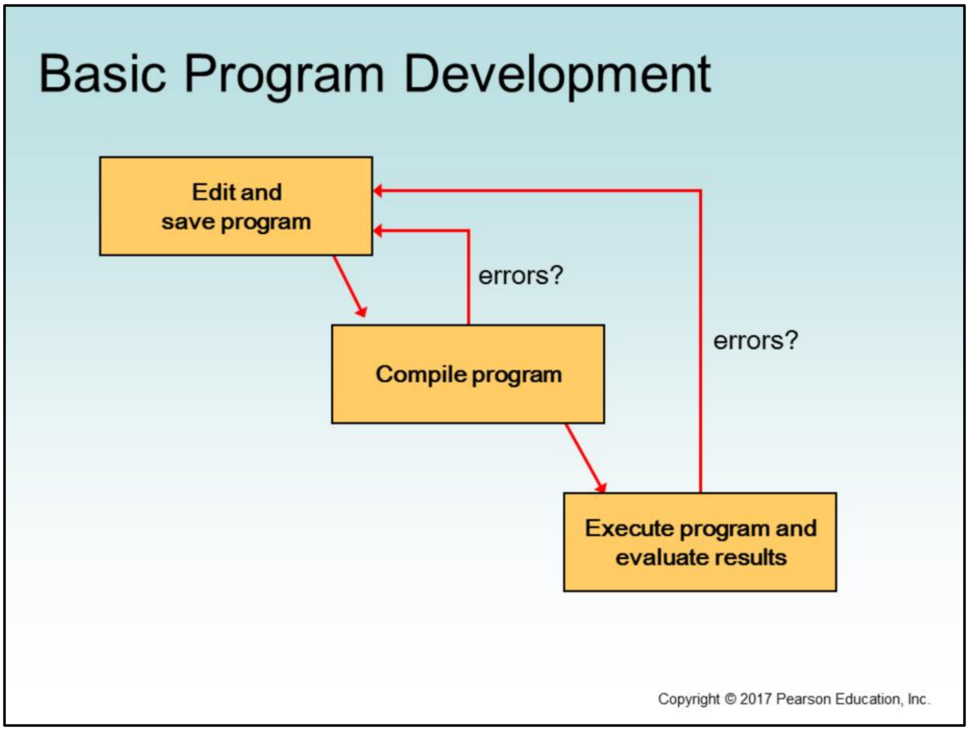
-When your program doesn’t work, don’t immediately give up

-Analyze your code line by line and examine what is happening

-This is part of what becoming a good software developer is all about!

-Instructor/CA will not debug your code for you, this is something you do!

-Instead, we may help direct you to the problem area



Basic Program Development

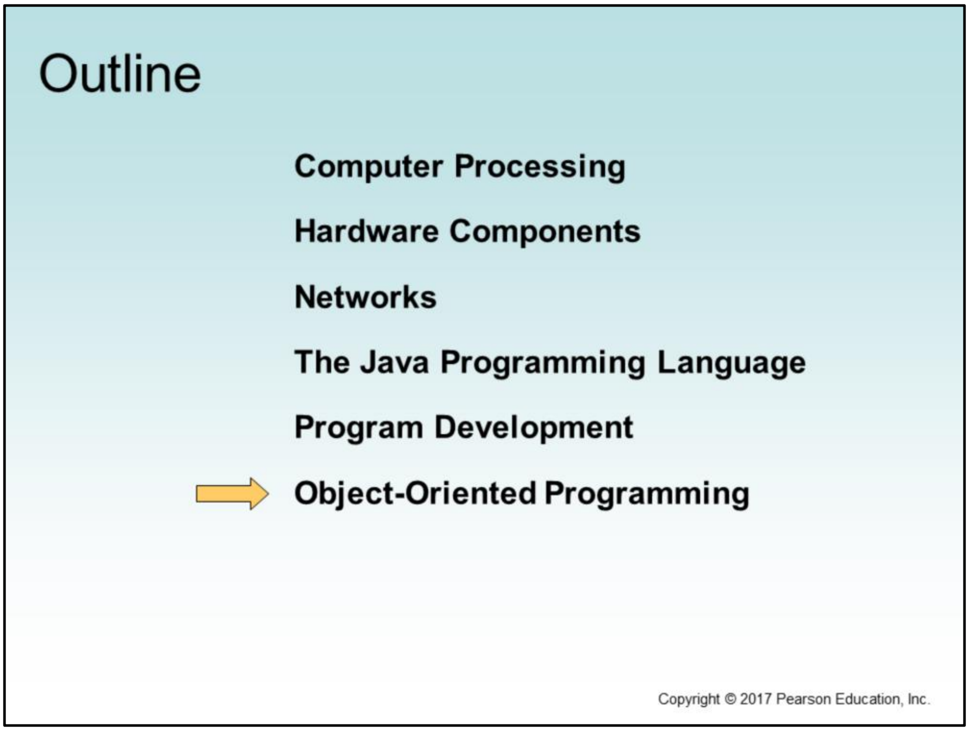
Edit and
save program

Compile program

Execute program and
evaluate results

errors?

errors?



Outline

Computer Processing

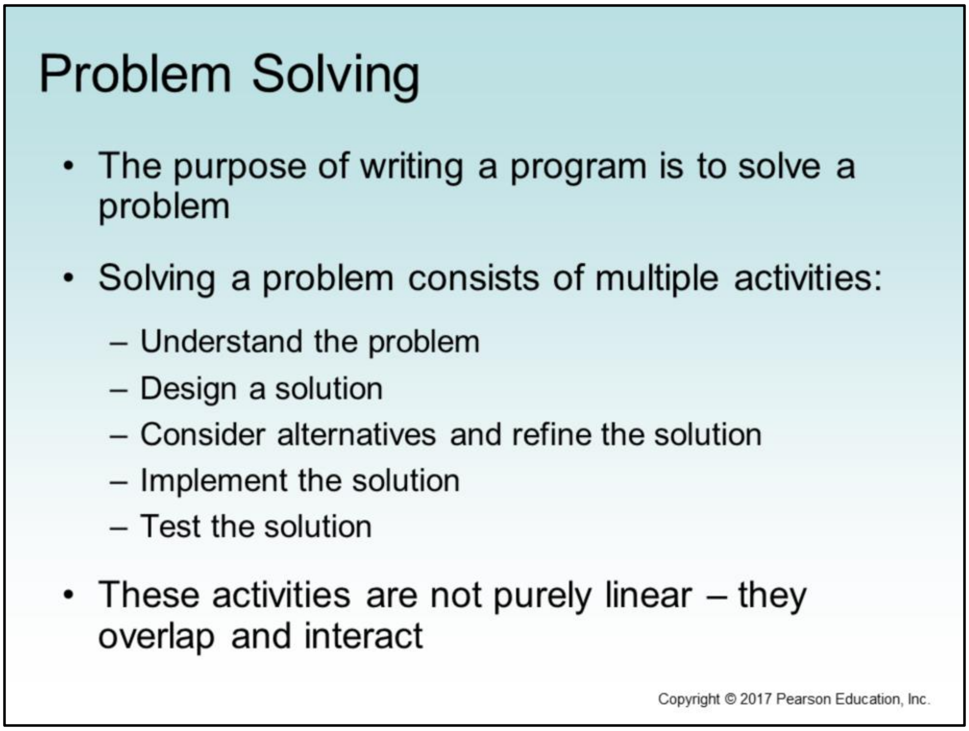
Hardware Components

Networks

The Java Programming Language

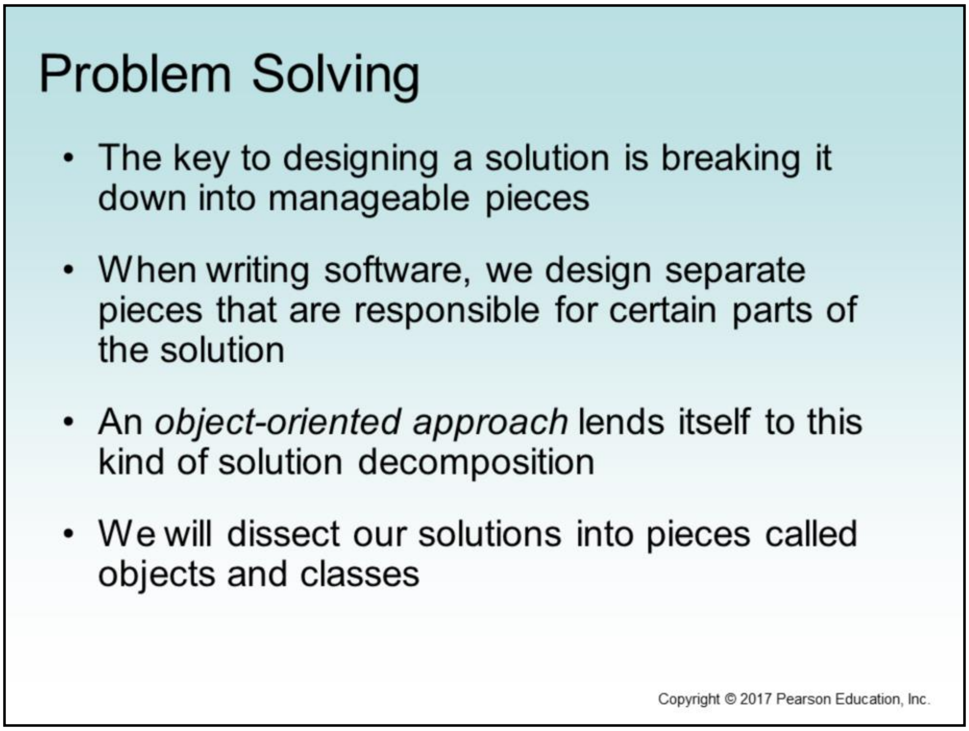
Program Development

➡ Object-Oriented Programming



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

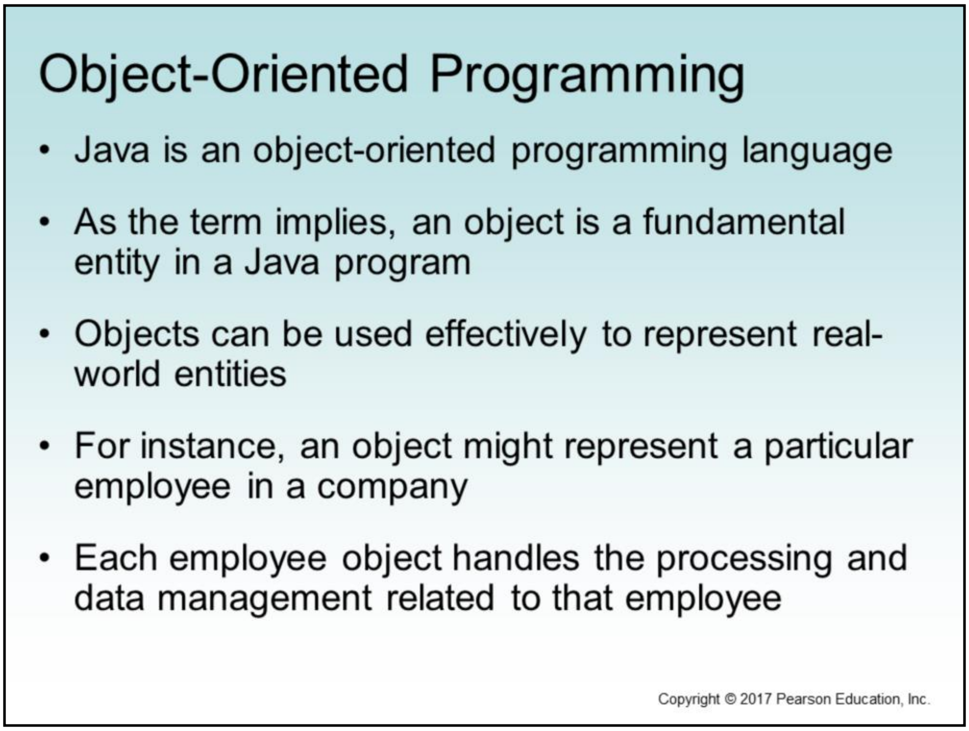
- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An *object-oriented approach* lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called objects and classes

-Our goal is to learn how to be software **designers** as well as programmers

-Software development is more about design than implementation

-We will learn how to design a solution to a problem using a certain method

-This method is based on **object-oriented** principles



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

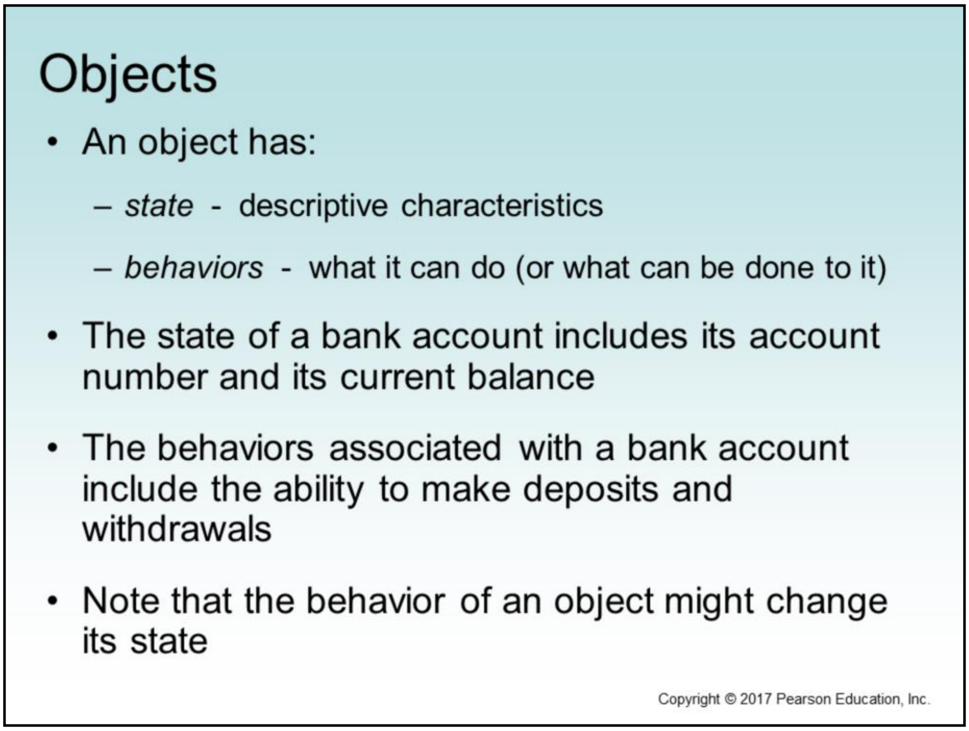
-We’ll learn how to look at a problem description and identify the **objects**

-Another term used for an object is a **data model**

-We’ll see that the objects in a problem description are typically the **nouns**

-Once identified, we’ll build software to describe these objects

-These are essentially the Java **classes** that we’ll use and write



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

-We'll design objects by what they "know" and what they "do"

-The object **state** is what it knows; The object **behaviors** are what they **do**

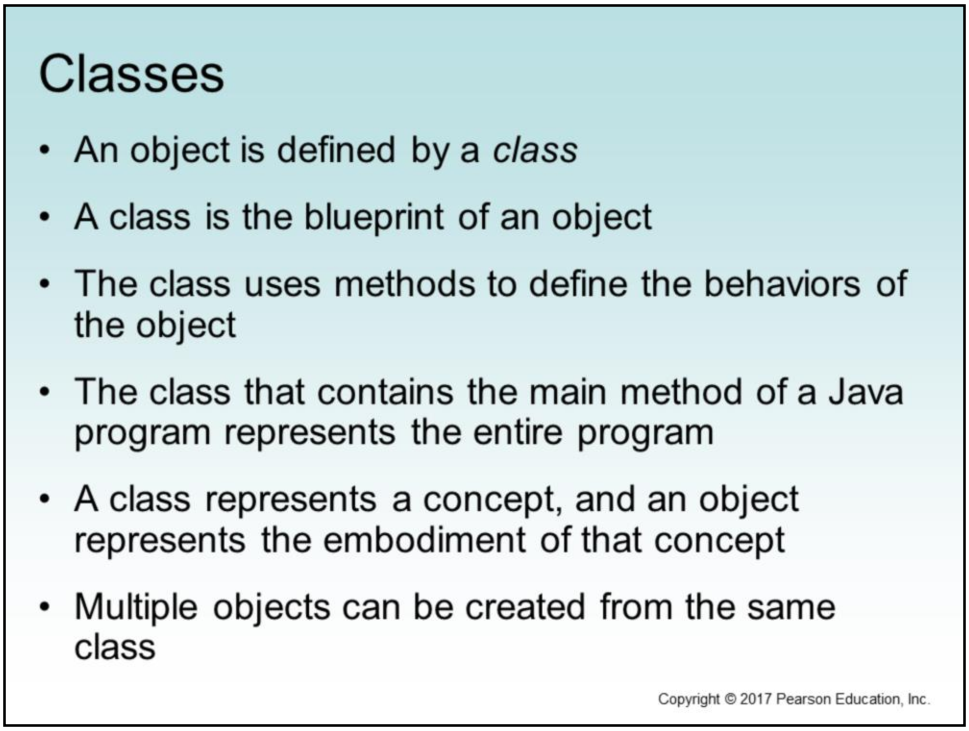
-The program is simply objects that interact with each other to solve the problem

-Consider, for example, a bowling scoring system

  -objects are the nouns (ball, pin, scorecard)

  -program consists of a ball, pins, and scorecard all interacting to compute the score

-Notice how we are thinking about the nouns (objects) to solve the problem



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

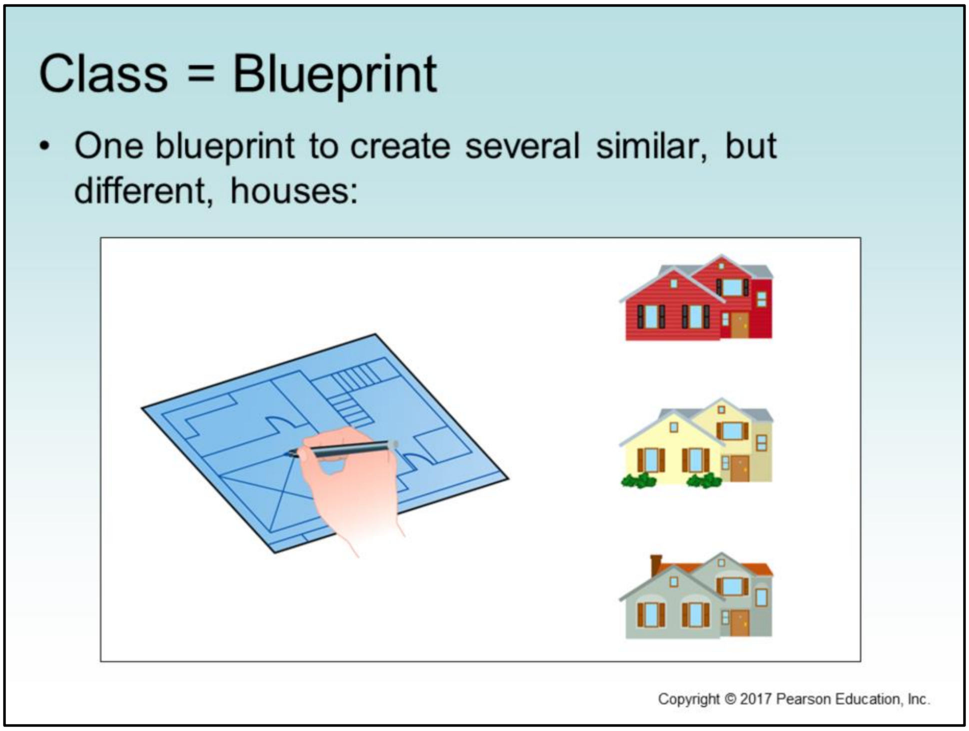
-We describe our object with a definition, similar to a blueprint, called a **class**

-The class defines how to create an actual object(s) we use in a program

-This is similar to how a blueprint is used to create an actual house

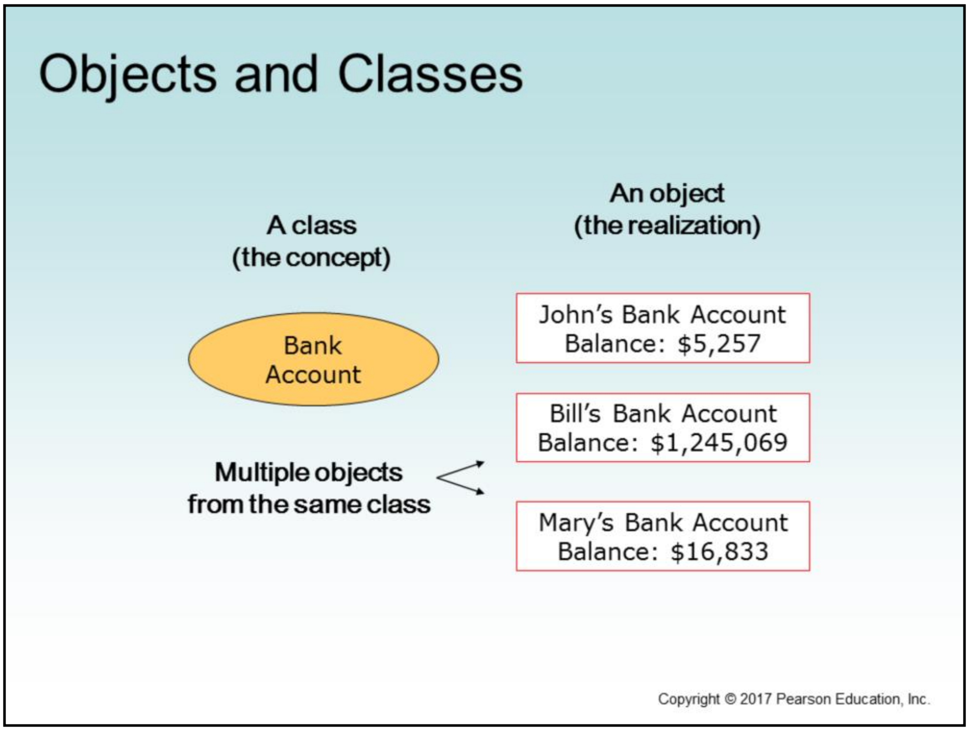
-The object actually “lives” in memory during a program

-The class is used by the CPU to describe how to build the object in memory



Class = Blueprint

- One blueprint to create several similar, but different, houses:



Objects and Classes

A class
(the concept)

An object
(the realization)

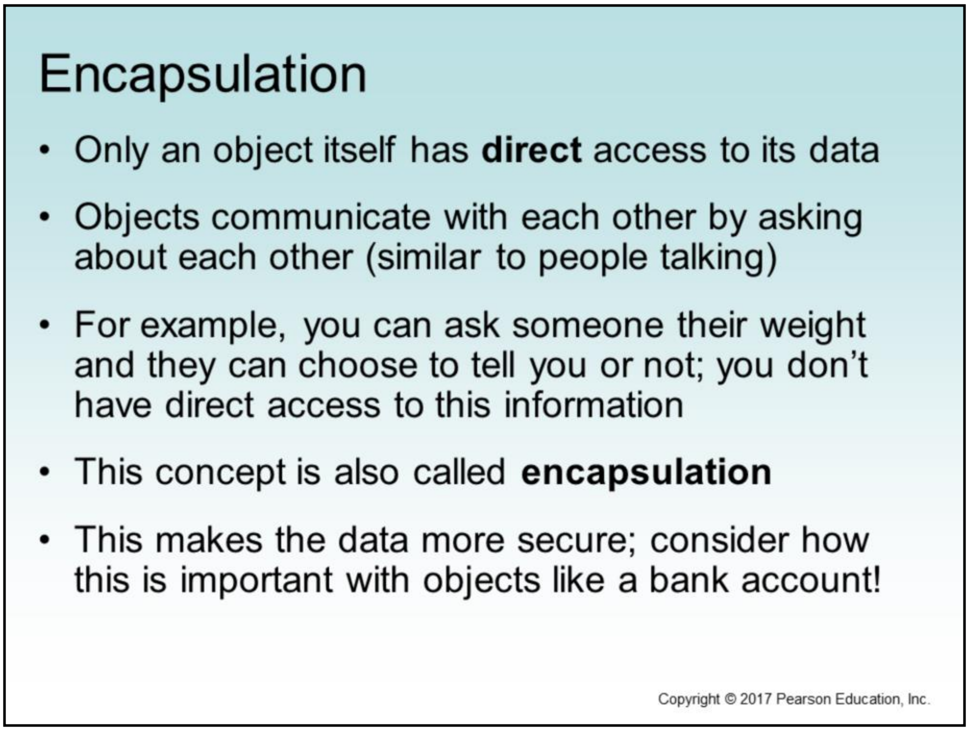
Bank
Account

John's Bank Account
Balance: $5,257

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

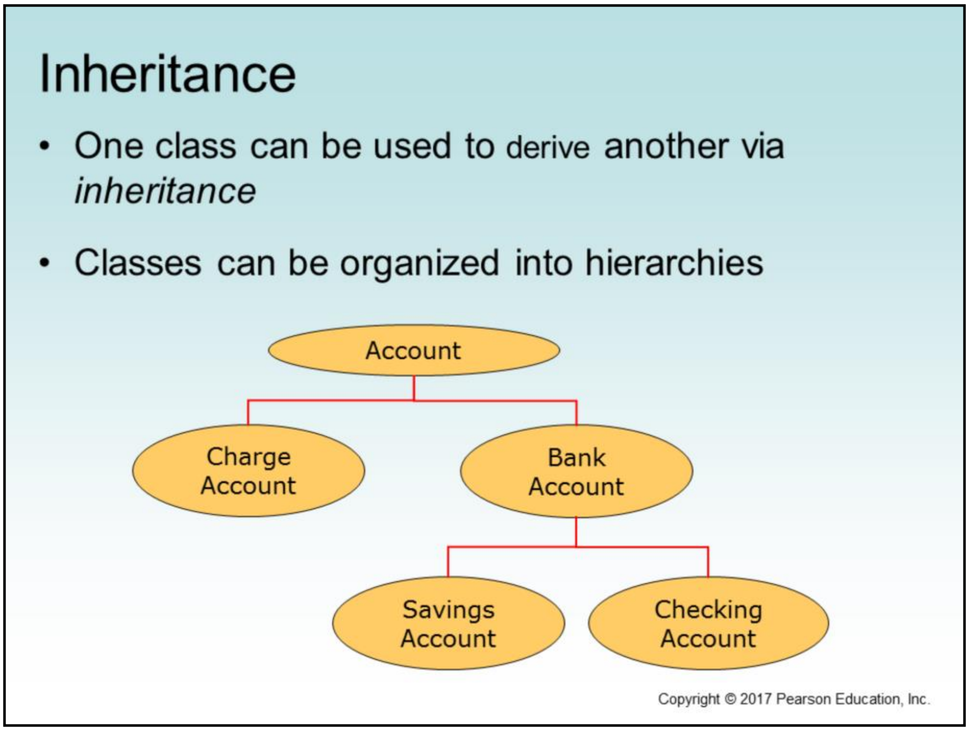
Mary's Bank Account
Balance: $16,833

Multiple objects
from the same class



Encapsulation

- Only an object itself has **direct** access to its data
- Objects communicate with each other by asking about each other (similar to people talking)
- For example, you can ask someone their weight and they can choose to tell you or not; you don’t have direct access to this information
- This concept is also called **encapsulation**
- This makes the data more secure; consider how this is important with objects like a bank account!



Inheritance

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

-Inheritance mimics the real-world; consider designing classes representing animals

-different **types of** animals; dogs, cats, horses

-different **types of** dogs, cats, horses

-Inheritance allows us to categorize and organize classes into hierarchies

-By deriving one class from another, we can reuse definitions

-**Reusability** is an important aspect in designing effective software

-**Polymorphism**: different objects respond differently using the same method name

-We'll be studying these concepts in detail as we begin designing our own classes