CS1007: Object Oriented Design and Programming in Java

Lecture #2

Sept 8

Shlomo Hershkop
shlomo@cs.columbia.edu

Announcements

- Class website updated
  - Homework assignments will be posted Sept 9.
- TA:
  - Edward Ishak
  - Amrita Rajagopal
- Unix review cources offered by CUIT
- Check courseworks tomorrow for office hours survey.
  - Let me know asap if you can not log in.

Unix classes

- 252 engineering
- Friday Sept 9
  - 10-12, 1-3, 3-5pm
- Monday Sept 12
  - 5-7pm
- Wed Sept 14
  - 5-7pm
  - Friday 16
  - 19, 21

Outline

- Review of Java basics.
- Writing classes in Java.
- Types
- Object reference vs. Object values

Reading

- Today: Chapter 1
- Next: Chapter 1, homework 0 (non-credit)

Status

- Should have plans on acquiring the text
- Should have tested your cunix access
- Should have seen the class website

- Cunix accounts, will be used for homework submissions
- CS accounts are useful for working in the clic lab and accessing the cs departments resources.
  - www.cs.columbia.edu/~crf/accounts

Suggestions

- Working outside of CUNIX:
  - Setup correct version of java
  - Use IDE
  - Save often
  - Don’t forget to test on cunix
- Working on CUNIX
  - Don’t telnet
  - Putty: available from acis
  - Work in the labs

Class background

- There is a wide variety of both JAVA and programming skills
- We will do a super fast overview of JAVA basics before starting the meat and potatoes of the course.
- Please bear with me.

Java Program Structure

- 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*
- A Java application always contains a method called **`main`**

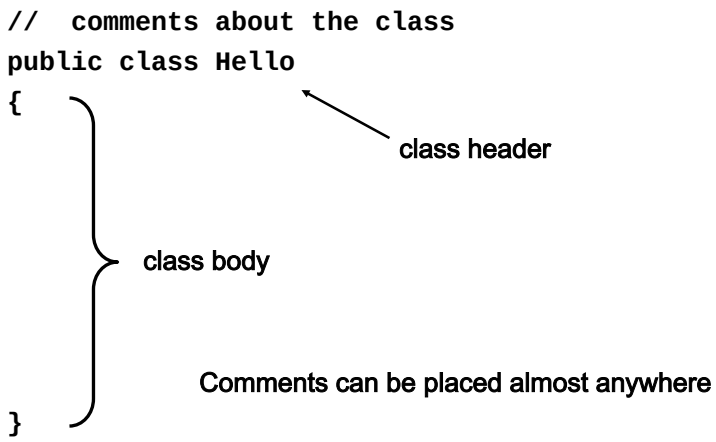
Java Program

- A Java program contains at least one class definition.

```
public class Hello {
  public static void main(String[] args) {
  System.out.println("Hello, world!") ;
  }
}
```

- This code defines a class named Hello. The definition of Hello must be in a file Hello.java.
- The method main is the code that runs when you call `java Hello'.

Java Program Structure



10

Java Program Structure

```
//  comments about the class
public class Hello
{
   //  comments about the method
   public static void main (String[] args)
   {
   }

}
```

11

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, they are simply ignored.
- 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   */
```

12

Identifiers

- Elements in a program are identified by some name. In Java, identifiers:
  - Always start with a letter.
  - Can include letters, digits, underscore (` ') and the dollar sign symbol ($).
  - Must be different from any Java reserved words (or keywords).
- Often we use special identifiers called reserved words that already have a predefined meaning in the language
  - Keywords that we've seen so far include: **`public`**, **`static`**, **`class`** and **`void`**.

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`

14

Case counts

- Identifiers and keywords in Java are case sensitive. In other words, capitalization matters. Keywords are always in lowercase. The following identifiers are all different:
  - **`SHLOMO`**
  - **`shlomo`**
  - **`SHlomO`**
- Bad idea: use all those in one program.
- WHY?

Spaces

- We use the word whitespace to describe blanks, tabs and newline characters. The Java compiler ignores whitespace except when it is used to separate words. E.g.:

```
y=m*x+b;total=total+y;
```

- Is the same as:

```
y = m*x + b ;
total = total + y ;
```

- Which is easier to read?
- Does anyone know the diffence between DOS and UNIX linebreaks? (hint: fixcrlf)

Types

The values a variable can take on and the operations we can perform on them is determined by its type. Java has the following type categories:

- Booleans
- Characters
- Integers
- Floating Points
- References to Objects

Integers

- The java integer type represents both positive and negative integers. An n-bit integer x, can represent the range:

$$-2^{n-1} \leq x \leq 2^{n-1}$$

- byte 8 bits
  short 16 bits
  int 32 bits
  long 64 bits

Integer Literal

- A integer value or literal can be specified in decimal, hex, or octal (base 8)
  - Decimal is a regular number which doesn't start with zero
  - Hex literals start with 0x…(0x1F = 31 base10)
  - Octal literals start with just zero (072 = 58 base10)
- Integer literals are by default of type int
- A long literal ends with L
- If an int is small enough to fit into a short, it will be automatically converted, else you need to cast. In general extra bits are thrown away (not always good).

Floating Point Type

- Floating point are used to represent the real numbers, i.e. fractional numbers
- 0.345 = 3.45 X 10

Program Development

- The mechanics of developing a program include several activities
  1. Skip design
  2. writing the program in a specific programming language (such as Java)
  3. translating the program into a form that the computer can execute
  4. investigating and fixing various types of errors that can occur
  5. Go back and design correctly
- Software tools can be used to help with all parts of this process

Development Environments

- There are many programs that support the development of Java software, including:
  - Sun Java Development Kit (JDK)
  - Sun NetBeans
  - IBM Eclipse
  - Borland JBuilder
  - MetroWerks CodeWarrior
  - Monash BlueJ
- 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

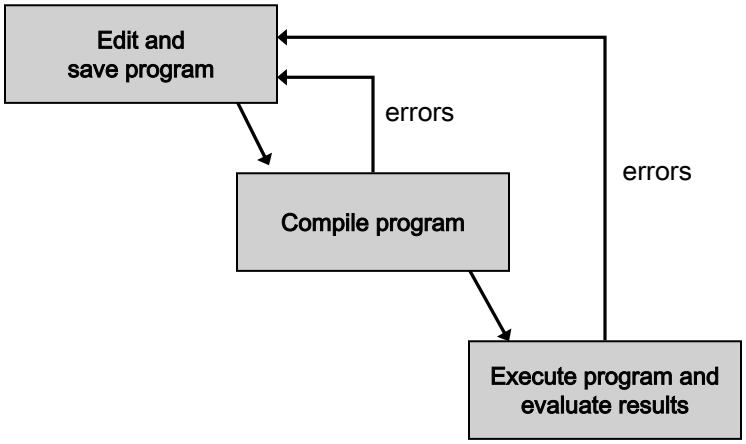
23

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*)

24

Basic Program Development



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
- We try to define all our data as objects, and define programs to work on those objects
- 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 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

29

Reusability

- OOP encourages the design of reusable components
- Vehicle as a general definition
- Mini-van as a more specific object

```
Public class miniVan{
  String manufacturer;
  String model;
  int year;
  Color color;
}
```

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

Instantiating

- Once we define a class we create an instance of the class.
- The constructor method is responsible for initializing the object
- `new` creates an instance

null

- null refers to no object
  - Uninitialized objects
  - Explicit assignment
- Can assign null to object variable:
  - `worldGreeter = null;`
- Can test whether reference is null
  - `if (worldGreeter == null) . . .`
- Dereferencing `null` causes `NullPointerException`

this

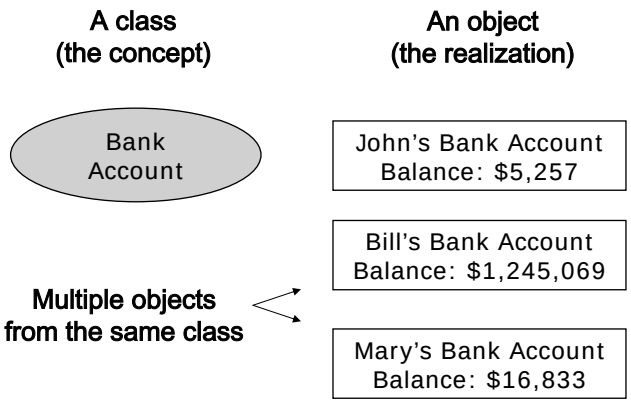
- Refers to implicit parameter of method call
- Example: Equality testing

```
public boolean equals(Greeter other)
{
   if (this == other) return true;
   return name.equals(other.name);
}
```

- Example: Constructor

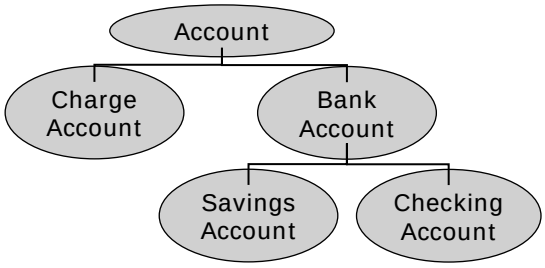
```
public Greeter(String name)
{
   this.name = name;
}
```

Objects and Classes



Inheritance

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

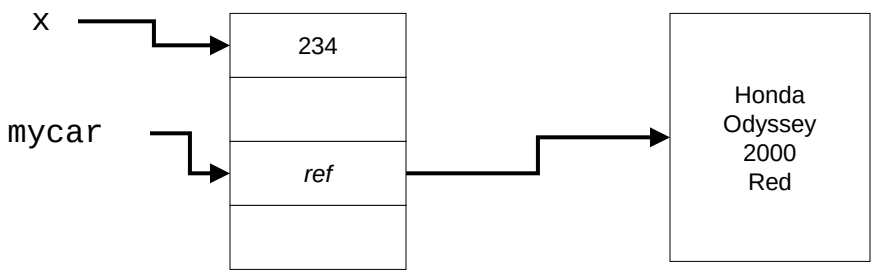


Java Objects

- Construct new objects with new operator
  - new Greeter("World")
- Can invoke method on newly constructed object
  - new Greeter("World").sayHello()
- More common: store object reference in object variable
  - Greeter worldGreeter = new Greeter("World");
- Then invoke method on variable:
  - String greeting = worldGreeter.sayHello();

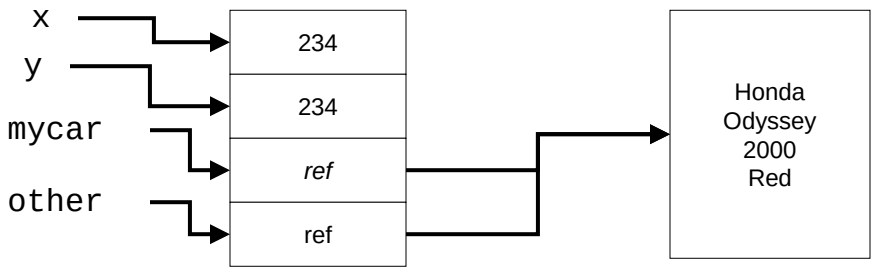
References

- A variable is a location in memory
- int x;
- x= 234;
- miniVan mycar;
- mycar = new miniVan(….)



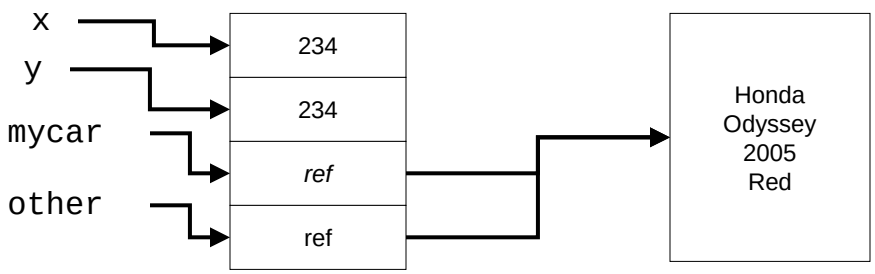
Difference References

- Create new variable y
- int y = x;
- Create another miniVan instance
  miniVan other = mycar;



Difference II

- Other.year = 2005;
- Surprise!



Why?

- Any ideas why we would want to create object using references?

References

- Object variable holds a reference
  - `Greeter worldGreeter = new Greeter("World");`
- Can have multiple references to the same object
  - `Greeter anotherGreeter = worldGreeter;`
- After applying mutator method, all references access modified object
  - `anotherGreeter.setName("Dave");`
  - `worldGreeter.sayHello() //returns "Hello, Dave!"`

Parameter Passing

- Java uses "call by value":
  Method receives copy of parameter value
- Copy of object reference lets method modify object

```
public void copyNameTo(Greeter other)
{
    other.name = this.name;
}
```

```
Greeter worldGreeter = new Greeter("World");
Greeter daveGreeter = new Greeter("Dave");
worldGreeter.copyNameTo(daveGreeter);
```

No reference passing

- No Reference Parameters

- Java has no "call by reference"

```
public void copyLengthTo(int n)
{
    n = name.length();
}
```

```
public void copyGreeterTo(Greeter other)
{
 other = new Greeter(name);
}
```

- Neither call has any effect after the method returns

```
int length = 0;
worldGreeter.copyLengthTo(length); // length still 0
worldGreeter.copyGreeterTo(daveGreeter) // daveGreeter unchanged
```

Java packages

- Collection of similar classes
- Package names are dot-separated identifier sequences

java.util
javax.swing
com.sun.misc
edu.columbia.cs.robotics

- Unique package names: start with reverse domain name
- Must match directory structure
- package statement to top of file
- Class without package name is in "default package“
- Full name of class = package name + class name

java.util.String

Importing Packages

- Tedious to use full class names
- import allows you to use short class name

```
import java.util.Scanner;
. . .
Scanner a; // i.e. java.util.Scanner
```

- Can import all classes from a package

```
import java.util.*;
```

Command Line Arguments

```
public static void main(String[] args)
```

- `args`, is an array of string.
- The elements of `args` are the command line arguments using in running this class.

```
Java testProgram –t –Moo=boo out.txt
0: ‘-t’
1: ‘-Moo=boo’
2: ‘out.txt’
```

Two dimensional arrays

- You can create an array of any object, including arrays
- An array of an array is a two dimensional array

```
public class TicTacToe{
   public static final int EMPTY = 0;
   public static final int x = 1;
   public static final int y = 2;

   private int[][] board =
   { {EMPTY, EMPTY, EMPTY},
     {EMPTY, EMPTY, EMPTY},
     {EMPTY, EMPTY, EMPTY}
   }
```

Two dimensions

- You can also initialize the inner array as a separate call.
- Doesn't have to be congruous memory locations

```
int [][]example = new int[5][];
for (int i=0;i<5;i++){
  example[i] = new int[i+1];
}
```

Multiple dimensions

- No reason cant create 4,5,6 dimension arrays
- Gets hard to manage
- Think about another way of representing the data
- Often creating an object is a better approach

Next Class

- Read Chapter 1
- Download and try Homework 0
- Get up to speed on Java
  - Read old notes
  - Dig out reference text