Lecture 3

First cup of JAVA

Winston & Narasimhan: Chapt 6 - 13

The Agenda for Today

• Review: A simple Java program
• Example: Quidditch
• The Math class: Arithmetic Expressions
• Properties of parameters (variables)
• Procedure abstraction
• Class variables
• Class instances
• Instance methods
• Constructors

Introduction to Objects

• Object Instance
  – Attributes (properties): associated data
  – Operations: actions that change attributes
    • Methods in JAVA
    • Member functions in C++

• Object Class
  – Category or set of objects
  – Share same attributes, operations and relationships to other objects
Quidditch

Quidditch is a game played in mid-air on broomsticks that is a cross between rugby, basketball and the emergency room. Each team consists of seven players; three Chasers, two Beaters, one Seeker and one Keeper. The Keeper acts as a goalie against the other team making a score in one of their three hoops. The Chasers work together to score, using a ball known as the Quaffle. Scoring with the Quaffle is worth ten points. Two other balls known as Bludgers are enchanted to try and knock the players from their brooms. It is up to the Beaters to keep this from happening. Equipped with bats similar to those used in Cricket, Beaters work furiously to keep their teammates from being injured by Bludgers. Last and never least, the Seekers have the important task of catching the Snitch. This ball is the size of a walnut with wings and flies around the Pitch independently. Capture of the Snitch is worth 150 points, and it automatically ends the game. However, this doesn't mean the team that catches the Snitch automatically wins. The game can also be ended with the mutual consent of both team captains.

From http://www.pensievemush.org/hogwarts.html

Example: Object Class

```
QuidditchTeam

name
quaffle
snitch
totalScore

setScore(…)
addQuaffle()
addSnitch()
```

Example: Object Instances

```
t1: QuidditchTeam

name = "Hufflepuff"
quaffles = 5
snitch = 0
totalScore = 50

t2: QuidditchTeam

name = "Gryffindor"
quaffles = 3
snitch = 1
totalScore = 180
```
Naming Conventions

- Convention:
  - Attributes/operation names begin with lower-case characters
  - Class and type names are capitalized
- Common Operations:
  - get*, set*, is*

Common Operations:
- get*, set*, is*

Some Variable Types

- char: 2 bytes: characters
- byte, short, int, long: increasing: integers
- float, double: 4, 8: floating point number
- OTHERS: boolean, String, arrays etc

```java
public class Demonstrate{
    public static void main (String args[]) {
        int script, acting, direction;
        ...
    }
}
```

ARITHMETIC EXPRESSIONS

- The usual operators: +, -, *, /
- Modulus: % (remainder in division)
- The usual precedence rules apply
- Mixing data types in expressions
- Casting an expression
  - Original data type remains unchanged
- Assignment operates from R to L
The Math Class

```java
public class Demonstrate {
    public static void main (String argv[])
    {  
        System.out.println("natural log of 10:");
        System.out.println(Math.log(10));
        System.out.println("abs value of -10"): Math.abs(-10));
        System.out.println("max of 2 and 3:");
        System.out.println(Math.max(2,3));
        System.out.println("5th power of 6:");
        System.out.println(Math.pow(6,5));
        System.out.println("sqrt of 7:");
        System.out.println(Math.sqrt(7));
        System.out.println("sine of 8 rad:");
        System.out.println(Math.sin(8));
        System.out.println("Random no.(0,1):");
    }
}
```

Defining a Simple Method

```java
public class Demonstrate {
    public static void main (String argv[])
    {  
        int script = 6, acting = 9, direction = 8;
        System.out.print("The rating of the movie is");
        System.out.println(movieRating(script,acting,direction));
    }
}
```

Defining Methods in Multiple Files

```java
public class Movie {
    public static int movieRating (int s, int a, int d) {
        return s + a + d;
    }
}
```

```java
public class Demonstrate {
    public static void main (String argv[])
    {  
        int s = 6, a = 9, d = 8;
        System.out.print("The rating of the movie is");
        System.out.println(Movie.movieRating(s,a,d));
    }
}
```
Other Things to Know

- You can define more than one class in the same file
- To call a method in a different class, preface the method name with the class name and join with a dot.
- You can define multiple methods with the same name in the same class as long as they have different signatures (i.e. arrangement of parameter data types)

Properties of parameters

- Call-by-value vs. call-by-reference
  - Java is a CALL-BY-VALUE language:
    - public class PassByValue {
      public static void main(String[] args){
        int i = 5;
        methodZ(i);
        // i = 5 (call by value); i = 6 (call by reference)
      }
      public static void methodZ(int j){
        j = 6;
      }
    }
  - “i” will still be equal to 5 in the main method

- Variables declared inside a block are said to be local variables
- A parameter is treated as a local variable inside the method block
- Parameters and local variables have
  - local scope
  - dynamic extent
  - (they use a temporary chunk of memory)
Find the BUGS

```java
public class Demonstrate {
    // define a method called adder:
    public static int adder() {
        return s + a + d;
    }
    // define a method called movieRating:
    public static int movieRating(int s, int a, int d) {
        return adder();
    }
    // define main:
    public static void main(String argv[]) {
        int script=6, acting=9, direction=8, result;
        result=movieRating(script, acting, direction);
        System.out.println("The rating of the movie is ");
        System.out.println(s + a + d);
    }
}
```

Procedure Abstraction

- What is it?
  Moving computational details into methods
- VIRTUES:
  - Makes it easy to reuse programs
  - Pushes details out of sight/mind
  - Makes debugging easier
  - Makes augmenting repetitive computations easier
  - Makes it easy to improve computation
  - Makes it easy to change computation

Class Variables

- Exist throughout program
- Defined inside class body but not inside any method
- Marked by keyword static

```
public class Movie {
    // define class variables:
    public static int wScript=6, wActing=13, wDirection=11;
    // define movieRating:
    public static int movieRating(int s, int a, int d) {
        return wScript*s + wActing*a + wDirection*d;
    }
}
```
Class Variables: Access & Scope

- To access a variable from a different class...
- To differentiate from a parameter or local variable of the same name...
- Public class variables have
  - Universal scope
  - Static extent
- Class variables' values can be changed to fix a variable at a constant value... use ‘final’

```java
public static final int lightSpeedms = 299792458
```

Class Instances

- Arithmetic types (int, double etc) are like simple atoms, and class instances are like molecules in chemistry.

```java
public class Movie {
    public int script, acting, direction;
}

public class Demonstrate {
    public static void main(String argv[]) {
        Movie m;
        m = new Movie();
        m.script = 3;
        m.acting = 8;
        m.direction = 6;
    }
}
```

Class Instances

- Define Instance Variables
- Assignment
Using Class Instance as Argument

```java
public class Movie {
    // define instance variables:
    public int script, acting, direction;
    // define rating:
    public static int rating (Movie m) {
        return m.script + m.acting + m.direction;
    }
    // public class Demonstrate {
    public static void main(String argv[]) {
        // Movie m = new Movie();
        // m.script=3; m.acting=8; m.direction=6;
        System.out.print("The rating of the movie is ");
        System.out.println(Movie.rating(m));
    }
}
```

Memory Allocation for Instances

- A class instance within a method is a reference-type variable
- b = new Button()
  
  b is NOT an object but only a “reference” for the new object
- Look here to clarify:
  www.javamain.com

Reference type variable

- Java is still a CALL-BY-VALUE language:
  ```java
  class PassByValue {
      public static void main(String[] args){
          Button b = new Button("AAA");
          methodX(b);
          // object referring to b IS affected
          y.setLabel("BBB");
          // object referring to b IS NOT affected
          y = new Button("CCC");
      }
  }
  ```
  - b.getLabel() will be equal to “BBB” in the main method
Reference type variable

1 - Assignment of b
   Reference Variable  
   Object
   b

2 - Passing b into a method
   Reference Variable
   Object
   b
   b

3 - Assignment of y
   b's object no more affected
   Reference Variable
   Object
   y
   b

Reference type variables can be

• used to pass objects to methods

• passed to other objects


Memory “management”

• Reassignment of variables such as:
  Movie m;
  Declaration
  m = new Movie();  
  Assignment
  m = new Movie();  
  Re-assignment
  (abandoned memory of first class instance)
  • Java has a built-in mechanism that return
    abandoned memory to the free-storage list
    (Garbage collection)

Instance Methods

• public class Movie {
  // define instance variables:
  public int script, acting, direction;
  public static int rating(Movie m) {
    return m.script + m.acting + m.direction;
  }
}

Movie m = new Movie();
m.script = 3; m.acting = 8; m.direction = 6;
System.out.println("The rating of the movie is "+
m.rating());
What is “This”?

- public class Movie {
  
  // define instance variables;
  // define instance method, rating:
  public int rating () {
    return this.script + this.acting + this.direction;
  }
}

- “this” is a copy of the address of an instance.
- Use “this” only when necessary or programs get bulky.

Ordinary arg in Instance Methods

public class Movie {

  // define instance variables:
  // define instance method, rating:
  public int rating (double scaleFactor) {
    return (int) (scaleFactor * (script + acting + direction));
  }

  Movie m = new Movie();
  m.script = 3; m.acting = 8; m.direction = 6;
  System.out.println("The rating of the movie is "+m.rating(0.75));
}

Constructors

- Constructor methods are called whenever a new class instance is created.
- Constructor method names are the same as the name of the class.
- Constructor methods return new class instances; no return type is specified.
A Constructor Method

```java
public class Movie {
    // define instance variables:
    // define constructor method, assigning 5 to all variables:
    public Movie() {
        script=5; acting=5; direction=5;
    }
    // define rating:
    public int rating() {
        return script+acting+direction;
    }
}
```

In main ...
```
Movie m = new Movie();
System.out.println("The default rating of the movie is "+ m.rating());
```

Another Constructor Method

```java
In Movie class ...

public Movie() {
    script=5; acting=5; direction=5;
}
public Movie(int s, int a, int d) {
    script=s; acting=a; direction=d;
}
```

In main ...
```
Movie m = new Movie();
System.out.println("The default rating of the movie is "+ m.rating());
Movie m = new Movie(8,9,6);
System.out.println("The new rating of the movie is "+ m.rating());
```

Ready, Get / Set, Go!

- A getter is a method that extracts information from an instance
- A setter (or mutator) is a method that inserts information into an instance
- By using getter/setter methods, we can include additional computations or output statements
- Get/Set methods do not have to deal directly with an actual instance variable
Get/Set Methods

In Movie class ...

................................................
public int getScript () { return script; }
public void setScript (int s) { script=s; }
................................................

In main ...

...........................................
Movie m = new Movie(8,9,6);
System.out.println("The script of the movie is "+m.getScript());
m.setScript (12);
System.out.println("The script of the movie is "+m.getScript());
...........................................

Notes on Access Methods

• Constructors, getters and setters are access methods
• When you move representation detail into a set of access methods, you are
  – practicing data abstraction, and
  – hiding details of data representation behind data-abstraction barriers

Advantages of Data-Abstraction

• Easy to reuse your work
• Push details out of sight / mind
• Easily augment what the class provides
• Easily improve way data is stored
Second Cup of JAVA Summary

- We have learnt to define:
  - Classes
  - Instances
  - Class Methods
  - Class Instances
  - Instance Constructors
  - Getter and Setter Methods