ISE 582: Information Technology for Industrial Engineers
University of Southern California
Department of Industrial and Systems Engineering

Lecture 3
Second cup of JAVA

Handouts
- Lecture 3 slides
- Homework 1 – part 2 (due next week)
- READ:
  Winston & Narasimhan: Ch. 6 - 13
- Java editor: jEdit 4.0.3 (www.jedit.org)
- Java compiler: J2SE 1.4.0_02 SDK
  (java.sun.com/j2se/1.4/download.html)

Our Second Cup of JAVA
- Properties of parameters (variables)
- Procedure abstraction
- Class variables
- Class instances
- Instance methods
- Constructors
- Homework hints
Properties of parameters

- Call-by-value vs. call-by-reference
- Java is a CALL-BY-VALUE language:
  ```java
  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

Properties of parameters

- 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
// define a class called Demonstrate
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[] args) {
    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

Example

```java
import java.util.Scanner;

public class GPA {
    public static void main(String[] args) {
        double iseX = 3, iseY = 4, iseZ = 2, result;
        result = (iseX + iseY + iseZ) / 3;
    }
}
```

```java
public class GPA {
    public static void main(String[] args) {
        double iseX = 3, iseY = 4, iseZ = 2.7, result;
        result = average(iseX, iseY, iseZ);
    }
}
```

```java
public static double average(double X, double Y, double Z) {
    double r = (X + Y + Z) / 3;
    return r;
}
```

Class Variables

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

```java
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.

```
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;
    }
}
```
Using Class Instance as Argument

```java
public class Demonstrate {
    public static void main(String argv[]) {
        Movie m = new Movie();
        m.script = 3; m.acting = 8; m.direction = 6;
        System.out.println("The rating of the movie is ");
        System.out.println(Movie.rating(m));
    }
}

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;
    }
}
```

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

1. Assignment of b
2. Passing b into a method
3. Assignment of y
   b's object no more affected

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

```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;
    }
}
```

```java
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:
  public int script, acting, direction;
  // 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:
  public int script, acting, direction;
  // define 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:
    public int script, acting, direction;
    // 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
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();
Movie m = new Movie(8,9,6);
System.out.println("The default rating of the movie is " + m.rating());
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 ...

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

In main ...

```java
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

Example: Object Class

<table>
<thead>
<tr>
<th>WIZARD</th>
<th>Class name</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td></td>
</tr>
<tr>
<td>age</td>
<td></td>
</tr>
<tr>
<td>house</td>
<td></td>
</tr>
<tr>
<td>claimToFame</td>
<td></td>
</tr>
<tr>
<td>changeName(...)</td>
<td>Instance</td>
</tr>
<tr>
<td>incrementAge()</td>
<td>Variables</td>
</tr>
<tr>
<td>calculateDOB()</td>
<td>Instance</td>
</tr>
<tr>
<td>Methods</td>
<td></td>
</tr>
</tbody>
</table>

Example: Object Instances

w1: Wizard
- name = “Albus Dumbledore”
- age = 161
- house = “Gryffindor”
- claimToFame = “Defeated dark wizard Grindelwald”

w2: Wizard
- name = “Harry Potter”
- age = 21
- house = “Gryffindor”
- claimToFame = “Survived deadly curse by Voldemort”
Homework hints

- **Form**: if you can, don't use Front-Page and don't worry about conditional statements
- **Gringotts**: only 2 classes in total for (a)+(b): Gringotts and Demonstrate
- **Weasleys**: “graduate” method must change the boolean instance variables too (is...)
- **TURN IN**: paper-copy of your codes put your files in your ise582 directory