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
• Assignment 3
• READ:
  – Winston & Narasimhan: Chapters 6 - 13 (pp 35 - 73)
  – Lau: Chapter 1 online java examples

Announcements

• Homework 1 is due today
• Winston “On to Java” is on order at the bookstore. Should be available by next Tuesday
The Agenda for Today

- Our second cup of JAVA
- Object-Oriented Concepts

Our Second Cup of JAVA

- The parameters of parameters
- The virtues of procedure abstraction
- Defining class variables and instances
- Defining instance methods
- Class instance constructors

The Parameter of the Parameters

- Call-by-value vs. call-by-reference
- 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
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 int main(String[] arg) {
        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);
    }
}
```

Virtues of Procedure Abstraction

- Makes it easy to reuse programs
- Pushes details out of sight/mind
- Makes debugging easier (divide & conquer)
- Makes augmenting repetitive computations easier
- Makes it easy to improve computation
- Makes it easy to change computation

Exercise 1

Write a method, named convert, that converts all wizarding money to knuts, the lowest common denominator. (17 silver Sickles = 1 gold Galleon; 29 bronze Knuts = 1 silver Sickle.)
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 method:
    public static int movierating(int s, int a, int d) {
        return wscript*s + wacting*a + wdirection*d;
    }
}
```

Class Methods

- CLASS
- INSTANCES
- EXAMPLES

Class Variables: Access & Scope

- To access a variable from a different class...
- To differentiate from a parameter or local variable of the same name...
- To fix a variable at a constant value...
- Public Class Variables have
  - Universal scope
  - Static extent

Class Instances

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

```java
public class Movie {
    // define instance variables:
    public int script, acting, direction;
    // define rating:
    public static int rating(int s, int a, int d) {
        return s + a + d;
    }
}
```

Movie m;
Movie m = new Movie();
m.script = 3;
m.acting = 8;
m.direction = 6;

Using Class Instance as Argument

```java
public class Demonstrate {
    // define instance variables:
    public static void main(String argv[]) {
        Movie m = new Movie();
m.script=8; m.acting=9; m.direction=6;
        System.out.print("The rating of the movie is ");
        System.out.println(Movie.rating(m));
    }
}
```

Memory Allocation for Instances

- Java is a call-by-value language: parameter reassignments inside method do not propagate outside method
- A class instance within a method is a reference-type variable
- What does this mean for reassignments of instance variables inside a method?
- What happens to memory for abandoned class instances?
### 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 ");
```

What are the diff bet calling a class or instance method?

### What is “This”?

```java
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

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

```java
Movie m = new Movie();
m.script = 3; m.acting = 8; m.direction = 6;
System.out.println("The rating of the movie is ");
```
Exercise 2

Define a Wizard class with variables for the number of Galleons, Sickles and Knuts as instance variables.

Write an instance method, that converts all that wizard's money to knuts.

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

In Movie class ...

```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 ...

```java
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

---

Exercise 3

Define constructor methods for the Wizard class
Create getter and setter methods for instance variables in the Wizard class
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
What’s the big deal about OOPs

- Encapsulation
- Polymorphism
- Inheritance

Encapsulation

- Encapsulation is the process of describing a class or object by giving only enough information to allow a programmer to use the class or object.

Polymorphism

- Polymorphism means that one method name can cause different actions depending on what kind of objects performs the action.
Inheritance

• Inheritance is a way of organizing classes so that classes with properties in common can be grouped. It allows the programmer to avoid repeating the same piece of programming instruction.