Lecture Set #6: Encapsulaton, “this”, junit testing and Libraries

1. Review of Parameter passing
2. this
3. public vs. private Choices
4. Libraries

Reference type Parameters

Recall that methods / constructors can have parameters

```
public int Student giveMore(Student s ) {
if (numOfTokens > s.numOfTokens){
      s.numOfTokens += 3;
    } else{
        numOfTokens += 3;
   }
}
```

Trace Calling assume there are Student objects stu1 and stu2

Where stu1 has 5 tokens and stu2 has 12 tokens

Called with

- stu1.giveMore(stu2);
- stu2.giveMore(stu1);

this

a reference to the current object. (Only makes sense in a non-static method.)
In an instance method, this is the object that is assumed

easy to refer to members (data or methods) using the assumed object

difficult to refer to the whole object without having a name to call it

Only use when needed – using it all the time makes the code more difficult to read

Public Declarations

public variables/methods and classes

Keyword `public` used in declaration

Every user of an object can access any `public` element

Sometimes access should be restricted!

To avoid giving object users unnecessary info (keep API small)

To enforce consistency on instance variables

Private Declarations

private variables, methods and classes

```
    private int tokenLevel = 3;
```

Private variables / members cannot be accessed outside the class definition

Declaring instance variables private means they can only be modified using public methods

Now getters (accessors) and setters (mutators) are required

What Should Be Public / Private?

Class interface = API = public variables / methods

Only make something public if there is a reason to

Why? Encapsulation

- As long as interface is preserved, class can change without breaking other code
- The more limited the interface, the less there is to maintain

Rule of thumb

- Make instance variables private
- Implement `set` / `get` methods
- Make auxiliary methods private

Separate:
API and the workings of the class

Design so that

you can change how the class works without having to change the API

the only things in the API are things the user will absolutely need (make the interface as simple as possible)

Demonstrations in Class

Significantly Modifying the Student class – without changing the API (or the driver)

The Cat class and its drivers

- with adding a copy constructor

Project 3

- API described – you are using those classes
- documentation / comments needed

The problem

Problems:

- need to be able to make sure all parts are tested
- need to know in testing exactly which part was not as expected
- need to be able to keep the tests for modifications made later

Unit testing helps overcome this problems of making sure everything is tested

- Unit testing: test each class and each part of the class (unit) individually
- Goal is to eliminate inconsistencies between the API and the actual working of the code

Floating Point Calculations

What will this print?

```
public class SimpleMath {
    public static void main(String[] args) {
        if (3.9 - 3.8 == 0.1) {
            System.out.println("I am a very smart computer.");
        } else {
            System.out.println("I can't do simple arithmetic.");
        }
    }
}
```

- à I can't do simple arithmetic.
- § Why?
- § Conversion of floating point to binary leads to precision errors!
- § What can we do?

Floating Point Calculations (cont.)

Two important rules:
You can never use == to compare floating point values. Instead, check if two numbers are within a certain tolerance of each other.
Never use floating point values to represent money, e.g., 3.52 to represent $3.52. Instead, use integer 352 to represent 352 pennies.

Documentation Types

Three Styles

```
// ...
/* ... */
/** ...  */
```

Two Purposes

Internal – those reading code

External – those using the class

Javadoc Documentation Standard

When documenting a method, list exceptions that method can throw

- Use `@exception` tag
- Be sure to include unhandled exceptions that operations in method may throw

Example:

```
/**
 * Returns the year part of a date string
 * @param d date string in mm/dd/yyyy format
 * @return an integer representing the date
 * @exception IndexOutOfBoundsException
 * @exception NumberFormatException
 */
public static int getYear(String d) {
  …
}
```

Libraries in Java

Library: implementation of useful routines that are shared by different programs
Java mechanism for creating libraries: packages

Package: group of related clases

Example: `java.util` (contains `Scanner` class)

To use a class from a package, you can use a fully qualified name (package name + class name):

```
java.util.Scanner s = new java.util.Scanner(System.in);
```

You can also import the class in the beginning of the file

```
import java.util.Scanner;
```

To import class in a package:

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

(Imports `Scanner` as well as other classes in package)

Package java.lang

A special package containing widely used classes:

- `String`
- `Math`
- etc.

`java.lang.*` is *automatically imported* by every Java program

Package Management

A class can be added to a package by including:

`package` <name of package>;

in source file (usually very first line)

The variables / methods provided by a class / package are often called its API (= Application Programmers Interface)

APIs should be documented

java.lang documentation:

http://java.sun.com/j2se/1.3/docs/api/java/lang/package-summary.html

On the resources page of the class web site – javadoc generated descriptions.

String API & Math API

`String` implements lots of string functions

`StringExample.java`

`Math` implements lots of mathematical functions

`MathExample.java`