Lecture 11: Libraries and Encapsulation

Last time:
1. for loops (from last lecture notes)
2. Static variables and methods

Today:
1. Parameter passing
2. Libraries
3. Public vs. private
Parameters and Methods

- Recall that methods / constructors can have parameters
  ```java
  public Student (String newName, int IDDesired) {
      name = newName;
      id = IDDesired;
      tokenLevel = 3;
  }
  ```

- What is printed by the following?
  ```java
  String newName = “Joe”;
  Student s = new Student(newName + “ Schmoe”, 123456789);
  System.out.println (s.name);
  System.out.println (newName);
  ```
  Joe Schmoe
  Joe

How Does Java Evaluate Method / Constructor Calls?

int newName = “Joe”;  
Student s = new Student  
   (newName + “ Schmoe”, 123456789);

1. Arguments are evaluated using stack in effect at call site (place where method called)
   ● newName + “ Schmoe”, evaluates to Joe Schmoe
   ● 123456789 evaluates to 123456789

2. Stack frame (temporary addition to stack) created to associate method parameters with values
3. Stack frame put into stack
4. Body of method executed in modified stack
5. Stack frame removed from stack
Example

- public Student (String newName, int IDDesired) {
  name = newName;
  id = IDDesired;
  tokenLevel = 3;
}

- String newName = "Joe";
  Student s = new Student (newName + " Schmoe", 123456789);
  System.out.println (s.name);
  System.out.println (newName);

Note: When two variables in the stack have the same name, Java uses the one in the active (most recently instantiated) frame. Only the variables that appear in the active frame are “in scope”. (All other stack variables are “out of scope.”)
Libraries in Java

- **Library**: implementation of useful routines that are shared by different programs
- Java mechanism for creating libraries: **packages**
  - Package: group of related classes
  - 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
  java.util.Scanner s = new java.util.Scanner(System.in);
  ```
- You can also import the class in the beginning of the file
  ```java
  import java.util.Scanner;
  ```
- To import class in a package:
  ```java
  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:
  
  \[ \text{package <name of package>;} \]
  
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

- \text{java.lang} documentation:
  
  \url{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
Public Declarations

• So far all classes / variables / methods have been public
  • 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

- Java also allows variables / methods to be declared **private**:
  
  ```java
  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
Example

```java
public class Student {
    private String name;
    private int id;
    private int tokenLevel = 3;

    ...
}
```
Example

// “acceptTokens” allows the tokenLevel to be set; an
// error is reported if the number of tokens is invalid

public void acceptTokens (int newTokens) {
    if ((newTokens > 0) || (newTokens <= 3)) {
        tokenLevel = newTokens;
    } else {
        System.out.println
            ("Bad argument to acceptTokens: " + newTokens);
    }
}

- Goal of acceptTokens: ensure tokenLevel values are valid
- But tokenLevel can still be made invalid
  Student s = new Student ();
  d.tokenLevel = 99;
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
  - Project 3
    - API described – you are using those classes
    - documentation / comments needed
Comments

- Explanations you add into your code
- Three forms in Java
  - /*
    Put text here
  */
  - // Put text here to end of line
  - /**
    Put text here
  */
- Last form is special comment for javadoc (utility for generating documentation from comments)
Comments (cont.)

- Some programmers hate them …
- … but they are essential for code understanding (it is called “code” for a reason)
- Beginning with Project #3, you will be graded on comments as well as indentation

Comment:
- Every variable (what is variable for?)
- Every method (what does method do?)
- Every class (what is class for? who wrote it?)
Project Commenting Examples

- Demonstration in Class
  - Project 1 and Project 2 to show
    - correct commenting style
    - correct method definition/calling style
    - efficient use of conditional and looping constructs
  - Octopus example
    - examples of methods calling other methods
    - differentiating static and instance
Example for Class

public class X{
    private String name;
    private static int MAX = 12;

    public void f(){
        X x = new X();

        /* valid or invalid & which is best? */
        System.out.println(name);
        System.out.println(x.name);
        System.out.println(X.name);
        System.out.println(MAX);
        System.out.println(x.MAX);
        System.out.println(X.MAX);
    }

    public static void g(){
        System.out.println(name);
        X x = new X();
        System.out.println(x.name);
        System.out.println(X.name);
        System.out.println(X.MAX);
        System.out.println(x.MAX);
    }
}