CIS 121—Data Structures and Algorithms with Java—Spring 2017

Java and OO Design - Interfaces, Inheritance, and more!—Tuesday (9/6)

Learning Goals

During this lab you will:

- Review interfaces and classes
- Review inheritance
- Discuss encapsulation and best practices
- Common design patterns: Factories

Interfaces vs Classes

What is an interface? In one sentence: It describes the properties and methods that an object should have, but not how to implement them. An interface is like a contract that these are methods that any implementing class is guaranteed to have.

Why do we have interfaces? They allow for more modular code. For example, let's say we have a Sorter interface. We may have many different implementations of the interface, but in a given application, we could swap them out since they all implement the interface and provide the guarantee that they have the same methods.

What is a class? Describes the properties and methods that an object should have, and how to implement them.

What is the difference between the two? There can be many classes that implement a single interface, like in the sorting example above. A single class can implement multiple interfaces. There is no reason it can't - it just means that it fulfills multiple of these contracts. Interfaces describe what is done, but not how it is done.

Subclasses and Inheritance

What is a subclass? Inherits all methods of parent class along with additional new methods. A class can only extend one parent class. All public methods are inherited (not private). All public instance variables are inherited (not private) Subclasses have indirect access to the parent class's variables - use public getState method to access state (they cannot directly access the private instance state variable), but they can access protected variables.

Encapsulation best practices / inner classes What is encapsulation? Keeping certain data hidden from external use. Why is encapsulation important? You don't want someone changing your object if it is returned in a get method (it could mess up your class) - return a copy instead. Also, it allows you to maintain invarients! You have a BST class? All the elements in a right tree better be greater than all the elements in a left tree. Only allow the client to access this data structure through methods that are guaranteed to maintain this invariant, or someone could break your BST. Simplifies the development process. Implement and then rigorously test one class at a time. Prove beyond a resonable doubt that the given class behaves as expected. If its well tested and encapsulated, clients can't break it, and it is robust. Debug elsewhere.

What should you encapsulate?

- All instance variables should be private
- All helper methods should be private

- When returning objects (arrays, collections, etc), never return actual reference, always return a copy
  - Pay attention to if the copy must be deep, are the contained objects modifiable to the client? Does capsulation exist there as well?
  - imagine a ScatterPlot class that contains a private set of Point objects, each of which has public void setX(int x), and setY(int y) methods. Returning a copy of the set could still allow a client to modify the plot throught the point objects.
- Return a copy of an array, for example, by copying every element into a new array. Same concept for a lists, sets, maps, etc.
- Collections has static methods that take mutable collections and maps and returns immutable versions (without having to copy every element over) i.e: return Collections.unmodifiableSet(someMutableSet);

What is an inner class? Class inside a larger class When to use an inner class? Usually, classes go in their own Java file, but inner classes are sometimes nice when they are not too complicated and tie in directly with the outer class

Java Design Patterns

- What is a factory? Class that returns an instance of an interface without specifying what implementation to use
- Why do we use factories? Because when we test your homework, you have freedom with naming your classes. We only ask that you fill out this factory method so we know what class to use when we test against methods in the interface
- Example of a factory Lightswitch factory has method to return instance of a lightswitch (can be many implementations for this)

Additional Topics

There are a few other topics that you should feel free to ask questions on, that we found students struggle wtih:

- Java Generics
- Abstract Classes vs Interfaces vs Classes
- Iterators and lazy iteration
- Other design patterns: Builder (Quadnode Homework), Facade (project)