1 Administrivia

- Lab 0.1 grades up
- Lab 0.2 on Friday
- Problem Set 1 due next Wednesday
- Find project partners before next Monday

2 Basic Concepts

Class A class is a “template” for an object. It contains a description of all of the variables and methods that objects of that class will be created with.

Instance An instance is a concrete object created at runtime from a class definition. This is known as instantiating an object. It is the job of the constructor to initialize new instances.

Constructor A constructor is a special type of method that is invoked when instances are created. It is generally responsible for initializing the member variables of the instance and any other setup that needs to happen before the instance is usable.

Method A method is simply a function that is defined in a class, either as an instance method or a static method. In Java, all code must reside inside of a method (unlike in Python, for example).

3 Instance vs. Static

<table>
<thead>
<tr>
<th>Associated with</th>
<th>Instance</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can access</td>
<td>Specific instance</td>
<td>Class</td>
</tr>
<tr>
<td>Can be overridden</td>
<td>Instance and static members</td>
<td>Static members</td>
</tr>
<tr>
<td>Useful for</td>
<td>Almost everything</td>
<td>Constants, utility methods, global state</td>
</tr>
</tbody>
</table>

4 Subclasses and Interfaces

<table>
<thead>
<tr>
<th>Expresses</th>
<th>Inheritance</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shares</td>
<td>Substitutability</td>
<td>Capability</td>
</tr>
<tr>
<td>Per class</td>
<td>Method signatures and bodies</td>
<td>Method signatures</td>
</tr>
<tr>
<td></td>
<td>Single</td>
<td>Multiple</td>
</tr>
</tbody>
</table>

- Use inheritance when you want to:
  - Share code between classes
  - Express substitutability, e.g. “a student is a person”
  - Override existing behavior in a class
- Use interfaces when you want to:
  - Enforce a contract other classes
  - Express a capability, e.g. “a cache can store pages”
  - Decouple the runtime type of your instances from calling code

Liskov Substitution Principle If $S$ is a subtype of $T$, then any instance of $T$ should be substitutable with an instance of $S$. 
Listing 1: Example code for demonstrating instance vs. static

```java
import java.util.ArrayList;
import java.util.List;

public class Counter {
    private final static List<Counter> counters = new ArrayList<Counter>();
    private int total;

    public Counter() {
        this.total = 0;
        // we can safely access a static variable from here
        counters.add(this);
    }

    public void increment() {
        this.total++;
    }

    public int getTotal() {
        return this.total;
    }

    public void reset() {
        this.total = 0;
    }

    public static void resetAll() {
        // this code will not compile because total is an instance variable
        // this.total = 0
        for (Counter counter : counters) {
            counter.reset();
        }
    }

    public static void main(String[] args) {
        Counter c1 = new Counter();
        c1.increment();
        System.out.println("Counter 1: " + c1.getTotal());
        Counter c2 = new Counter();
        c2.increment();
        System.out.println("Counter 2: " + c2.getTotal());
        Counter.resetAll();
        System.out.println("Counter 1: " + c1.getTotal());
        System.out.println("Counter 2: " + c2.getTotal());
    }
}
```