Announcements
• Project 2 due 11/01

Material
• Interfaces
• Anonymous classes
Let's see abstractions...

- You have seen “abstractions” in many places, let's consider them from the ground up.
  - Expressions
  - Functions (with no arguments)
  - Functions with arguments
  - Functions in a separate class
  - Functions in an interface

- Each has different forms of reuse
  - compilation, invocation, execution, programs, etc.
Different levels

Expressions

```java
System.out.println(3 + 4);
```

Functions (with no arguments)

```java
function void doAdd() {
    System.out.println(3 + 4);
}
```

Functions with arguments

```java
function int doAdd() {
    return 3 + 4;
}
System.out.println(doAdd());
```

```java
function int doAdd(int a, int b) {
    return a + b;
}
```

Functions in a separate class

```java
class Add {
    function int doAdd(int a, int b) {
        return a + b;
    }
}
Add a = new Add();
System.out.println(a.doAdd());
```
Interfaces one more level

• Famous saying in CS
  • Anything can be done with one more level of indirection

If someone didn't understand their code and its likely uses well enough to write brief useful comments, why should I imagine that they understood it well enough to write code that works? -- Richard A. O'Keefe

When I am working on a problem, I never think about beauty. I think only of how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong. -- R Buckminster Fuller

As soon as we started programming, we found out to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs. -- Maurice Wilkes
LinearSearch Revisited

```java
068:     static boolean linearSearch(Object[] data, Object target) {
069:         
070:         assert(data != null);
071:         assert(target != null);
072:         int index = 0;
073:         boolean found = false;
074:         
075:         while (!found && index < data.length) {
076:             if (data[index].equals(target))
077:                 found = true;
078:             index++;
079:         }
080:         return found;
081:     }
```

Abstract out a comparison between `data[index]` and `target`. Need True if they are same, False otherwise.
Abstraction

- Comparator interface in Java

```java
interface Comparator {
    int compare(Object o1, Object o2);
}
```

- We had to abstract out

```java
if (data[index].equals(target))
```

- Then it becomes

```java
comp.compare(data[index++], target) == 0
```

- Where `comp` is of type Comparator and somehow must be provided
```java
static boolean linearSearch(Object[] data, Object target, Comparator comp) {
    assert data != null; assert target != null; assert comp != null;
    int index = 0;
    boolean found = false;

    while (!found && index < data.length) {
        if (comp.compare(data[index++], target) == 0)
            found = true;
        index++;
    }

    return found;
}
```

Linear w/ Comparator Function Object
Clients of linear search

• Before the change...
  
  if (Search.linearSearch(array, target))
  
  found it

• Now...
  
  if (Search.linearSearch(array, target, comp))
  
  found it

• Where comp.compare() must do this
  
  (data[index].equals(target))

• So what we have is:

  class myCompare implements Comparator
  {
      int compare(Object o1, Object o2) {
          return (o1.compareTo(o2));
      }
  }
class myCompare implements Comparator {
    int compare(Object o1, Object o2) {
        return o1.compareTo(o2);
    }
}

public class Main {
    public static void main(String[] args) {
        Comparator comp = new myCompare();
        // code omitted here
        if (Search.linearSearch(array, target, comp))
            System.out.println("Found it");
        else
            System.out.println("No found");
    }
}
class myCompare implements Comparator {
    int compare(Object o1, Object o2) {
        return o1.compareTo(o2);
    }
}

public class Main {
    public static void main(String[] args) {
        // code omitted here
        if (Search.linearSearch(array, target, new myCompare())) {
            System.out.println("Found it");
        } else {
            System.out.println("No found");
        }
    }
}
```java
public class Main {
    public static void main(String[] args) {
        // code omitted here
        if (Search.linearSearch(array, target,
                new Comparator() {
            int compare(Object o1, Object o2) {
                return o1.compareTo(o2);
            }
        })) {
            System.out.println("Found it");
        } else {
            System.out.println("No found");
        }
    }
}
```