CSE 143
Computer Programming II

Questions From Last Time

- Do you recommend reading the textbook?
- Are TAs allowed to help with “style” at the IPL?
- Is there extra credit?
- Do you like eclipse?
- Do you have to use the “this” keyword?

```java
public class Example {
    int number;
    public incrementNumber() {
        // Both of the following two lines work!
        // this.number++;
        // number++;
    }
}
```

More Questions From Last Time

- How many programmers does it take to change a light bulb? (none, that’s a hardware problem)
- What is your favorite pizza flavor? (I’m not sure. I’ll get back to you on this one.)
- Why is it called Piazza?
- What is the meaning of life? (42)
- Knock Knock (Who’s there?)
- What’s up? How was your day? (the ceiling; good)

Drawings

WTF’s per Minute

http://cocodebot.com
Rubber Ducky, You’re The One!

**Rubber Duck Debugging**

Rubber Duck Debugging is the idea that when your code doesn’t work, you talk to an inanimate object about what it does to find the error.

The idea is to explain what your code is supposed to do vs. what it is doing. Many times, the action of saying it out loud helps solve the problem.

### Why Use Fields Directly vs. Using Instance Methods?

**Why Use Fields Directly?**

- It’s sometimes more readable to use the fields directly
- The code is sometimes shorter

**Why Use Getters and Setters?**

- What happens if we change the implementation (e.g. Point location instead of int x, y)?
- If there is code that checks validity of inputs, then we only put it in one place

### Duplicated Code: Constructors

We’d like to have two constructors for ArrayIntList:

- One that uses a default size
- One that uses a size given by the user

**Redundant Constructors**

1. public ArrayIntList() {
2.   this.data = new int[10];
3.   this.size = 0;
4. }
5. public ArrayIntList(int capacity) {
6.   this.data = new int[capacity];
7.   this.size = 0;
8. }

This is a lot of redundant code! How can we fix it?

**Fixed Constructor**

Java allows us to call one constructor from another using this(...):

```
public ArrayIntList() {
  this(10);
}
```

### Using Fields Directly vs. Using Instance Methods

```
1 public class Circle {
2  int radius;
3  int x, y;
4
5  public Circle() {
6    this.radius = 10;
7    this.x = 0;
8    this.y = 0;
9  }
10 }
```

**Redundant add Methods**

```
1 public void add(int value) {
2   add(this.size, value);
3 }
```

The fix is to call the more general add method from the less general one. (As a rule of thumb, methods with fewer arguments are less general.)

So, we’d replace the first method with:

**Fixed add Method**

```
public void add(int value) {
  add(this.size, value);
}
```

### Duplicated Code: Methods

**Implementing remove**

```
(size = 5) 3 8 2 45 6
```

1. list.remove(2):
2. (size = 3) 3 8 45 6
3. 0 0 0

How do we remove from the middle of the list?

- Shift over all elements starting from the index to remove at
- Set the last element to 0 (Do we need to do this?)
- Decrement the size
Illegal Arguments

1. public class Circle {
2.   int radius;
3.   int x, y;
4.   ...,
5.   public void moveRight(int numberOfUnits) {
6.       this.x += numberOfUnits;
7.   }
8. }
9. 
10. Are there any arguments to moveRight that are "invalid"?
11. Yes! We shouldn't allow negative numbers.
12. The implementor is responsible for (1) telling the user about invalid ways to use methods and (2) preventing a malicious user from getting away with using their methods in an invalid way!

Illegal Arguments

1. public class Circle {
2.   int radius;
3.   int x, y;
4.   ...,
5.   public void moveRight(int numberOfUnits) {
6.       this.x += numberOfUnits;
7.   }
8. }
9. 
10. Are there any arguments to moveRight that are "invalid"?
11. Yes! We shouldn't allow negative numbers.
12. The implementor is responsible for (1) telling the user about invalid ways to use methods and (2) preventing a malicious user from getting away with using their methods in an invalid way!

Preconditions

A precondition is an assertion that something must be true for a method to work correctly. The objective is to tell clients about invalid ways to use your method.

Example Preconditions:

- For moveRight(int numberOfUnits):
  ```java
  // pre: numberOfUnits >= 0
  ```

- For minElement(int[] array):
  ```java
  // pre: array.length > 0
  ```

- For add(int index, int value):
  ```java
  // pre: capacity >= size + 1, 0 <= index <= size
  ```

Preconditions are important, because they explain method behavior to the client, but they aren't enough! The client can still use the method in invalid ways!

Why Use Exceptions?

Exceptions prevent the client from accidentally using the method in a way it wasn't intended. They alert them about errors in their code!

An Example

```java
public void set(int index, int value) {
    if (index < 0 || index >= size) {
        throw new IndexOutOfBoundsException(index);
    }
    this.data[index] = value;
}
```

Uh oh! We have MORE redundant code!

Private Methods

A private method is a method that only the implementor can use. They are useful to abstract out redundant functionality.
Hitting Capacity

Example ArrayList

Client View: 29 1 3 9 8

Impl. View: 29 1 3 9 8

Let’s run add(3, 8)! Uh oh! There’s no space left. What do we do?

Create a new array of double the size, and copy the elements!

Resizing (Implementor View)

Before: 29 1 3 9 8

Resize: 29 1 3 9 8 0 0 0 0

Insert: 29 1 3 8 9 8 0 0 0 0

Arrays Reference

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarySearch(array, val)</td>
<td>Returns the index of val in array if array is sorted; (or &lt; 0 if not found)</td>
</tr>
<tr>
<td>toString()</td>
<td>Returns a string representation of the array such as [3, 42, -7, 15]</td>
</tr>
<tr>
<td>sort(array)</td>
<td>Sorts the elements of array (this edits the original array!)</td>
</tr>
<tr>
<td>copyOf(array, len)</td>
<td>Returns a new copy of array with length len</td>
</tr>
<tr>
<td>equals(array1, array2)</td>
<td>Returns true precisely when the elements of array1 and array2 are identical (according to .equals)</td>
</tr>
</tbody>
</table>

Call these with Arrays.method(arg1, arg2, ...)

Postconditions

A postcondition is an assertion that something must be true after a method has run. The objective is to tell clients what your method does.

Example Postconditions:

- For moveRight(int numberOfUnits):
  // post: Increases the x coordinate of the circle by numberOfUnits

- For minElement(int[] array):
  // post: returns the smallest element in array

- For add(int index, int value):
  // post: Inserts value at index in the ArrayList; shifts all elements from index to the end forward one index; ensures capacity of ArrayList is large enough

Postconditions are important, because they explain method behavior to the client.