Questions From Last Time

1. Can you cover which guidelines are being used for style?
2. What is the type of an ArrayList that has 3.5, 2.7, etc. (ArrayList<Double>)
3. I took CSE 142 (or equivalent) a long time ago. What should I do? What do I need to know from 142?
4. This is a test to see if you actually read all of the notecards.
5. Do different companies have different style guidelines? (Yes.)
6. What defines a “good comment”? (length? can they be too long?)
7. What are design decisions? (See Piazza)

Questions From Last Time

1. Coming to class late? Leaving early?
2. Are exams open note? (No, but we give you a cheatsheet attached to the exam)
3. Is the class curved?
4. Lighter color on slides?
5. JGrasp vs. Eclipse?
6. Is there a style guide?
7. Is there such a thing as too many comments?
Wrapper Classes

int vs. Integer   char vs. Character   double vs. Double

The lowercase versions are primitive types; the uppercase versions are "wrapper classes".

The following is valid code:
1 int a = 5;
2 Integer b = 10;
3 int c = a + b; //You can treat ints and Integers as the same

When we create ArrayList’s, we must use non-primitive types. So:
1 ArrayList<int> bad1 = new ArrayList<int>(); // This won’t compile!
2 // v This will work.
3 ArrayList<Integer> better = new ArrayList<Integer>();
4 better.add(5); // We can add an 'int' to an 'Integer' ArrayList

Clients and Implementors

Client vs. Implementor: Medication
For a tylenol pill, who is the client? Who is the implementor?

Java Examples
You’ve already been a client!
■ DrawingPanel
■ ArrayList
You’ve already been an implementor!
■ Critter

Classes, Objects, and Instances

Class
A Class is
■ a complete program, or
■ a "template" for a type
(Examples: ArrayList, ReverseFile, ...)

The class explains what an object is, an instance is a particular version of the object.

1 ArrayList<String> list1 = new ArrayList<String>();
2 ArrayList<String> list2 = new ArrayList<String>();
3 //list1 and list2 are instances of ArrayList

Object
An Object combines state and behavior.
Java is an "object-oriented" programming language (OOP); programs consist of objects interacting with each other.

Example Class

A class is made up of field(s), constructor(s), and method(s).
Let’s make an object Circle that represents a circle...
■ with a size
■ that can be moved right
■ at a particular location
1 public class Circle {
2 // Fields
3 private int radius;
4 private int x;
5 private int y;
6 }
7 // Constructor
8 public Circle(int radius, int x, int y) {
9  this.radius = radius;
10  this.x = x;
11  this.y = y;
12 }
13 // Methods
14 public void moveRight(int numberOfUnits) {
15  this.x += numberOfUnits;
16 }
17 }
18

Implementor View of ArrayList

What behavior should we support? (Methods)
add, remove, indexOf, etc.

What state do we keep track of? (Fields)
■ Elements stored in the ArrayList (probably stored as an array!)
■ Size of ArrayList

Two Views of an ArrayList

Client View: 3 -23 -5 222 35 ...
Impl. View: 3 -23 -5 222 35 0 0 0

ArrayIntList

■ No generics (only stores ints)
■ Fewer methods: add(value), add(index, value), get(index),
set(index, value), size(), isEmpty(), remove(index),
indexOf(value), contains(value), toString()
Implementing add

(size = 4) | 3 | 8 | 2 | 45 | 0 | 0 | 0 | 0
---|---|---|---|---|---|---|---|---

```
ls = [3, 8, 2, 45, 0, 0, 0, 0]
```

```
lst.add(222):
(size = 5) | 3 | 8 | 2 | 45 | 222 | 0 | 0 | 0
```

How do we add to the end of the list?
- Put the element in the last slot
- Increment the size

```
1 public void add(int value) {
2     this.data[size] = value;
3     size++;
4 }
```

Printing an ArrayIntList

System.out.println automatically calls toString on the given object.
toString looks like:

```
1 public String toString() {
2     if (size == 0) {
3         return "[]";
4     } else {
5         String result = "[" + this.data[0];
6         for (int i = 1; i < this.size; i++) {
7             result += ", " + this.data[i];
8         }
9         result += "]";
10        return result;
11    }
12 }
```

Implementing add #2

(size = 4) | 3 | 8 | 2 | 45 | 0 | 0 | 0 | 0
---|---|---|---|---|---|---|---|---

```
l = [3, 8, 2, 45, 0, 0, 0, 0]
```

```
list.add(1, 222):
(size = 5) | 5 | 222 | 8 | 2 | 45 | 0 | 0 | 0
```

How do we add to the middle of the list?
- Shift over all elements starting from the end
- Put the new element in its index
- Increment the size

```
1 public void add(int index, int value) {
2     for (int i = size; i > index; i--) {
3         this.data[i] = this.data[i - 1];
4     }
5     this.data[index] = value;
6     size++;
7 }
```

Today’s Takeaways!

- Understand the difference between client and implementor
- Always use wrapper classes when creating an ArrayList of a primitive type
- Understand how ArrayList is implemented