Adam Blank Lecture 2 Autumn 2016

CSE 143

Computer Programming II

CSE 143: Computer Programming II

ArrayIntList

Questions From Last Time

- Is the book required?
  - I want you to do what's best for you. I rarely recommend textbooks in classes I teach, but I actually like this one a lot.
- How do I review CSE 142?
  - Review session on Monday at 3:30pm - 5:00pm in EEB 105
  - http://courses.cs.washington.edu/courses/cse143/16sp/lectures/03-28/slides/review.pdf
  - http://practiceit.cs.washington.edu/problem/list
- Is it bad form to make an ArrayList where an Array would do?
  - Yes. We'll get there, but the short answer is "if you know the size upfront, using an ArrayList is a waste".
- How much help on HW can we get from the IPL and office hours?
  - Great question. We won't "pre-grade" your homework, but any other sorts of questions are fair game. If you ask a question, the worst that can happen is we'll say "we can't answer that".
- How can I help others without cheating?
  - You can work together on anything that isn't homework. Practice-it, practice exams, lecture problems, ...

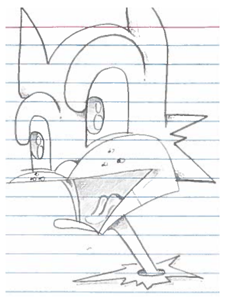
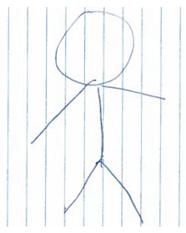
Questions From Last Time

- emacs or vi?
  - vim. full stop.
- Favorite programming language?
  - It depends on the situation. For quick and dirty scripts, Python. For teaching intro, Java. For writing clean code, Haskell. ...
- Favorite color?
  - Green (see slides.)
- What's your favorite candy?
  - Not sure. I have celiac disease; so, I'm severely limited in what I can eat.
- Macbook or surface?
  - My main machine is a macbook, but I lecture from the surface. I'd recommend them both!
- Do you want to jump rope in class?
  - Nope. I'm good. Do you?
- Does CSE have a room "CSE 404"?
  - It doesn't. I have joked about this before.

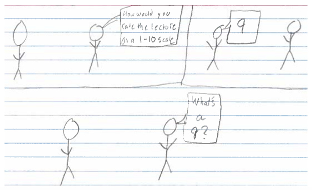
Questions From Last Time

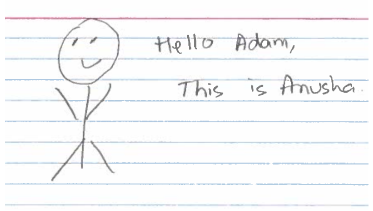
- At what level should my pikachu be before I use a thunderstone to evolve it?
  - http://www.gamefaqs.com/boards/696959-pokemon-x/67478014
  - http://www.gamefaqs.com/boards/696959-pokemon-x/69305531
  - http://pokemondb.net/pokebase/84886/when-is-a-good-time-to-evolve-pikachu-into-raichu
  - http://www.pokecommunity.com/showthread.php?t=173760
  - http://www.pokecommunity.com/showthread.php?t=173760

Drawings

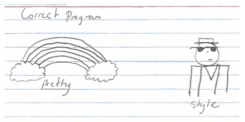



Drawings (continued)

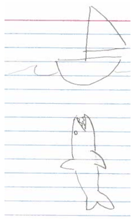




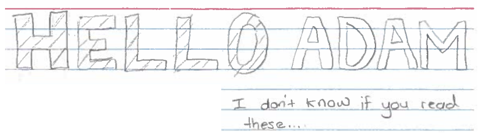
Drawings (continued)



Drawings (continued)



Drawings (continued)



An `ArrayList` Client

```java
public class ArrayIntListClient {
   // Notice the main! That's usually a signal that we're a client!
   public static void main(String[] args) {
      // BUG: We haven't specified what "type" the list contains!
      ArrayList list = new ArrayList();
      for (int i = 0; i < 20; i++) {
         list.add(i);
      }
     System.out.println(list);
   }
}
```

Today's Goal

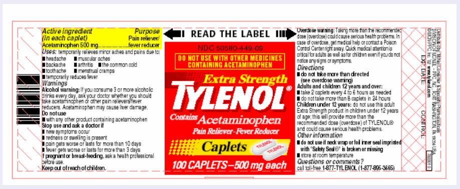
Replace Java's `ArrayList` with our own version!

Another way of saying this: we will **implement** `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`

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:

```java
int a = 5;
Integer b = 10;
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:

```java
ArrayList<int> bad1 = new ArrayList<int>(); // This won't compile!
// v This will work.
ArrayList<Integer> better = new ArrayList<Integer>();
better.add(5); // We can add an 'int' to an 'Integer' ArrayList
```

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.

```java
ArrayList<String> list1 = new ArrayList<String>();
ArrayList<String> list2 = new ArrayList<String>()
//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

```java
public class Circle {
   /* Fields */
   private int radius;
   private int x;
   private int y;

   /* Constructor */
   public Circle(int radius, int x, int y) {
      this.radius = radius;
      this.x = x;
      this.y = y;
   }

   /* Methods */
   public void moveRight(int numberOfUnits) {
      this.x += numberOfUnits;
   }
}
```

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	···
0	1	2	3	4	

Impl. View:

3	-23	-5	222	35	0	0	0
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]	arr[5]	arr[6]	arr[7]

`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
lst[0]	lst[1]	lst[2]	lst[3]	lst[4]	lst[5]	lst[6]	lst[7]

`lst.add(222)`:
(size = 5)

3	8	2	45	**222**	0	0	0
lst[0]	lst[1]	lst[2]	lst[3]	lst[4]	lst[5]	lst[6]	lst[7]

How do we add to the end of the list?

- Put the element in the last slot
- Increment the size

```java
public void add(int value) {
   this.data[this.size] = value;
   this.size++;
}
```

Printing an `ArrayIntList`

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

```java
public String toString() {
   ...
}
```

`ArrayIntList toString`:

```java
public String toString() {
   if (this.size == 0) {
      return "[]";
   }
   else {
      String result = "[" + this.data[0];
      for (int i = 1; i < this.size; i++) {
         result += ", " + this.data[i];
      }
      result += "]";
      return result;
   }
}
```

Implementing `add` #2

(size = 4)

3	8	2	45	0	0	0	0
list[0]	list[1]	list[2]	list[3]	list[4]	list[5]	list[6]	list[7]

`list.add(1, 222)`:

(size = 5)

3	**222**	**8**	**2**	**45**	0	0	0
list[0]	list[1]	list[2]	list[3]	list[4]	list[5]	list[6]	list[7]

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

```java
public void add(int index, int value) {
   for (int i = this.size; i > index; i--) {
      this.data[i] = this.data[i - 1];
   }
   this.data[index] = value;
   this.size++;
}
```

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