Adam Blank | Lecture 2 | Winter 2015

CSE 143

Computer Programming II

CSE 143: Computer Programming II

ArrayIntList

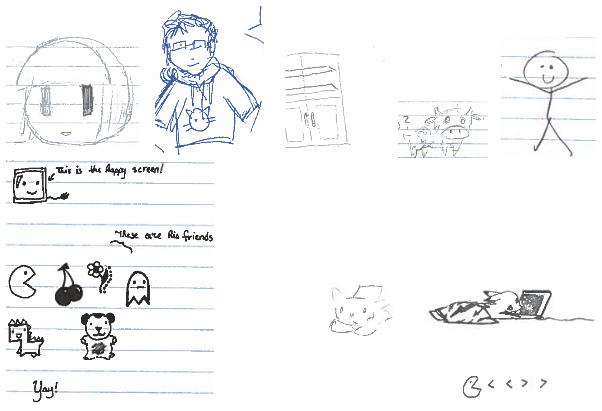
Questions From Last Time

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

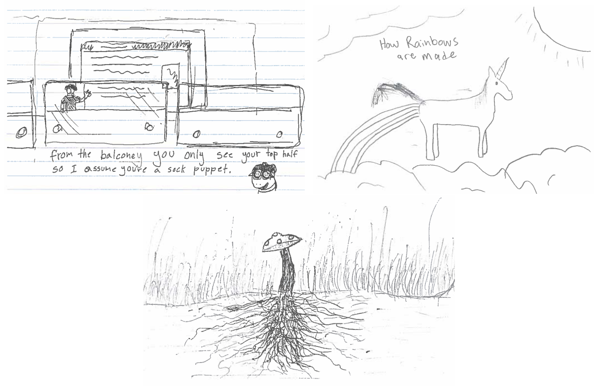
Questions From Last Time

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

Drawings



Drawings (continued)


From the balcony you only see your top half so I assume you're a sock puppet.
How Rainbows are made

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
```

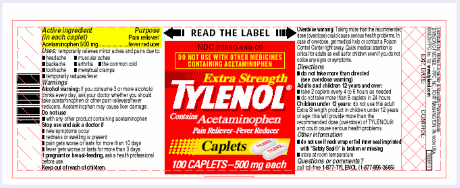
When we create `ArrayList`’s, we must use **non-primitive types**. So:

```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
```

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.

```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[size] = value;
   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 (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 = size; i > index; i--) {
      this.data[i] = this.data[i - 1];
   }
   this.data[index] = value;
   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