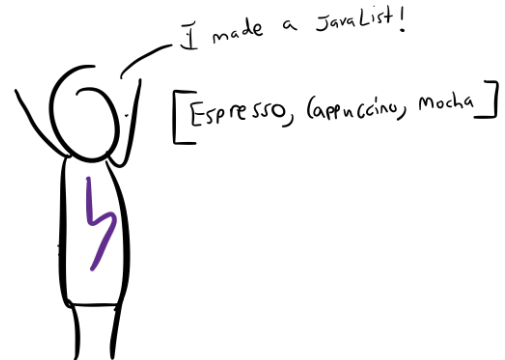


CSE 143

Computer Programming II

ArrayList



Questions From Last Time

1

- Increase Text Size in JGrasp (done!)
- I took CSE 142 a long time ago. What should I do?
We're holding a review session of CSE 142 material sometime at the end of this week or the beginning of next week!
- Will slides be online? (yup!)
- Will programs from lecture be posted? (yup!)
- Can you repeat questions out loud? (yes, sorry!)
- Where is the IPL? (MGH room 334 & 342)
- What is your favorite color? (green)
- "Hello" (Hi!)



Wrapper Classes

2

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

3

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

4

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

5

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
14    /* Methods */
15    public void moveRight(int numberOfUnits) {
16        this.x += numberOfUnits;
17    }
18 }

```

Implementor View of ArrayList

6

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

7

- 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

8

(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(222):

(size = 5)

3	8	2	45	222	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 end of the list?

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

```

1 public void add(int value) {
2     list[size] = value;
3     size++;
4 }

```

Printing an ArrayIntList

9

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

```

1 public String toString() {
2     ...
3 }

```

ArrayIntList toString:

```

1 public String toString() {
2     if (size == 0) {
3         return "";
4     }
5     else {
6         String result = "[" + list[0];
7         for (int i = 1; i < size; i++) {
8             result += ", " + list[i];
9         }
10        result += "]";
11        return result;
12    }
13 }

```

Implementing add #2

10

(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

```

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

```