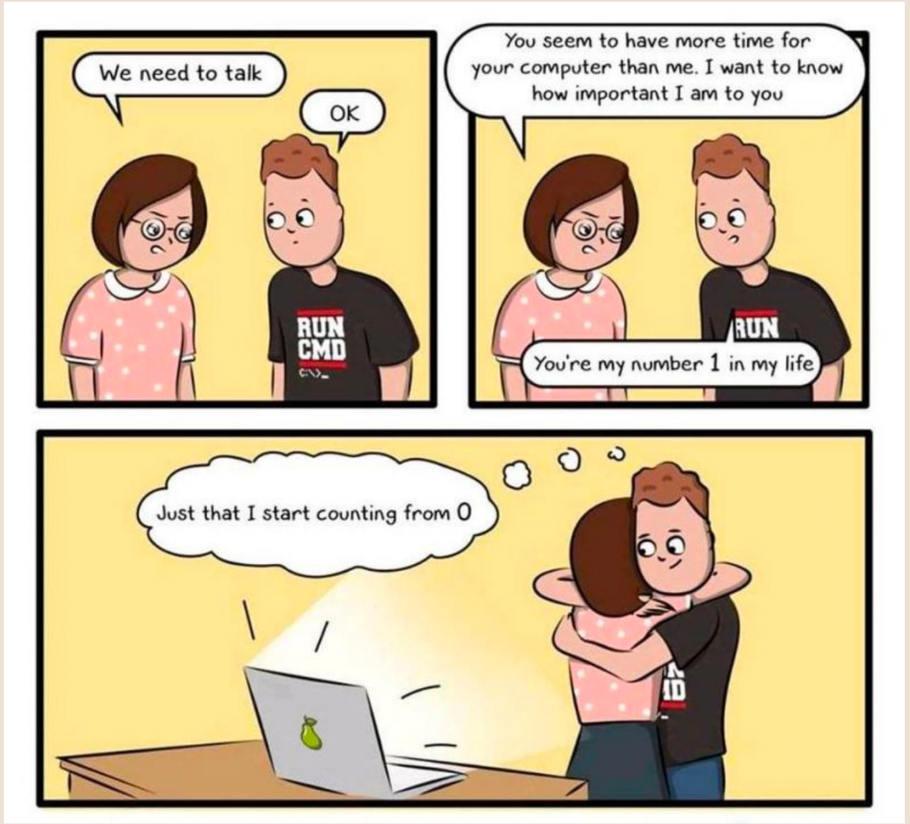
Topic 5.0: ArrayLists & Collections

Learning Goals (Week 5):

- **Create instances of a built-in Java data type such as an ArrayList.**
- **Use instances of a built-in Java data type such as an ArrayList.**
- **Compare and contrast arrays and a Java-defined data type.**
- Use wrapper classes to manipulate primitive types as objects.

Reminder: Partially Filled Arrays

- Array of pre-set maximum size and an integer to help us track the current number of elements being stored
- Cons:
  - size limitations
  - may need to be constantly shifting data in it (lots of loops)
  - constantly tracking two variables for a single array



devrant

ArrayLists

- Java has a built **dynamic** array called ArrayList
- works a bit like a partially-filled array but much easier for us!
- *import java.util.ArrayList* is required (just like when we needed java.util.io for File I/O stuff

```
ArrayList<String> arrlist = new ArrayList<String>();
```

- Those weird <> are called generics but we won't be covering generics as a whole in this class.
- Defined like this:

```
ArrayList arrlist = new ArrayList();
```

- Let's us store any type of data under the Object umbrella
  - it's functionally the same as writing:

```
ArrayList<Object> arrlist = new ArrayList<Object>();
```

ArrayLists: One major “con”

- ArrayLists can **ONLY** hold Objects in it, not primitive types

```
ArrayList<String> // is OK
ArrayList<int>    // will cause error
```

- Can’t use int, double, boolean, char, float, long, byte, short

ArrayLists: One major “con”

- ArrayLists can **ONLY** hold Objects in it, not primitive types

```
ArrayList<String> // is OK
ArrayList<int>    // will cause error
```

- Can’t use int, double, boolean, char, float, long, byte, short

ArrayLists: Not actually a con

- Those primitive types can be **wrapped** in classes that represent these primitive types
  - Integer, Double, Boolean, Character, Long are classes that give Object “versions” of the primitive types, allowing them to be used in ArrayLists, too!

Objects CRUD

- Whenever we think about objects, we should think of the acronym **CRUD**

Objects CRUD

- Whenever we think about objects, we should think of the acronym **CRUD**

Create

Read

Update

Delete

- These are the things we should be able to do with objects.
- Arrays are objects
  - create: int[] example = new int[10];
  - read: System.out.println(example[0]);
  - update: example[0] = 100;
  - delete element: (shift everything over, overwrite it, etc)
  - delete array: (set to null, use ‘new’ again)

MyArrayList: CRUD

- Let’s understand what ArrayLists are secretly doing under the java hood by writing our own!
- Create a Java class named MyArrayList with An array of objects to store the elements and an integer variable to keep track of the number of elements currently in the array
- Create a default constructor for your MyArrayList class that initializes the array with a fixed initial capacity (e.g., 10) and sets the size to 0. Add the following methods to your class:
- public void add(Object element): This method should add the given element to the end of the partially filled array. If the array is full, you should dynamically resize it to accommodate more elements. (**reminder: using Object is like the umbrella for all datatypes so you can store ANYTHING in your class this way, just like a real ArrayList)**
- public Object get(int index): This method should return the element at the specified index.
- public void remove(int index): This method should remove the element at the specified index and shift the remaining elements to fill the gap.
- public int size(): This method should return the current number of elements in the list.
- public boolean isEmpty(): This method should return true if the list is empty (size is 0), and false otherwise.

ArrayLists: The actual class

Let’s look at some methods:

- add(Object e), add(int i, Object e)
  - returns True always

```
ArrayList<String> a = new ArrayList<String>( );
a.add("testing"); //adds to the end
a.add("hippo"); //now "testing" "hippo"
a.add(1, "second"); //add “second” to index 1
// now a = "testing" "second" "hippo"
a.add(10,"far"); //IndexOutOfBoundsException: can't leave "gaps"
```

ArrayLists: The actual class

Let’s look at some methods:

- size()
  - arrays (.length) and Strings (.length()) #sorry

```
ArrayList<String> a = new ArrayList<String>( );
a.add("testing"); //adds to the end
a.add("hippo"); //now "testing" "hippo"
int size = a.size(); // 2
```

ArrayLists: The actual class

Let’s look at some methods:

- remove(int i), remove(Object e)
  - index one returns deleted element, Object one returns true/false

```
a.remove(0); //Removes the first element.
             //All others move left one place.
            //returns the deleted element
a.remove("hippo"); //removes that String
                   //returns a boolean ("was it there?")
a.remove(10); //IndexOutOfBoundsException
a.clear(); //a is now empty. This method is void
```

ArrayLists: The actual class

Let’s look at some methods:

- get(int i)/set(int i , Object e)
  - get works just like array[i]
  - set works just like array[i] = e BUT returns the old element

```
a.get(0)  //Just like a[0] would be for an array
a.get(a.size()-1)  //gets the last one.
a.set(0, "new") // replaces the first one.
                //returns the old value that was deleted.
a.set(10, "new") // IndexOutOfBoundsException
```

Remember: Since ArrayLists contain only Objects, the result is always a reference to an Object

ArrayLists: The actual class

Let’s look at some methods:

- printing (a toString goes through each element and prints it out nicely)
  - if storing your own objects, it will call your object’s toString() for each element

```
a.toString( )
System.out.println(a) //This uses toString( ), too.
```

ArrayLists: The actual class

- Searching for things in ArrayLists:

```
int indexOf(Object e)
// returns the position of the first occurrence of that object, or -1
```

```
int lastIndexOf(Object e)
// searches from the other end
```

- Both of these call the object's *boolean equals(Object el)* method, if it doesn't exist, defined JUST like that (Object parameter) it uses ==

ArrayLists: The actual class

- Searching for things in ArrayLists:

```
boolean contains(Object e)
// Simply detects whether it's there or not
// equivalent to:
    // indexOf(Object) >= 0
```

ArrayLists vs Arrays: Method Overview

Array	ArrayList
String[] a = new String[10];	ArrayList<String> a = new ArrayList<String>();
a.length //cannot change	a.size( ) //changes after each modification
a[0]	a.get(0)
a[0] = “test”	a.set(0, "test")
….?	a.add("new")
…?	a.remove(0)
Contains any type	Contains objects only

Pause & Practice (with me)

- Example 1: Build an alphabetical list of words
- Example 2 : Remove duplicates from a list of words