# What if Arrays Weren't So Rigid?

Or: ArrayLists.

## Rules of Arrays

- Once an array object is declared, its size can't change.
- An array can only contain elements of one declared type.
- Array elements are accessed/modified using `[]`, and we get array length with `.length`

## Rules of ArrayLists

- **ArrayLists can change size to accomodate more elements.**
- An ArrayList can only contain elements of one declared type.
- ArrayList elements are accessed/modified with **methods**, and that's also how we check the number of elements.

## Declaring ArrayLists

In order to declare a variable for an ArrayList and initialize an empty one:

```
ArrayList<DataType> values = new ArrayList<DataType>();
```

- In the angle brackets ( `<>` ) goes the data type that the ArrayList will store
  - For example, we say that `ArrayList<String>` is an ArrayList of `Strings`.
  - For now, the contents of the angle brackets on the left and right should match.

## ArrayList types

ArrayLists cannot support primitive types like `int`, `double`, `boolean`, or `char`.
Java has a bunch of *wrapper* classes that can be used instead.

| Wrong! | Right. |
|---|---|
| `ArrayList<int>` | `ArrayList<Integer>` |
| `ArrayList<double>` | `ArrayList<Double>` |
| `ArrayList<char>` | `ArrayList<Character>` |
| `ArrayList<boolean>` | `ArrayList<Boolean>` |

## Adding to an ArrayList

`add(T element)` adds the value `element` to the ArrayList, making sure there's space for it first.

```java
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
System.out.println(staffNames);
```

➡️ `["Adrian", "Dana", "Jintong", "Philipp", "Vivian"]`

## Did you catch that?

🚨🚨🚨🚨🚨

## You can print out the contents of an ArrayList with a single call to `System.out.println()`!

🚨🚨🚨🚨🚨

## Anyway...

`add(T element)` adds the value `element` to the ArrayList, making sure there's space for it first.

- Elements are added to the end
- The ArrayList can grow arbitrarily large, although there are occasionally performance penalties when growing the List.

## Getting from an ArrayList

`get(int index)` returns the element at the specified location. Valid indices range from `0` to `list.size() - 1` (just like in arrays.)

```java
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
System.out.println(staffNames.get(2));
```

➡ `"Jintong"`

# Getting from an ArrayList

`get(int index)` returns the element at the specified location. Valid indices range from `0` to `list.size() - 1` (just like in arrays.)

```
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
System.out.println(staffNames.get(0));
```

➡️ `"Adrian"`

# Getting from an ArrayList

`get(int index)` returns the element at the specified location. Valid indices range from `0` to `list.size() - 1` (just like in arrays.)

```java
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
System.out.println(staffNames.get(-1));
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException:
Index -1 out of bounds for length 5 🚨🚨🚨🚨
```

## Getting from an ArrayList

`get(int index)` returns the element at the specified location. Valid indices range from `0` to `list.size() - 1` (just like in arrays.)

```java
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
System.out.println(staffNames.get(10));
```

```
Exception in thread "main" java.lang.IndexOutOfBoundsException:
Index 10 out of bounds for length 5 🚨🚨🚨🚨
```

## Updating Values in an ArrayList

`set(int index, T element)` sets the value at the specified location to be `element` . Valid indices range from `0` to `list.size() - 1` still.

```
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
staffNames.set(2, "Evil Jintong")
System.out.println(staffNames);
```

➡️ `["Adrian", "Dana", "Evil Jintong", "Philipp", "Vivian"]`

## Updating Values in an ArrayList

`set(int index, T element)` actually returns the value that's being replaced, if you want it.

```
ArrayList<String> staffNames = new ArrayList<String>();
staffNames.add("Adrian");
staffNames.add("Dana");
staffNames.add("Jintong");
staffNames.add("Philipp");
staffNames.add("Vivian");
String oldName = staffNames.set(2, "Evil Jintong")
System.out.println(oldName);
```

➡️ `"Jintong"`

## The Length of an ArrayList

`size()` returns the number of elements inside of the specified ArrayList.

```
ArrayList<String> staffNames = new ArrayList<String>();
System.out.print(staffNames.size() + " ");
staffNames.add("Adrian");
System.out.print(staffNames.size() + " ");
staffNames.add("Dana");
System.out.print(staffNames.size() + " ");
```

➡️ `0 1 2`

# There's more where that came from...

Check out the documentation if you want to use other methods of an ArrayList. Here are a few handy ones.

| Method | Return Type | Purpose |
|---|---|---|
| `add(int index, T element)` | void | Put an element at a specific place in the ArrayList |
| `clear()` | void | Remove all elements from the list |
| `remove(int index)` | element type | Delete and return whatever lives at the specified index. |

## Iterating over ArrayLists

It's pretty similar to array iteration

```java
ArrayList<Rectangle> shapes = new ArrayList<Rectangle>();
// assume that we add some stuff to the ArrayList here!

for (int i = 0; i < shapes.size(); i++) {
    Rectangle currentElement = shapes.get(i);
    System.out.println("shapes has " + currentElement + " at index " + i);
}
```

Remember to use `.size()` instead of `.length`, though.

## Enhanced Iteration over ArrayLists

You can also use the *Enhanced For Loop* to iterate over ArrayLists

```
ArrayList<Rectangle> shapes = new ArrayList<Rectangle>();
// assume that we add some stuff to the ArrayList here!

for (Rectangle currentElement : shapes) {
    System.out.println("currentElement is now " + currentElement);
}
```

The iteration behavior is the same: it'll proceed from index `0` to `shapes.size() - 1`.
Technically, you can do this over plain old arrays, too.

## Problem: Acceptable Names

Given an ArrayList of names, make sure that they start with an uppercase letter. If they don't, print that you're fixing the name and then modify the list to have the correctly formatted name.

You'll find that `Character.isUpperCase(char c)` and `Character.toUpperCase(char c)` will come in handy.

Make sure to change the ArrayList *in-place*, which is to say that you shouldn't create a new ArrayList to complete this task.

## Solution:

```java
public void fixFormatting(ArrayList<String> names) {
    for (int i = 0; i < names.size(); i++) {
        String currentName = names.get(i);
        char firstLetter = currentName.charAt(0);
        if (!Character.isUpperCase(firstLetter)) {
            String fixedName = Character.toUpperCase(firstLetter) + currentName.substring(1);
            names.set(i, fixedName);
            System.out.println("Fixing name " + currentName + " at position " + i + " to " + fixedName);
        }
    }
}
```

## Problem: Concatenate

Given two ArrayLists, create a new ArrayList containing first the values from the first ArrayList followed by the values from the second ArrayList

Make sure to create and return a new ArrayList

```
public ArrayList<Integer> concatenate(ArrayList<Integer> first, ArrayList<Integer> second) {

}
```

## Solution:

```java
public ArrayList<Integer> concatenate(ArrayList<Integer> first, ArrayList<Integer> second) {
    ArrayList<Integer> newList = new ArrayList<Integer>();

    for (int i : first) {
        newList.add(i);
    }
    for (int i : second) {
        newList.add(i);
    }
    return newList;
}
```

## Problem: Weave

Given two ArrayLists, create a new ArrayList with the values from the input lists woven together like so:

```
a = [1, 3, 5, 7];
b = [2, 4, 6, 8, 10, 12];
weave(a, b) --> [1, 2, 3, 4, 5, 6, 7, 8, 10, 12]
```

Make sure to create and return a new ArrayList.

```java
public ArrayList<Integer> weave(ArrayList<Integer> first, ArrayList<Integer> second) {

}
```

## Solution:

```
public ArrayList<Integer> weave(ArrayList<Integer> first, ArrayList<Integer> second) {
    ArrayList<Integer> newList = new ArrayList<Integer>();
    int firstIdx = 0;
    int secondIdx = 0;
    while (firstIdx < first.size() || secondIdx < second.size()) {
        if (firstIdx < first.size()) {
            newList.add(first.get(firstIdx));
            firstIdx++;
        }
        if (secondIdx < second.size()) {
            newList.add(second.get(secondIdx));
            secondIdx++;
        }
    }
    return newList;
}
```