ARRAYLISTS

Introduction

When we work with arrays in Java, we've been limited by the fact that once an array is created, it has a fixed size. We can't add or remove elements.

But what if we needed to add to the book lists, newsfeeds, and other structures we were using arrays to represent?

To create mutable and dynamic lists, we can use Java's

`ArrayList`

class. `ArrayList` allows us to:

- Store object references as elements
- Store elements of the same type (just like arrays)
- Access elements by index (just like arrays)
- Add elements
- Remove elements

Remember how we had to import `java.util.Arrays` in order to use additional array methods? To use an `ArrayList` at all, we need to import them from Java's `util` package as well:

```
import java.util.ArrayList;
```

Let's learn how to make use of this powerful object...

Instructions

1. In **Shopping.java** we've defined two arrays:
    - `groceryItems`, a `String` array
    - `prices`, a `double` array

We've tried to add a new item to the end of each. Run the code — does it work?

```
import java.util.Arrays;

class Shopping {

  public static void main(String[] args) {

    String[] groceryItems = {"steak", "milk", "jelly beans"};
    double[] prices = {25.00, 2.95, 2.50};

    // Adding ham to the groceries
    groceryItems[3] = "ham";
    prices[3] = 4.99;

  }

}
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds
for length 3
        at Shopping.main(Shopping.java:11)
```

Creating ArrayLists

To create an `ArrayList`, we need to declare the type of objects it will hold, just as we do with arrays:

```
ArrayList<String> babyNames;
```

We use angle brackets `<` and `>` to declare the type of the `ArrayList`. These symbols are used for *generics*. Generics are a Java construct that allows us to define classes and objects as parameters of an `ArrayList`. For this reason, we can't use primitive types in an `ArrayList`:

```
// This code won't compile:
ArrayList<int> ages;

// This code will compile:
ArrayList<Integer> ages;
```

The `<Integer>` generic has to be used in an `ArrayList` instead. You can also use `<Double>` and `<Character>` for types you would normally declare as `double`s or `char`s.

We can initialize to an empty `ArrayList` using the `new` keyword:

```
// Declaring:
ArrayList<Integer> ages;
// Initializing:
ages = new ArrayList<Integer>();

// Declaring and initializing in one line:
ArrayList<String> babyNames = new ArrayList<String>();
```

Instructions

1.Import the `ArrayList` package from `java.util`.

2.Create a new `ArrayList` that will contain `String` elements and call it `toDoList`.

```
// import the ArrayList package here:
import java.util.ArrayList;
class ToDos {

  public static void main(String[] args) {

    // Create toDoList below:
    ArrayList<String> toDoList = new ArrayList<String>();
}}
```

Adding an Item

Now we have an empty `ArrayList`, but how do we get it to store values?

`ArrayList` comes with an `add()` method which inserts an element into the structure. There are two ways we can use `add()`.

If we want to add an element to the end of the `ArrayList`, we'll call `add()` using only one argument that represents the value we are inserting. In this example, we'll add objects from the `Car` class to an `ArrayList` called `carShow`:

```
ArrayList<Car> carShow = new ArrayList<Car>();

carShow.add(ferrari);
// carShow now holds [ferrari]
carShow.add(thunderbird);
// carShow now holds [ferrari, thunderbird]
carShow.add(volkswagen);
// carShow now holds [ferrari, thunderbird, volkswagen]
```

If we want to add an element at a specific index of our `ArrayList`, we'll need two arguments in our method call: the first argument will define the index of the new element while the second argument defines the value of the new element:

```
// Insert object corvette at index 1
carShow.add(1, corvette);
// carShow now holds [ferrari, corvette, thunderbird, volkswagen]

// Insert object porsche at index 2
carShow.add(2, porsche);
// carShow now holds [ferrari, corvette, porsche, thunderbird,
volkswagen]
```

By inserting a value at a specified index, any elements that appear after this new element will have their index value shift over by 1.

Also, note that an error will occur if we try to insert a value at an index that does not exist.

Keep Reading: AP Computer Science A Students

When using `ArrayList` methods (like `add()`), the reference parameters and return type of a method must match the declared element type of the `ArrayList`. For example, we cannot add an `Integer` type value to an `ArrayList` of `String` elements.

We've discussed how to specify the element type of an `ArrayList`; however, it is possible to create an `ArrayList` that holds values of different types.

In the following snippet, `assortment` is an `ArrayList` that can store different values because we do not specify its type during initialization.

```
ArrayList assortment = new ArrayList<>();
assortment.add("Hello"); // String
assortment.add(12); // Integer
assortment.add(ferrari); // reference to Car
// assortment holds ["Hello", 12, ferrari]
```

In this case, the items stored in this `ArrayList` will be considered `Objects`. As a result, they won't have access to some of their methods without doing some fancy casting. Although this type of `ArrayList` is allowed, using an `ArrayList` that specifies its type is preferred.

Instructions

1. We've created an empty `ArrayList` called `toDoList`. Time to add some to-dos!

Below where we've initialized `toDo1`, initialize two new `String` variables: `toDo2` and `toDo3`.

Set their values to any tasks you like.

2. Use `.add()` to add `toDo1`, `toDo2`, and `toDo3` to `toDoList`.

```
import java.util.ArrayList;

class ToDos {

  public static void main(String[] args) {

    ArrayList<String> toDoList = new ArrayList<String>();
    String toDo1 = "Water plants";
    // Add more to-dos here:
    String toDo2 = "fuck like horse";
    String toDo3 = "my bottle is red";


    // Add to-dos to toDoList
    toDoList.add(toDo1);
       toDoList.add(toDo2);
          toDoList.add(toDo3);
    System.out.println(toDoList);
  }
}
```

```
[Water plants, fuck like horse, my bottle is red]
```

ArrayList Size

Let's say we have an ArrayList that stores items in a user's online shopping cart. As the user navigates through the site and adds items, their cart grows bigger and bigger.

If we wanted to display the number of items in the cart, we could find the size of it using the size() method:

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

shoppingCart.add("Trench Coat");
System.out.println(shoppingCart.size());
// 1 is printed
shoppingCart.add("Tweed Houndstooth Hat");
System.out.println(shoppingCart.size());
// 2 is printed
shoppingCart.add("Magnifying Glass");
System.out.println(shoppingCart.size());
// 3 is printed
```

In dynamic objects like ArrayLists, it's important to know how to access the amount of objects we have stored.

Instructions

1. Detectives do a lot to solve a case. But who has more to do?

Print out the size of each detective's to-do ArrayList:

- sherlocksToDos for Sherlock Holmes
- poirotsToDos for Hercule Poirot

2. So who has more to do? Print the name of the detective whose to-do list is longer. Was it Sherlock or Poirot?

```
import java.util.ArrayList;

class ToDos {
```

```
  public static void main(String[] args) {

    // Sherlock
    ArrayList<String> sherlocksToDos = new ArrayList<String>();

    sherlocksToDos.add("visit the crime scene");
    sherlocksToDos.add("play violin");
    sherlocksToDos.add("interview suspects");
    sherlocksToDos.add("solve the case");
    sherlocksToDos.add("apprehend the criminal");

    System.out.println(sherlocksToDos.size());

    // Poirot
    ArrayList<String> poirotsToDos = new ArrayList<String>();

    poirotsToDos.add("visit the crime scene");
    poirotsToDos.add("interview suspects");
    poirotsToDos.add("let the little grey cells do their work");
    poirotsToDos.add("trim mustache");
    poirotsToDos.add("call all suspects together");
    poirotsToDos.add("reveal the truth of the crime");

    // Print the size of each ArrayList below:
    System.out.println(poirotsToDos.size());

    // Print the name of the detective with the larger to-do list:
if(sherlocksToDos.size() > poirotsToDos.size())
{
  System.out.println("sherlock");
} else {
  System.out.println("poirot");
}}}
```

```
5
6
poirot
```

Accessing an Index With arrays, we can use bracket notation to access a value at a particular index:

```
double[] ratings = {3.2, 2.5, 1.7};

System.out.println(ratings[1]);
```

This code prints `2.5`, the value at index `1` of the array.

For `ArrayLists`, bracket notation won't work. Instead, we use the method `get()` to access an index:

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

shoppingCart.add("Trench Coat");
shoppingCart.add("Tweed Houndstooth Hat");
shoppingCart.add("Magnifying Glass");

System.out.println(shoppingCart.get(2));
```

This code prints `"Magnifying Glass"`, which is the value at index 2 of the `ArrayList`.

Instructions

1. Use `get()` to access the third to-do element of `sherlocksToDos` and print the result.

2. Use `get()` to access the second to-do element of `poirotsToDos` and print the result.

```
import java.util.ArrayList;
```

```
class ToDos {

  public static void main(String[] args) {

    // Sherlock
    ArrayList<String> sherlocksToDos = new ArrayList<String>();

    sherlocksToDos.add("visit the crime scene");
    sherlocksToDos.add("play violin");
    sherlocksToDos.add("interview suspects");
    sherlocksToDos.add("solve the case");
    sherlocksToDos.add("apprehend the criminal");

    // Poirot
    ArrayList<String> poirotsToDos = new ArrayList<String>();

    poirotsToDos.add("visit the crime scene");
    poirotsToDos.add("interview suspects");
    poirotsToDos.add("let the little grey cells do their work");
    poirotsToDos.add("trim mustache");
    poirotsToDos.add("call all suspects together");
    poirotsToDos.add("reveal the truth of the crime");

    System.out.println("Sherlock's third to-do:");
    // Print Sherlock's third to-do:
   System.out.println( sherlocksToDos.get(2));

    System.out.println("\nPoirot's second to-do:");
    // Print Poirot's second to-do:
    System.out.println(poirotsToDos.get(1));
  }
}
```

```
Sherlock's third to-do:
interview suspects

Poirot's second to-do:
interview suspects
```

Changing a Value

When we were using arrays, we could rewrite entries by using bracket notation to reassign values:

```
String[] shoppingCart = {"Trench Coat", "Tweed Houndstooth Hat",
"Magnifying Glass"};

shoppingCart[0] = "Tweed Cape";

// shoppingCart now holds ["Tweed Cape", "Tweed Houndstooth
Hat", "Magnifying Glass"]
```

ArrayList has a slightly different way of doing this, using the set() method:

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

shoppingCart.add("Trench Coat");
shoppingCart.add("Tweed Houndstooth Hat");
shoppingCart.add("Magnifying Glass");

shoppingCart.set(0, "Tweed Cape");

// shoppingCart now holds ["Tweed Cape", "Tweed Houndstooth
Hat", "Magnifying Glass"]
```

Instructions

1. Modify `sherlocksToDos` so that the value at `"play violin"` becomes `"listen to Dr. Watson for amusement"`.

2. Modify `poirotsToDos` so that the value at `"trim mustache"` becomes `"listen to Captain Hastings for amusement"`.

```
import java.util.ArrayList;

class ToDos {

  public static void main(String[] args) {

    // Sherlock
    ArrayList<String> sherlocksToDos = new ArrayList<String>();

    sherlocksToDos.add("visit the crime scene");
    sherlocksToDos.add("play violin");
    sherlocksToDos.add("interview suspects");
    sherlocksToDos.add("solve the case");
    sherlocksToDos.add("apprehend the criminal");

    // Poirot
    ArrayList<String> poirotsToDos = new ArrayList<String>();

    poirotsToDos.add("visit the crime scene");
    poirotsToDos.add("interview suspects");
    poirotsToDos.add("let the little grey cells do their work");
    poirotsToDos.add("trim mustache");
    poirotsToDos.add("call all suspects together");
    poirotsToDos.add("reveal the truth of the crime");

    // Set each to-do below:
    sherlocksToDos.set(1,"listen to Dr. Watson for amusement");
```

```
    poirotsToDos.set(3,"listen to Captain Hastings for amusement");

    System.out.println("Sherlock's to-do list:");
    System.out.println(sherlocksToDos.toString() + "\n");
    System.out.println("Poirot's to-do list:");
    System.out.println(poirotsToDos.toString());
  }
}
```

```
Sherlock's to-do list:
[visit the crime scene, listen to Dr. Watson for amusement,
interview suspects, solve the case, apprehend the criminal]

Poirot's to-do list:
[visit the crime scene, interview suspects, let the little grey cells
do their work, listen to Captain Hastings for amusement, call all
suspects together, reveal the truth of the crime]
```

Removing an Item

What if we wanted to get rid of an entry altogether? For arrays, we would have to make a completely new array without the value.

Luckily, `ArrayList`s allow us to remove an item by specifying the index to remove:

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

shoppingCart.add("Trench Coat");
shoppingCart.add("Tweed Houndstooth Hat");
shoppingCart.add("Magnifying Glass");
```

```
shoppingCart.remove(1);
// shoppingCart now holds ["Trench Coat", "Magnifying Glass"]
```

We can also remove an item by specifying the value to remove:

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

shoppingCart.add("Trench Coat");
shoppingCart.add("Tweed Houndstooth Hat");
shoppingCart.add("Magnifying Glass");

shoppingCart.remove("Trench Coat");
// shoppingCart now holds ["Tweed Houndstooth Hat", "Magnifying Glass"]
```

Note: This command removes the FIRST instance of the value `"Trench Coat"`.

Instructions

1. Sherlock Holmes and Hercule Poirot have each already visited their respective crime scenes.

Remove `"visit the crime scene"` from `sherlocksToDos` and `poirotsToDos` using `remove()`.

Moreover, Sherlock Holmes has also gotten some violin playing done.

So you can remove `"play violin"` from `sherlocksToDos` as well.

```
import java.util.ArrayList;

class ToDos {

  public static void main(String[] args) {

    // Sherlock
    ArrayList<String> sherlocksToDos = new ArrayList<String>();
```

```
    sherlocksToDos.add("visit the crime scene");
    sherlocksToDos.add("play violin");
    sherlocksToDos.add("interview suspects");
    sherlocksToDos.add("solve the case");
    sherlocksToDos.add("apprehend the criminal");

    // Poirot
    ArrayList<String> poirotsToDos = new ArrayList<String>();

    poirotsToDos.add("visit the crime scene");
    poirotsToDos.add("interview suspects");
    poirotsToDos.add("let the little grey cells do their work");
    poirotsToDos.add("trim mustache");
    poirotsToDos.add("call all suspects together");
    poirotsToDos.add("reveal the truth of the crime");

    // Remove each to-do below:
    poirotsToDos.remove(0);
     sherlocksToDos.remove(0);
     sherlocksToDos.remove("play violin");

    System.out.println(sherlocksToDos.toString() + "\n");
    System.out.println(poirotsToDos.toString());
  }

}
```

```
[interview suspects, solve the case, apprehend the criminal]

[interview suspects, let the little grey cells do their work, trim mustache, call all suspects together, reveal the truth of the crime]
```

Getting an Item's Index

What if we had a really large list and wanted to know the position of a certain element in it? For instance, what if we had an `ArrayList` `detectives` with the names of fictional detectives in chronological order, and we wanted to know what position `"Fletcher"` was.

```
// detectives holds ["Holmes", "Poirot", "Marple", "Spade",
"Fletcher", "Conan", "Ramotswe"];
System.out.println(detectives.indexOf("Fletcher"));
```

This code would print `4`, since `"Fletcher"` is at index `4` of the `detectives` `ArrayList`.

Instructions

1. After visiting the crime scene, the ever-impatient Dr. Watson wants to know how many to-dos are left until Sherlock solves the case.

To help Dr. Watson figure this out, use `indexOf()` to determine where in the to-do list `"solve the case"` is.

Print this information out. That's the number of to-dos remaining before Sherlock reaches `"solve the case"`.

```
import java.util.ArrayList;

class ToDos {

  public static void main(String[] args) {

    // Sherlock
    ArrayList<String> sherlocksToDos = new ArrayList<String>();

    sherlocksToDos.add("visit the crime scene");
    sherlocksToDos.add("play violin");
```

```
    sherlocksToDos.add("interview suspects");
    sherlocksToDos.add("listen to Dr. Watson for amusement");
    sherlocksToDos.add("solve the case");
    sherlocksToDos.add("apprehend the criminal");

    sherlocksToDos.remove("visit the crime scene");

    // Calculate to-dos until case is solved:
    int solved = sherlocksToDos.indexOf("solve the case");

    // Change the value printed:
    System.out.println("solved");

  }
}
```

```
solved
```

Review

Nice work! You now know the basics of

ArrayLists including:

- Creating an `ArrayList`.
- Adding a new `ArrayList` item using `add()`.
- Accessing the size of an `ArrayList` using `size()`.
- Finding an item by index using `get()`.
- Changing the value of an `ArrayList` item using `set()`.
- Removing an item with a specific value using `remove()`.
- Retrieving the index of an item with a specific value using `indexOf()`.

Now if only there were some way to move through an array or `ArrayList`, item by item...

Instructions

We've included a workspace for you to test out your newfound knowledge of arrays and `ArrayList`s.

Remember: To run your code, enter the following commands in the terminal:

```
javac List.java
```

and then:

```
java List
```

```
import java.util.ArrayList;

class List {

  public static void main(String[] args) {

  }

}
```