Learning Objectives: ArrayLists

- **Create an empty ArrayList**
- **Add and remove ArrayList elements**
- **Get and set ArrayList elements**
- **Iterate through ArrayLists using both a regular `for` loop and an *enhanced* `for` loop**
- **Determine ArrayList output**
- **Determine key differences between ArrayLists and arrays**

Creating an ArrayList

What Is an ArrayList?

Although arrays are very useful for data collection, they are considered **static**, meaning once they are created, you cannot add or remove elements from them without changing the way they are initialized. **ArrayLists**, on the other hand, are **dynamic**, meaning you can make changes to them while the program is running. ArrayLists are particularly helpful when you don't know how large your collection of elements will become. Since ArrayLists are dynamic, you can add and remove elements later on if needed. In order to use ArrayLists, you must import it using `import java.util.ArrayList;` in the header of your program. For convenience, the program file to your left already contains the import statement.

ArrayList Creation

To create an ArrayList, you need to include the following:
* The keyword `ArrayList` followed by the data type in angle brackets `<>`.
* Note that unlike arrays, ArrayList types are labeled slightly differently (e.g. `Integer`, `Double`, `Boolean`, and `String`).
* A variable name that refers to the ArrayList.
* The assignment operator `=` followed by the keyword `new`.
* Another `ArrayList` keyword followed by the data type in angle brackets `<>` followed by empty parentheses `()`.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();

System.out.println(numbers);
```

Code Visualizer

important

IMPORTANT

Unlike printing arrays, printing an ArrayList will output its elements instead of its memory address. When an ArrayList is initialized, it is empty by default. This is why printing a new ArrayList results in empty brackets `[]`.

Determining ArrayList Size

ArrayLists use the method `size()` to determine the number of elements that exist instead of `length` which is used for arrays. When an ArrayList is initially created, its size is automatically `0` since all new ArrayLists are empty by default.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();

System.out.println(numbers.size());
```

Code Visualizer

Adding and Removing Elements

Adding ArrayList Elements

To add elements to the ArrayList, simply use the `add()` method. The `add()` method will add whatever element that is specified inside parentheses `()` to the end of the ArrayList by default. If an element is added to an empty ArrayList, that element will be the first and only element in the ArrayList.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(50); //add 50 as an element to end of ArrayList

System.out.println(numbers);
```

challenge

What happens if you:

- add `numbers.add(100);` directly below `numbers.add(50);`?
- add `numbers.add(12.3);` below `numbers.add(50);` but before the print statement?

important

IMPORTANT

Much like arrays, ArrayLists can only store one type of data. For example, you cannot store `12.3` into an `Integer` ArrayList.

To add an element to a *specific index* in the ArrayList, you can use the `add()` method with two parameters inside the parentheses `()`. The first parameter is the index where you want the element to be stored at and the second parameter is the element itself. For example, `numbers.add(0, 12)` will add the number `12` to index `0` causing `12` to become the first element in the ArrayList. This will cause all of the elements to the right of `12` to move up by 1 index number.

```
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(50);
numbers.add(100);
System.out.println(numbers);

numbers.add(0, 12); //add 12 as an element to index 0
System.out.println(numbers);
```

challenge

What happens if you:

- add `numbers.add(2, 75);` directly below `numbers.add(0, 12);`?
- add `numbers.add(4, 250);` below `numbers.add(0, 12);` but before the second print statement?
- add `numbers.add(8, 320);` below `numbers.add(0, 12);` but before the second print statement?

important

IMPORTANT

Adding `numbers.add(8, 320);` produces the familiar `IndexOutOfBoundsException` error. This occurs because the ArrayList does not contain index number `8`. However, you can add an element to the ArrayList if you specify the last available array index plus 1. For example, if the last available index is 3, then you can use `numbers.add(4, 250);` to add `250` to index 4 which did not exist previously.

Removing ArrayList Elements

To remove an element from an ArrayList, use the `remove()` method and specify the ArrayList index of the element you want to be removed as a parameter inside the parentheses `()`. Deleting an element will cause all elements to the right of that element to move down by 1 index number.

```java
ArrayList<Integer> numbers = new ArrayList<Integer>();
numbers.add(12);
numbers.add(50);
numbers.add(75);
numbers.add(100);
System.out.println(numbers);

numbers.remove(2); //remove element at index 2
System.out.println(numbers);
```

challenge

What happens if you:

- add another `numbers.remove(2);` directly below `numbers.remove(2);`?
- add a third `numbers.remove(2);` directly below the other two `numbers.remove(2);`s?

Getting and Setting Elements

Getting ArrayList Elements

To get or access ArrayList elements, use the `get()` method and include the index as a parameter inside parentheses `()`.

```
ArrayList<String> contact = new ArrayList<String>();
contact.add("First name");
contact.add("Last name");
contact.add("Phone number");

System.out.println(contact.get(0)); //gets element at index 0
        and prints
```

challenge

What happens if you:

- change `contact.get(0)` in the code above to `contact.get(1)`?
- change `contact.get(0)` in the code above to `contact.get(2)`?
- change `contact.get(0)` in the code above to `contact.get(3)`?

Setting ArrayList Elements

To set or modify ArrayList elements, use the `set()` method which includes two parameters within parentheses `()`. The first parameter specifies the ArrayList index and the second parameter specifies the element that will replace the current value at the index. For example, `contact.set(2, "Email")` will modify the element at index 2 and change it to `Email`.

```java
ArrayList<String> contact = new ArrayList<String>();
contact.add("First name");
contact.add("Last name");
contact.add("Phone number");
System.out.println(contact);

contact.set(2, "Email"); //change element at index 2 to "Email"
System.out.println(contact);
```

challenge

What happens if you:

- add `contact.set(0, "Full name");` to the line directly before `contact.set(2, "Email");`?
- change `contact.set(2, "Email");` in the code above to `contact.set(1, "Address");`?
- change `contact.set(2, "Email");` in the code above to `contact.set(3, "Alternative name");`?

important

IMPORTANT

Both `get()` and `set()` methods require that the ArrayList already has an element that exists at the specified index. Otherwise, the `IndexOutOfBoundsException` error will occur.

Iterating an ArrayList

Iterating ArrayList Elements

Iterating through an ArrayList is very similar to iterating through an array. The main difference is that in an ArrayList, we use `get()` to access the elements instead of brackets `[]`. Both of the code blocks below use a regular `for` to produce the exact same results. The first code block contains an array and the second contains an ArrayList.

```
//iterating through an array
int[] grades = {85, 95, 48, 100, 92};

for (int i = 0; i < grades.length; i++) {
  System.out.println(grades[i]);
}
```

```
//iterating through an ArrayList
ArrayList<Integer> grades = new ArrayList<Integer>();
grades.add(85);
grades.add(95);
grades.add(48);
grades.add(100);
grades.add(92);

for (int i = 0; i < grades.size(); i++) {
  System.out.println(grades.get(i));
}
```

Enhanced For Loop in ArrayList

We can also use an **enhanced for loop** to iterate through an ArrayList.

```java
//iterating an ArrayList with Enhanced For Loop
ArrayList<Integer> grades = new ArrayList<Integer>();
grades.add(85);
grades.add(95);
grades.add(48);
grades.add(100);
grades.add(92);

for (Integer i : grades) { //Integer is required instead of int!
  System.out.println(i);
}
```

important

IMPORTANT

When using an enhanced `for` loop for an ArrayList, you must label the iterating variable accordingly. Remember that ArrayLists use `Integer`, `Double`, and `Boolean` instead of `int`, `double`, and `boolean`. Only `String` is consistently labeled between ArrayLists and arrays. Therefore, `for (Integer i : grades)` is required instead of `for (int i : grades)`.

ArrayList vs. Array

ArrayList vs. Array

Which one is better: ArrayList or array? The answer is, it really *depends*. If you know how many elements you need in your collection and you don't intend on changing the order of those elements, then it is better to use an **array**. On the other hand, if you don't know how many elements you need and want to modify the order of elements later on, then it is better to use an **ArrayList**.

Although an array is shorter to write and arguably easier to use, it is **static**, meaning it is not possible to add additional elements to the array after it has already been initialized. In contrast, an ArrayList is more **dyanamic**, meaning you can add, remove, and reorganize elements as needed later on.

Here is a table showing the differences between ArrayLists and arrays. Note that uppercase `Type` stands for compatible *ArrayList* types while lowercase `type` stands for compatible *array* types. Also note that `var` stands for ArrayList or array name, `num` stands for an integer number, `index` stands for index or position number, and `element` stands for an ArrayList or array element.

Method/Types	ArrayList	Array
Create	ArrayList<Type> var = new ArrayList<Type>()	type[] var = new type[nu or type[] var = {element, element...}
Find number of elements	var.size()	var.length
Access an element	var.get(index)	var[index]
Modify and element	var.set(index, element)	var[index] = element
Add an element	var.add(element) or var.add(index, element)	n/a
Remove an element	var.remove(index)	n/a
for loop	for (int i = 0; i < var.size(); i++) {System.out.println(var.get(i));}	for (int i = 0; i < var.lengt i++) {System.out.println(var[

Enhanced for loop	for (`Type i : var`) {System.out.println(i)}	for (`type i : var`) {System.out.println(i)}
Common compatible types	Integer, Double, Boolean, Strings	int, double, boolean, Strings

Using Both an ArrayList and Array

ArrayLists and arrays can be used in tandem with each other. For example, the following code keeps track of the top five students in a class.

```
String[] top = {"First: ", "Second: ", "Third: ", "Fourth: ",
        "Fifth: "};
ArrayList<String> names = new ArrayList<String>();

names.add("Alan");
names.add("Bob");
names.add("Carol");
names.add("David");
names.add("Ellen");

for (int i = 0; i < 5; i++) {
  System.out.println(top[i] + names.get(i));
}
```

Code Visualizer

challenge

Without deleting any existing code, try to:

- switch `Alan` and `Carol`'s places.
- replace `David` with `Fred`.
- make `Grace` get "Fifth" place and remove `Ellen` from the list.

Code Visualizer

▼ **Sample Solution**

```
String[] top = {"First: ", "Second: ", "Third: ", "Fourth: ",
        "Fifth: "};
ArrayList<String> names = new ArrayList<String>();

names.add("Alan");
names.add("Bob");
names.add("Carol");
names.add("David");
names.add("Ellen");

names.set(0, "Carol"); //switch Alan with Carol
names.set(2, "Alan");  //and vice versa

names.set(3, "Fred"); //Fred replaces David

names.add(4, "Grace"); //Grace takes Ellen's place
names.remove(5); //Ellen's "Sixth" place gets removed

for (int i = 0; i < 5; i++) {
  System.out.println(top[i] + names.get(i));
}
```

Helpful ArrayList Algorithms

ArrayList Algorithms

Like arrays, ArrayLists can be used to search for a particular element and to find a minimum or maximum element. Additionally, ArrayLists can reverse the order of elements rather than just simply printing the elements in reverse order.

Searching for a Particular Element

```
ArrayList<String> cars = new ArrayList<String>();
String Camry = "A Camry is not available."; //default String
        value

cars.add("Corolla");
cars.add("Camry");
cars.add("Prius");
cars.add("RAV4");
cars.add("Highlander");

for (String s : cars) { //enhanced for loop
  if (s.equals("Camry")) { //if "Camry" is in ArrayList
    Camry = "A Camry is available."; //variable changes if
        "Camry" exists
  }
}

System.out.println(Camry); //print whether Camry exists or not
```

challenge

What happens if you:

- add `cars.remove(1);` to the line directly below `cars.add("Highlander");`?
- try to modify the code above so that the algorithm will look for `Prius` in the array and will print `A Prius is available.` if `Prius` is an element and `A Prius is not available.` if it is not an element.

▼ **Sample Solution**

```
ArrayList<String> cars = new ArrayList<String>();
String Prius = "A Prius is not available.";

cars.add("Corolla");
cars.add("Camry");
cars.add("Prius");
cars.add("RAV4");
cars.add("Highlander");

for (String s : cars) {
  if (s.equals("Prius")) {
    Prius = "A Prius is available.";
  }
}

System.out.println(Prius);
```

Finding a Minimum or Maximum Value

```
ArrayList<Integer> grades = new ArrayList<Integer>();
grades.add(72);
grades.add(84);
grades.add(63);
grades.add(55);
grades.add(98);

int min = grades.get(0); //set min to the first element in the
        array

for (int i : grades) { //enhanced for loop
  if (i < min) { //if element is less than min
    min = i; //set min to element that is less
  }
}
//elements are not modified so enhanced for loop can be used

System.out.println("The lowest grade is " + min); //print lowest
        element
```

challenge

What happens if you:

- add `grades.set(0, 42);` to the line directly below `grades.add(98);`?
- try to modify the code so that the algorithm will look for the **maximum** element instead?

▼ **Sample Solution**

```
ArrayList<Integer> grades = new ArrayList<Integer>();
grades.add(72);
grades.add(84);
grades.add(63);
grades.add(55);
grades.add(98);

int max = grades.get(0);

for (int i : grades) {
  if (i > max) {
    max = i;
  }
}

System.out.println("The highest grade is " + max);
```

Reversing the Order of Elements

```java
ArrayList<String> letters = new ArrayList<String>();
letters.add("A");
letters.add("B");
letters.add("C");
letters.add("D");
letters.add("E");

int original = letters.size(); //original size

//regular for loops needed to access element indices

for (int i = letters.size() - 1; i >= 0; i--) {
  letters.add(letters.get(i));
} //add elements in reverse order to the ArrayList

for (int j = 0; j < original; j++) {
  letters.remove(0);
} //remove all the original elements

System.out.println(letters); //print new ArrayList
```

important

IMPORTANT

Note that we used `letters.remove(0)` rather than `letters.remove(j)` in the code above because `remove()` deletes both the **element** and the **index**. Thus, the next element in the ArrayList becomes the *new* 0th index which we want to continue to delete.