Objectives

- Recognize the limitations of arrays.
- Define and use the **ArrayList** class from the Java Collections Framework.

Limitations of Arrays

Having used an array in your previous assignment to store a list of **Fruit** objects, you have learned that arrays have some serious limitations:

- Once an array is created, its size cannot be changed, forcing the programmer to know in advance how many elements the array will hold. (There are ways around this limitation, but they are all rather complicated.)
- Arrays provide inadequate support for inserting, deleting, sorting, and searching operations.

The ArrayList Class to the Rescue

The collection class **ArrayList** is one of several valuable classes in Java's collection classes. The **ArrayList** class eliminates limitations of regular arrays by modeling an abstract data type (ADT) which represents a dynamic structure that can grow and shrink as the program executes.

As a client of the **ArrayList** class we are not interested in the implementation details of its underlying structure and supported operations. As a client, we are interested in learning about how we can create an array list of a specific type of object and about how we can use the operations it supports to accomplish our tasks.

Your Task

Repeat Assignment 1-B, but this time use an **ArrayList** instead of an array as the underlying data structure for your **FruitList** class.

Examples of defining and using an ArrayList<E>

The E in ArrayList<E> is a placeholder for the type of Elements you want the array list to hold. Examples:

1. Create an ArrayList<Friut> object to represent an empty fruit list:

```
ArrayList<Fruit> fruitList = new ArrayList<Fruit>(); // an empty list
```

2. Create an ArrayList<Book> object to represent an empty bookshelf:

```
ArrayList<Book> bookshelf = new ArrayList<Book>(); // an empty list
```

3. Create a pet list and add some pet names to it:

```
1 ArrayList<String> pets = new ArrayList<String>(); // []
2 pets.add("cat"); // [cat]
3 pets.add("fish"); // [cat, fish]
4 pets.add("dog"); // [cat, fish, dog]
5 System.out.println("pet list: " + pets);
```

   Executing the code above produces the following output:

output

```
1 pet list: [cat, fish, dog]
```

4. Add an element at index 1:

```
6 System.out.println("before adding bunny = " + pets);
7 pets.add(1, "bunny");
8 System.out.println("after adding bunny = " + pets);
```

output

```
2 before adding bunny = [cat, fish, dog]
3 after adding bunny  = [cat, bunny, fish, dog]
```

5. Remove the element at index 0 and then the element at index 1.

```
System.out.println("before remove(0) : " + pets);
pets.remove(0);
System.out.println("after remove(0) : " + pets);
System.out.println();
System.out.println("before remove(1) : " + pets);
pets.remove(1);
System.out.println("after remove(1) : " + pets);
```

output

```
before remove(0) : [cat, bunny, fish, dog]
after remove(0)  : [bunny, fish, dog]

before remove(1) : [bunny, fish, dog]
after remove(1)  : [bunny, dog]
```

6. Add "parrot" to the list and set the second element to "pig".

```


System.out.println();
System.out.println("before adding parrot : " + pets);
pets.add("parrot");
System.out.println("after adding parrot : " + pets);
System.out.println();
System.out.println("before setting 2nd element to pig : " + pets);
pets.set(1, "pig");
System.out.println("after setting 2nd element to pig : " + pets);
```

output

```

before adding parrot : [bunny, dog]
after adding parrot  : [bunny, dog, parrot]

before setting 2nd element to pig : [bunny, dog, parrot]
after setting 2nd element to pig  : [bunny, pig, parrot]
```

7. To demonstrate `ArrayList`'s **get** method, compute the total length of all names

in the pet list:

```
System.out.println();
int totalLength= 0;
for (int i = 0; i < pets.size(); i++)
{
    String petName = pets.get(i); // showing off ArrayList's get() method
    totalLength += petName.length();
}
System.out.println("Total of lengths = " + totalLength);
```

output

```
Total of lengths = 14
```

Here are some frequently used methods in the `ArrayList<E>` class:

`void add(`*value*`)`	Adds a new element to the end of this list.
`void add(`*index*`, `*value*`)`	Inserts the specified element at the specified index position.
`void remove(`*index*`)`	Removes the element at the specified index position.
`void remove(`*value*`)`	Removes the first instance of the specified element, if any.
`void clear()`	Removes all elements from this list.
`E get(`*index*`)`	Returns the object at the specified index.
`void set(`*index*`, `*value*`)`	Sets the element at the specified index to the new value.
`int indexOf(`*value*`)`	Returns the index of the first occurrence of the specified value, or -1 if it does not appear.
`boolean contains(`*value*`)`	Returns true if this list contains the specified value.
`boolean isEmpty()`	Returns true if this list contains no elements.
`int size()`	Returns the number of elements in this list.

For more examples and information see section 11.11 "The ArrayList Class" of the course textbook on page 432. For complete information see JDK API documentation for `ArrayList<E>`.

Evaluation Criteria	
Correctness of execution of your program	60%
Proper use of required Java concepts	20%
Java API documentation style	10%
Comments on nontrivial steps in code, Choice of meaningful variable names, Indentation and readability of program	10%