Java Collections

Lab setup

1. Create a new Java project for this lab in Eclipse.
2. Add the Bicycle class to the project.
3. Add a main class to the project.

Array

Arrays are objects used to store a fixed number of elements of the same type. These elements can be any of the primitive types or other objects.

We can assign values to the array at declaration using an array literal. We can also use the `new` operator to create an array of a specific size.

```
// Declare an String array names with Alice, Bob, and Cora
String[] names = {"Alice", "Bob", "Cora"};
// Create a new String array with 4 elements. Loses reference to the original
String[] names = new String[4];
```

Java Collections

The Collection framework provides architecture to store and manipulate groups of objects. A collection is a single unit of objects. The framework provides interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, etc).

The collections are part of the `java.util` package.

Iterable and Collection

Iterable is the root interface in the Collection framework. Iterable contains only the abstract method.

```
// Returns an iterator over elements of type T
Iterator<T> iterator();
```

Collection is the next interface in the hierarchy. It extends Iterable and has abstract methods for manipulating collections and getting information about them.

List

List is one of three interfaces that extend Collection. It represents the list kind of data structure which can store an ordered collection of objects.

ArrayList

ArrayList is an implementation of the List interface. It is a dynamic array, meaning its size is not static. Unlike a normal array, an ArrayList can add or remove elements as much as needed.

We will use the ArrayList collection to model a storage facility for Bicycles. `BicycleStorage` will be a wrapper of `ArrayList` where we take advantage of a class that has already been defined with the functionality we need.

4. Add a new class to the project called BicycleStorage
5. Add import statements for List and ArrayList to the beginning of BicycleStorage.java.
6. We will start with the class definition now. Inside the `BicycleStorage` class, add a field for an `ArrayList`. The `ArrayList` is a generic class and needs a type parameter.
7. In the constructor, we can create the `ArrayList` object. We will again need to use a type parameter.

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

public class BicycleStorage {

    private ArrayList<Bicycle> bicycles;


    // Constructor
    public BicycleStorage() {
        bicycles = new ArrayList<Bicycle>()

    }
}
```

Figure 1 BicycleStorage class (st.4,5,6,7)

Graded Submission Task 1:

We can use ArrayList methods to implement the functionality we want. The methods we will use in the this task are `add()`, `remove()`, and `get()`.

Add methods to BicycleStorage for the following:

- Add a `Bicycle` to the `BicycleStorage to the end.`
- Remove a `Bicycle` at a position in the `BicycleStorage.`
- Access a `Bicycle` at a position in the `BicycleStorage` without removing it.

Note: Removing and accessing at a position in the `ArrayList` will require methods which throw exceptions. For now, you can accept Eclipse's suggestions for handling them.

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 1. You will submit this copy at the end of this lab.

`List` type or `ArrayList` type

Now, why did we import `List` when we did not need it? Suppose we wanted to define a `BicycleStorage`, but not specify what kind of `List` it would use. Maybe we would create multiple classes with different implementations. Could we do this?

Answer: We could. We can change the field of the class to be `List<Bicycle>`. Please note that the `List<T>` is an interface so it cannot be instantiated, but you will notice that instantiating it as an `ArrayList<Bicycle>` is still valid. We can use any class which implements `List`.

A caveat with this is because the field `bicycles` is declared to be a `List` we would not be able to use any methods the implementing class defines which are not part of the interface.

Custom iterables

We may also create our own classes which `implements Iterable<T>`. We will show this with a class called Inventory. Add the class shown in Figure 2 to the project.

```
import java.util.Iterator;
class Inventory implements Iterable<Bicycle> {
    private Bicycle[] arr;

    Inventory(Bicycle[] arr){
        this.arr = arr;
    }

    public Iterator<Bicycle> iterator() {
        Iterator<Bicycle> it = new Iterator<Bicycle>() {
            private int currIndex = 0;
            public boolean hasNext() {
                return currIndex < arr.length && arr[currIndex] != null;
            }
            public Bicycle next() {
                return arr[currIndex++];
            }
            public void remove() {
                throw new UnsupportedOperationException();
            }
        };
        return it;
    }
}
```

Figure 2 Inventory class implementing Iterable.

Here Inventory is defined to work only with Bicycle objects. We could also define Inventory with `<T>` so that it can work with any type as well.

Drone application

The previous implementation of DroneController used static arrays for the collections of drones. We used the `activeDrones` static array as a partially filled array and had to keep track of the end of the partially filled array. The `dronesInRange` static array was recreated at each time step. We can simplify these by implementing them with ArrayLists.

Graded Submission Task 2:

Modify the previous implementation of the DroneController class to use `List<T>`. Instantiate both `activeDrones` and `dronesInRange` as ArrayLists and remove the field `dronesInSystem`. You will need to rewrite some methods now that the collections of drones are no longer static.

The previous implementation iterates over activeDrones twice: first to count the number of drones in range, then to record the drones in range. Consider if it's still necessary to iterate twice using ArrayList. Also, consider if it's necessary to recreate `dronesInRange` every time step or if there is a simpler way.

Modify the provided driver to have the same functionality as before now that DroneController has been changed.

The Javadoc for ArrayList is provided here:
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/ArrayList.html

** STOP **

Ensure you have saved the source files in your project. Make a copy of your Eclipse project at this point (you can do this from your system's file explorer). Name the file labelling it as Task 2. You will submit this copy at the end of this lab.

Lab wrap-up

In this lab we discussed the Java Collections framework. We focused specifically on `List` and `ArrayList` with a small example for creating your own `Iterable` class.

The Collections framework defines interfaces and classes which hold collections of objects. The root interface is `Iterable`. Any class which is `Iterable` must implement the method `iterator()`, which returns an iterator over a generic type. The `List` interface extends `Iterable`. It provides the outline for list-type collections. `Iterable` and `List` cannot be instantiated because they are interfaces.

`ArrayList` implements the `List` interface as a dynamically sized array. It supports adding and removing elements and can resize as necessary. We implemented `BicycleStorage` as a wrapper around `ArrayList` for `Bicycle` objects. We also discussed that we could instantiate `ArrayList` objects from `List` variables.

We also provided an example of implementing our own `Iterable` for `Bicycle` objects.

Submission notes for graded tasks:

- Submit a zipped folder containing the Eclipse projects for each graded task.
- For each project, ensure you have included the src folder containing the .java file, the .classpath file, and the .project file.
- Ensure that each project is named appropriately to correspond with each task.
- Zip the projects together and name the zipped folder “Lab4_firstName_lastName”.
- Submit the zipped file to the respective Canvas link.

Rubrics	
Task 1 (allotted points) - Compilation (4) - Correctly use ArrayList add method (2) - Correctly use ArrayList remove method (2) - Correctly use ArrayList get method (2)	Task 2 (allotted points) - Compilation (2) - Replace array fields with List or ArrayList (2) - Reimplementation of methods to use ArrayList (4) - Driver functions like original (2)

Appendix

This appendix discusses the Collections class and the modifiability of collections.

Collections class

The Collections class is a class in the java.util package that exclusively contains static methods that operate on or return Collection objects. These include methods implementing polymorphic algorithms and wrappers which return new collections backed by a specified collection.

Modifiability of collections

Many modification methods in the collection interfaces are labeled optional. These collections have terms that are used to refer to them in the documentation.

Unmodifiable

An unmodifiable collection is one that does not support modification operations such as add, remove, or clear. Often, this means you have a reference to a view of the original collection.

Immutable

An immutable collection is one that is unmodifiable, that also guarantees no changes to the collection will be visible. Often, this means you have a reference to a copy of the original collection.

Fixed-size lists

Fixed-size lists are lists which cannot change size even while the elements themselves can change.