1. Follow the lab lecture about JavaDoc and unit tests then continue from step 2.
2. Create a new project in Android Studio named ListyCity.
3. Create a new java class named City that has two variables city and province. Create getters of these variables.

```
public class City {
    private String city;
    private String province;

    City(String city, String province){
        this.city = city;
        this.province = province;
    }

    String getCityName(){
        return this.city;
    }

    String getProvinceName(){
        return this.province;
    }
}
```

4. Add Javadoc comments for City, its variables and functions. e.g.

```
/**
* This class defines a city.
*/
public class City { ... }
```

5. Create a new class CityList and write javadoc comments for this class. Follow the instructor for further details regarding javadoc comments.

```
/**
* This is a class that keeps track of a list of city objects
*/
public class CityList { ... }
```

6. Declare a list to hold the city objects in CityList and write a javadoc comment to explain the list.

```
private List<City> cities = new ArrayList<>();
```

7. Implement a method to add city objects to this list, if a city already exists then throw Exception. Write JavaDoc comments with @param tag. Follow the instructor for further details regarding javadoc comments.

```
/**
* This adds a city to the list if the city does not exist
* @param city
*     This is a candidate city to add
*/
public void add(City city) {
   if (cities.contains(city)) {
      throw new IllegalArgumentException();
   }
   cities.add(city);
}
```

8. Create another method to get a list of city objects sorted according to the city name. Here we also have written JavaDoc comments with @return tag. Follow the instructor for further details regarding javadoc comments.

```
/**
* This returns a sorted list of cities
* @return
*    Return the sorted list
*/
public List getCities() {
   List list = cities;
   Collections.sort(list);
   return list;
}
```

9. In the CityList class the Collections.sort(list); line shows an error. We want to sort the name of the cities but we are trying to sort the city objects. To sort an Object by its property, we have to make the Object implement the Comparable interface and override the compareTo() method.

   In order to remove the error, go to the City class and make sure City implements Comparable interface and compareTo() method. It will look something like this:

```
public class City implements Comparable {
    ...
    @Override
    public int compareTo(Object o) {
        City city = (City) o;
        return this.city.compareTo(city.getCityName());
    }
}
```

Edit the Javadoc comments for city class to reflect the changes made and add javadoc comments for compareTo method. Follow the instructor for further details regarding javadoc comments.

Note: Lists (and arrays) of objects that implement Comparable interface can be sorted automatically by Collections.sort(). We also need to implement the method compareTo(). All wrapper classes and String classes implement Comparable interface. Wrapper classes are compared by their values, and strings are compared lexicographically. To know more: https://howtodoinjava.com/java/collections/java-comparable-interface/

10. Select **Tools -> Generate JavaDoc** to create HTML java documentation from your JavaDoc comments. Select the "Module app" and select the output directory.

11. Now we will move towards testing. We will use junit 5 for this lab. By default junit 4 is included, remove that line (`testImplementation 'junit:junit:4.13.2'`). To use junit 5, Include following lines in the dependencies section on app Gradle (build.gradle(Module:app)) file and sync it.

```
testImplementation 'org.junit.jupiter:junit-jupiter-api:5.3.2'
testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.3.2'
```

12. Under `com.abc.xyz.listycity(test)` folder, create a new class **CityListTest** to test the functionalities of our **CityList** class.
13. Create two private methods in **CityListTest** for creating a mock city object and adding to the cityList.

```
public class CityListTest {
   private CityList mockCityList() {
      CityList cityList = new CityList();
      cityList.add(mockCity());
      return cityList;
   }
   private City mockCity() {
      return new City("Edmonton", "Alberta");
   }
}
```

14. Write a test for the add() method which is in CityList class. Write it under the mockCity() method. Add city objects using the add() method and check if the addition is successful by assertEquals() and assertTrue(). We need to add @Test before any test method to identify it as a junit test.

```
@Test
void testAdd() {
   CityList cityList = mockCityList();
   assertEquals(1, cityList.getCities().size());
   City city = new City("Regina", "Saskatchewan");
   cityList.add(city);
   assertEquals(2, cityList.getCities().size());
   assertTrue(cityList.getCities().contains(city));
}
```

15. Write another test method for Exception while adding a city that already exists in the list.

```
@Test
void testAddException() {
   CityList cityList = mockCityList();
   City city = new City("Yellowknife", "Northwest Territories");
   cityList.add(city);
   assertThrows( IllegalArgumentException.class, () -> {
cityList.add(city); });
}
```

If you get an error on using lambda expressions then add the following lines in the android section on app Gradle (build.gradle(Module:app)) file and sync it.

```
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
```

16. Write another test method to test the getCities() method in the CityList class.

```
@Test
void testGetCities() {
   CityList cityList = mockCityList();
   assertEquals(0,
mockCity().compareTo(cityList.getCities().get(0)));
   City city = new City("Charlottetown", "Prince Edward Island");
   cityList.add(city);
   assertEquals(0, city.compareTo(cityList.getCities().get(0)));
   assertEquals(0,
mockCity().compareTo(cityList.getCities().get(1)));
}
```

17. Now run the tests by (right-clicking on the test folder), and pressing Run "Tests" in "<package_name>". There might be an error, to remove the error add following lines in the android section on app Gradle (build.gradle(Module:app)) file and sync it.

```
testOptions {
        unitTests.all {
            useJUnitPlatform()
        }
}
```

18. The tests can be written before the real implementation of the CityList class. We can implement all the tests first and they must fail as there is no implementation of CityList class and its methods. Then we can give implementation of the class and methods. Then our tests would pass. This is called test-driven development. You can also follow this method.