## Lab 2

1. **Accept** the Lab 2 assignment using this GitHub Classroom link with your Github account.

2. Clone your GitHub Classroom repository using IntelliJ IDEA.

3. Complete **Exercises 1–5** as described in the comments in code and the requirements in this file.

4. Test your solutions in **Main.java**.

5. Once your code runs correctly in **Main.java**, use the provided **JUnit tests** to verify your work.

   * If the tests fail, use the **debugger** to fix your code.

   *** Do not modify the JUnit test files. Any changes will be detected.**

6. Use **Git** to submit your code to your GitHub Classroom repository.

   * Check the **GitHub Actions results** to confirm that your code passes all tests.

7. After completing all the steps above, be prepared to **demo** your code to me.

## Exercise 1

In this exercise, you will use the file:

- **Q1.java**

**Which contains 3 classes, namely:**

- **class Array**
- **class Stack**
- **class Q1**

In this exercise, you will use class **Array** to implement class **Stack**.

**Hint:** Use an instance of class Array as a data member of class Stack and complete the functionality of Stack.

Uncomment the line, **Q1.q1()** in **main() method** in **Main.java** and compile the program. Run the program, so that no error message show up.

## Exercise 2

Examine the Java code below:

```
package edu.scu.coen160.lab2;

class Asset { // etend something?
	// ToDo: Implement powerDown
}

class EmergencyLight extends ... { // what to extend?
	// ToDo: Implement powerDown
}

class PersonalComputer extends ... { // what to extend?
	// ToDo: Implement powerDown
}

class TV extends ... { // what to extend?
	ToDo: Implement powerDown
}

class BuildingManager {
	Asset things[] = new Asset[24];
	int numItems = 0;

	public void powerDownAtNight() {
		for (int i = 0; i < things.length; i++)
			if (this.things[i] != null)
				this.things[i].powerDown();
	}

	// Add an Asset to this building
	public void add(Asset thing) {
		this.things[this.numItems] = thing;
		this.numItems=this.numItems+1;
	}
}

public class Q2 {
	public static void q2() {
		BuildingManager b1 = new BuildingManager();

		b1.add(new EmergencyLight());
		b1.add(new PersonalComputer());
		b1.add(new TV());
		b1.powerDownAtNight();
	}
}
```

The scenario above (incomplete) is an attempt to represent the fact that things like **EmergencyLight, Computer** and **TV** are **Asset**s and are added to the building manager which powers down all the assets at night.

Currently, because of the incomplete type definitions and relationships, the code will not compile. Make the necessary corrections to make the code compile and run. You may use one line of code:

- `System.out.println("powering down Asset!");`
- `System.out.println("powering down EmergencyLight");`
- `System.out.println("powering down PersonalComputer");`
- `System.out.println("powering down TV");`

**The program is in Q2.java**

## Exercise 3

## In this Exercise, you will use Java interfaces to create Lockable objects.

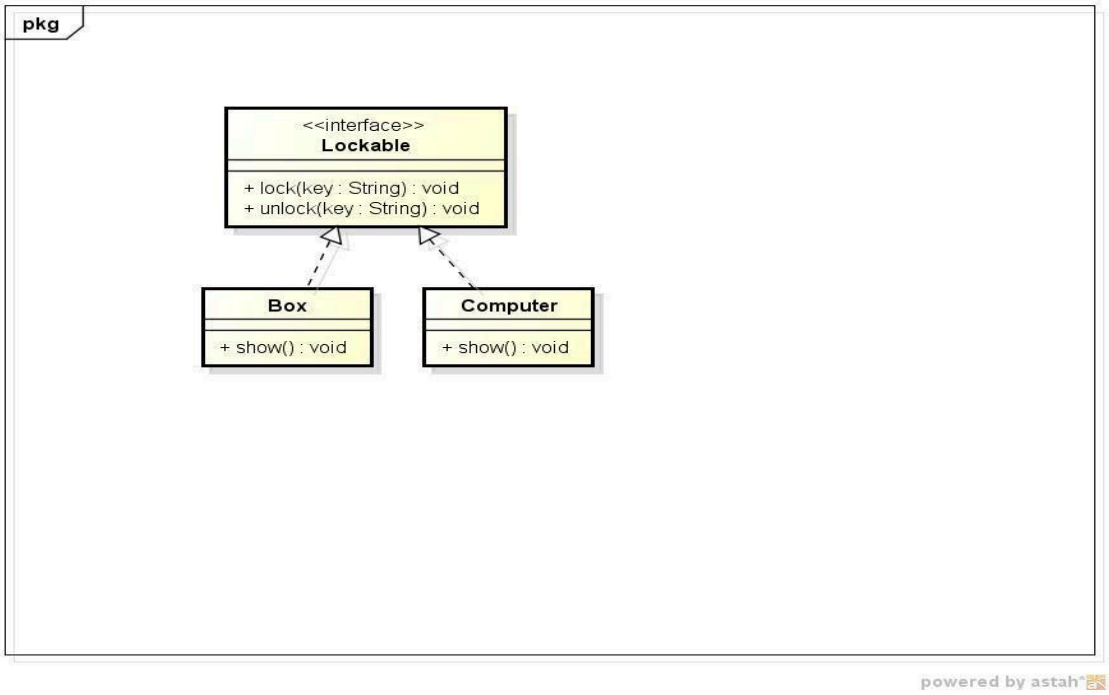
You will use the file, **Q3.java which contains 3 classes, namely, class Box, class Computer and class Q3.**

- Objects of classes Box and Computer are considered Lockable and should have methods, lock() and unlock().
- A box and Computer are really not closely related things, therefore, we use an interface called Lockable to enforce that both are Lockable.

**The UML diagram below shows the relationship among Box, Computer and Lockable types.**

**Complete the code in Q3.java using the comments given.**

To compile the program, uncomment the line, **Q3.q3()** in **main() method** in **Main.java**. Run the program.



## Exercise 4

In object-oriented programming, a java singleton class is a class that can have only one object (an instance of the class) at a time. After the first time, if we try to instantiate the Java Singleton classes, the new variable also points to the first instance created. So whatever modifications we do to any variable inside the class through any instance, affects the variable of the single instance created and is visible if we access that variable through any variable of that class type defined.

Remember the key points while defining a class as a singleton class that is while designing a singleton class:

- Make a constructor private.
- Write a static method that has the return type object of this singleton class.

**Purpose of Singleton Class**

The primary purpose of a java Singleton class is to restrict the limit of the number of object creations to only one. This often ensures that there is access control to resources, for example, socket or database connection.

A Singleton implementation is given. Implement a test in **Q4_q4().java** where you create three instances and then check that they are the same?

## Exercise 5

This exercise covers abstract classes.

In Java, abstract class is declared with the abstract keyword. It may have both abstract and non-abstract methods(methods with bodies). An abstract is a Java modifier applicable for classes and methods in Java but not for Variables. In this article, we will learn the use of abstract classes in Java.

The class Q5abstract is an abstract class that inherits and implements the missing methods and tests it in Q5.q5().

Using the comments given, complete the code in **Q5.java**.