The University of the West Indies, St. Augustine
COMP 2603 Object Oriented Programming 1
2020/2021 Semester 2
Lab 3

In Lab 2, we created an **AirConditioner** class that simulated how an air conditioner works to cool a room. Today, we will extend our example with a **Room** class that aims to model the attributes and behaviour of a room in a building. Then, we will set up simple associations between these classes in our object-oriented application in a main class called **FSTCoolingSystem**.

Part 1: Class & Instance Variables, Constructors, Method Overloading

1. Create a new project in BlueJ and download the **Room** and **FSTCoolingSystem** Java classes from myElearning or the network folder. Compile both classes to ensure there are no errors.

2. In the **FSTCoolingSystem** class:
   a. Create an array that can hold 10 **Room** objects.
   b. Add four **Room** objects, with the state in Table 1, to the array.

Name	Floor	Square Footage
Lecture Room 1	1	120
Computer Science Lab 1	2	100
Computer Science Lab 2	1	150
Lecture Room 2	1	300

Table 1

   c. Print the details of the **Room** objects by invoking to **toString( )** methods. What do you observe?

3. In the Room class, uncomment the supplied **toString( )** method. Print the details of the **Room** objects by invoking to **toString( )** methods. What do you observe now?

4. Suppose the number of seats in a **Room** can vary from the default calculation, perhaps due to safety regulations. Add an overloaded constructor to the **Room** class that accepts a **name**, **floor**, **square footage** and **number of seats** for the **Room.** This constructor should initialize the appropriate attributes with the supplied values and should also initialise the **roomID** to “FST”.

5. Using the new constructor from Step 5, create a new **Room** called “Lecture Room 3”, located on Floor 2, with a size of 175 square feet and seating for 70 people. Add this new **Room** to the array and print an updated list of Rooms.

6. In the **Room** class, write a method with the signature:
   public void generateRoomID (int floor).
   This method should automatically generate a **roomID** for a Room object based on the floor that a Room is located on. Table 2 shows how this is to be achieved.
   Note: Two class variables are required: **floor1IDGenerator** and **floor2IDGenerator**.

TIP: Class variable format

<access modifier> static <variable type> <variable name>

Floor	ID Start	Increment	Examples of roomID values
1	100	10	FST100, FST110, FST120…
2	200	10	FST200, FST210, FST220…

Table 2

7. Iterate through the Rooms array in the **FSTCoolingSystem** class, generate and set the **roomID** for each **Room** object, and then print the details of each **Room** object.

8. In the **FSTCoolingSystem** class, print the values of the two class variables (**floor1IDGenerator, floor2IDGenerator**) in the **Room** class. What do you notice? Change these variables to instance variables then print the contents of the Rooms array and the values of the generator variables once more. What do you observe?

9. Locate and print the details of the **Room** object with **roomID FST100** and the **Room** object with the **roomID FST210**.

Part 2: Association Relationships

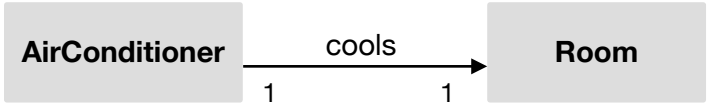
Now, we will model the relationship between a **Room** object and an **AirConditioner** object. Let's start with the premise that one room is cooled by one air conditioner.

1. Retrieve the **AirConditioner.java** from the network folder (or myElearning). In your **AirConditioner** class:
   a. Observe that the following constructor is provided:
      public AirConditioner (String brand, int btu) which generates unique values for the acID starting from 1000.
   b. Create an object reference to a **Room** object as follows:
      private Room room;
   c. Write a method in the **AirConditioner** class which associates an **AirConditioner** object with a **Room** object:

```
public void cools (Room r){
        room = r;
}
```

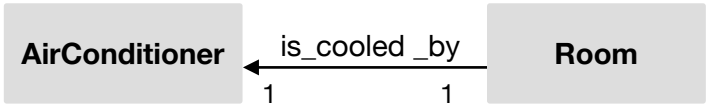
   d. Write a method **public String listRoom( )** which returns the details of the **Room** that the **AirConditioner** cools, or a message indicating that the **AirConditioner** is currently not installed in any room.

Note: By creating an object variable of type **Room** in the **AirConditioner** class in 1(b), we have essentially created an association between the **AirConditioner** class and the **Room** class. This is an example of a one-to-one relationship. The method **cools(Room r)** in 1(c), allows the Room object variable to be set to a real object. The **listRoom( )** method in 1(d) simply allows a client to get details of the **Room** object if it has been set or a message if the **Room object has not been set (i.e. it is null).**



2. Within your **Room** class:
    a. Create an object reference to an **AirConditioner** object as follows:
        private AirConditioner airConditioner;
    b. Write a method in the **Room** class which associates a **AirConditioner** object with a **Room** object:
```
public void addAirConditioner(AirConditioner ac){
   airConditioner = ac;
 }
```
    c. Write a method **public String listAirConditioner**( ) which returns the details of the **AirConditioner** installed in the **Room** or a message indicating that there is no **AirConditioner** installed in the **Room**.

> Note: Step 1 set up the directed relationship from the AirConditioner class to Room class. This allows an AirConditioner object to know about the Room it cools. Step 2 sets up another relationship, this time directed from the Room class to the AirConditioner class in the same way. This now allows the Room object to know about the AirConditioner object. Another example of a one-to-one relationship. The method name does not necessarily have to be the same as the relationship name. The method is merely an implementation of the design concept.



3. In your **FSTCoolingSystem** class:
    a. Create an **AirConditioner** object **ac1** with the following details:
    Brand: Carrier
    BTU: 5000
    b. Associate the **AirConditioner** object **ac1** with the **Room** object with **roomID FST210**. This is done as follows:
        - Locate the **Room** object with roomID FST130 (Let's call this object **room210** for reference)
        - Invoke the **cools(..)** method on the **AirConditioner** object, passing the **Room** object as an input parameter.
    c. Invoke the **listRoom( )** method on **ac1**. Observe what happens.
    d. Invoke the **listAirConditioner( )** method on **room210**. Observe what happens.
    e. Associate the **AirConditioner** object **ac1** with the **Room** object **room210**. This is done as follows:
        - Invoke the **addAirConditioner**(..) method on the **Room** object **room210**, passing the **AirConditioner** object **ac1** as an input parameter.
    f. Invoke the **listRoom( )**( ) method again on **ac1**. Observe what happens.
    g. Invoke the **listAirConditioner( )** method on **room210**. Observe what happens now.
    h. Create a second **AirConditioner** object **ac2** with the following details:
    Brand: Carrier
    BTU: 6000
    Add this **AirConditioner ac2** to the **Room room210** and repeat steps (f) and (g). What happens here? (We have been handling One-to-One relationships but now need to be able to handle One-to-Many relationships!)

4. Modify your **Room** class so that it keeps track of many **AirConditioner** objects. You need to add an array.
    a. Fix the **addAirConditioner(AirConditioner ac)** method appropriately
    b. Try to add **AirConditioner ac2** to **Room room210** and observe what happens now.
    c. Create a third **AirConditioner ac3** on your own and add it to a room of your choice.
    d. Print out the details of all the rooms now with air conditioner details.

For some more realistic modelling of AC and room association, see: https://learn.compactappliance.com/selecting-a-room-air-conditioner/