**The University of the West Indies, St. Augustine**
**COMP 2603 Object Oriented Programming 1**
**Assignment 2**
**2022/2023 Semester 2**

**Due Date: March 17, 2023 at 10:00 p.m.**
**Lockout Date: March 19, 2023 at 10:00pm**

**Overview:** This assignment requires you to write code for a cooling simulation involving rooms and devices. Three types of devices can be used to cool a room: air conditioners, ceiling fans, and standing fans. Air conditioners and ceiling fans cannot be removed from a room once they are added. Standing fans however are portable devices that can be swapped between rooms only if they are not in use. The table below shows the different properties of each device: portability, breeziness, noisiness, and cooling power.

| Device | Portable | Breeziness | Noisiness | Cools by |
|---|---|---|---|---|
| Air Conditioner | no | 0 | 0 | 5 |
| Ceiling Fan | no | 2 | 0 | 3 |
| Standing Fan | yes | 2 | 2 | 1 |

Table 1

You are required to create 4 rooms with the following properties such that the breeziness, noisiness, and ending temperature of the rooms can be quantified and assessed for correctness.

| Room | Starting Temperature | Required Properties | Air Conditioners | Ceiling Fans | Standing Fans |
|---|---|---|---|---|---|
| 1 | 30 | Only Breezy or Noisy Devices turned on | 1 | 1 | 3 |
| 2 | 35 | No Breezy and No Noisy Devices turned on | 2 | 1 | 4 |
| 3 | 37 | All Noisy Devices turned off | 1 | 2 | 2 |
| 4 | 25 | Very Breezy and Very Noisy and Very Cold | 1 | 1 | 1 |

Table 2

**Submission Instructions**
The code for each class in the application should be written in separate source files.

Ensure your student ID is documented in each file. Upload a zipped file of your source to the myElearning course page by the deadline.

### UML Diagram of Domain Classes

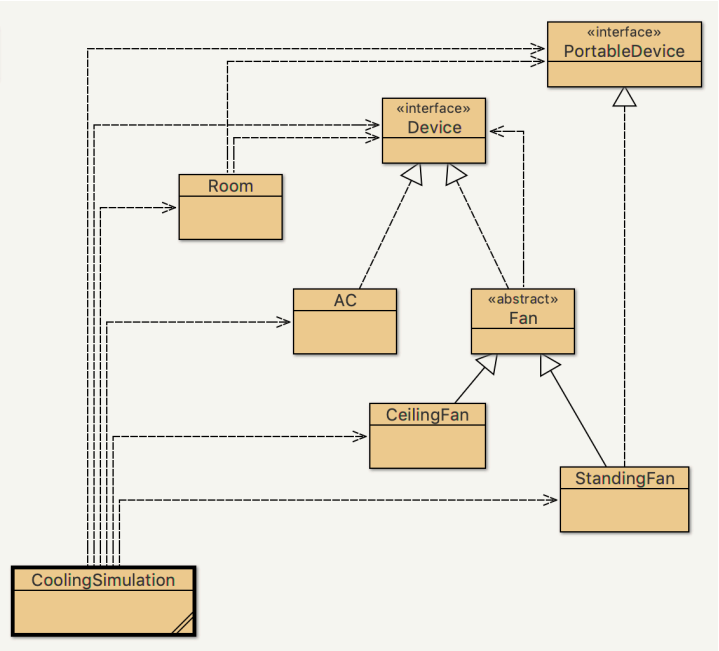



Figure 2: UML Diagram of Domain Classes

### Device Interface

The Device interface specifies the following behaviour for a device:

| Method Signature | Return Type | Purpose |
|---|---|---|
| getID( ) | String | Returns the ID of a device |
| isBreezy( ) | boolean | Returns true if the device has a breeziness greater than zero, false otherwise |
| isNoisy( ) | boolean | Returns true if the device has a noisiness greater than zero, false otherwise |
| isOn( ) | boolean | Returns true if the device is turned on, false otherwise |
| turnOn( ) | void | Turns a device on |
| turnOff( ) | void | Turns a device off |
| coolsBy( ) | int | Returns the temperature drop produced by a device |
| getBreeziness( ) | int | Returns the breeziness of a device |
| getNoisiness( ) | int | Returns the noisiness of a device |

### PortableDevice Interface

The PortableDevice interface does not specify any behaviour. It allows a standing fan to be subtyped as a PortableDevice.

### AC class

The AC class implements the Device interface with the behaviour required for air conditioners in Table 1. IDs take the following pattern: AC1, AC2, AC3 for AC objects.

### Fan class

The Fan class is abstract and it implements the Device interface with only the common behaviour implemented for fans in Table 1. IDs are controlled by this class as 1, 2, 3. Subclasses then modify these values by prefixing appropriate characters based on the type of fan. The Fan constructor accepts an integer parameter for a fan's noisiness which can vary.

### CeilingFan class

The CeilingFan class is a subclass of the Fan class and implements the Device interface with the behaviour required for ceiling fans in Table 1. IDs take the following pattern: CFAN1, CFAN2, CFAN3 for CeilingFan objects.

### StandingFan class

The StandingFan class is a subclass of the Fan class and implements the PortableDevice interface and the Device interface with the behaviour required for standing fans in Table 1. It cannot be subclassed. IDs take the following pattern: SFAN1, SFAN2, SFAN3 for StandingFan objects.

### Room class

Supply attributes of your own for the Room class which must have the following behaviour:

| Method Signature | Return Type | Purpose |
|---|---|---|
| Room(int startingTemperature) | | Constructor |
| getDevices( ) | ArrayList<Device> | Returns an ArrayList of devices |
| addDevice( Device d) | boolean | Return true if a device was successfully inserted into the devices ArrayList, false if the device already exists in the collection. |
| removeDevice(Device d) | boolean | Removes a device from the devices ArrayList if it is a PortableDevice and returns true, false otherwise |
| getTemperatureDrop( ) | int | Returns the temperature drop produced by all of the devices turned on in the room |
| getTemperature( ) | int | Returns the temperature of the room (based on the devices turned on in the room) |
| getBreeziness( ) | int | Returns the breeziness of the room (produced by the devices turned on in the room) |
| getNoisiness( ) | int | Returns the noisiness of the room (produced by the devices turned on in the room) |
| printState( ) | void | Prints the state of the room and the devices. |

**CoolingSimulation class**

The CoolingSimulation class has an ArrayList of Rooms. It has the following behaviour:

| Method Signature | Return Type | Purpose |
|---|---|---|
| CoolingSimulation( ) | | Constructor |
| getRooms( ) | ArrayList<Room> | Returns an ArrayList of rooms |
| getRoom( int i) | Room | Returns a Room object stored in the rooms ArrayList at the particular index if the index is between 1-4 inclusive. |
| createRooms( ) | void | Creates the Room objects in Table 2 and inserts them into the rooms ArrayList |
| createDevices( ) | void | Creates the Device objects in Table 2 and associate them with the appropriate room objects stored in the rooms ArrayList |
| swapPortableDevices (Room r1, Room r2) | void | Removes PortableDevices that are turned off in Room r1, and add them to Room r2 |
| coolRoom1( ) | void | Turns on the appropriate devices according to the room's requirements in Table 2 |
| coolRoom2( ) | void | Turns on the appropriate devices according to the room's requirements in Table 2 |
| coolRoom3( ) | void | Turns on the appropriate devices according to the room's requirements in Table 2 |
| coolRoom4( ) | void | Turns on the appropriate devices according to the room's requirements in Table 2. Additional devices from other rooms may be necessary |
| printStates( ) | void | Prints the state of all room objects in the rooms ArrayList |

In your CoolingSimulation class, provide the following main method:

```
public static void main(String[] args){
        CoolingSimulation sim = new CoolingSimulation();
        sim.createRooms();
        sim.createDevices();
        sim.coolRoom1();
        sim.coolRoom2();
        sim.coolRoom3();
        sim.coolRoom4();
        sim.printStates();
 }
```

**UML Diagram of Domain and Test Classes**

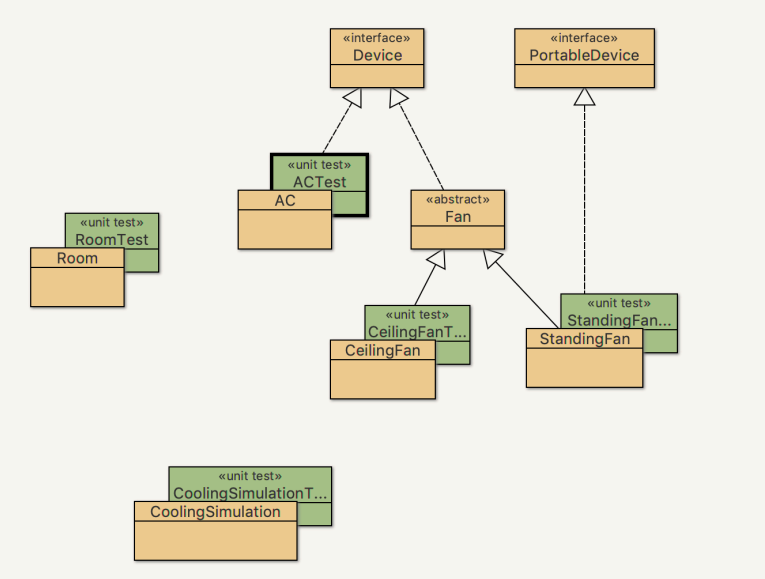



Figure 2: UML Diagram of Domain Classes and Test Classes

You will be provided with unit classes (indicated by green classes in Figure 2) that will check the functionality of your classes and test whether they conform to the assignment specifications.

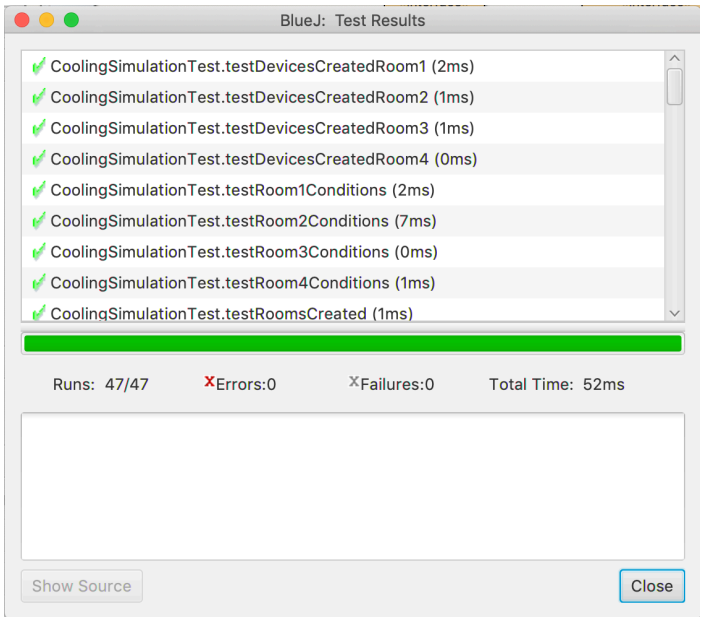
Add these incrementally to your project as you build your solution. The dependencies are listed in the table below:

| TestClass | Required Classes |
|---|---|
| ACTest | Device, AC |
| CeilingFanTest | Device, PortableDevice, Fan, CeilingFan |
| StandingFanTest | Device, PortableDevice, Fan, StandingFan |
| RoomTest | Device, Room, CeilingFan, StandingFan, AC, |
| CoolingSimulationTest | CoolingSimulation, Room |

To test individual classes: Right-click on the appropriate test class and select “Test All”

To test the entire program systematically: Click on “Run Tests” button.

Ensure that your program passes all of the tests supplied.




**Submission Instructions**

- Write the Java code for each class in the application using BlueJ.
- Document your student ID at the top of each file within a comment block.
- Upload a ZIP file of your compiled project source and class files to the myElearning course page by the deadline. Submissions that do not compile will receive 0 marks. Empty submission files or faulty zip files will receive 0 marks. Check your submitted files for completeness and correctness.
- Name your ZIP file as follows: **FirstName_LastName_ID_A2.zip.** Marks will be deducted for submissions that do not conform to this naming convention.
- Sign and submit the University Plagiarism declaration confirming that you are submitting your own original work and that you have not copied or collaborated with other students.
- Early submission points can be earned if you submit before the due date. No penalties will be applied but no early points will be earned if you submit by the lockout date.

**Important Information Regarding Academic Conduct**

- This is an individual assignment. You must attempt this assignment by yourself without any help from others, aside from the course lecturer, tutor or marker.
- You may use legitimate resources on the Internet, in books, or from the course notes to assist (unless prohibited by a question). Copying or modifying such content does not make it yours. Indicate on your submission any or all sources used in your answers within comments at the end of your files. Identify code snippets that you did not write yourself but reused from some source.
- You are not allowed to communicate, share or disclose any aspect of your solutions/work in any form to anyone except for the course lecturer, tutors, markers, and examiners.
- You are not allowed to assist others with this assignment.
- University plagiarism and academic misconduct policies apply fully.
- No part of your submission should be made publicly available even after the due date without permission from the course lecturer.
- The use of Chegg.com, CourseHero.com or any tutoring or homework assistance services, online or not, are prohibited.
- If you are faced with extenuating circumstances, contact your lecturer right away. Concessions may be arranged on a case-by-case basis. Be honest with your claims and provide evidence where possible.