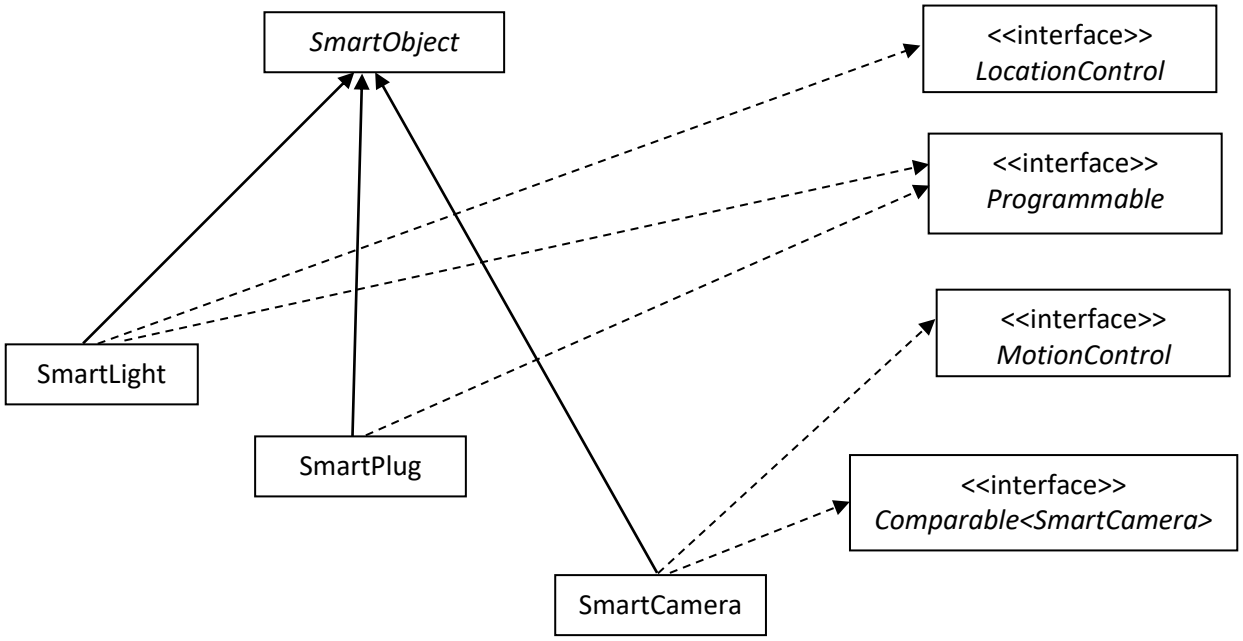
In this homework, you are expected to implement a simple smart home system. In a smart home system, there might be several intelligent objects such as smart light, smart plug, smart camera, etc. These devices can be connected to the internet and they can be controlled by households remotely. Nowadays, we can use such devices to improve smart life experience thanks to internet technology and smartphones.

Our simple smart home system has the following OOP class hierarchy:



Please find the class details below.

1) Implement an abstract class ***SmartObject*** using the following UML diagram.

SmartObject	
-	alias: String
-	macId: String
-	IP: String
-	connectionStatus: boolean
+	SmartObject()
+	connect(IP: String): boolean
+	disconnect(): boolean
+	SmartObjectToString(): void
+	controlConnection(): boolean
+	*testObject(): boolean*
+	*shutDownObject(): boolean*
+	getter/setter methods for data fields

- ***SmartObject*** is an abstract superclass of **SmartLight**, **SmartPlug** and **SmartCamera** classes.
- The data field **alias** represents the name of a smart device. Each device has a unique **macId** to connect to the internet. The data field **IP** (Internet Protocol) a networking protocol to communicate with other devices. The data field **connectionStatus** represents whether the smart object is connected to the internet or not.
- It has a concrete method **connect** which connects the smart object with the given **IP** value. It sets the **IP** and **connectionStatus** properties of the smart object and prints a message such as:

  "SmartCam1 connection established"
- It has a concrete method **disconnect** which disconnects the smart object. It sets **IP** and **connectionStatus** properties with appropriate values.
- It has a concrete method of **SmartObjectToString** which prints the details about a smart object as the following:

```
This is SmartCamera device SmartCam1
  MacId: AA:BB:CC
  IP: 10.0.0.100
```

- It has a concrete method of **controlConnection** which controls the connection of the smart object. If the device is not connected it prints the following message and return false.

```
This device is not connected. SmartCamera -> SmartCam1
```

- It has two abstract methods, ***testObject*** and ***shutdownObject***, with the given signatures above.
- There are setter/getter methods.

2) Implement a ***LocationControl*** interface using the following UML diagram.

	<<interface>> ***LocationControl***
+	*onLeave(): void*
+	*onCome(): void*

- ***LocationControl*** interface represents the property of controlling a smart device based on the location of a household.
- For example, a household may choose to turn on a device automatically before coming (with method ***onCome()***) to the house or turn off a device automatically after leaving the house (with method ***onLeave()***).
- The ***onLeave()*** and ***onCome()*** abstract methods should be implemented by the classes that implements ***LocationControl*** interface.

3) Implement a ***Programmable*** interface using the following UML diagram.

	<<interface>> ***Programmable***
+	*setTimer(seconds: int): void*
+	*cancelTimer(): void*
+	*runProgram(): void*

- ***Programmable*** interface represents the property of programming a smart device.
- A household may choose to turn on a device automatically based on setting a timer (with method ***setTimer***). The device should be turned on or turned off after given amount of **seconds** value.
- A household may choose to cancel the timer of a smart device (with method ***cancelTimer***).
- The method ***runProgram*** should check the program time of a smart device when its timer is set and it should turn on or turn off the device if the program time matches with the current time.
- The abstract methods ***setTimer***, ***cancelTimer*** and ***runProgram*** should be implemented by the classes that implements ***Programmable*** interface.

4) Implement a ***MotionControl*** interface using the following UML diagram.

	<<interface>> ***MotionControl***
+	*controlMotion(hasMotion: boolean, isDay: boolean): boolean*

- ***MotionControl*** interface represents the property of motion capturing for a smart device.
- The method ***controlMotion*** takes two **boolean** parameters, in which the first one represents the presence of motion and the second one represents whether it is daytime or night time. It should start recording if there is a captured motion.
- The abstract method ***controlMotion*** should be implemented by the classes that implements ***MotionControl*** interface.

5) Implement a **SmartLight** class using the following UML diagram.

	SmartLight
-	hasLightTurned: boolean
-	programTime: Calendar
-	programAction: boolean
+	SmartLight(alias: String, macId: String)
+	turnOnLight(): void
+	turnOffLight(): void
+	testObject(): boolean
+	shutDownObject(): boolean
+	onLeave(): void
+	onCome(): void
+	setTimer(seconds: int): void
+	cancelTimer(): void
+	runProgram(): void
+	getter/setter methods

- **SmartLight** is a subclass of **SmartObject** class and it implements ***LocationControl*** and ***Programmable*** interfaces.

- It has three data fields: **hasLightTurned** takes the value of true if the light is turned on, **programTime** keeps the exact time of automatic activation of the smart device, **programAction** keeps the next action of the smart device (either turn on or turn off).
- The method **turnOnLight** should check the connection of a smart light firstly and it should turn on the light by printing the opening time (i.e. current time). In this method, you should update the **hasLightTurned** property appropriately. If the light is newly turned on, it should print a message as follows:

```
Smart Light - Living Room Light is turned on now (Current time: 10:14:39)
```

  - If the smart light object has been already turned on, then it should print a message as the following:

```
Smart Light - Living Room Light has been already turned on
```

- The method **turnOffLight** should check the connection of a smart light firstly and it should turn off the light by printing the power-off time (i.e. current time). In this method, you should update the **hasLightTurned** property appropriately. If the light is newly turned off, it should print a message as the following:

```
Smart Light - Living Room Light is turned off now (Current time: 11:15:45)
```

  - If the smart light object has been already turned off, then it should print a message as the following:

```
Smart Light - Living Room Light has been already turned off
```

- The method **testObject** should check the connection of a smart light firstly and it should test the functionalities of the smart light by invoking methods **SmartObjectToString**, **turnOnLight**, and **turnOffLight**. An example output of the **testObject** method is as follows:

```
This is SmartLight device Living Room Light
   MacId: AA:BB:CC
   IP: 10.0.0.100
Smart Light - Living Room Light is turned on now (Current time: 10:14:39)
Smart Light - Living Room Light is turned off now (Current time: 11:15:45)
Test completed for SmartLight
```

  - If the smart light object was not connected to the system, then this method should return the value of **false**.

- The method **shutDownObject** should check the connection of a smart light firstly. Then, it should turn off the light (if it has been already turned on) after calling **SmartObjectToString** method. If the smart light object was not connected to the system, then this method should return the value of **false**.
- The method **onLeave** should check the connection of a smart light firstly and then it should turn off the light. The method **onCome** should check the connection of a smart light firstly, and then it should turn on the light. The example outputs for both methods should be as the following:

```
On Come -> Smart Light - Living Room Light
Smart Light - Living Room Light is turned on now (Current time: 11:15:45)
On Leave -> Smart Light - Living Room Light
Smart Light - Living Room Light is turned off now (Current time: 11:16:45)
```

- The method **setTimer** should set the timer of a smart light with the given amount of seconds. Firstly, it should check the connection of a smart light, and then it should set the **programTime** property of the smart light by using the current time. Then, it should add the given amount of seconds to the **programTime** property. Lastly, it should print the following messages by checking the **hasLightTurned** property:

```
Smart light - Living Room Light will be turned off 5 seconds later!
(Current time: 11:15:45)
```

OR

```
Smart light - Living Room Light will be turned on 5 seconds later!
(Current time: 11:15:45)
```

- The method **cancelTimer** should check the connection of a smart light, and then it should cancel the timer of a smart light by assigning the value of **null** to the **programTime** property.
- The method **runProgram** should check the connection of a smart light firstly. Then, it should either turn on or turn off the light by checking the **programAction** property of the smart light if the **programTime** value equals to the current time.
- An example flow of **setTimer** and **runProgram** methods are given below:

```
Smart light - Living Room Light will be turned off 5 seconds later!
(Current time: 11:32:14)

RunProgram -> Smart Light - Living Room Light
Smart Light - Living Room Light is turned off now (Current time: 11:32:19)
```

- You should assign the value **null** to the **programTime** property after printing messages.
- There are setter/getter methods.

6) Implement a **SmartPlug** class using the following UML diagram.

SmartPlug	
-	status: boolean
-	programTime: Calendar
-	programAction: boolean
+	SmartPlug(alias: String, macId: String)
+	turnOn(): void
+	turnOff(): void
+	testObject(): boolean
+	shutDownObject(): boolean
+	setTimer(seconds: int): void
+	cancelTimer(): void
+	runProgram(): void
+	getter/setter methods

- **SmartPlug** is a subclass of **SmartObject** class and it implements ***Programmable*** interface.
- It has three data fields: **status** takes the value of true if the plug is turned on, **programTime** keeps the exact time of automatic activation of the smart device, **programAction** keeps the next action of the smart device (either turn on or turn off).
- The **turnOn** method should check the connection of a smart plug firstly and it should turn on it by printing the power on time (i.e. current time). In this method, you should update the **status** property appropriately. If the plug is newly turned on, it should print a message as the following:

```
Smart Plug - Kitchen Plug 1 is turned on now (Current time: 11:32:14)
```

  - If the smart plug object has already been turned on, then it should print a message as follows:

```
Smart Plug - Kitchen Plug 1 has been already turned on
```

- The **turnOff** method should check the connection of a smart plug firstly and it should turn off it by printing the power off time (i.e. current time). In this method, you should update the **status** property appropriately. If the plug is newly turned off, it should print a message as the following:

```
Smart Plug - Kitchen Plug 1 is turned off now (Current time: 11:49:41)
```

  - If the smart light object has already been turned off, then it should print a message as follows:

```
Smart Plug - Kitchen Plug 1 has been already turned off
```

- The **testObject** method should check the connection of a smart light firstly and it should test the functionalities of the smart light by invoking **SmartObjectToString**, **turnOn**, and **turnOff** methods. An example output of the **testObject** method is as the following:

```
This is SmartPlug device Kitchen Plug 1
   MacId: DD:KK:FF
   IP: 10.0.0.102
Smart Plug - Kitchen Plug 1 is turned on now (Current time: 11:49:41)
Smart Plug - Kitchen Plug 1 is turned off now (Current time: 11:49:41)
Test completed for SmartPlug
```

  - If the smart plug object was not connected to the system, then this method should return the value of **false**.

- The **shutDownObject** method should check the connection of a smart light firstly. Then, it should turn off it (if it has already been turned on) after calling **SmartObjectToString** method. If the smart plug object was not connected to the system, then this method should return the value of **false**.

- The **setTimer** method should set the timer of a smart plug with the given amount of seconds. Firstly, it should check the connection of a smart plug, and then it should set the **programTime** property of the smart plug by using the current time. Then, it should add the given amount of seconds to the **programTime** property. Lastly, it should print the following messages by checking the **status** property:

```
Smart plug - Kitchen Plug 1 will be turned on 5 seconds later! (Current
time: 11:49:41)
```

OR

```
Smart plug - Kitchen Plug 1 will be turned off 5 seconds later! (Current
time: 11:49:41)
```

- The **cancelTimer** method should check the connection of a smart plug, and then it should cancel the timer of a smart plug by assigning the value of **null** to the **programTime** property.

- The **runProgram** method should check the connection of a smart plug firstly. Then, it should either turn on or turn off it by checking the **programAction** property of the smart plug if the **programTime** value equals to the current time.

- An example flow of setTimer and runProgram methods are given below:

```
Smart plug - Kitchen Plug 1 will be turned on 5 seconds later! (Current
time: 11:49:41)

RunProgram -> Smart Plug - Kitchen Plug 1
Smart Plug - Kitchen Plug 1 is turned on now (Current time: 11:49:46)
```

  - You should assign the value **null** to the **programTime** property after printing messages.

- There are setter/getter methods.

7) Implement a **SmartCamera** class using the following UML diagram.

SmartCamera	
-	status: boolean
-	batteryLife: int
-	nightVision: boolean
+	SmartCamera(alias: String, macId: String, nightVision: boolean, batteryLife: int)
+	recordOn(isDay: boolean): void
+	recordOff(): void
+	testObject(): boolean
+	shutDownObject(): Boolean
+	controlMotion(hasMotion: Boolean, isDay:boolean): boolean
+	compareTo(smartCamera: SmartCamera): int
+	toString(): String
+	getter/setter methods

- **SmartCamera** is a subclass of **SmartObject** class and it implements ***MotionControl*** and ***Comparable*** interfaces.
- It has three data fields: **status** takes the value of true if the camera is recording, **batteryLife** represents the battery life of the camera, **nightVision** represents the night vision feature of the camera.
- The **recordOn** method should check the followings firstly: the connection of a smart camera, the **isDay** value and the **nightVision** feature of the smart camera. Based on these controls it should start recording. In this method, you should update the **status** property appropriately. If the camera is newly turned on, it should print a message as the following:

```
Smart Camera - Garden Cam is turned on now
```

  - If the smart camera object has already been turned on, then it should print a message as the following:

```
Smart Camera - Garden Cam has been already turned on
```

  - If the time of the day is night time (i.e., **isDay** is false) and there is no night vision feature of the camera (i.e., **nightVision** is false), then it should display the following message:

```
Sorry! Smart Camera - Garden Cam does not have night vision feature.
```

- The **recordOff** method should check the connection of a smart camera firstly and it should stop recording. In this method, you should update the **status** property appropriately. If the camera is newly turned off, it should print a message as the following:

```
Smart Camera - Child Room Cam is turned off now
```

  - If the smart camera object has already been turned off, then it should print a message as the following:

```
Smart Camera - Child Room Cam has been already turned off
```

- The **testObject** method should check the connection of a smart camera firstly and it should test the functionalities of the smart camera by invoking **SmartObjectToString**, **recordOn(true)** (i.e daytime), and **turnOff** methods firstly. Then, it should invoke **recordOn(false)** (i.e night time), and **turnOff** methods. An example output of the **testObject** method is as the following:

```
This is SmartCamera device Child Room Cam
   MacId: JJ:KK:LL
   IP: 10.0.0.107
```

```
Test is starting for SmartCamera day time
Smart Camera - Child Room Cam is turned on now
Smart Camera - Child Room Cam is turned off now
Test is starting for SmartCamera night time
Sorry! Smart Camera - Child Room Cam does not have night vision feature.
Smart Camera - Child Room Cam has been already turned off
Test completed for SmartCamera Test completed for SmartCamera
```

  - If the smart camera object was not connected to the system, then this method should return the value of **false**.

- The **shutDownObject** method should check the connection of a smart camera firstly. Then, it should turn off it (if it has been already turned on) after calling **SmartObjectToString** method. If the smart camera object was not connected to the system, then this method should return the value of **false**.
- The **controlMotion** method should check the **hasMotion** parameter, and it should print "Motion not detected!" if it is false; "Motion detected!" otherwise. Then, it should check **isDay** parameter, if it is true (i.e., daytime) it should start recording. If it is false, it should check the **nightVision** property of the camera firstly, and then it should start recording or not.
- The **compareTo** method should check the **batteryLife** of the smart camera with the given parameter **smartCamera**. If the battery life of the smart camera is greater than the battery life of the **smartCamera** parameter, it should return the value of 1. If they are equal, return the value of 0. If it is smaller, then return the value of -1.
- The **toString** method should return a representative string for the smart camera as the following:

```
SmartCamera -> Child Room Cam's battery life is 30 status is recording
```

- There are setter/getter methods.

8) Implement a **SmartHome** class using the following UML diagram.

SmartHome	
-	smartObjectList: ArrayList<SmartObject>
+	SmartHome()
+	addSmartObject(smartObject: SmartObject): boolean
+	removeSmartObject(smartObject: SmartObject): boolean
+	controlLocation(onCome: boolean): void
+	controlMotion(hasMotion: boolean, isDay: boolean): void
+	controlProgrammable(): void
+	controlTimer(seconds: int): void
+	controlTimerRandomly(): void
+	sortCameras(): void
+	getter/setter methods

- **SmartHome** represents a smart house containing several smart objects (**smartObjects** are kept in **smartObjectList**).
- The **addSmartObject** method adds the given **smartObject** to the **smartObjectList**. Firstly, it invokes **connect** method of the **smartObject** by sending its **IP** value.
  - The **IP** value is set as "10.0.0.x", where x represents the index of the smart object in the **smartObjectList** starting from 100. For example, if it is the first smart object added to the list, the IP value should be "10.0.0.100". If it is the second smart object added to the list, the IP value should be "10.0.0.101", etc.

- Then, it should invoke the **testObject** method for the given **smartObject**.
- An example output of **addSmartObject** method is as the following:

```
--------------------------------------------------------------------------
--------------------------------------------------------------------------
Adding new SmartObject
--------------------------------------------------------------------------
Living Room Light connection established
Test is starting for SmartLight
This is SmartLight device Living Room Light
   MacId: AA:BB:CC
   IP: 10.0.0.100
Smart Light - Living Room Light is turned on now (Current time: 12:25:36)
Smart Light - Living Room Light is turned off now (Current time: 12:25:36)
Test completed for SmartLight
```

- The **removeSmartObject** method removes the given **smartObject** from the **smartObjectList**.
- The **controlLocation** method should traverse the **smartObjectList** and if it finds an object implementing ***LocationControl*** interface, then, it should invoke either **onCome** or **onLeave** method of it by checking the **onCome** boolean parameter. If **onCome** boolean parameter is true, then it should invoke **onCome** method, and **onLeave** method otherwise. An example output of this method is given below:

```
--------------------------------------------------------------------------
--------------------------------------------------------------------------
LocationControl : OnCome
--------------------------------------------------------------------------
On Come -> Smart Light - Living Room Light
Smart Light - Living Room Light is turned on now (Current time: 12:25:36 )
On Come -> Smart Light - Kitchen Light
Smart Light - Kitchen Light is turned on now (Current time: 12:25:36 )
```

- The **controlMotion** method should traverse the **smartObjectList** and if it finds an object implementing ***MotionControl*** interface, then, it should invoke its method **controlMotion** by sending **hasMotion** and **isDay** boolean parameters. An example output of this method is given below:

```
--------------------------------------------------------------------------
--------------------------------------------------------------------------
MotionControl: HasMotion, isDay
--------------------------------------------------------------------------
Motion detected!
Smart Camera - Garden Cam is turned on now
Motion detected!
Smart Camera - Child Room Cam is turned on now
Motion detected!
Smart Camera - Gate Cam is turned on now
```

- The **controlProgrammable** method should traverse the **smartObjectList** and if it finds an object implementing ***Programmable*** interface, then, it should invoke its method **runProgram**. An example output of this method is given below:

```
--------------------------------------------------------------------------
--------------------------------------------------------------------------
Programmable: runProgram
--------------------------------------------------------------------------
runProgram -> Smart Light - Living Room Light
Smart Light - Living Room Light is turned off now (Current time: 12:42:36)
runProgram -> Smart Plug - Living Room Plug 1
Smart Plug - Living Room Plug 1 is turned on now (Current time: 12:42:36)
runProgram -> Smart Plug - Living Room Plug 2
Smart Plug - Living Room Plug 2 is turned on now (Current time: 12:42:36)
```

- The **controlTimer** method should traverse the **smartObjectList** and it should search for a smart object implementing ***Programmable*** interface. In case of finding such an object, it should invoke **setTimer** method of it if the given seconds value is greater than 0, and it should invoke **cancelTimer** method if the given seconds value is equal to 0. An example output of this method is given below:

```
------------------------------------------------------------------------
------------------------------------------------------------------------
Programmable: Timer = 10 seconds
------------------------------------------------------------------------
Smart light - Living Room Light will be turned off  10 seconds later!
(Current time: 12:44:24)
Smart light - Kitchen Light will be turned off  10 seconds later! (Current
time: 12:44:24)
Smart plug - Kitchen Plug 1 will be turned on 10 seconds later! (Current
time: 12:44:24)
Smart plug - Kitchen Plug 2 will be turned on 10 seconds later! (Current
time: 12:44:24)
Smart plug - Living Room Plug 1 will be turned on 10 seconds later!
(Current time: 12:44:24)
Smart plug - Living Room Plug 2  will be turned on 10 seconds later!
(Current time: 12:44:24)
```

- The **controlTimerRandomly** method should traverse the **smartObjectList** and it should search for a smart object implementing ***Programmable*** interface. In case of finding such an object, it should invoke its method **setTimer** with the value of 5 or 10 seconds randomly. If the random number is 0, then it should invoke **cancelTimer** method. Here, the random number should be 0, 5, or 10. An example output of this method is given below:

```
------------------------------------------------------------------------
------------------------------------------------------------------------
Programmable: Timer = 0, 5 or 10 seconds randomly
------------------------------------------------------------------------
Smart light - Living Room Light will be turned off  10 seconds later!
(Current time: 12:44:24)
Smart light - Kitchen Light will be turned off  10 seconds later! (Current
time: 12:44:24)
Smart plug - Kitchen Plug 1 will be turned on 10 seconds later! (Current
time: 12:44:24)
Smart plug - Kitchen Plug 2 will be turned on 5 seconds later! (Current
time: 12:44:24)
Smart plug - Living Room Plug 1 will be turned on 5 seconds later!
(Current time: 12:44:24)
Smart plug - Living Room Plug 2  will be turned on 10 seconds later!
(Current time: 12:44:24)
```

- The **sortCameras** method should traverse the **smartObjectList** and it should search for smart cameras implementing ***Comparable*** interface. Then, it should invoke **Arrays.sort** method to sort smart cameras based on the battery life. An example output of this method is given below:

```
------------------------------------------------------------------------
------------------------------------------------------------------------
Sort Smart Cameras
------------------------------------------------------------------------
SmartCamera -> Child Room Cam's battery life is 30 status is recording
SmartCamera -> Gate Cam's battery life is 50 status is recording
SmartCamera -> Garden Cam's battery life is 60 status is recording
```

9) Use the given **Test** class for your program. Please analyze the Test class carefully by reading the comments. The test class creates a **SmartHome** instance and adds several smart objects into it. It invokes most of the methods in the **SmartHome** class. There is a special method of **sleepSystem** which sleeps the system for 5 seconds. This method is used for checking **setTimer** and **runProgram** methods of the smart objects implementing ***Programmable*** interface. An example console output of the test program is given in "consoleOutput.txt". You are not required to do File I/O, the output is given in a file due to long output size.

This is a simple scenario to test your class implementations. There might be other test cases, too. Therefore, please pay attention to use the same class, method and variable names in your implementations. You are allowed to increase the number of methods in the classes; however, you cannot decrease the number of them. Additionally, you are not allowed to increase the number of data fields in each class. It should be noted that selected parts will be graded in your solution.