**Programming 2019/20**

**Java Coursework: “Smart Home App”**

**Deadline**

**11 pm, Thursday 14th May 2020**

**Background**

A fictional UK-Finnish company, **Alikoti,** develops eco-friendly smart home technologies. Since 1999 they have developed fully integrated Smart Home Solutions. Since 2011 they have begun selling standalone smart home devices, including smart devices, smart lights and smart fridges to be supportedby third-party software. Last year the company made the decision to build their own Smart Home App. The Smart Home App can register devices in a home and control devices.

**Overview**

This coursework focuses on the classes required to support the app. It is not concerned with building a front end (i.e. a graphical user interface is not required). All interaction and testing will be carried out in a Java console. The coursework is broken down into several steps, each with accompanying testing. The testing will be used by the assessors to evaluate functionality. The coursework must be developed as a single Netbeans **project**. The core class for the App is a **SmartDevice** class. It represents a generic smart device and should have three member variables: a name, a location and a switchedOn value. The switchedOn value refers to whether the device is switched on or off. The location uses the following system:



The location, represented by a double, is made of two parts: the integer part (12 in the example above) and a fractional part (4 in the example above). The integer part corresponds to the **room** in a particular house. The fractional part refers to the actual wall **socket** in that room that the device is plugged into. Each location is assumed to be unique. The **SmartDevice** will be managed within a **SmartHome** class. The **SmartHome** class provides a number of methods to populate, display, analyse and update values of an array of SmartDevice objects. The **SmartHome** classwill also allow new smart devices to be registered, and control the switchedOn value for all smart devices, both at the home level and at a room level. The SmartDevice class will be extended to add a **SmartFridge** classand **SmartLamp** class, both of which have additional properties and behaviours.

**Core Tasks**

|  |  |  |
| --- | --- | --- |
| **Step 1** | **SmartDevice class** (SmartDevice.java) | **5 marks** |
| **Tests** (Step1.java) | **20 marks** |

This step focuses on the design, creation and testing of the **SmartDevice** class. Assessment of code quality will be based on the **SmartDevice** class design and code. Assessment of functionality will be based on execution of a separate test class (Step1.java).

**Design** and **write** the **SmartDevice** class (SmartDevice.java). This class should have three private member variables: name (String), location (double) and switchedOn (boolean). Name and location member variables should only be accessed and updated through standard get/set methods which you should also write. The switchedOn member variable should only be accessed through switchOn/switchOff/isSwitchedOn methods which you should also write.

**Design** and **write** a **constructor** which must allow all three member variables to be populated.

Design and write a **toString()** method**.** This method should return a well-formatted String containing the variable name-value pairs stored inside the object. For example:

**Name:** device

**Location:** **12.1**

**Switched On:** **false**

Create a test script file (**Step1.java)** to **test** the **SmartDevice** class. All test sequences should be placed inside the main() method of this class and use the console for input and output. This class will be executed when we assess the functionality of your work.

Your testing must include all of the following tests, one after the other.

1. Create object -
   1. Create a SmartDevice object with suitable parameter values.
   2. Use the toString() method to display the object’s.
2. Create an array of SmartDevices in the main() method -
   1. Using console input, allow the user to specify the number of Smart Devices to store in an array. Create an array with the given size in main(). Populate each element of the array using a loop and user input for each value. Test that the array is populated correctly by using a loop combined with the toString() method to display the details of all the SmartDevices.

(Example output)

----------

-DEVICE **1**-

----------

**Name:** device **1**

**Location:** **12.1**

**Switched On:** **false**

----------

----------

-DEVICE **2**-

----------

**Name:** device **2**

**Location:** **6.2**

**Switched On:** **true**

----------

----------

-DEVICE **3**-

----------

**Name:** device **3**

**Location:** **1.1**

**Switched On:** **false**

----------

1. Update the switchedOnvalue for one of the devices in the array -
   1. Allow the user to select an item (SmartDevice) from the array by entering an index value.
   2. Execute a switchOn()() or switchOff() operation on the selected SmartDevice to change the switchedOn value (from true to false, or from false to true).
   3. Verify that the update procedure worked correctly by using a loop combined with the SmartDevice toString() method to display details of all the SmartDevices.

|  |  |  |
| --- | --- | --- |
| **Step 2** | **SmartHome class** (SmartHome.java) | **5 marks** |
| **Tests** (Step2.java) | **20 marks** |

This step focuses on the design, creation and testing of the **SmartHome** class. Assessment of the code quality will be based on the **SmartHome** class design and code. Assessment of functionality will be based on execution of a separate test class (Step2.java). The test script in **Step 1** will help you code this step.

**Design** and **write** the **SmartHome** class (SmartHome.java). This class should have only one private member variable: an array of SmartDevice objects.

**Design** and **write** two constructors. The first constructor should take one parameter: a size value. The constructor should create a new instance of the array using the size parameter. The initial array should be **unpopulated** - it should contain null values at every index. The second constructor should take one parameter: an array of pre-populated SmartDevices. This parameter should be assigned to the member variable.

Design and **write** two **insertDevice()** methods. The first insertDevice() method should take one parameter: a SmartDevice object. It should insert the SmartDevice at the first empty (null) array position and fail if there is no empty position in the array. The second insertDevice() method takes 3 parameters: name, location and switchedOn, parameters corresponding to the three member variables of the SmartDevice class. It creates a SmartDevice from the parameter values. It should then insert the SmartDevice at the first empty (null) position in the array and fail if there is no empty position.

**Design** and **write** two **getDevice()** methods. The first getDevice() method should take one parameter: an index value. It should return the SmartDevice object at the index of the array. If the index is too big or small for the array, or if there is no SmartDevice at the given index, then it should return null. The second getDevice() method should also have one parameter: a location value. The method returns the SmartDevice object at the given location. If no device is at the given location, then it should return null.

Design and write a **toggle()** method. This method takes one parameter: the location (a search parameter). The method attempts to locate the SmartDevice using the location parameter. If it finds a SmartDevice at the location then it should flip the switchedOn value of the SmartDevice (from false to true, or from true to false)

Design and write a **toString()** method**.** This method should return a well-formatted String containing the name-value pairs of the SmartDevices stored inside the array. If there are empty positions, these should be skipped (not appear in the returned String). For example:

----------

-DEVICE **1**-

----------

**Name:** device **1**

**Location:** **12.1**

**Switched On:** **false**

----------

----------

-DEVICE **2**-

----------

**Name:** device **2**

**Location:** **6.2**

**Switched On:** **true**

----------

----------

-DEVICE **3**-

----------

**Name:** device **3**

**Location:** **1.1**

**Switched On:** **false**

----------

Create a test script file (**Step2.java)** to **test** the **SmartHome** class. All test sequences should be placed inside the main() method and use the console for input and output. This class will be executed when we assess the functionality of your work.

Your testing must include all of the following tests, one after the other.

1. Create SmartHome object - the first constructor.
   1. Using console input, ask the user to specify the number of smart devices (size) to be held in the SmartHome object.
   2. Create a SmartHome object using the size value given by the user.
   3. Populate the SmartHome object with Smart devices using a loop, user inputs for each new SmartDevice, and the insertDevice() method.
   4. Verify that the SmartHome is correctly populated using the SmartHome object toString() method.
2. Create SmartHome object - using the second constructor.
   1. Using console input, ask the user to specify the number of items to be held in an array of SmartDevices. Create the array in main(). Populate each item in the array using a loop and user input for each new SmartDevice object.
   2. Create a SmartHome object using the populated array.
   3. Verify that the SmartHome object is correctly populated using the SmartHome object toString() method.
3. Get a device from the SmartHome object using an index and update switchedOn value.
   1. Ask the user for an index, and using the getDevice() method, return the specific SmartDevice at that index to the main().
   2. Execute a switchOn()() or switchOff() operation on this SmartDevice to change the switchedOn value (from true to false, or from false to true).
   3. Verify the values are updated using the SmartHome object’s toString() method.
4. Get a device from SmartHome object using a location and update SwitchedOn value
   1. Ask the user for a location and using the getDevice() method, return the specific SmartDevice at that location to the main().
   2. Execute a switchOn() or switchOff() operation on this SmartDevice to change the switchedOn value. ate switchedOn value
   3. Verify this using the SmartHome object’s toString() method.

|  |  |  |
| --- | --- | --- |
| **Step 3** | **Add functionality to SmartHome Class** (SmartHome.java) | **5 marks** |
| **Tests** (Step3.java) | **15 marks** |

This step focuses on the design, creation and testing of additional functionality to the **SmartHome** class. Assessment of the code quality will be based on the **SmartHome** class design and code. Assessment of functionality will be based on execution of a separate test class (Step 3.java).

The code should be inserted into the SmartHome class designed, built and tested in step 2.

Design and write an **addDevice()** method. The addDevice takes one parameter: a SmartDevice object. This method increases the size of the array by one, using the technique covered in the lectures, if the array is already full. It inserts the new device at the new final (empty) location of the array.

Design and write a **setAllInRoom()** method. This method takes two parameters: a room location search parameter (an integer corresponding to the room part of the location variable) and a new switchedOn value. The **setAllInRoom()** method attempts to locate a room location using the room parameter. If it finds such a room then it updates all the SmartDevices in that room to the new switchedOn value.

Design and write a **shutdown()** method. This method changes all the switchedOn values of all SmartDevices in the SmartHome array to false (i.e. switches them all off). (Please note that you will have to override this behaviour if you design and write the SmartFridge in a later step.)

Design and write the following change into the **insertDevice()** method. If the array is fully populated then the method executes (and passes the parameter value to) the **addDevice()** method.

Create a test script file (**Step3.java)** to **test** the additional functionality of the **SmartHome** class. All test sequences should be placed inside the main() method and use the console for input and output. This class will be executed when we assess the functionality of your work.

Your testing must include all of the following tests, one after the other.

1. Register a new device -
   1. Create a SmartHome object and fully populate it. See **Step 2 Test 1** or **Step 2 Test 2** for guidance.
   2. Given the array is now fully populated, create a new SmartDevice and attempt to add it to the SmartHome. This should invoke the addDevice() method and the SmartDevice object should be added, resulting in an increase in the size of the array in the SmartHome object.
   3. Verify that the additional the object is added and the array is increased using the SmartHome object’s toString() method.
2. Room level change -
   1. Run setAllInRoom() method for one of the existing rooms to change all the switchedOn values for the SmartDevices in that room.
   2. Verify the changes made using the SmartHome object’s toString() method.
3. Shutdown all devices
   1. Run shutdown()
   2. Verify the changes made using the SmartHome object’s toString() method.

**Inheritance Tasks**

These tasks all build and extend upon what has been done so far. The classes in the remaining tasks **extend** the SmartDevice. You should use inheritance mechanisms throughout the code and testing to create objects from the SmartFridge and SmartLamp classes.

|  |  |  |
| --- | --- | --- |
| **Step 4** | **SmartFridge Class extends SmartDevice** (SmartFridge.java) | **5 marks** |
| **Tests** (Step4.java) | **10 marks** |

This step focuses on the design, creation and testing of the **SmartFridge** class. Assessment of code quality will be based on the **SmartFridge** class design and code. Assessment of functionality will be based on execution of a separate test class (Step4.java).

**Design** and **write** a **SmartFridge** class(SmartFridge.java) which directly inherits **(extends)** from the **SmartDevice** class. This class has only one private member variable: currentTemperature (double). This member variable should only be accessed and updated through standard get/set methods.

Design and write a constructor. The constructor takes 3 parameters: currentTemperature, name and location. The constructor should call the superclass constructor with name, location and a true value for the superclass switchedOn member variable. It should set the value of currentTemperature from the parameter value.

Design and write **increment()** and **decrement()** methods. The increment() method increases the temperature by 1 degree, The decrement method decreases the temperature by 1 degree.

Design and write a **SmartFridge switchOff()** method, overriding the one inherited from SmartDevice. The switchOff() method should not change the switchedOn variable. The SmartFridge should not be switched off ever!

Design and write a **toString()** method. This method uses the superclass toString() method and adds the currentTemperature so it returns:

**Name:** fridge 1

**Location:** **3.1**

**Switched On:** **true**

**Current Temperature** 3

Create a test script file (**Step4.java)** to **test** the **SmartFridge** class. All test sequences should be placed inside the main() method and use the console for input and output. This class will be executed when we assess the functionality of your work. In order to complete these tests you must first complete **Step 3** above.

Your testing must include all of the following tests, one after the other.

1. Create SmartHome object - using the first constructor from **step 2**.
   1. Using console input, allow the user to specify the number of smart devices (size) to be held in a SmartHome object.
   2. Create a SmartHome object using the size value given by the user.
   3. Using **insertDevice()**, populate the SmartHome object using a loop and user input for each value **with a mixture of SmartDevice and SmartFridge** objects.
   4. Verify that the SmartHome is correctly populated using the SmartHome object’s toString() method.
2. Create SmartHome object - using the second constructor.
   1. Using console input, allow the user to specify the number of items to be held in an array. Create an array in main(). Populate each item in the array using a loop and user input for each value **with a mixture of SmartDevice and SmartFridge** objects.
   2. Create a SmartHome object using the populated array.
   3. Verify that the SmartHome object is correctly populated using the SmartHome object’s toString() method.
3. Using the shutdown() method, attempt to switch off a fridge.
   1. Using the SmartHome shutdown() method, attempt to shut all devices down.
   2. Verify that the SmartFridge(s) remain(s) switched on using the SmartHome object’s toString() method.
4. Increment and decrement temperature.
   1. For a selected fridge (use an approriate index and the getDevice() method), increment the temperature. Verify that the current temperature has been updated using the SmartHome object’s toString() method.
   2. For the selected fridge decrement the temperature. Verify that the current temperature has been updated using the SmartHome object toString() method.

|  |  |  |
| --- | --- | --- |
| **Step 5** | **SmartLamp Class extends SmartDevice** (SmartLamp.java)  **LightModes Custom Data Type** (LightModes.java) | **5 marks** |
| **Tests** (Step5.java) | **10 marks** |

This task focuses on the design, creation and testing of the **SmartLamp** class and a custom data type **LightModes** (LightModes.java). Assessment of code quality will be based on the **SmartLamp** class and **LightModes** custom data type design and code. Assessment of functionality will be based on execution of a separate test class (Step5.java).

**Design** and **write** a custom data type **LightModes** (LightModes.java). This custom data type should only allow three values:

|  |  |
| --- | --- |
| **LightModes** custom data type | |
| **Value** | **Detail** |
| NIGHT\_MODE | This mode turns the Lamp brightness into a night light, primarily for sleeping |
| SOFT\_MODE | This mode turns the Lamp brightness into a room background mode, used for watching television, eating, etc. |
| STANDARD\_ MODE | This mode turns the Lamp brightness into standard brightness, used for reading, doing work, etc. |

**Design** and **write** a **SmartLamp** class (SmartLamp.java) which directly inherits **(extends)** from the **SmartDevice** class. This class has only one private member variable: lightMode, built from the custom data type **LightModes** (see above). This member variable should only be accessed and updated through standard get/set methods inside the SmartLamp class.

Design and write a constructor for the **SmartLamp** class. The constructor takes 4 parameters: mode, name, location and switchedOn. The constructor calls the superclass constructor with name, location and switchedOn values and sets the lightMode variable from the mode parameter.

Design and write a **toString()** method for the **SmartLamp** class. This method uses the superclass toString() method and adds the mode, so it return strings such as:

**Name:** lamp1

**Location:** **3.1**

**Switched On:** **false**

**Mode:** STANDARD

Create a test script file (**Step5.java)** to **test** the **SmartLamp** class. All test sequences should be placed inside the main() method and use the console for input and output. This class will be executed when we assess the functionality of your work.

Your testing must include all of the following tests, one after the other.

1. Light mode changes test .
   1. Create SmartHome object and fully populate it with a mixture of SmartDevices, a SmartFridge and two Smart Lamps in different rooms.
   2. Using console input to obtain a valid index for the SmartHome object from the user, use getDevice() to select a smart lamp and with the returned object change the lighting mode.
   3. Verify these changes using the SmartHome object toString() method.
2. Light mode changes test 2.
   1. Using console input to obtain a valid location for the SmartHome object from the user, use getDevice() to select a smart lamp and with the returned object change the lighting mode.
   2. Verify these changes using the SmartHome object toString() method.
3. Light mode changes test 3.
   1. Using console input, obtain a valid location in the SmartHome object from the user. Use getDevice() to select a smart lamp and with the returned object change the lighting mode to a different lighting mode: BRIGHT\_MODE.
   2. Verify these changes to the lighting mode **cannot** be done using the SmartHome object toString() method.

**Java Libraries Permitted**

This coursework should only use:

* java.util.Scanner;
* an agreed library. Contact [nadim.bakhshov@port.ac.uk](mailto:nadim.bakhshov@port.ac.uk) for clarification

You should not use:

* java.util.Arrays - especially for toString()
* arraylists - as an alternative to arrays

**Moodle Submission**

Project name: **JavaCourseworkUPXXXXXX**

You must upload your Java Project as a NetBeans 8.1 project in a zipped form to Moodle by **11 pm, Thursday 14th May, 2020**:

The filename you upload should take the form: **JAVA\_UPXXXXXX.zip** where XXXXXX is replaced by your student number.

**Please note:** The functionality of the code will be tested on NetBeans 8.1. If you build your project in any other IDE then you must rebuild it on Netbeans 8.1 so it should execute during assessment. If the project does not run on NetBeans 8.1 it will receive 0 for functionality and will only be judged on code quality.

**Assessment**

This coursework will be assessed in the following way:

1. The coursework will be unzipped and opened in NetBeans 8.1
2. Each **step** test script will be executed, eg. Step1.java, Step2.java, etc**.**
3. **Functionality** will be judged by script execution and input and output at the console.
4. **Code quality** will be judged on the underlying classes.