School of Computing

Module Coordinator Other lecturers	Nadim Bakhshov <nadim.bakhshov@port.ac.uk>
Date Issued	2021-04-13
Code	M30299
Title	SmartHome Dashboard Console

Schedule and Deliverables

Item	Value	Format	Deadline
One ZIP Moodle Upload	The assignment carries 25% of the module marks	ZIP TO INCLUDE: 1. Code/Project. Verify the unzipped project/code does work. 2.Technical Report (PDF) 3. A link to the unlisted YouTube video included in the report document or a separate txt file containing the link	2021-05-13 23:00 BST

Notes and Advice

- The Extenuating Circumstances procedure is there to support you if you have had any circumstances (problems) that have been serious or significant enough to prevent you from attending, completing or submitting an assessment on time.
- ASDAC are available to any students who disclose a disability or require additional support for their academic studies with a good set of resources on the ASDAC moodle site
- The University takes plagiarism seriously. Please ensure you adhere to the plagiarism guidelines.
- Any material included in your coursework should be fully cited and referenced in APA format (sixth edition). Detailed advice on referencing is available from http://referencing.port.ac.uk/
- Any material submitted that does not meet format or submission guidelines, or falls outside of the submission deadline could be subject to a cap on your overall result or disqualification entirely.
- If you need additional assistance, you can ask your personal tutor, learning supportana.baker@port.ac.uk and xia.han@port.ac.uk or your lecturers.

Programming Coursework:
SmartHome Dashboard Console

Note

The coursework directly emerges out of **Week 7 material.** Make sure you have gone through that before starting. Many issues and questions are covered in that material

---

Case Study Scenario

You have recently signed up for small scale freelancing development project work. You have registered your email on the website **newcoders.org** and after a week you receive the following project notification:

"A UK-Finnish company, **Alikoti,** are looking to develop a console prototype for an app they want to put out on the market. Click on the project details below."

You click on the link and this takes you to the Alikoti post on the **newcoders** website. It reads:

"The UK-Finnish company Alikoti have been developing eco-friendly smart home technologies since 1999. Their main product is a fully integrated Smart Home Solution which comes with bespoke smart technologies. These are large scale, expensive and are fully integrated and specifically designed to control the smart home from within the house and through the bespoke handheld devices. In 2019 they decided to broaden their market by selling standalone smart plug devices independently of the integrated solution. These are designed for any household to be used with standard televisions, lights, computers, heaters and phones rechargers and so on. The smart plugs are all of a standard design and deliver standard UK voltage for most standard appliances. The smart plugs are relatively cheap and a homeowner can buy as many as they need to replace their standard plugs. Currently, Alikoti ship out their bespoke touch screen control device with purchases of the plugs. This has become uneconomical as Alikoti has realised that some households are happy to buy just one or two smart plugs. This bespoke control device has slowed down the sales of their smart plugs as it is expensive and only becomes cost effective if a homeowner buys over 30 smart plugs. The business team, after undertaking some market research, have agreed that providing a mobile app to control the smart plugs instead would be a more cost-effective solution. The Alikoti Dev Team has no time to begin looking at this and so have farmed out a series of projects to **newcoders** and other freelancer websites. James Baldwin, the Head of the Alikoti Dev Team , has requested this small scale project to allow his team to judge the **feasibility** of further development.

He and his team have drafted the requirements and expectations for the project. Briefly, they require a Java console dashboard (desktop) application. Once this project is completed and submitted they intend to examine the complexity of the backend requirements and user interface complexity. The following project, with report and demonstration video must be uploaded by the given deadline. There will be further opportunities to produce the follow-on projects if they are impressed by what you build.

Project

The Alikoti Dev Team requirements will be presented and detailed in what follows. Please note they have inserted comments and expectations alongs the way to help you understand the relevant development contexts. The project requires you to design, build and demonstrate a SmartHome Console Dashboard in Java. They have provided a partial mockup of the dashboard console interface to help you visualise the frontend:

```
                          ---------------DASHBOARD--------------
ROOM: 0
SmartPlug |attached to: lamp                |room: study 1|ID: 0|status: off|
ROOM: 1
SmartPlug |attached to: TV                  |room: living room|ID: 1|status: off|
ROOM: 2
SmartPlug |attached to: recharger           |room: kitchen|ID: 2|status: off|
SmartPlug |attached to: heater              |room: kitchen|ID: 3|status: off|
ROOM: 3
SmartPlug |attached to: computer            |room: dining room|ID: 4|status: off|

                          -------------MENU OPTIONS-------------
                          ---------please select option:--------
1 - house level options
2 - room level options
3 - plug level options
4 - system options
```

The above dashboard is expected to be placed within a repeating loop to both (a) ensure all the updates are displayed after each option change and (b) to simulate a graphical user interface. Before the dashboard runs there should be initial configuration steps. The following document details the necessary tasks (step-by-step) to complete the project.

Constraints

There are a number of key constraints. These will be listed below. **These will influence grading decisions.** Any questions come to your practical sessions, also see the FAQ or email nadim.bakhshov@port.ac.uk

Infrastructure	The project must separate the responsibilities of backend objects and frontend objects. See **appendix** for more details.	The backend objects should store and manage the data for smart plugs. The frontend objects should only be used to output to the console and manage input from the console. The frontend can wrap processes, for example having a populate method in a console helper object will tidy up the complexity of the frontend. *The Alikoti Dev Team expects **no** backend object to include any reference to the console.*
Room & Plug Combinations	You can have zero, one or many smart plugs in a single room	You can have zero, one or many smart plugs in any room. Some rooms can be left empty of smart plugs. *You cannot have one smart plug in several rooms.*
User interface	The application interface must be in the form of a Java console	It is encouraged to allow most if not all input to be **integer** based input. The Alikoti Dev Team is not concerned with wasting development over managing string input.
Language	Java	
Exception Handling	Exception handling is not required	The Alikoti Dev Team is not concerned with exception handling through bad input. For this project they are **assuming good input throughout**. They expect exception handling to be used only where absolutely necessary. **See Week 7 material for suggested guidance**. If your code is excessively or arbitrarily wrapped in try-catch blocks this will be seen as poor code quality. If you do decide to use

		try-catch blocks then ensure you have fully justified their use in the technical report. Without such justification this will be seen as irrelevant coding. Alikoti will strip out all user interface exception code when it takes your code and moves it into a graphical user interface.
Libraries	This project must not use Java specific libraries to carry out functionality other than the ones permitted. The software team at Alikoti does not want to be constrained by Java specific libraries.	This project should **only** use: ● java.util.Scanner ● StringBuilder This project should **not** use: ● java.util.Arrays - especially for toString() ● arraylists - as an alternative to arrays **You must build all core processes/mechanisms yourself.**
Complexity	You are also expected to code with clear functional code. No introduction of arbitrary complexity is asked for.	The Alikoti Dev Team does not require your code to be overcomplicated through arbitrary or subjective preferences. This is a commercial project and the code will be used in further projects. Secondly, you must absolutely not add additional functionality to the project or deviate from the requirements given.
Extra Functionality	You should not include any extra functionality not specifically covered in the case study or this document	

Tasks

The assignment is marked out of 100 and will account for **25%** of your overall module mark. The table below shows a more detailed breakdown of marks by task

Task overview

Details of the tasks and mark breakdown will follow.

Tasks		Marks
Initial Development Tasks (10 marks)	The code must comply with the **requirements**. Responsibilities must be correctly distributed amongst the objects. It is your responsibility to ensure this. See Appendix 1 for System components	80
Core Development tasks (70 marks)		
Technical Report Task	The report will be used to mark the code quality. It is your responsibility to present all key coding decisions and justifications.	10
Video Demonstration Task	The video will be used to mark the functionality. It is your responsibility to make a video that demonstrates all the functionality you have completed.	10

For more information on core system components see **appendix**

Initial Development Tasks (10 marks)

Task	Detail	Requirements
1 (5 marks)	Design and build a console helper frontend object	You are allowed to choose an appropriate **name** for your console helper class. This class itself can vary in some details. For example, you can attach the SmartHome to it or use it as a static class for basic level console interaction. The ConsoleHelper *(or equivalent)* class should be responsible for managing console interactions: - outputting to console - receiving input from console It should also be responsible for combining input and output to create **wrapper** methods to simplify frontend coding. It can also support your coding by wrapping processes and mechanisms that are needed in your dashboard. You can pass the SmartHome object to it as a parameter to help simplify your code but if you do so you must use it as an object. I*t is recommended to use this as an object to help usability and managing backend objects. The Alikoti Dev Team will be relying on this to look at the separation of frontend issues and backend issues.*

2
(5 marks)
	Design and build a dashboard class	The application entry point should lie in the Dashboard class. It is here that the SmartHome object is created as is the ConsoleHelper object (if you use it as an object). The Dashboard Class should host and execute the **`public static void main (String[] args)`** method. This method should have the following structure:
		
		`//BUILD SMARTHOME & CONSOLE HELPER OBJECTS`
		`//POPULATE SMARTHOME`
		`while(true){`
		`  //DISPLAY DASHBOARD`
		`  //DISPLAY OPTIONS`
		`  //PROCESS OPTIONS/ACTIONS`
		`}`
		
		It is highly recommended that you extensively manage the frontend console interaction through a console helper object.
		
		The Alikoti Dev Team will treat this as part of your core 'testing' of the backend objects.

3	*Ongoing*	*As you develop your backend objects you will inevitably add more frontend functionality to support input and output processes.*

Core Development Tasks (70 marks)

The Alikoti Dev Team recommends that you build the backend in conjunction with parts of the frontend console dashboard. Each task below will specify the relationship. They would rather receive a well designed and built project rather than a poorly designed one, rushed to complete every requirement. Note: many of the requirements have sample screenshots. These are based on an early interface design sketched out by James Baldwin, the Head of the Alikoti Dev Team. Please do not deviate from them.

See **Appendix 1** for details of System Components.

Part 1 - System Setup (20 marks)

Task	**Backend Detail**	**Requirements**

4
(5 marks)
	For your backend components:
	
	Design and build constructors, member variables
		This system should work by first asking the user through the console to set up (a) the number of rooms which require smart plugs and (b) the number of smart plugs to be distributed across these rooms. For example:
		
		`How many rooms are there in this property?`
		`>> 4`
		`How many plugs do you want to place in this property`
		`>> 5`
		
		This data is used to create the backend objects.
		
		The Alikoti Dev Team has assumed that only one backend object is exposed at the frontend - the SmartPlug objects must be designed to be internal to the SmartHome object.

5
(5 marks)
	For your backend components:
	
	Design and build methods to populate room names and room
		The console should then ask the user for the rooms. These should be stored in the backend objects. For example:
		
		`Please provide a name for your first room`
		`>> study 1`
		`Please provide a name for your second room`
		`>> living room`

	IDs
	
	You must design how your SmartPlug will store Room data. See **System Component Specification** for options.
		`Please provide a name for your third room`
		`>> kitchen`
		`Please provide a name for your fourth room`
		`>> dining room`
		
		The backend objects should attach a numeric ID to each room and store this information. This ID is used throughout the system to allocate rooms to smart plugs. When you arrive at the step (see below) to allocate rooms to smart plugs you should not use the room name but the numeric ID.
		
		The Alikoti Dev Team recognises the backend objects can manage this data in several ways. They will examine your code to see how reusable and expansible your code is. You should consider seriously building Room and AttachedDevice classes and working with these objects in the SmartPlugs. If this is too difficult then ensure you still have a working solution. See Appendix.

6
(10 marks)
	For your backend components:
	
	Design and build methods to create each smart plug, populate it and add it to the SmartHome
	
	You must design how your SmartPlug will store AttachedDevice data. See **System Component Specification** for options.
		After defining the rooms the system should ask the user to configure and add each smart plug. This will be stored inside the backend objects. Firstly, each plug will need to be assigned to a room. The system should list the available rooms and ask the user to select a room for the smart plug. The user should enter the room ID to select a room for a smart plug.
		
		`ENTER PLUG INFORMATION BELOW`
		
		`ROOMS AVAILABLE: 1 - study 1 | 2 - living room | 3 - kitchen | 4 - dining room |`
		
		`Using the above list, please select the room for this plug (integer only)`
		`>> 1`
		
		It should then provide a list of available attached devices for the smart plug. These should be stored in the backend objects. *The Alikoti Dev Team have added this: Initially you*

		should think of them as predefined. The user should not need to set up this initial list through the console.
		
		`AVAILABLE DEVICE LIST OPTIONS`
		`These are standard devices that can be attached to the smart plug:`
		`1-Lamp`
		`2-TV`
		`3-Computer`
		`4-Phone Recharger`
		`5-Heater`
		`Using the above list, please select the device to attach to the smart plug (integer only)`
		`>> 2`
		
		The backend objects should build the SmartPlug and add it to the array in the SmartHome. This is then repeated for each plug. All plugs should be set up as **switched off** in the setup process.
		
		The Alikoti Dev Team does not want the SmartPlug to be created in the client and passed to the SmartHome. They are interested in how you manage the list of attached devices.

Part 2 - Dashboard Display & Options (50 marks)

The Alikoti Dev Team recommends that, within the given time constraints, you should attempt to build as many options as you can. However, the priority is the House Level Options, followed by The Room Level Options and, if well designed, the Plug Level options. They will be looking at your code to see how well you have designed various options with the use of parameters.

Task	**Backend Detail**	**Requirements**

7
(5 marks)
	For your backend components:
	
	Design and build a method to allow all SmartPLugs to be switched off
	
	Design and build a method to allow all SmartPlugs to be switched on
		House Level Options
		
		For the house level actions there should be two actions. All the plugs can either be switched on or off. Relevant methods in the backend objects must be called. The frontend code selects which method to call. For example:
		
		`HOUSE LEVEL OPTIONS`
		`1 - Switch all plugs off`
		`2 - Switch all plugs on`
		`Select an option`
		`>> 2`
		
		After the specific house level option has been selected the console should return to the main dashboard with updated data from the backend objects displayed.

8
(5 marks)
	For your backend components:
	
	Design and build the combination of methods to select a room and switch off all SmartPlugs in that room
	
	Design and build the combination of methods to select a room and switch on all SmartPlugs in that room
	
	Design and build the combination
		Room Level Options
		
		For the room level options you should first be asked to select a room. With a selected room you are given 3 actions for the plugs in that room. Relevant methods in the backend objects must be called. For example:
		
		`ROOMS AVAILABLE: 0 - study 1 | 1 - living room | 2 - kitchen | 3 - dining room |`
		
		`Please select room (integer only)`
		`>> 2`
		
		`SmartPlug |attached to: recharger |room: kitchen|ID: 2|status: on|`
		`SmartPlug |attached to: heater |room: kitchen|ID: 3|status: on|`
		
		`ROOM LEVEL OPTIONS`
		`1 - Switch all devices off in room`

	of methods to select a room and toggle all SmartPlugs in that room	`2 - Switch all devices on in room`
		`3 - Select a device in the room and toggle its on/off status`
		
		`Select an option`
		`>> 1`
		
		After the room level option has been selected the console should return to the main dashboard with updated data displayed.

9
(5 marks)
	For your backend components:
	
	Design and build the combination of methods to select a smart plug and switch off
	
	Design and build the combination of methods to select a smart plug and switch on
		Smart Plug Level Options - options 1, 2
		
		For the plug level options you should be given a *room independent* listing of the plugs. From this list you should be able to select a smart plug from its ID. For example:
		
		`SmartPlug |attached to: lamp |room: study 1|ID: 0|status: on`
		`SmartPlug |attached to: TV |room: living room|ID: 1|status: on`
		`SmartPlug |attached to: recharger |room: kitchen|ID: 2|status: off`
		`SmartPlug |attached to: heater |room: kitchen|ID: 3|status: off`
		`SmartPlug |attached to: computer|room: dining room|ID: 4|status: on`
		
		`Please select plug (integer only)`
		`>> 1`
		`PLUG LEVEL OPTIONS`
		`1 - Switch plug off`
		`2 - Switch plug on`
		`3 - Change attached device`
		`4 - Move plug to different room`
		
		`Select an option`
		`>> 3`
		
		Actions 3 and 4 require further details

10 (10 marks)	*For your backend components:* Design and build the combination of methods to select a smart plug and change the attached	**Smart Plug Level Options - option 3** **3 - Change attached device option** For the *change attached device action* you should be given a listing of the available devices to allow the user to select a different attached device. For example: `AVAILABLE DEVICE LIST OPTIONS`

	device for that plug	These are standard devices attached to the smart plug, unless otherwise stated 1-Lamp 2-TV 3-Computer 4-Phone Recharger 5-Heater Enter device to attach to smart plug (integer only) >> 1
11 (10 marks)	*For your backend components:* Design and build the combination of methods to select a smart plug and change the attached device for that plug	**Smart Plug Level Options - option 4** (10 Marks) **4 - Move plug to different room action** For the *move plug to different room action* you should be given a listing of available rooms to allow the user to select a different room for the plug. For example: ROOMS AVAILABLE: 0 - study 1 | 1 - living room | 2 - kitchen | 3 - dining room | Please select room for device from list (integer only) >>
12 (15 marks)	*For your backend components:* Design and build the combination of methods to allow more smart plugs to be added and stored Design and build the combination of methods to allow more attached devices to be added and stored Design and build the combination of methods to allow more	**System Level options** (15 marks) The system options allow (a) more smart plugs to be added, (b) more attached devices to be listed and (b) more rooms to be added. *The Alikoti Dev Team has added this requirement to test your thinking capacity. They will be looking carefully at your solution and justification in your technical report.*

	rooms to be added and stored	

Technical Report Task (10 marks)

The technical report will also contribute to the grading of code quality

Maximum Report Length: 1000 Words ± 10%

Task	**Detail**	**Context**
13 (10 marks)	Design Decisions for object design	The technical report should present the reasoning process in the design process for the whole project you have built and designed. Consider where data is stored, how it is accessed and updated and displayed. Consider carefully how you complied to the constraints (given above). **Please note if you only partially complete the project but document this clearly you will still achieve the grades.**

Video Demonstration Task (10 marks)

Maximum Video Length: 2 - 3 minutes

Task	**Detail**	**Context**
14 (10 marks)	Demonstrate the console dashboard	This task requires you to screen record a demonstration and discussion of how your system works. This video will play a crucial role in determining the success of the project as the Alikoti Dev Team does not expect to execute all submitted projects.

For those projects where the video or technical report are absent the relevant marks will be lost and the grading for the whole project will rely on code inspection only.

Appendix: System Component Specification

The Alikoti Dev Team have supplied supplementary detail to support your project.

The SmartHome and SmartPlug will be referred to as **backend components**. The Dashboard and Console Helper will be referred to as **frontend components**. The Dashboard will have a single public static void main(String[] args) entry point for the system. The Console Helper will manage console output and input processes.

Overview

This system should be made up of four core classes: (a) two **backend/backend** classes - a SmartHome class and a SmartPlug class and (optional) Room and (optional) AttachedDevice class. (b) two **frontend/frontend** classes - a Dashboard class which contains the application entry point - *public static void main()* - and a consoleUtilities/Manager/Library type class to wrap and manage functionality for console interaction. The backend classes must not have any internal mechanisms using console based code. The frontend classes should make up the console interface. No data should be stored in the frontend classes. All data should be managed by the SmartHome object public methods.

Frontend Class Dashboard

The application entry point should lie in the Dashboard class. It is here that the SmartHome object is created as is the ConsoleHelper object. The Dashboard Class should execute the **`public static void main (String[] args)`** method. This method should have the following structure:

```
//BUILD SMARTHOME & CONSOLE HELPER OBJECTS
//POPULATE SMARTHOME
while(true){
    //DISPLAY DASHBOARD
    //DISPLAY OPTIONS
    //PROCESS OPTIONS/ACTIONS
}
```

It is highly recommended that you extensively manage the frontend console interaction through a console helper object.

Frontend Class Console Helper

You are allowed to choose an appropriate **name** for your console helper class. This class itself can vary in some details. For example, you can attach the SmartHome to it or use it as a static class for basic level console interaction:

- outputting to console
- receiving input from console

It is also responsible for combining input and output to create **wrapper** methods to simplify frontend coding. It can also support your coding by wrapping processes and mechanisms that are needed in your dashboard. You can pass the SmartHome object to it as a parameter to help simplify your code but if you do so you must use it as an object.

I*t is recommended to use this as an object to help usability and managing backend objects.*

Backend Class SmartHome

The SmartHome class manages a private array of SmartPlug objects. It is the core backend object of the system. When writing all the methods ensure you delegate responsibilities to the SmartPlug class where appropriate. This data should include:

- a private array of smart plug objects

The public methods should include methods to

- **Initialise** and **construct** object, using the size of the array
- **Populate** the SmartPlug array
- **Display** relevant information from the array of SmartPLugs, including data from with the SmartPlug objects
- **Update** - this can include all SmartPlugs, some SmartPlugs or only one. The Update can update the appropriate variable inside the SmartPlug object

It is your responsibility to design these methods to be used in the frontend. you can create private methods to help support your class processes.

Backend Class SmartPlug

The SmartPlug should be responsible for maintaining data for each SmartPlug object. The SmartPlug class contains all the private data for an individual SmartPlug. It is used as an object in the array of MSartPlug objects in the SmartHome. This data should include:

- the attached device* `(String name, int ID)`
- the room* `(String name, int ID)`
- An ID value for the smart plug - this is internally generated by the system when the smartplug is added.
- A boolean status (off/on) for the plug

The public methods should include:

- a constructor - to initialise a SmartPlug object with all appropriate variables. You may wish to build more than one constructor to cover different scenarios.
- get/set methods for each variable
- a display method - to display the data in the variables and return as a String.

*Both the objects are optional (see below) and are made up of two pieces of information. Both objects can be replaced with a simpler pair of variables for attached device and room : `String name, int ID`

Optional Classes

As with many software designs, there are a variety of ways the Alikoti system could be designed. If you choose, you can add the two additional classes to your system. See below for details. The alternative is to have the variables all in the SmartPlug class.

(Optional) backend Class Room

This is the class that organises room data into a single object. It is used as a member variable of the SmartPlug class. The Room class should contain all the private data for an individual Room. This data includes:

- the name of the room
- An ID value for the room

The public methods should include:

- constructor - to initialise a Room object
- get/set methods for each variable
- a display method - to display the data in the variables

(Optional) backend Class AttachedDevice

This is the class that organises attached device data into a single object. It is used as a member variable of the SmartPlug class. The AttachedDevice class should contain all the private data for an individual AttachedDevice. This data includes:

- the name of the attached device
- An ID value for the attached device

The public methods should include:

- constructor - to initialise an attachedDevice object
- get/set methods for each variable
- a display method - to display the data in the variables