[Please read all 6 pages of this document first]

COSC1519 Introduction To Programming

Assignment 2: Object Oriented OffGridManager (20 marks)

[spec version 2020.04.20; ensure you have the latest version]

Due dates and times: See section 3 of this document

Note: You are awarded marks based on your ability to follow and fulfill the given requirements of this assignment. **Every word in this document matters.** If it is not written, it is not a requirement but you must use the necessary concepts shown in class materials as the list of things you must not do is too long to list. If there are changes to any requirements or additional clarifications, announcements will be made via *Canvas→Announcements*.

Plagiarism warning: This is an individual assignment; all submitted work must be your own. Do not ask others to write any part of your code. **Do not even let others see your code.** The university takes plagiarism and intellectual property matters very seriously; plagiarism attracts academic penalties (e.g. zero marks or more severe penalties) and disciplinary warnings that go on your record. See RMIT Academic Integrity information for specific details.

Tip: Books and lecture notes do not contain solutions to programming tasks. Instead, programmers learn concepts and apply them to solve unforeseen programming tasks. In this assignment, you will be demonstrating your understanding of the concepts covered in first few week of Introduction To Programming lectures. There is no need to wait to get started as you can attempt some parts already. Students are expected to start on released assignments and are expected to manage their time and minimise risks that might prevent timely completion of the work. As this is tertiary education, you are also expected to independently investigate additional examples and perform the necessary research to complete this assignment. Hints and tips will be given during classes and the class work will help you complete this assignment. You are expected to do the class work and the assignment, simultaneously.

1. Introduction

In this assignment you will create an application that allows an off-the-grid user to manage their electrical devices (similar to HomeAutoBot) and manage power production/consumption need.

Stop! It will be easier to make sense of the rest of the specs if you watch the live demonstration given during the 20/April/2020 lecture between approximately 03:30-16:00 segment in the recording:

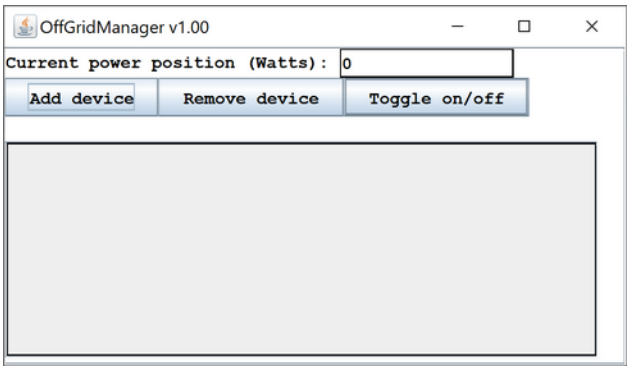
https://au-lti.bbcollab.com/recording/50c464796d38495782c00ee9eeb7fb91

2. Requirements

2.1 Functional Requirements

The marks for functional requirements are approximately equally divided between each of the following functional criteria (given in bold).

The main window of the application: Must have the following elements (you are free to re-order them as you wish)



'Current power position (Watts)' field: This is not a value that should be entered directly. It is calculated and updated based on what power generating devices and what power consuming devices are currently turned on/off. When a power generating device is turned on/off, this value must increase/decrease by the rated power of that device. When a power consuming device is turned on/off, this value must decrease/increase by the rated power of that device. The program must not allow the user to turn on power consuming devices or turn off power consuming devices that would result in the power position becoming negative. (The power position must be zero when all devices are turned off.)

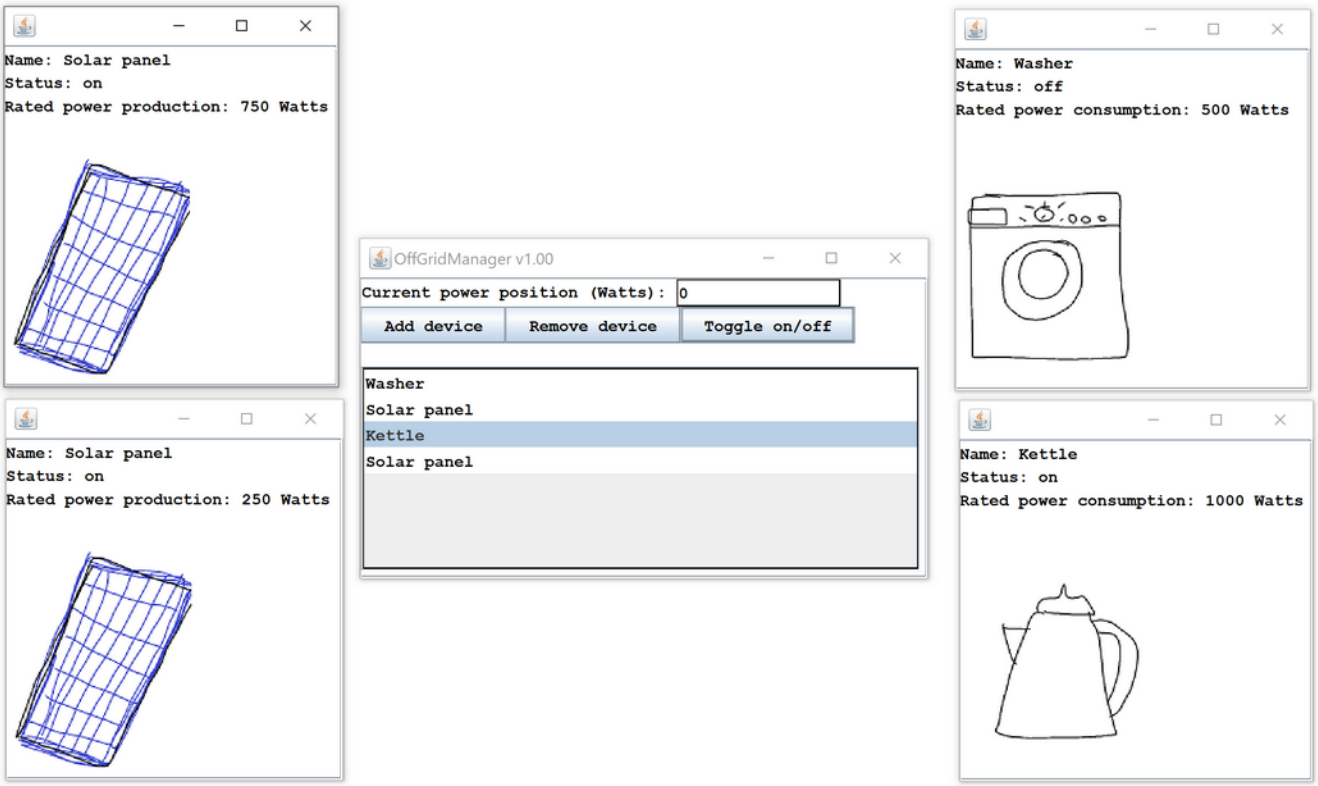
'Add device' button: Allows the user to add an electrical device that either uses energy or produces energy. First, the user selects an image of the device and the name of the device is extracted from the filename. Then the user must enter its power rating in Watts (a negative input here means it produces electricity). It is then added to the list (the grey area in the illustration above). At first, the device is turned off and therefore it does not affect the power position. Finally, a separate GTerm window must be created for the device, showing its name, on/off status, rated power production/consumption figure and image.

'Remove device' button: Allows the user to remove the device that is currently selected in the list. Before removal, the device must be turned off and if the removal of a currently turned on power generating device would result in a negative power position, the user must be told that they cannot remove the device. Additionally, the program must gracefully handle situations where the user selects this button without selecting an item from the list.

'Toggle on/off' button: Allows the user to turn on the selected device if it is currently turned off or turn off a device that is currently turned on. The on/off status of that device must be shown in that device's window. Your code must ensure that the *power position* is correctly maintained (refer to description for 'current power position' field).

Device list: This list shows only the names of the devices that are currently in the system. These devices may be turned on or off. Selecting devices on this list is necessary before the user is allowed to toggle on/off or remove a device.

Example:



The example above shows how two solar panels with a total power production of 1000 Watts is used to power the kettle which consumes 1000 Watts (power position is 0). Note the washer cannot be turned on without turning the kettle off in the above scenario or without adding more power generating devices. The solar panels cannot also be removed while the kettle is ‘on’ as doing so would lead to a negtive power position.

2.2 Code Justification Requirements

The mark for 'code justification' are approximately equally divided for the justification types shown in column 3. Deductions proportional deductions will be made if criteria in column 2 are not met when meeting functional requirements (section 2.1). The code alone will not receive marks for this section.

Concept	**Code must implement...**	**At each implemented location, the code comments must explain and justify why it is more suitable to...**
Classes	Multiple classes. Each class must be in its own file. Must follow naming conventions shown in lectures. All code must be formatted using Eclipse→Source menu→Format. Comments must start on a new line, before the code being commented (do not include in-line comments). There must be no unreachable code. If a class has a red dot, it will result in 0 marks for the entire submission.	… use the name given for your class over other names that may be somewhat acceptable. … keep the composition of your class that way instead of splitting it further (to create more classes) or merging it with another (and reducing the overall number of classes).
Member variables (and arrays)	Each class must only have private object member variables (and/or arrays) that are not static. Only declarations should be at class level (There should be no = signs near declaration). Names of these must be descriptive of their contents and must follow naming conventions shown in lectures.	… name this member variable that way instead of using other potentially suitable names. … use this data type chosen instead of other compatible data types. … declare this variable as a member of the whole object as opposed to having it declared inside a method or several methods.
Constructors	Each class must have one constructor and it must be public. Constructor must have corresponding parameters for the most important object member variables declared in that class and assign them. The constructor must ensure that it initialises all member variables before doing anything else.	… use your choice of constructor parameters (which maybe none) over other potentially suitable choices. … perform only what you have performed in the rest of the constructor (excluding parameter declarations and initialiations).
Methods (Other than constructors)	Methods must be explicitly public and not static. Method names must describe actions and must follow method naming conventions shown in lectures. Methods may take parameters and/or return values. A method can only have one 'return' statement at most and this must be the very last statement in the method (no	… use the name given for your method over other names that may be somewhat acceptable. … use your choice of parameters (which maybe none) over other potentially suitable choices. … use your choice of return type (or void) over other potentially suitable choices. … keep the composition of your method that way instead of splitting it further (to create

	spaghetti code). All references to member variables and/or arrays from methods must begin with "this.". Only one class must have the 'public static void main...' method and it must only create an object of the class in which it is contained (this would start the application).	more methods) or merging it with another (and reducing the number of methods).
Arrays	The program must use standard Java array concepts to maintain added devices and the data type of the array must be of a class type that you have created (you need the array in addition to the any GTerm list that might hold the same data). The array may have a maximum capacity but the program must work any number of devices added. Must only use concepts covered in lectures (e.g. do not use Array, Arrays, ArrayList, etc. classes).	… name the array that way instead of using other potentially suitable names. … use the data type chosen instead of other compatible data types. … declare the array as an object member/inside a method as opposed to declaring it inside a method/as an object member.
Loops	The code must exclusively use while-loops when repetition is necessary. The while loop condition must fail eventually and 'break', 'continue', 'return' or similar statements must not be used to terminate loops.	… use the condition you have used as opposed to formulating it in a different way that would also work. … include only what's included in the loop's body as opposed to moving any of those statements outside of the loop.
if-statements	Must exclusively use if-statements for non-repeating conditional execution (e.g. no *switch*). If-statement conditions must not be redundant.	… use the condition you have used as opposed to formulating it in a different way that would also work. … include only what's included in the if-statement's body as opposed to moving any of those statements outside of the if-statement.
Declarations	Data type choices in the entire program must demonstrate ability to use primitive types over class types when possible. Identifier names must be descriptive and must follow conventions shown in lectures.	… name this identifier/variable that way instead of using other potentially suitable names. … use this data type chosen instead of other compatible data types.

3. Delivery/Submission Requirements

You to perform two deliveries of your work. There are **three important dates/actions** to remember.

- *Delivery 1*: Demonstrate tentative justification comments – need not be final version. You will need to create potential classes and relevant code blocks to place comments. **Submit your tentative .java file(s) with comments** via Canvas→Assignments→Assignment 2 **by end of week 9** and demo it during your allocated practical **in week 10**. Completion of *Delivery 1* attracts **7.5 marks**. Your demo instructor will ask you questions related to your work and you need to be able to answer these. No submission means no demo. No demo means no mark.
- *Delivery 2*: Demo the final functionality of the application. **Submit your final .java file(s), device images and a screenshot** via Canvas→Assignments→Assignment 2 **by end of week 11** and demo it during your allocated practical **in week 12**. Completion of *Delivery 2* attracts **12.5 marks** (5 for submission + 7.5 for demo). Your demo instructor will ask you questions related to your work and you need to be able to answer these. No submission means no demo.

Important note: If there is no submission before the start of your class, you will not get a turn to demo. If there is no demo, there will be no mark for that entire delivery. Code with any invalid Java (e.g. red dots in Eclipse) will receive 0. Code that crashes at run-time will receive a 50% penalty of the eligible mark of that delivery.

Late submissions: Late submissions incur a standard 10% penalty for each working day late for that delivery; submissions will not be accepted after 5 working days from deadline. If you wish to organise special consideration, please contact RMIT special consideration.

End of Document