Practical 5

COS214

Department of Computer Science

Deadline: 07/10/2024 at 23:59

Marks: 120

1 General Instructions

- This assignment should be completed in pairs.
- Be ready to upload your practical well before the deadline, as no extensions will be granted.
- You can use any version of C++.
- You can import any library but it can't do the pattern for you:
- You will upload your code with your main to FitchFork as proof that you have a working system.
- **If you meet the minimum FitchFork requirements (working code with at least 60% testing coverage), you will be marked by tutors during your practical session. Please book in advance**
- **You will not be allowed to demo if you do not meet the minimum FitchFork requirements.**

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with permission) and copying material from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm. **If you have any questions regarding this, please ask one of the lecturers to avoid misunderstanding.**

3 Mark Distribution

Activity	**Mark**
Task 1: UML Diagrams	30
Task 2: Implementation of Patterns	40
Task 3: FitchFork Testing Coverage	10
Task 4: Unit Testing and Memory Management	10
Task 5: Github and Doxygen	10
Task 6: Demo Preparation	20
Total	**120**

Table 1: Mark Distribution

Smart Home Automation System Practical Assignment

You are tasked with designing and implementing a **smart home automation system** that integrates multiple devices, such as lights, thermostats, door locks, and security alarms. This system must allow seamless communication between devices and adapt older devices (e.g., legacy thermostats) into the modern system.

Scenario Overview

Your smart home automation system must:

- Track and control the states of smart devices like lights, thermostats, and door locks.
- Implement commands for common tasks like locking all doors, turning off all lights, etc.
- Allow sensors to observe and notify other devices of environmental changes (e.g., turning on lights when motion is detected).
- Adapt legacy devices that do not have smart functionality, allowing them to be controlled by the central smart home system.
- Implement traversal strategies for iterating through collections of devices to manage the home efficiently.

System Components and Tasks

Each section describes the minimum requirements to complete that task. You will often need to implement additional functions to meet the requirements.

Component 1: Smart Devices and State Management

The smart home is composed of various devices. Your task is to:

1. Create a structure representing smart devices like `Light`, `Thermostat`, and `DoorLock`. They can be grouped by rooms, and rooms may be grouped into sections of the house.
2. Each device should:
   - Respond to commands (e.g., `ToggleOn`, `ToggleOff`, `Toggle`, etc.).
   - Maintain its state (on/off for lights, locked/unlocked for door locks, etc.).
3. Implement the following methods:
   - `getStatus()`: Returns the current status of the device (e.g., "on", "off", "locked", "unlocked").
   - `performAction()`: Execute actions like turning on lights or locking doors.
   - `getDeviceType()`: Returns the type of device (e.g., "Light", "Thermostat", etc.).

Component 2: Legacy Device Integration

Not all devices are "smart." Your task is to:

1. Use the appropriate pattern to integrate legacy devices (like an old thermostat) into the smart home system.
2. Implement a `SmartThermostatIntegrator` that maps the old thermostat's interface to the new one.
3. Implement the following methods:
   - `setTemperature()`: Set the temperature on the adapted legacy thermostat.
   - `getTemperature()`: Get the current temperature reading from the legacy thermostat.
4. You may add other legacy devices as well.

Component 3: Automation of Smart Device Control

Users want to automate common tasks. Your task is to:

1. Implement the appropriate to create automatic procedures such as `TurnOffAllLights` and `LockAllDoors` (these are just examples - you can have any controls).
2. Implement a **MacroRoutine** that executes multiple steps in a sequence (e.g., "Goodnight" routine turns off all lights and locks all doors).
3. We suggest that you implement the following methods:
   - `execute()`: Execute the specific command (e.g., turn off lights, lock doors).
   - `addProcedure()`: Add a command to a `MacroRoutine` object.
   - `removeProcedure()`: Remove a command from a `MacroRoutine` object.

Component 4: Sensor Observations and Notifications

Smart homes use sensors to detect changes in the environment. Your task is to:

1. Implement the appropriate pattern to notify devices when certain environmental conditions are met.
2. For example, when a motion sensor detects movement, it should notify connected devices (e.g., lights or alarms) to take action.
3. Implement the following methods:
   - `addDevice()`: Add an device to the sensor.
   - `removeDevice()`: Remove a device from the sensor.
   - `notifyDevices()`: Notify all registered devices when a condition is triggered (e.g., movement detected).

4 Task 1: UML Diagrams (30 marks)

In this task, you need to construct UML Class, Activity and Sequence Diagram for the given scenario.

4.1 Class Diagram (10 marks)

- Create a single comprehensive UML class diagram that shows how the classes in the scenario interact and participate in various patterns.
- Ensure it includes all relevant classes, their attributes, and methods.
- Identify and indicate the design patterns each class is involved in, as well as their roles within these patterns. (Remember a class can participate in more than one pattern)
- Display all relationships between classes.
- Add any additional classes that are necessary for a complete representation.

4.2 Activity Diagram (10 marks)

- Draw a UML activity diagram modelling the overall running of the syste,.

4.3 Sequence Diagram (10 marks)

- Draw a UML sequence diagram showing how the smart devices are updated when sensors pick up changes.

5 Task 2: Implementation (40 marks)

In this task, you need to implement the system components as described in the scenario.

- Use your UML class diagram as a guide to integrate the design patterns. Note that a single class can be part of multiple patterns.
- The provided scenario outlines the minimum requirements for this practical. **You may need to add extra classes/functions to complete the pattern.** Feel free to add any additional classes or functionalities to better demonstrate your understanding of the patterns and tasks.
- Thoroughly test your code (more details will be provided in the next task). Tutors will only mark code that runs.
- You may use arrays, vectors, or other relevant containers. Avoid using any libraries not mentioned in the general instructions, as your code must be compatible with FitchFork. If you really need a specific library please email u21689432@tuks.co.za at least a week before the due date.
- Tutors may require you to explain specific functions to ensure you understand the pattern being implemented.

6 Task 3: FitchFork Testing Coverage (10 marks)

In this task, you are required to upload your completed practical to FitchFork.

- Ensure that your code has at least **60% coverage** (6/10) in your testing main. This is necessary for demonstrating your work to the tutors.
- You will need to show your FitchFork submission with sufficient coverage to the tutor before being allowed to demo.
- You will be required to download your code from FitchFork for demonstration purposes.
- It might be useful to have two main files: a testing main for FitchFork and a demo main with a more user-friendly interface (e.g., a terminal menu) for demoing. This is just a suggestion, not a requirement. **You are required to have at least a testing main.** If you choose to have a demo main, upload it to FitchFork as well, but ensure it does not compile and run with `make run`.
- Name your testing main file `TestingMain.cpp`.
- If you create a demo main, name it `DemoMain.cpp` so it can be excluded from the coverage percentage. Note that you do not have to test your demo main.
- Additionally, upload a makefile that will compile and run your code with the command `make run`.

7 Task 4: Unit Testing and Memory Management (10 marks)

You are required to write unit tests using a testing framework of your choice (not your own).

You will also be assessed on memory management using valgrind (or leaks). Ensure this command is incorporated into your makefile.

8 Task 5: GitHub and Doxygen (10 marks)

Utilize both Github and Doxygen comprehensively to be awarded the full 10 bonus marks.

At least 10 meaningful commits per person and a good branching strategy will be required to get full credit.

9 Task 6: Demo Preparation (20 marks)

You will have 7-10 minutes to demo your system and answer any questions from the tutors. Proper preparation is crucial.

- Ensure you have all relevant files open, such as UML diagrams and source code.
- Set up your environment so that you can easily run the code during the demo after downloading it.
- Be ready to demonstrate specific function implementations upon request.
- Practice your demo to make sure it fits within the allotted time and leaves time for questions.
- Prepare a brief overview (30 seconds) of your system to quickly explain its functionality and design.
- Make sure your testing and demo mains (if applicable) are ready to show their respective functionalities.
- Have a clear understanding of the design patterns used and be prepared to discuss how they are implemented in your code. Also, think about other potential use cases for the patterns.
- Be prepared to answer questions. If you do not know the answer, inform the marker and offer to return to the question later to avoid running out of time.

10 Submission Instructions

You will submit on both ClickUp and FitchFork.

- FitchFork submission:
  - Zip all files.
  - Ensure you are zipping the files and not the folder containing the files.

  - Upload to the appropriate slot on FitchFork well before the deadline, as no extensions will be granted.
  - **Ensure that you have at least 60% coverage.**

- ClickUp submission. You should submit the following in an archived folder:
  - Your UML diagrams (both as images and Visual Paradigm projects).
  - Your source code.