Sample Exam

Software Engineering (IT)

Q1:*Design patterns* and *testing*

Scenario: You are developing software for a pollution monitoring device. The device measures the pollution level once daily, stores this value, and broadcasts the result to all of the pollution services that have subscribed to updates from the device.

a) What design pattern should be applied to represent the relationship between the ***pollution monitoring device*** and the ***pollution services***. Please provide the name of the pattern and a description of the problem this pattern solves.

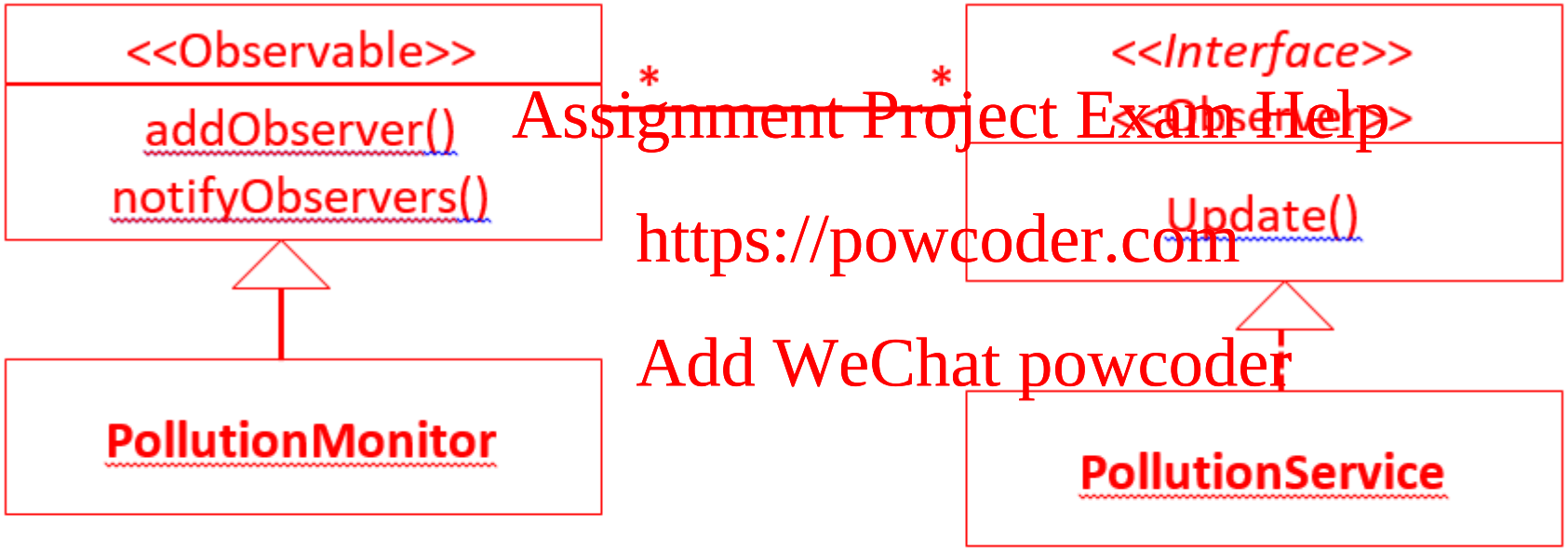
b) Give a **UML class diagram for the proposed system**. Ensure you include all the required methods and multiplicities.

c) The system needs to be tested to ensure that pollution services can subscribe and unsubscribe to updates from the pollution device. **Write a JUnit test case that tests how pollution services can subscribe to updates.**

a)

- Relationship should be implemented with the Observer pattern [1 mark]
- Solve the problem of how to create relationships between classes in different modules/subsystems [2 marks]
- So that objects can communicate with [2 marks]
- Without needing to know the class of each module in advance [1 mark].

b)



Marks awarded for each correct box [4 marks], for correct multiplicity [1 mark], and for correct arrows [2 marks]

c)

```
@Test
public void testAddObserver() {
        assertEquals("no elements", 0 , pollutionMonitor.getServiceListLength());
        pollutionMonitor.addObserver(new PollutionService());
        assertEquals("one element", 1 , pollutionMonitor.getServiceListLength());
}
```


1 mark test case declaration
1 mark for checking null case
1 mark for adding observer
1 mark for checking it was added.

Q2:*Software architectures*

a) Model View Controller is a software architecture pattern. **Give a graphical representation and a description of the role/responsibilities of the three components of this pattern**.

b) Describe how Model View Controller supports software engineering design principles with three examples. For each example, provide a distinct design principle and give a brief description of how Model View Controller adheres to that design principle.

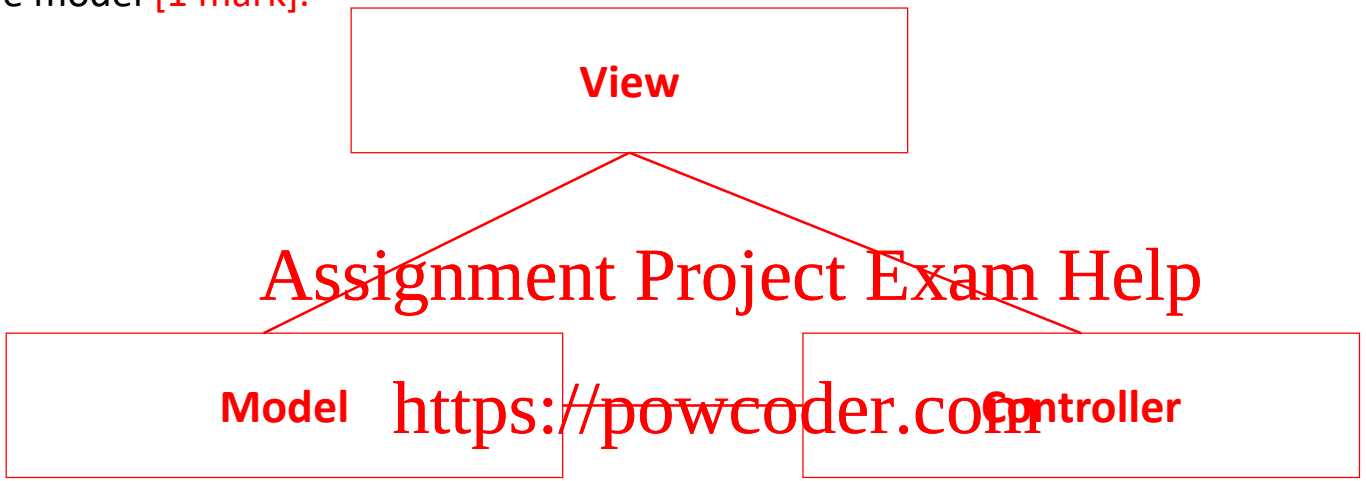
c) The Multilayer software architecture pattern is related to Model View Controller. **Describe a key difference between these two patterns**

d) Procedural and Sequential actions are a kind of cohesion. **Describe how each type of action increases cohesion and identify the key difference between these two types of cohesion.**

a) The Model is responsible for handling all the underlying data of the system [1 mark].

The View is responsible for provide the graphical user interface of the system [1 mark].

The Controller is responsible for handling the majority of the functionality of the system and/or interfacing between the view and the model [1 mark].



b) Any one of [2 marks for each correct response]:

Divide and Conquer: Three components can be developed independently

Increase Cohesion: Components have stronger layer cohesion when UI and control are separated.

Reduce Coupling: Communication channel between components are minimised.

Increase Reuse: View and controller make extensive use of reusable components

Design for Flexibility: UI can be easily changed out for other Views

Design for Testability: You can test the application separately from the UI

c)

The multilayer architecture does not allow for bi-directional communication between layers, so can be thought of as a specialised or more “strict” version of MVC [2 marks].

d)

Procedural cohesion keeps together functions which are called one after another, but do not depend on each other [1 mark].

Sequential cohesion keeps together methods which are called on after another, where the input from one action feeds directly into the next [1 mark].

The key difference is that sequential cohesion depends on the input from methods in a sequence [2 marks].

Q3: *Coupling* and *refactoring*

```
1   public class Student {
2         public int enrolledCredits;
3         public string name;
4         public string email;
5         public boolean fullTime;
6         public string address;
7         public string advisor;
8         public string degreeProgramme;
9
10        //  Constructors and other methods…
11  }
12
13  public class GlobalVs {
14        public MAX_CREDITS = 10;
15        public MAX_YEARS = 5;
16  }
17
18  public class StudentController {
19
20        public void enrollStudent(Student student) {
21
22              if (student.fullTime && students.credits < GlobalVs.MAX_CREDITS ) {
23
24                    credits += 1
25
26                    //  Notify Student of successful enrolment
27                    String header = generateEmail();
28                    String emailLog = prepareEmailLog("Successful Enrolment");
29                    String  signature = addEmailSignature();
30                    String email = header . emailLog . signature;
31
32                    emailStudent(Student student, String email);
33
34              } else {
35
36                    String header = generateEmail();
37                    String emailLog = prepareEmailLog("Unsuccessful Enrolment");
38                    String  signature = addEmailSignature();
39                    String email = header . emailLog . signature;
40
41                    emailStudent(Student student, String email);
42
43              }
44        }
45  }
```


a) Software designers should aim to reduce coupling where possible. **What are two challenges of working with software that is highly coupled?**

b) Review the partial code sample above. Identify four examples of coupling by providing the **type of coupling** present, a **description** of what the coupling is, and the **line numbers** where it is present in the code.

c) Select **two of your examples from part *b*** of this question and describe how you would reduce this kind of coupling.

a)

Changes in code with a high level of coupling can produce unexpected changes elsewhere [2 marks]

Code with a high level of coupling can be difficult to interpret and understand [2 marks]

```
1  public class Student {
2        public int enrolledCredits;
3        public string name;
4        public string email;
5        public boolean fullTime;
6        public string address;
7        public string advisor;
8        public string degreeProgramme;
9
10       //  Constructors and other methods…
11 }
12
13 public class GlobalVs {
14       public MAX_CREDITS = 10;
15       public MAX_YEARS = 5;
16 }
17
18 public class StudentController {
19
20       public void enrollStudent(Student student) {
21
22             if (student.fullTime && students.credits < GlobalVs.MAX_CREDITS ) {
23
24                   credits += 1
25
26                   //  Notify Student of successful enrolment
27                   String header = generateEmail();
28                   String emailLog = prepareEmailLog("Successful Enrolment");
29                   String  signature = addEmailSignature();
30                   String email = header . emailLog . signature;
31
32                   emailStudent(Student student, String email);
33
34             } else {
35
36                   String header = generateEmail();
37                   String emailLog = prepareEmailLog("Unsuccessful Enrolment");
38                   String  signature = addEmailSignature();
39                   String email = header . emailLog . signature;
40
41                   emailStudent(Student student, String email);
42
43             }
44       }
45 }
```


b)

- Common Coupling [1 mark] using global variables [2 marks] – Lines 13-15 [1 mark]
- Stamp coupling [1 mark] when a class is passed as an argument [2 marks] – Line 20, 32 and 41 [1 mark]
- Control coupling [1 mark] when another method is controlled by flags [2 marks] – Lines 22 and 34 [1 mark]
- Routine Coupling [1 mark] when a series of methods need to be called together repeatedly [2 marks] – Lines 27-30 and Line 36-39 [1 mark]
- Content coupling [1 mark] when internal component of one class is surreptitiously modified by another [2 marks] – Lines 2-8 and line 24 [1 mark]

C)

- Common Coupling: Encapsulate any global values in classes that restrict access and editing of values [2 marks].
- Stamp Coupling: Reduce coupling by only passing data that is needed as opposed to whole classes [2 marks].
- Control Coupling: Use polymorphism to control behaviour, and remove Boolean flags [2 marks].
- Routine Coupling: Encapsulate repeated method calls within a single method [2 marks].
- Content Coupling: Set instance variables to private and use getters/setters [2 marks].