Question 1. [25 MARKS]

Software Design.

The EduLand college system has traditionally maintained a `Student` class as follows:

```
public class Student {
    protected String name;
    protected int gpa;
    public Student(String name) {
        this.name = name;
    }
    public void computeGPA() {
       this.gpa = 370; // hardcoded for simplicity
    }
}
```

Later EduLand Government introduced *diploma* and *degree* streams. A student is admitted in either *stream*, but not both. Meanwhile, the Government is pondering to introduce a *certificate* stream as well. Different streams use different criteria to compute the students GPA. A quick fix to the `Student` class is shown below.

Study carefully the following code and answer questions stated in the next page.

```
public class Student {
    protected enum Stream {DIPLOMA, DEGREE};
    protected String name, stream;
    protected int gpa;
    public Student(String name, Stream stream) {
        this.name = name;
        this.stream = stream;
    }
    public void computeGPA() {
        switch(this.stream) {
        case DEGREE:
            this.gpa = 370; // hardcoded for convenience
        case DIPLOMA:
            this.gpa = 360; // same as the above
        default:
        throw new IllegalArgumentException();
        }
    } // equals and hashcode not shown here
}
```

Part (a) [5 MARKS]

Recall the concept of *cohesion* from the lectures. Please answer the following questions:

Solution (bolded items are the correct answer):

(i) What is the level of cohesion between the members of the new `Student class`? Tick one of the boxes below:

☐ Low

☐ **High**

(ii) What is (in general) the desired level of cohesion in a class? Tick one of the boxes below:

☐ Low

☐ **High**

Part (b) [5 MARKS]

Recall the concept of *coupling* from the lectures. Please answer the following questions:

Solution:

(i) What is the level of coupling between the members of the new `Student class`? Tick one of the boxes below:

☐ Loose

☐ **Tight**

(ii) The enumeration type `Stream` is tightly coupled with the `Student` class. Additionally, the computation of gpa would need to change if a new program is introduced.

(iii) What is (in general) the desired level of coupling in a class? Tick one of the boxes below:

☐ **Loose**

☐ Tight

The software team hired for refactoring the code, in an effort to fix the design problems, identifies the difference in the GPA computing scheme needs be addressed by abstracting the `computeGPA` behaviour of `Student class`, writing a `GPA` class that contains various static methods to perform necessary GPA computation and calling them as needed in various extensions of `Student` class. (A particular extension, `DegreeStudent`, is shown below).

```
public abstract class Student {
    protected enum Stream {DIPLOMA, DEGREE};
    protected String name, stream;
    protected int gpa;
    public Student(String name, Stream stream) {
        this.name = name;
        this.stream = stream;
    }
    public abstract void computeGPA();
    @Override
    public boolean equals(Object other) {
        if (other == null || other.getClass() != this.getClass())
            return false;
        Student student = (Student)other;
        return student.name.equals(this.name) &&
                student.stream.equals(this.stream) && student.gpa == this.gpa;
    }
    @Override
    public int hashCode() {
        return 3*this.stream.hashCode() + 11 * this.name.hashCode() + 19 * this.gpa;
    }
}
public class DegreeStudent extends Student {
    public DegreeStudent(String name) {
        super(name, Stream.DEGREE);
    }
    @Override
    public void computeGPA() {
        GPA.gpaDegreeStudent(this);
    }
}
public class GPA {
    public static void gpaDegreeStudent(Student student) {
        if (student.stream != Stream.DEGREE)
            throw new IllegalArgumentException();
        student.gpa = 370; // hardcoded for convenience
    }
// Additional methods here to handle other streams GPA computation
}
```

Part (c) [5 MARKS]

Please answer the following questions:

Solution (bolded items are the correct answer):

(i) Has the level of coupling of changed in the latest implementation of `Student` class? Tick one of the boxes below:

☐ **Loosened**

☐ Tightend

☐ No change

(ii) Open-Close.

(iii) Specifically, if we want to extend `Student` class to either `DiplomaStudent` or `CertificateStudent` we need to add new methods to `Mark` class.

Now it is time to look at the additional classes employed by the college software system:

```
public class Course {

    protected String cID;
    protected int enrollmentCap;

    public Course(String cID, int eCap) {
        this.cID = cID;
        this.enrollmentCap = eCap;
    }

    @Override
    public boolean equals(Object object) {
        if (object == null || object.getClass() != this.getClass())
            return false;
        Course course = (Course) object;
        return this.cID.equals(course.cID) &&
                this.enrollmentCap == course.enrollmentCap;
    }

    @Override
    public int hashCode() {
        return 17*this.cID.hashCode() +
            19*this.enrollmentCap;
    }
}
```

Part (d) [5 marks]

Consider two `Course` objects, (`c1` and `c2`), and the following statement:

```
c1.equals(c2) implies c1.hashCode() == c2.hashcode()   (*)
```

Please answer the following questions:

Solution:

(i) True, because if two Course objects are equal, they have necessarily same hashcode computed by the given formula (which is well defined).

(ii) False. A counterexample can be give by two Courses that have totally different cID values, but with the resulting hashcodes of the cID values falling in the same bucket.

Example: "Aa" and "BB" have the same hash code value of 2112.

Part (e) [5 MARKS]

Now consider the `College` class.

```
public class College {

    protected String cName;
    protected ArrayList<Student> students;
    protected HashSet<Course> courses;
    protected Map<Student, Course> courseEnrollment;

    public College(String cName) {
        this.cName = cName;
        this.students = new ArrayList<Student>();
        this.courses = new HashSet<Course>();
        this.courseEnrollment = new HashMap<Student, Course>();
    }
    public void admission(Student student) {
        if (!this.students.contains(student))
            this.students.add(student);
    }
    public void addCourse(Course course) {
        if (!this.courses.contains(course))
            this.courses.add(course);
    }
    public void enroll(Student student, Course course) {
        // add a student if it is not in the course
        this.courseEnrollment.putIfAbsent(student, course);
    }
}
```

Answer the following questions:

Solution:

(i) **(2.5 marks)** Dependency Inversion Principle is violated by the attribute declarations - we should program towards an interface, not towards an implementation (example: need to use List instead of ArrayList).

(ii) **(2.5 marks)** To stop violating the dependency inversion principle,

replace the declaration of `ArrayList<Student> students` with `List<Student> students`,

and `HashSet<Course> courses` with `Set<Course> courses`.

Of course, for their instantiation you still need to use a concrete class, but for their declarations as members of College, they should be the interface,

just like how `Map<Student, Course> courseEnrollment` is declared.

They may also point out other valid bugs - feel free to award a mark if you feel it has merit. One example is below.

Another thing to note - the enroll method should either:

1. Automatically also attempt to admit the student and add the course provided, as it would not make sense to do an enrollment for a student who is not admitted or a course that is not at the school; or
2. Do not do anything if student is not admitted or course is not in the course set.

Question 2. [15 MARKS]

Software Testing.

Recall from the lecture the black box testing strategy. Suppose we have to test a function that takes as input three numbers (you may assume integers for simplicity), interprets them as length of the sides of a triangle, and outputs the the type of the triangle: scalene/isoceles/equilateral, or outputs an error message if it is not possible to interpret the given numbers as the sides of some triangle (so you can think of the return type of the function as a string).

Part (a) [5 MARKS]

Identify the domain of the values we need to test for.

Solution: The domain is cartesian product of three copies of integers available on the specific language the peogram has been written.

Part (b) [10 MARKS]

Partition the domain in the correct subdomains. For each subdomain, using the provided sample in the table below, list the subdomains, the necessary test cases for each domain, and the expected outcome for each test case. Your mark will be proportional to the number of correct test cases. Frivolous test cases will cause a reduction in your mark earned from correct test cases.

Solution:

Subdomain	Test Case Sample	Test Outcome
Equilateral		
All sides equal	5, 5, 5	Equilateral
Scalene		
Increasing size	3,4,5	Scalene
Decreasing size	5,4,3	Scalene
Largest in the middle	4,5,3	Scalene
Isoceles		
a=b&other side larger	5,5,8	Isoceles
a=c&other side larger	5,8,5	Isoceles
b=c&other side larger	8,5,5	Isoceles
a=b&other side smaller	8,8,5	Isoceles
a=c&other side smaller	8,5,8	Isoceles
b=c&other side smaller	5,8,8	Isoceles
Not a triangle		
Largest first	6,4,2	Not a triangle
Largest second	4,6,2	Not a triangle
Largest third	1,2,3	Not a triangle
Bad inputs		
One bad input	-1,2,4	Bad input
Two bad inputs	3,-2,-5	Bad input
Three bad inputs	0,0,0	Bad input

Note: Equilateral case already provided to students as sample.

Question 3. [10 MARKS]

Use Cases.

Consider an order-processing system for an online order company that is a reseller of products from several suppliers. When customers want to purchase products, they place an order accompanied with payment information. While adding products for an order, customers are able to save it for later use. Orders can be freely cancelled after being placed, but only if not being shipped yet. When an order is placed, the customer's account owing balance is increased with the order amount. When an order is cancelled, the customer's account owing balance is reduced for the full cancelled amount. The customer's account may also be updated by an accounting clerk which processes customer phone calls.

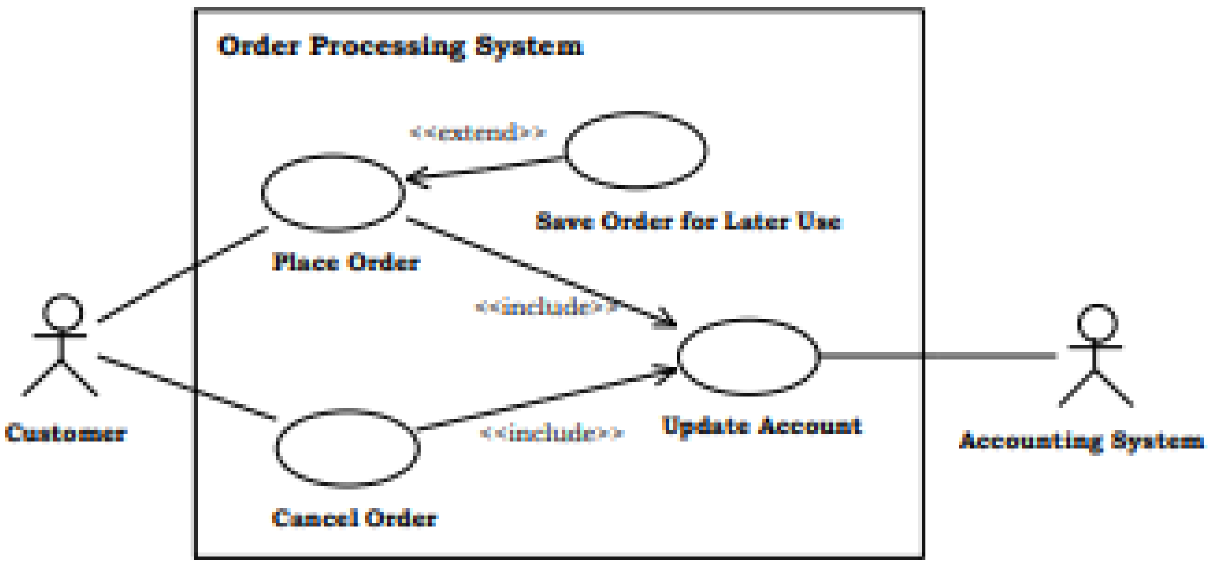
Part (a) [3 MARKS]

From the narrative, fill in the following table:

Goal Use Case	Actor	Include	Extend
Place Order	Customer	Update Account	Save Order for Later Use
Cancel Order	Customer	Update Account	
Update Account	Accounting Clerk		

Part (b) [7 MARKS]

Based on the table, draw the UML Use Case diagram. Make sure to draw the boundary of the system, the name of the system, actors, use cases, and relationships between use cases. There is no need to use different types of arrows - simply write the relationship name (`<<include>>` or `<<extend>>`) on top of the corresponding arrow.



Question 4. [5 MARKS]

Design patterns.

Let's say we have a requirement to sort an array of numbers, so we implement BubbleSort, because it's relatively easy. In a few days, we realize the input data sometimes is much larger than we expected, so we implement QuickSort instead, to speed things up. Now, QuickSort turns out to be great for the large data sets, but for small arrays it implies a lot of overhead. What if we took the best of all approaches and picked the right sorting method at runtime, based on the information we have about the data set?

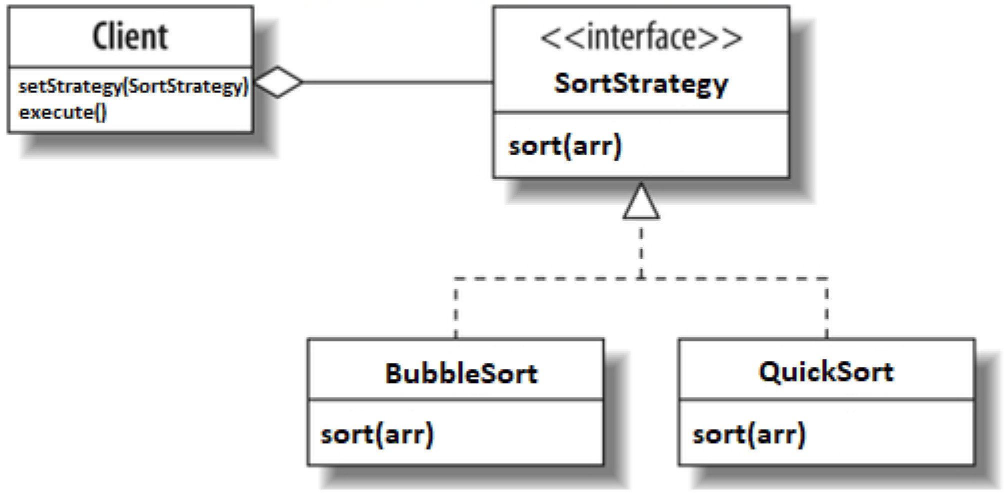
Part (a) [1 MARK]

Identify the correct design pattern that can be used to implement a class that can efficiently sort arrays of numbers.

Solution: Strategy Pattern

Part (b) [4 MARKS]

Student's solution should look similar to this (class and method names may vary, but we should have exactly this many methods per class doing approximately those tasks).



The interface's method and the method named in its implementing classes must be an exact match for the name, otherwise no mark for the interface.

An attribute for the strategy instead of a method to set the strategy can also be accepted (per the slides from Tue Jun 20).
However, there must either be that attribute, or the method to set the Strategy.
If there is only a method that would be for executing the strategy, it can only get full marks if it requires an object of the strategy interface type as an argument, otherwise 0.5 marks awarded only.
I guess what I'm trying to say here is - there has to be some acknowledgement that the strategy could be changed.

If the wrong design pattern is identified, and no methods are identified, then no marks are given.
Else: if implemented correctly then 0.5 marks given, otherwise 0 marks.

Question 5. [15 MARKS]

In each of the scenarios given below, name the design pattern that would be the most appropriate to implement.

(a) If you have many classes that make CRUD operation calls to the database, but you need to be able to swap out between multiple databases, e.g MySQL, PostgreSQL, Oracle, what is the best design pattern to use?

Adapter

(b) Suppose you work on Google Maps and you sync a user's labeled places data every few minutes. If there is new data, you want to update the basemap, user's labeled places list view, and any place details views open. What is the best pattern to use?

Observer

(c) Which is the best pattern to use if you are designing a DirectionsService that serves cached offline results if the user has downloaded this city's map area, else calls the Maps server.

Proxy

(d) Let's consider an Employee object that has a salary property. We'd like to be able to change their salary and keep the payroll system informed about any modifications. What is the best pattern to achieve that?

Observer

(e) Imagine that you are asked to build a simulation of life in a pond that has plenty of ducks. But how would we model our Pond if we wanted to have frogs instead of ducks? You may assume both Duck and Frog classes are subclasses of a WaterAnimal class.

Factory

Question 6. [5 MARKS]

This function implements implication truth table.

3 marks for correct JML code, 2 marks for saying it is implication.

```
//@ ensures \result <==> (p==>q);
```

Question 7. [5 MARKS]

Asnwer and Explanation: First we need to see if we understand what the function is doing. The function runs from 2 to num/2 since we know that num will always be divisible by one. Therefore, we don't want to call a false negative on the number passed in because it is divisible by one. However, note that we only loop up to num/2 because a number is never divisible by more than half of itself. For example, let's say we want to find the factors of 12, we have 1 and 12, 2 and 6, and 3 and 4, notice that the greatest factor other than the number itself is 6, which is half of 12 and no other factor is greater than this factor.

So, we can ensure that the result of the function will be equivalent to the negation of whether a value exists for i that proves num % i == 0. We can write this as seen below:

```
//@ ensures \result <==> !(\exist int i; i >= 2; num % i == 0);
```

3 marks for the JML code

2 marks for saying something along the lines of "checks if a number is prime" (can be as roundabout as the explanation provided above).

Question 8. [5 MARKS]

Consider this code:

```
import java.util.*;
public class ArrayBlaster {

        public static void main(String args[]) {
                ArrayList<Integer> a1 = new ArrayList<Integer>();
                ArrayList<Integer> a2 = new ArrayList<Integer>();

                a1.add(new Integer(10));
                a1.add(new Integer(4));

                a2.add(new Integer(7));
                a2.add(new Integer(9));
                a2.add(new Integer(11));

                blastArray(a1, a2);

                System.out.println("Contents of a1: ");
                for(int i = 0; i < a1.size(); i++)
                        System.out.println(a1.get(i));

                System.out.println("Contents of a2: ");
                for(int i = 0; i < a2.size(); i++)
                        System.out.println(a2.get(i));
        }

        public static void blastArray(ArrayList<Integer> a1, ArrayList<Integer> a2) {
                a1.add(new Integer(13));

                ArrayList<Integer> a3;
                a3 = a1;
                a1 = a2;
                a2 = a3;

                a1.add(new Integer(1));
                a2.add(new Integer(2));
        }
}
```

Note: you cannot switch the array references in main, because of pass-by-value (the value being the memory address of the ArrayList)

The switching of the array references is only applicable to the scope of the method.

All new Integers added to the ArrayLists here are able to persist outside of the scope of the function, because of being referenced to by the ArrayLists which belong to the main function.

Print the output generated by running the above program.

```
Contents of a1:
10
4
13
2
Contents of a2:
7
9
11
1
```

Each array is worth 2.5 marks total, so each element inside is worth 0.625 marks.

If the contents are swapped (i.e. what is supposed to be in a1 is in a2 and vice-versa), halve whatever mark they have.

If a1 is the exact same as a2, then a1 gets no marks.

Question 9. [5 marks]

Which is FALSE about the observer pattern:

A. The observer doesn't need to be any special type of class.

B. There can only be one observer.

C. The subject doesn't need to be any special type of class.

D. A subject can add itself as an observer.

Answer: B

Question 10. [5 marks]

Unit tests should:

A. Test that the UI behaves correctly when a user interacts with it (e.g clicks a button).

B. Test that the output of a method is correct for a given input.

C. Test that an entire module performs the correct behavior for a given input.

D. Test that your feature correctly implements the customer's user stories.

Answer: B

Question 11. [5 marks]

We're creating a new app that allows students to schedule appointments with faculty at specific times and days. Which BEST describes how to model appointments between faculty and students:

A. Faculty has-many appointments; Student has-many appointments

B. Faculty has-many Appointments, through Students

C. Faculty belongs-to Appointment; Student belongs-to Appointment

D. Faculty has-and-belongs-to-many Students; Student has-and-belongs-to-many Faculty

Answer: A