

# BCS1430 Object-Oriented Modelling

**Program:** BSc Computer Science

**Course code:** BCS1430

**Examiners:** dr. Ashish Sai & Spriha Joshi

**Date/time:** Practice Exam

**Format:** Closed book exam

**Allowed aids:** Pens, simple (non-programmable) calculator from the DACS list of allowed calculators

©copyright 2023 - 2024 dr. Ashish Sai & Spriha Joshi - you are not allowed to redistribute this exam, nor any part thereof, without prior written permission of the authors

## Q1 Object Oriented Programming and Software Engineering

You have been tasked with developing a new online e-commerce platform for a large retail company. The system should allow customers to browse and search for products, add items to a virtual shopping cart, proceed through a secure checkout process, and track their order status. It should also enable inventory management, order fulfilment, and reporting capabilities for the company. The platform needs to be accessible across different devices and integrate with various payment gateways and shipping carriers.

4p

**1a** Provide two examples of functional requirements and two examples of non-functional requirements for this e-commerce platform.

3p **1b** Create a class called ExpressShipping that extends the following abstract class:

```
public abstract class ShippingMethod {  
    public abstract double calculateCost(Order  
order);  
}
```

The ExpressShipping class should have a constructor that takes a flat rate (e.g., Euro 9.99) as a parameter. Implement the constructor in the ExpressShipping class.

|  |
|--|
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

Select True or False for the following statements

1.5p **1c** A class in Java can be both abstract and final at the same time.

☐ a True ☐ b False

1.5p **1d** Every object of a subclass is also an instance of its superclass, and thus, it can be assigned to a variable of its superclass type.

☐ a True ☐ b False

**Q2 Object Oriented Design Principles**

You are tasked with creating a university course registration system that facilitates student enrolment, course management, and academic scheduling. The system should enable students to search for available courses, register for classes, and view their schedules. Faculty members should be able to manage course details, enrol students, and access teaching schedules. Administrators should have the ability to oversee the course catalogue, monitor enrolment statistics, and generate comprehensive academic reports.

4p **2a** Consider the following class definition within the university course registration system.

```
public class CourseManagement {  
    public void addCourse(String courseType,  
        CourseDetails details) {  
        if ("Online".equals(courseType)) {  
            // Add an online course  
        } else if ("Onsite".equals(courseType)) {  
            // Add an onsite course  
        }  
        // Assume further else-if blocks for other course  
        types  
    }  
}
```

Which design principle does this class potentially violate?

- ☐ a Open/Closed Principle (OCP)
- ☐ b Single Responsibility Principle (SRP)
- ☐ c Dependency Inversion Principle (DIP)
- ☐ d Interface Segregation Principle (ISP)

4p **2b** Consider the following approach to register students:

```
public class RegistrationHandler {  
    public void registerStudentForCourse(Student student, Course course) {  
        // Code to register student  
    }  
  
    public void generateScheduleForStudent(Student student) {  
        // Code to generate student's schedule  
    }  
  
    public void generateEnrollmentReport() {  
        // Code to generate enrollment statistics report  
    }  
}
```

Identify the code smell in this code snippet.

- ☐ a Open/Closed Principle (OCP)
- ☐ b Single Responsibility Principle (SRP)
- ☐ c Dependency Inversion Principle (DIP)
- ☐ d Interface Segregation Principle (ISP)

4p **2c** Analyze the design of the class below within the university course registration system:

```
public class  
    AcademicReportGenerator  
    {  
        private DatabaseLogger  
        logger = new  
        DatabaseLogger();  
  
        public void  
        generateReport() {  
            try {  
                // Code to generate  
                academic report  
            } catch (Exception e) {  
  
                logger.log(e.getMessage());  
                // Directly using a specific  
                logger  
            }  
        }  
    }  
}
```

- ☐ a Liskov Substitution Principle (LSP)
- ☐ b Open/Closed Principle (OCP)
- ☐ c Dependency Inversion Principle (DIP)
- ☐ d Interface Segregation Principle (ISP)

4p **2d** Given the code below identify the clear violation of a design principle:

```
public interface CourseOperations {
    void addCourse(Course course);
    void updateCourseDetails(Course course);
    void enrollStudent(Student student, Course course);
    void unenrollStudent(Student student, Course course);
    void generateCourseReport();
}
```

- ☐ a Interface Segregation Principle (ISP)
- ☐ b Single Responsibility Principle (SRP)
- ☐ c Law of Demeter (LOD)
- ☐ d Dependency Inversion Principle (DIP)

9p **2e** Consider the following code :

```
public class Course {
    public void
    scheduleClassroom(String room)
    {
        System.out.println("Classroom
        scheduled: " + room);
    }
}

public class OnlineCourse extends
    Course {
    @Override
    public void
    scheduleClassroom(String room)
    {
        throw new
        UnsupportedOperationException("
        Online courses do not require a
        classroom.");
    }
}
```

This approach violates one of the SOLID design principles. Refactor the code to adhere to the SOLID design principles. Explain which principle you are applying and provide the refactored code snippet.

[illegible]

### Q3 Code Smells

- 3p **3a** In the OnlineCourse class below, what code smell is indicated by the use of string to represent the course start time?

```
public class OnlineCourse extends Course
{
    private String courseId;
    private String platformLink;
    private String courseStartTime; // Format: "HH:MM"

    public OnlineCourse(String courseId, String platformLink, String courseStartTime) {
        this.courseId = courseId;
        this.platformLink = platformLink;
        this.courseStartTime = courseStartTime;
    }
    // Methods to schedule and conduct the course
}
```

- ☐ a Primitive Obsession
- ☐ b Large Class
- ☐ c Divergent Change
- ☐ d None of the above
- 3p **3b** Examine the EmailSender and SMSSender classes used in the notification system of the university course registration platform. Identify the code smell present in this scenario.

```
public class EmailSender {
    public void sendEmail(String
recipientEmail, String message) {
        // Logic to send email
    }
}

public class SMSSender {
    public void sendSMS(String
recipientPhoneNumber, String message)
{
    // Logic to send SMS
}
}
```



- ☐ a Alternative Classes with Different Interfaces
- ☐ b Duplicated Code
- ☐ c Message Chains
- ☐ d Extract Subclass or Interface

3p **3c** In the CourseRegistration system, two different methods in separate classes are used for adding a student to a course.

```
public class OnlineCourseRegistration {
    public void registerStudent(Student student, Course
        course) {
        // Check if course is not full
        // Add student to course
        System.out.println("Student registered for online
            course.");
    }
}

public class OnsiteCourseRegistration {
    public void registerStudent(Student student, Course
        course) {
        // Check if course is not full
        // Add student to course
        System.out.println("Student registered for onsite
            course.");
    }
}
```

- ☐ a Long Method
- ☐ b Duplicated Code
- ☐ c Message Chains
- ☐ d Large Class

3p **3d** Consider the following way of accessing a student's enrolled course's instructor's email in the university course management system. What code smell is demonstrated by this example?

```
public class StudentService {  
    public String getInstructorEmail(Student student,  
        String courseName) {  
        return student.getEnrolledCourses().stream()  
            .filter(course ->  
                course.getName().equals(courseName))  
            .findFirst()  
            .get()  
            .getInstructor()  
            .getEmail();  
    }  
}
```

- ☐ a Inappropriate Intimacy
- ☐ b Message Chains
- ☐ c Data Clumps
- ☐ d Primitive Obsession

3p **3e** What code smell could the following code snippet indicate?:

```
public class NotificationService {  
  
    public void sendNotification(String type, String message) {  
  
        if (type.equals("Email")) {  
  
            // Send email  
  
        } else if (type.equals("SMS")) {  
  
            // Send SMS  
  
        } else if (type.equals("Push")) {  
  
            // Send push notification  
  
        }  
  
        // Additional else-if blocks for other notification types  
  
    }  
}
```

- a Switch Statements
- b Long Parameter List
- c Primitive Obsession
- d Speculative Generality

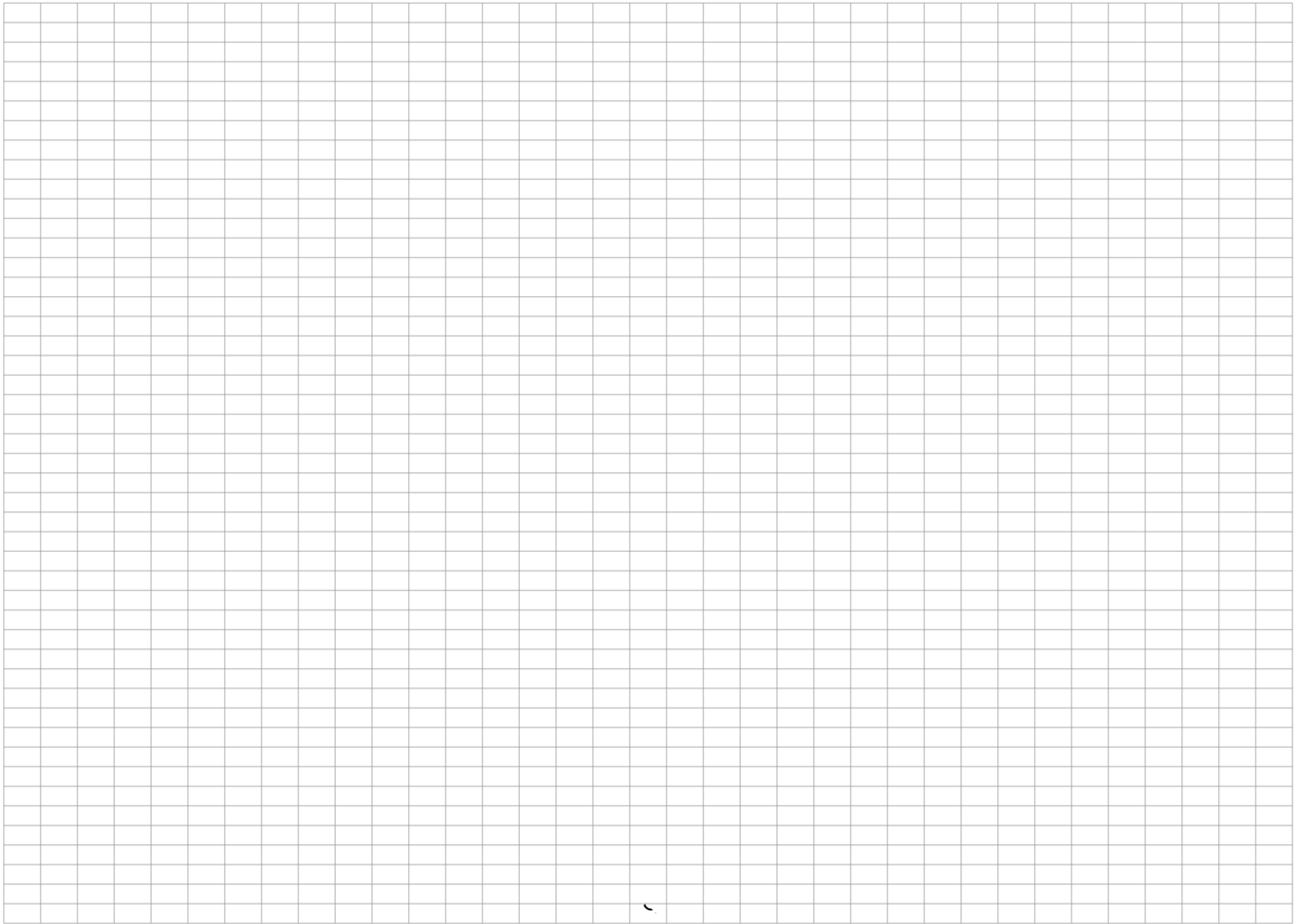
Q4. UML

10p 4.1

In an online book rental system, a book can go through various states: Available, Reserved, Rented, and Returned. A customer can reserve an available book, which then changes its state to Reserved. A reserved book can be rented, changing its state to Rented. Finally, once the book is returned by the customer, its state changes back to Available. If a reserved book is not rented within a certain period, it automatically goes back to the Available state.

Draw a UML state diagram for a book in the online book rental system. Your diagram should clearly show all the states a book can be in, along with the transitions between these states based on the actions performed (e.g., reserve, rent, return).

A large grid for drawing a UML state diagram. The grid is 20 columns wide and 20 rows high, providing a space for the student to draw the state diagram for the book rental system.

A large rectangular area filled with a light gray grid, resembling graph paper, intended for writing the use case.

10p **4.2**

In a university library system, students can search for books based on titles, authors, or subjects. Once a student finds a book they are interested in, they can check its availability. If the book is available, the student can choose to borrow it by entering their student ID and the expected return date. The system then updates the book's status to checked out and records the transaction.

Write a detailed use case for the "Borrow a Book" functionality in the university library system. Include the following elements in your use case: Use Case Name, Actors, Preconditions, Main Flow, Alternative Flows, and Postconditions. Make sure your use case captures all steps involved in borrowing a book, from searching for it to successfully checking it out.



## Q5. Design Patterns

15p **5a** In a software development project management tool, tasks are assigned to different categories (e.g., Development, Testing, Design). The tool currently uses a single method in the TaskManager class to create tasks based on a string parameter indicating the category. As the project complexity increases, there's a need to introduce more categories and associated task types without continuously modifying the TaskManager class.

```
public class TaskManager {  
  
    public Task createTask(String category) {  
  
        if (category.equals("Development")) {  
  
            return new DevelopmentTask();  
  
        } else if (category.equals("Testing")) {  
  
            return new TestingTask();  
  
        } else if (category.equals("Design")) {  
  
            return new DesignTask();  
  
        }  
  
        // Additional categories...  
  
        return null;  
  
    }  
  
}
```

- a) Identify the problem with the current approach.
- b) Suggest a design pattern that can solve this problem, making the creation of tasks more flexible and extensible.
- c) Provide a high-level UML diagram or pseudocode to illustrate how your suggested design pattern can be implemented to solve the issue.

15p     **5b**     In a content management system (CMS), you are tasked with enhancing the functionality for rendering pages. The system currently uses a single method in the PageRenderer class to render pages. You anticipate the need to support different rendering styles (e.g., HTML, Markdown, XML) in the future without altering the existing rendering code.

```
public class PageRenderer {  
  
    public String renderPage(String pageContent, String  
        renderStyle) {  
  
        if (renderStyle.equals("HTML")) {  
  
            // Render page as HTML  
  
            return "<html>" + pageContent + "</html>";  
  
        } else if (renderStyle.equals("Markdown")) {  
  
            // Render page in Markdown  
  
            // Markdown rendering logic  
  
            return pageContent; // Simplified for example  
  
        } else if (renderStyle.equals("XML")) {  
  
            // Render page as XML  
  
            return "<xml>" + pageContent + "</xml>";  
  
        }  
  
        // Additional rendering styles...  
  
        return pageContent;  
    }  
}
```

- a) Identify the potential issue with the current approach.
- b) Suggest a design pattern that can solve this problem, enhancing the system's flexibility and extensibility in supporting various rendering styles.
- c) Provide a high-level UML diagram or pseudocode to illustrate how your suggested design pattern can be implemented to address the issue.

### Extra Space

If you use these extra answer boxes, **please mention clearly in your main answer that part of your answer can be found here!**

**6a**

**6b**



