Examination (with solutions)
Object-oriented programming

Course code: DIT044

Date:	2025-03-17
Questions:	5
Results:	Posted within 15 working days.
Allowed aids:	No aids (books or calculators) are allowed.

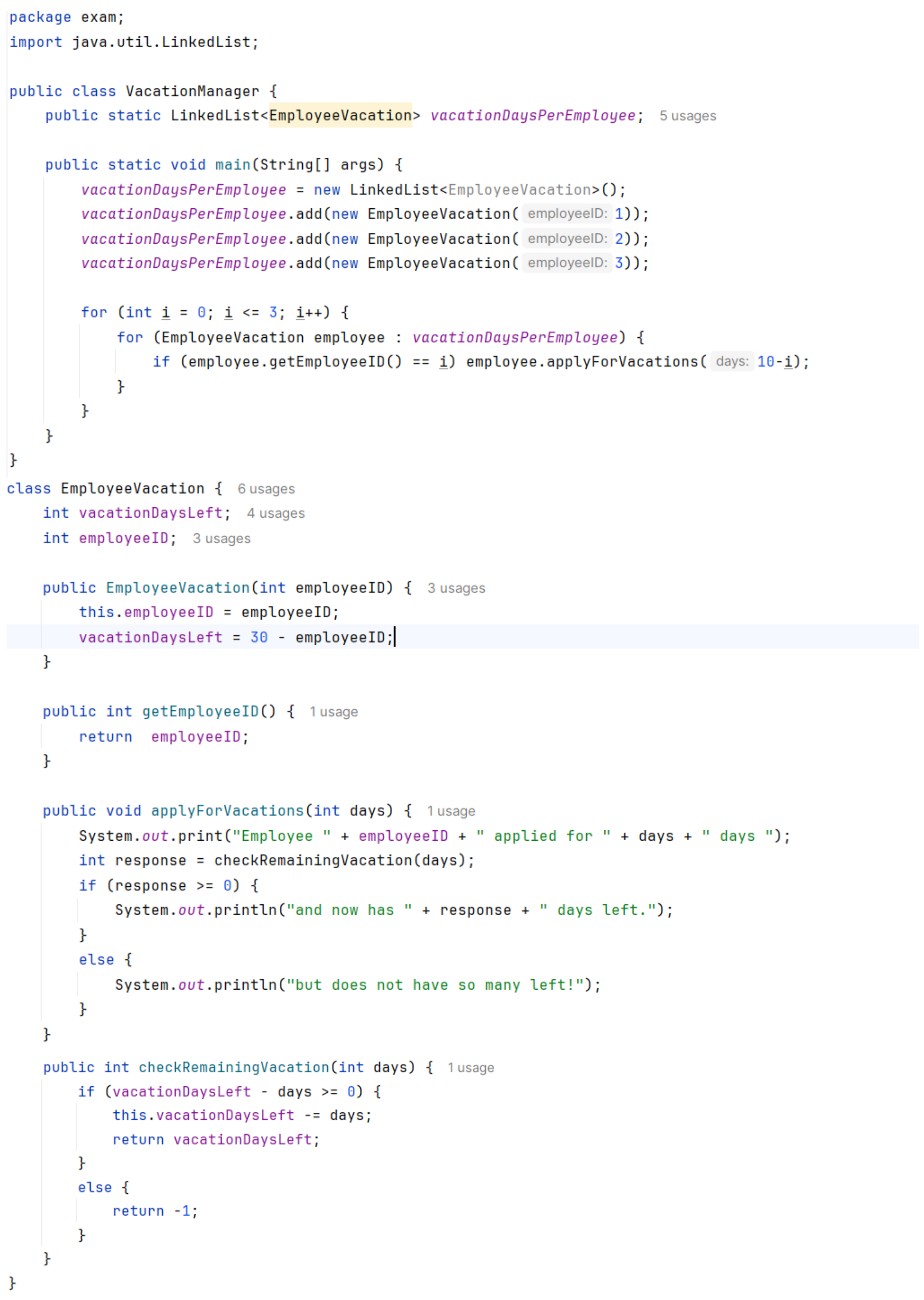
Please observe carefully the following:

- Write in legible English (illegible translates to “no points”).
- Motivate your answers, and clearly state any assumptions made.
- Start each task on a new sheet and write on only one side of the paper.
- Before handing in your exam, number and sort the sheets in task order.
- Write your anonymous code and page number on every page.
- The exam is anonymous, do not leave any information that would reveal your name on your exam sheets.
- For all class diagrams in your exam, write all methods and attributes relevant to your code, including constructors, getters and setters.

Not following these instructions will result in the deduction of points.

Question 1 (15pt)

1.1. Write the output of this program. (9pt)



```java
package exam;
import java.util.LinkedList;

public class VacationManager {
    public static LinkedList<EmployeeVacation> vacationDaysPerEmployee;  5 usages

    public static void main(String[] args) {
        vacationDaysPerEmployee = new LinkedList<EmployeeVacation>();
        vacationDaysPerEmployee.add(new EmployeeVacation( employeeID: 1));
        vacationDaysPerEmployee.add(new EmployeeVacation( employeeID: 2));
        vacationDaysPerEmployee.add(new EmployeeVacation( employeeID: 3));

        for (int i = 0; i <= 3; i++) {
            for (EmployeeVacation employee : vacationDaysPerEmployee) {
                if (employee.getEmployeeID() == i) employee.applyForVacations( days: 10-i);
            }
        }
    }
}
class EmployeeVacation {  6 usages
    int vacationDaysLeft;  4 usages
    int employeeID;  3 usages

    public EmployeeVacation(int employeeID) {  3 usages
        this.employeeID = employeeID;
        vacationDaysLeft = 30 - employeeID;
    }

    public int getEmployeeID() {  1 usage
        return  employeeID;
    }

    public void applyForVacations(int days) {  1 usage
        System.out.print("Employee " + employeeID + " applied for " + days + " days ");
        int response = checkRemainingVacation(days);
        if (response >= 0) {
            System.out.println("and now has " + response + " days left.");
        }
        else {
            System.out.println("but does not have so many left!");
        }
    }

    public int checkRemainingVacation(int days) {  1 usage
        if (vacationDaysLeft - days >= 0) {
            this.vacationDaysLeft -= days;
            return vacationDaysLeft;
        }
        else {
            return -1;
        }
    }
}
```

Solution: 3pt per fully correct line
Employee 1 applied for 9 days and now has 20 days left.
Employee 2 applied for 8 days and now has 20 days left.
Employee 3 applied for 7 days and now has 20 days left.

1.2. Describe the relationship between these two OOP concepts. (6pt)

1. Class attribute
2. Constructor

Solution: 3pt for correct answer (relationship explanation makes sense, 0 points if these are independent definitions), 3pt for using right terms (e.g., method, parameters, initialize, etc.).

Question 2 (15pt)

2.1. Define composition and provide a short Java or pseudo-code example. (5pt)

Solution: 2pt for explanation, 3pt for the example

2.3. Consider abstract class B as a child of another abstract class A. If A has an abstract method x(), does B have to provide an implementation for that method? Or can it remain abstract? Justify your answer. (10pt)

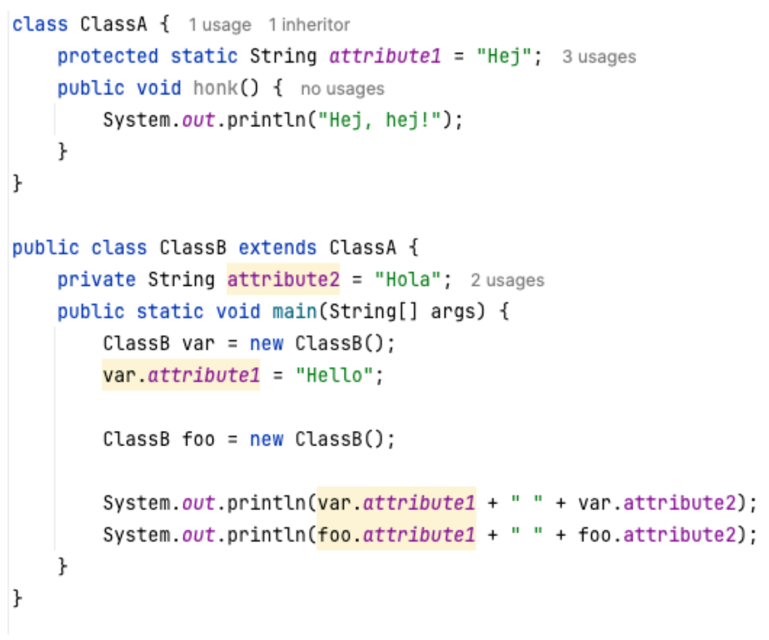
Solution: 5pt sensible explanation (makes sense), 5pt for using precise terminology.

Question 3 (15pt)

3.1. Describe inheritance. How do we use it to achieve polymorphism? (10pt)

Solution: 5pt sensible explanation (makes sense), 5pt for using precise terminology

3.2. What will the following code print and why? (5pt)



```
class ClassA {  1 usage  1 inheritor
    protected static String attribute1 = "Hej";  3 usages
    public void honk() {  no usages
        System.out.println("Hej, hej!");
    }
}

public class ClassB extends ClassA {
    private String attribute2 = "Hola";  2 usages
    public static void main(String[] args) {
        ClassB var = new ClassB();
        var.attribute1 = "Hello";

        ClassB foo = new ClassB();

        System.out.println(var.attribute1 + " " + var.attribute2);
        System.out.println(foo.attribute1 + " " + foo.attribute2);
    }
}
```

Solution: 2pt for "Hello Hola\nHello Hola", 3pt for explaining static

Question 4 (30pt)

4.1. What is the intent of the Decorator pattern? Illustrate it with an example (you can use code, but it is not mandatory). (15pt)

Solution: 5pt for correct intent, 5pt for using right terminology, 5pt for example (code or not).

4.2. We are coding a system that manages notifications of different types for different apps. We want to separate the product construction from the code that uses the product, so that it is easier to extend the construction code independently from the rest. We thought of having an interface for creating objects while allowing subclasses to deal with the type of objects to create (versions). What design pattern should we follow? What is its intent? Write a pseudo-code implementation for the key methods. (15pt)

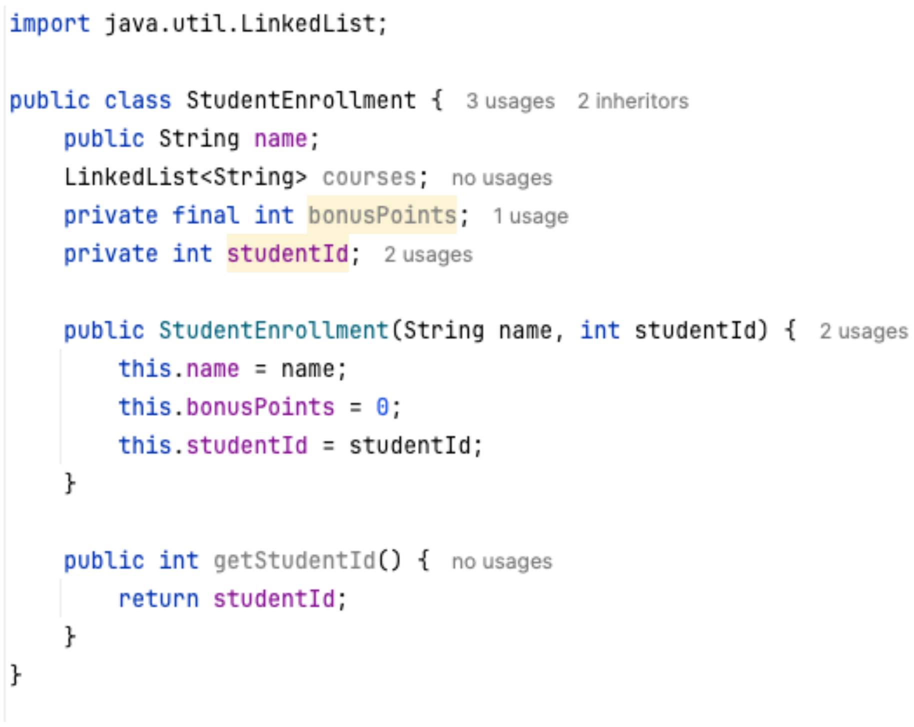
Solution: 1pt for saying "observer" (if explanation is provided BUT this is not what the exercise is about, the word "notification" could be changed for "icon" and then this would not hold) or 2pt "factory", 3pt for stating intent (for factory), 10pt for code or pseudo-code that makes sense, i.e., there is an interface (5pt) and same method signatures (5pt).

Question 5 (25pt)

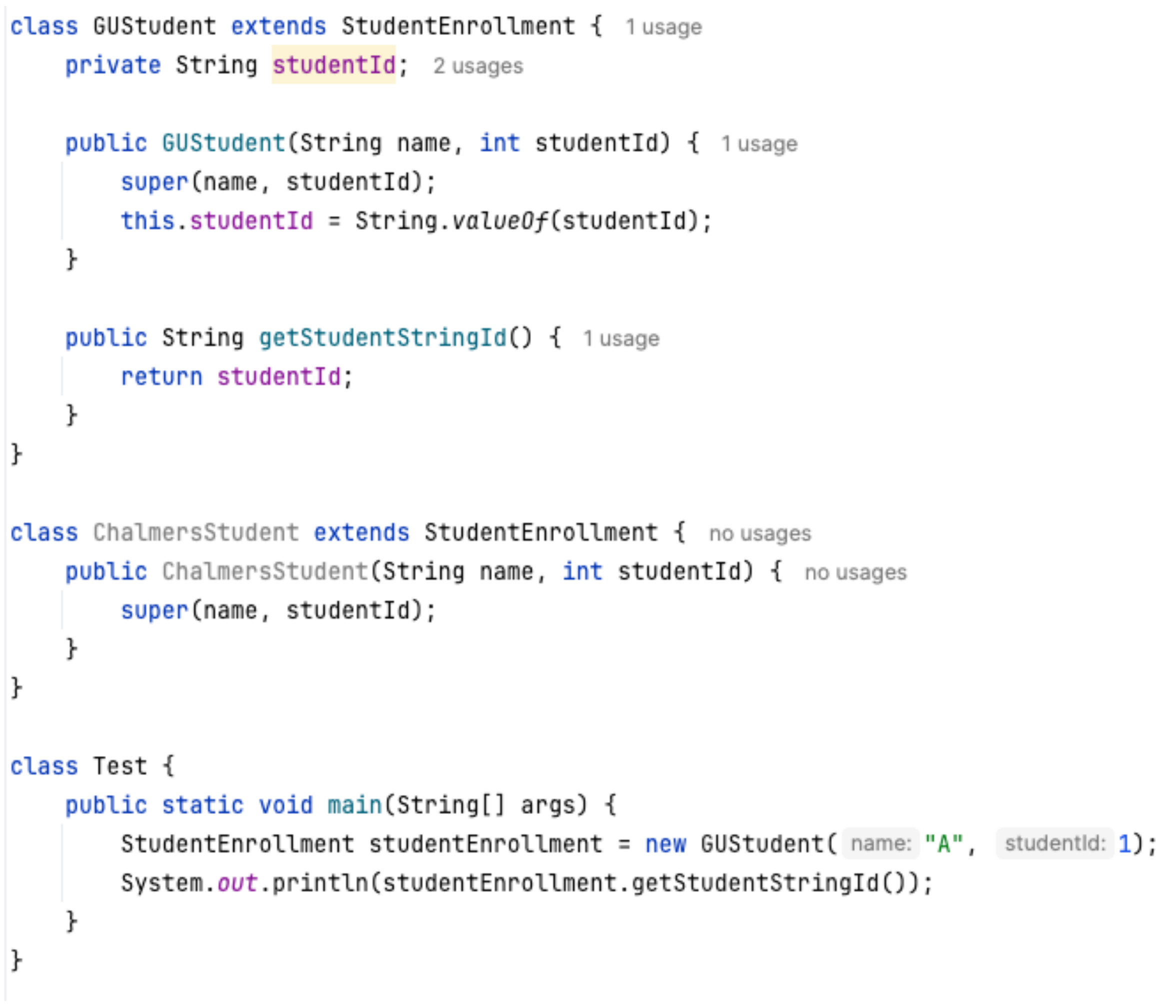
5.1. You and your friend Bea were hired to implement a program for managing course enrollments and storing "bonus points" to students. Bea implemented the program

below. Does the code fulfil the specification? Justify your answer pointing to elements of the code. Does the code throw any error or exception? Which and why? (10pt)

The system must support two types of student enrollments: GU and Chalmers. Both types of membership share common features like storing the member's name, a list of courses, and accumulated bonus points (from taking extracurricular activities). GU students have a numeric ID while Chalmers students have an alphanumeric ID. The implementation by Bea:



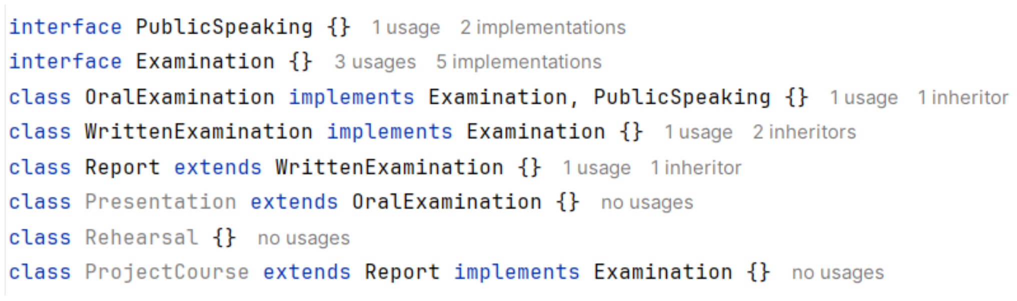
```java
import java.util.LinkedList;

public class StudentEnrollment {  3 usages   2 inheritors
    public String name;
    LinkedList<String> courses;  no usages
    private final int bonusPoints;  1 usage
    private int studentId;  2 usages

    public StudentEnrollment(String name, int studentId) {  2 usages
        this.name = name;
        this.bonusPoints = 0;
        this.studentId = studentId;
    }

    public int getStudentId() {  no usages
        return studentId;
    }
}
```



```
class GUStudent extends StudentEnrollment {  1 usage
    private String studentId;  2 usages

    public GUStudent(String name, int studentId) {  1 usage
        super(name, studentId);
        this.studentId = String.valueOf(studentId);
    }

    public String getStudentStringId() {  1 usage
        return studentId;
    }
}

class ChalmersStudent extends StudentEnrollment {  no usages
    public ChalmersStudent(String name, int studentId) {  no usages
        super(name, studentId);
    }
}

class Test {
    public static void main(String[] args) {
        StudentEnrollment studentEnrollment = new GUStudent( name: "A",  studentId: 1);
        System.out.println(studentEnrollment.getStudentStringId());
    }
}
```

Solution: 2pt for interpretation of the code, 2pt for pointing at elements in the code in explanation (0pt if they do not), 2pt for "it is Chalmers that has an alphanumeric ID" and "cannot assign a value to final variable bonusPoints", 2pt for "StudentEnrollment does not have method getStudentStringId()", 2pt for using right terminology (override, overload, inherit, extend, polymorphism, parent class, etc.)

5.2. Draw a UML diagram representing the relationship between these classes. (15pt)



```
interface PublicSpeaking {}  1 usage   2 implementations
interface Examination {}  3 usages   5 implementations
class OralExamination implements Examination, PublicSpeaking {}  1 usage   1 inheritor
class WrittenExamination implements Examination {}  1 usage   2 inheritors
class Report extends WrittenExamination {}  1 usage   1 inheritor
class Presentation extends OralExamination {}  no usages
class Rehearsal {}  no usages
class ProjectCourse extends Report implements Examination {}  no usages
```

Solution: 1pt per each of 8 classes, 1pt for each correct arrow (7 in total)