DIT044 exam with solutions

Question 1 (W45, 15pt)

1.1. Define these OOP concepts. (5pt)

a. Access modifiers
b. Constructor

Solution: ok definitions (can be short) = 5pt, only one definition or definitions are incomplete or a bit incorrect = 2pt, definitions are incorrect = 0pt.

1.2. Fill in the gaps in this explanation. Please provide your answers using the numbers in the pool of words (e.g., "A1, B2, C3"). (5pt)

Imagine you have many identical (A) of a single class and you want to change all of them. You can convert all those objects to a single (B) object. That one object will contain all the most current information about a particular entity. If the object is changed, these changes are accessible from the other parts of the (C) that have a (D) to it. If the objects that you want to access are created in advance, you need to decide how to handle (E) when a non-existent object is requested.

Pool of words: (1) reference, (2) instances, (3) list, (4) syntax, (5) program, (6) errors, (7) class, (8) Java, (9) constructor, (10) index

Solution: A2, B1, C5, D1, E6

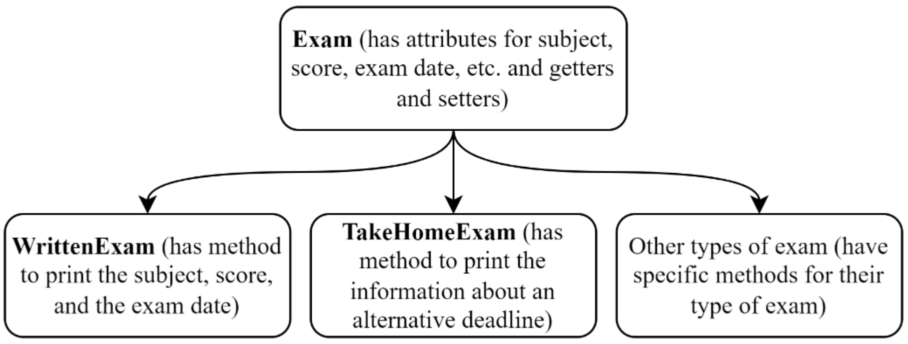
1.3. Write the output of this program. (5pt)

```
class BinaryConverter {
    public static void main(String[] args) {
        int[] numbers = {1, 15};

        for (int number : numbers) {
            String binary = convertToBinary(number);
            System.out.println(binary);
            System.out.println(countDigits(number));
        }
        System.out.println("The end!");
    }

    public static String convertToBinary(int number) {
        String binary = "";
        while (number > 0) {
            binary = (number % 2) + binary;
            number = number / 2;
        }
        return binary;
    }

    public static int countDigits(int number) {
        return String.valueOf(number).length();
    }
}
```

Solution: 1 \n 1 \n 1111 \n 2 \n The end! (1pt per correct line)

Question 2 (W46, 15pt)

2.1. Write the class WrittenExam so that it inherits all the attributes and methods from a class Exam (not provided), including getters and setters for each attribute. On top of those, WrittenExam has an extra field for the number of questions. Create a method for the WrittenExam class that prints the subject, score, and the exam date, all of which are attributes of the Exam class. (5pt)


Exam (has attributes for subject, score, exam date, etc. and getters and setters)
WrittenExam (has method to print the subject, score, and the exam date)
TakeHomeExam (has method to print the information about an alternative deadline)
Other types of exam (have specific methods for their type of exam)

Grading: 1pt for each of [“extends”, super constructor, has questions attribute, print details method (single method for 3 attributes in Exam), and overall quality/readability]

2.3. Write two differences between abstract classes and interfaces. What happens if a class does not override an abstract method of its abstract parent class? (10pt)

Grading, part 1: 5pt for two differences, 2pt for 1 difference or 2 "questionable" or unclear differences. Part 2: 5pt for "either subclass is also abstract, or the compiler will throw an error." Grading, part 2: both alternatives = 5pt, one alternative or both are somewhat correct = 2pt, no keywords or incorrect answer = 0pt.

Question 3 (W47, 15pt)

3.2. Describe polymorphism. How do we use inheritance and other mechanisms to achieve polymorphism? (10pt)

Grading: what and why polymorphism = max 5pt, part two: brief explanations of [inheritance, overriding, overloading, casting, late binding] = max 5pt

3.1. Write the abstract classes EntertainmentMovie and EducationalMovie. (5pt)

```
interface Movie {
    void play();
}

class ActionMovie extends EntertainmentMovie {
    public void play() {
        relax();
        provideSnacks();
        recommendSimilarMovies();
    }
    void relax() { System.out.println("Sit back and enjoy the movie!"); }
}

class HistoryDocumentary extends EducationalMovie {
    public void play() {
        learn();
        provideReferences();
        quizAfterMovie();
    }
}

public class MovieLibrary {
    public static void main(String[] args) {
        Movie[] movies = {new ActionMovie(), new HistoryDocumentary()};
        for (Movie movie : movies) { movie.play(); }
    }
}
```

Solution: classes are abstract (2pt), keyword implements used (2pt), learn method does not have a body (1pt),

```
abstract class EntertainmentMovie implements Movie {
    abstract void relax();
    void provideSnacks() { System.out.println("Snacks are provided"); }
    void recommendSimilarMovies() { System.out.println("Enjoy more hits!"); }
}

abstract class EducationalMovie implements Movie {
    void learn() { System.out.println("Expand your knowledge!"); }
    void quizAfterMovie() { System.out.println("Test your knowledge."); }
    void provideReferences() { System.out.println("Additional references."); }
}
```

Question 4 (W48-49, 30pt)

4.1. What is the intent of the Strategy pattern? (5pt)

Grading: intent is not there but description somewhat ok or examples = max 2 pt, intent is there and well explained (can be short) = 5pt

4.2. What is the name of the design pattern that lets you attach new behaviors to objects by placing them inside special objects that contain the extra behaviors? (5pt)

Solution: decorator (5pt) or wrapper (2pt).

4.3. We want to ensure that a class Database has only one instance and that a global access point to this instance is available throughout the code. What design pattern should we follow? Is it creational, behavioral, or structural? Write the class Database (no need to write the logic inside methods, a comment or a print is enough). (10pt)

Solution: Singleton (3pt), creational (2pt), private constructor (3pt), static getInstance or similar method (2pt).

4.4. Fill in the gaps in this explanation. Please provide your answers using the numbers in the pool of words (e.g., "A1, B2, C3"). (5pt)

The (A) pattern suggests that you replace direct object construction with calls to a special method: the objects are still created, but it's being called from within a separate (B). Objects returned by this method are often referred to as products. The code that uses this method (often called the (C) code) doesn't see a difference between the actual products returned to it (e.g., cars, motorbikes, trucks, etc.), instead it treats all the products as abstract "Transport". The code knows that all transport objects are supposed to have the same methods so it can use them (D), but exactly how it works isn't important. To achieve this, this pattern recommends making all products follow the same (E), which should declare methods that make sense in every product.

Pool of words: (1) Factory Method, (2) Observer, (3) Façade, (4) client, (5) interface, (6) contract, (7) aggregation, (8) class, (9) interchangeably, (10) uniquely.

Solution: A1, B8, C4, D9, E5 or E6

4.5. This code follows the Model-View-Controller design pattern. Which class contains the needed data operations for the application? (5pt)

```java
class ClassA {
    private String info1;

    public ClassA(String name) { this.info1 = name; }

    public String getInfo1() { return info1; }
    public void setInfo1(String info1) { this.info1 = info1; }
}

class ClassB {
    private ClassA info1;
    private ClassC info2;

    public ClassB(ClassA classAinstance, ClassC classCinstance) {
        this.info1 = classAinstance;
        this.info2 = classCinstance;
    }

    public void updateInfo(String name) { info1.setInfo1(name); }
    public void refreshView() { info2.display(info1.getInfo1()); }
}

class ClassC {
    public void display(String info1) {
        System.out.println("Details: " + info1);
    }
}
```

Solution: ClassA, the Model, etc. Comment: class B is clearly the controller; it has A and C!

Question 5 (W50, 25pt)

5.2. You and your friend Bea were hired to implement a program for managing gym memberships, and gifting "gym points". 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 exception? Which and why? (10pt)

The system must support two types of memberships: Gothenburg and All-Sweden. Both types of membership share common features like storing the member's name, their regular gym, and accumulated points. In the Gothenburg membership, members can visit any gyms in Gothenburg and earn 10 points for visiting gyms outside their regular one. In the latter, members can also visit gyms outside their regular one, but across the country, earning 20 points for each visit to a gym outside their regular one. All-Sweden members start with 50 points as a bonus.

The implementation by Bea:

```java
public class Membership {
    private static final double BONUS = 0.10;
    private static final int EXTRA_POINTS = 20;
    private static String type;
    private String name;
    private double gymPoints;
    private int memberPoints;

    public Membership(String name, double initialPoints, String type) {
        this.name = name;
        this.gymPoints = initialPoints;
        if (this.type.equals("Country Membership")) { this.memberPoints = 20; }
        else { this.memberPoints = 0; }
        this.type = type;
    }

    public void addPoints(double additionalPoints) {
        double bonusPoints = additionalPoints;
        if (type.equals("Country Membership")) {
            bonusPoints = additionalPoints + (BONUS * additionalPoints);
        }
        this.gymPoints = this.gymPoints + bonusPoints;
    }

    public void addMemberPoints(int additionalPoints) {
        int bonusPoints = additionalPoints;
        if (type.equals("City Membership") || type.equals("Country Membership"))
            bonusPoints = bonusPoints + EXTRA_POINTS;
        }
        this.memberPoints += bonusPoints;
    }

    // getters and setters (well implemented) here
}
```

Grading: student states whether implementation is ok or not and why (2pt), student uses parts of the code in explanation (3pt). Student writes that code throws an exception "A NullPointerException occurred: Cannot invoke "String.equals(Object)" because "Membership.type" is null" or similar = 5pt. Comment: type vs this.type ← still "empty" when used!

5.1. Fill in the gaps in this explanation. Please provide your answers using the numbers in the pool of words (e.g., "A1, B2, C3"). (5pt)

(A) is the degree of interdependence between (B), a measure of how closely connected they are, and the strength of the relationships between the classes. On the other hand, (C) is the strength of the relationship between the (D) and data of a class and some unifying purpose or concept served by that class. This is measurable! One example metric to measure it is (E). If this metric equals to 1, the class is cohesive.

Pool of words: (1) cohesion, (2) LCOM4, (3) coupling, (4) classes, (5) instances, (6) composition, (7) methods, (8) counting, (9) intuition, (10) coders.

Solution: A3, B4, C1, D7, E2

5.3. Draw a UML diagram representing the relationship between these classes. (10pt)

Several classes are used to represent some real-life containers and real-life contents. An example of this could be a necklace, that is a physical object and can be stored inside a box. Other types of contents, though, are digital (such as pictures taken with a digital camera). These concepts are related to one another and the relationship between them can be expressed through code, as seen below:

```java
public class Box {
    Content content;
}
public interface Content {
    abstract void describe();
}
public abstract class PhysicalObject implements Content {
}
public class Necklace extends PhysicalObject {
    public void describe() {
        System.out.println("I could be in a box.");
    }
}
public class DigitalContent implements Content {
    public void describe() {
        System.out.println("Like and subscribe!");
    }
}
public class JewelryBox {
}
```

Solution: 6 classes (Box, Content, JewelryBox, DigitalContent, PhysicalObject, and Necklace). Grading: 2pt x 4 correct connections (aggregation/composition both count as correct in this case), JewelryBox is not connected to anything (2pt).