Ne rien inscrire dans ce cadre

Prénom .....

Nom .....

Promotion .....

Groupe .....

Promotion : L3-APP-LSI

Module : Programmation en Java Avancée

ALSI62

DE - 1h45 min Horaire : 10H20-12H10

Examen sur table

Sujet rédigé par : Monsieur Arsène Lapostolet

Supports autorisés :

Calculatrice autorisée : **non**

Documents autorisés : non

Traducteur électronique : non

Dictionnaire : non

Consigne :

Merci de restituer : votre copie quadrillée

Rappel :

- Tous les appareils électroniques (téléphones portables, ordinateurs, tablettes, montres connectées, accès à internet ...) doivent être éteints et rangés.
- Il est interdit de communiquer.
- Toute fraude ou tentative de fraude fera l'objet d'un rapport de la part du surveillant et sera sanctionnée par la note zéro, assortie d'une convocation devant le conseil de discipline. Aucune contestation ne sera possible. Tous les documents et supports utilisés frauduleusement devront être remis au surveillant.
- Aucune sortie de la salle d'examen ne sera autorisée avant la moitié de la durée de l'épreuve.

Partie I – Questions de Cours (10 points)

1. Expliquez en quelque phrases à quoi sert Gradle dans un projet Java (1 point)

2. Expliquez en quelque phrases l'intérêt des tests automatiques (1 point)

3. Expliquez de manière détaillée ce qu'est le polymorphisme et l'intérêt de ce dernier (2 point)

4. Dans le code suivant :

```
public LinkedList<Integer> annee_bissextile(int y, int n) {
   LinkedList<Integer> ly = new LinkedList<>();

   for (var i = y; i < y + n; i++) {
      if ((i % 4 == 0) && (i % 100 != 0) || i % 400 == 0) {
         ly.add(i);
      }
   }

   return ly;
}
```

Expliquez quels éléments contreviennent à des principes de propreté du code, citez ces derniers (2 points).

5. Expliquez ce qu'est une interface fonctionnelle en Java (1 point)

6. Expliquez le principe de l'opérateur fonctionnel *flatMap* (1 point)

7. Expliquez de manière détaillée ce qu'est une section critique et comment éviter les problèmes avec ces dernières (2 points).

Partie II –Exercice (10 points)

Le code fourni est un ensemble de classes Java permettant de traiter les absences des étudiants de l'EFREI. Après avoir examiné le code, proposez une stratégie de refactoring permettant d'améliorer la maintenabilité du code, notamment en réduisant le couplage entre la classe *StudentAbsenceService* et le caractéristiques d'un étudiant. Pas besoin d'écrire de code, décrivez simplement de manière détaillée les étapes à suivre ainsi que votre raisonnement et citez les techniques et *design patterns* utilisés.

Classe StudentAbsenceService

```
public class StudentAbsenceService {

   private final NotificationService notificationService;

   public StudentAbsenceService(NotificationService notificationService) {
      this.notificationService = notificationService;
   }
```

```
public void processAbsence(Student student, LocalDate date) {
    final var absence = student.getAbsenceByDate(date);

    if (hasValidJustificationAttachment(absence, student)) {
        absence.setJustified(true);
    }

    if (student.getProgram() == Student.PROGRAM.APPRENTICE && !absence.isJustified()) {
        notificationService.notifyCompany();
    }

    checkSanctions(student);
}

public boolean hasValidJustificationAttachment(Absence absence, Student student) {
    if (student.getProgram() == Student.PROGRAM.APPRENTICE) {
        return absence
                .getAttachments()
                .stream()
                .anyMatch(attachment -> attachment.type == Attachment.TYPE.SICK_LEAVE);
    }
    return absence
            .getAttachments()
            .stream()
            .anyMatch(attachment -> attachment.type == Attachment.TYPE.PARENT_LETTER);
}

private void checkSanctions(Student student) {

    if (student.getUnjustifiedAbsences().size() >= 10) {
        student.getSanctions().add("Avertissement");

        if (student.getLevel() == Student.LEVEL.LICENCE) {
            notificationService.notifyParents();
        }

        if (student.getUnjustifiedAbsences().size() >= 15) {
            if (student.getDiplomaKind() == Student.DIPLOMA.ENGINEER) {
                if (student.getLevel() == Student.LEVEL.LICENCE) {
                    student.getSanctions().add("Conseil de Discipline");
                }
                if (student.getLevel() == Student.LEVEL.MASTER) {
                    student.getSanctions().add("Rattrapages interdits");
                }
            } else if (student.getDiplomaKind() == Student.DIPLOMA.EXPERT) {
                student.getSanctions().add("Conseil de Discipline");
                student.getSanctions().add("Rattrapages interdits");
            }

        }

    }

}
```

Classe Student

```
public class Student {

    private final String name;
```

```
    private final String email;
    private final DIPLOMA diplomaKind;
    private final PROGRAM program;
    private final LEVEL level;
    private final List<Absence> absences;


    private final List<String> sanctions;

    public Student(String name, String email, DIPLOMA diplomaKind, PROGRAM program, LEVEL level, List<Absence> absences,
List<String> sanctions) {
        this.name = name;
        this.email = email;
        this.diplomaKind = diplomaKind;
        this.program = program;
        this.level = level;
        this.absences = absences;
        this.sanctions = sanctions;
    }

    public List<String> getSanctions() {
        return sanctions;
    }

    public enum LEVEL {LICENCE, MASTER}
    public enum PROGRAM {INITIAL, APPRENTICE}
    public enum DIPLOMA {ENGINEER, EXPERT}

    public Absence getAbsenceByDate(LocalDate date) {
        return absences
                .stream()
                .filter(absence -> absence.getDate().equals(date))
                .findFirst()
                .orElseThrow();
    }


    public List<Absence> getUnjustifiedAbsences() {
        return absences
                .stream()
                .filter(absence -> !absence.isJustified())
                .toList();
    }


    public String getName() {
        return name;
    }

    public String getEmail() {
        return email;
    }

    public DIPLOMA getDiplomaKind() {
        return diplomaKind;
    }

    public PROGRAM getProgram() {
        return program;
    }

    public LEVEL getLevel() {
        return level;
```

```
    }
}
```

Classe Absence

```
public class Absence {
    private final LocalDate date;
    private boolean justified = false;

    public Absence(LocalDate date) {
        this.date = date;
    }

    public List<Attachment> getAttachments() {
        return null;
    }

    public LocalDate getDate() {
        return date;
    }

    public boolean isJustified() {
        return justified;
    }

    public void setJustified(boolean justified) {
        this.justified = justified;
    }
}
```

La classe Attachment *est omise par soucis de brieveté, et n'est pas requise dans le cadre de l'exercice.*