Ne marquez pas les imports. Ne rendez pas l'énoncé avec votre copie. N'indiquez que le code qui change (ne recopiez pas ce qui ne change pas, mais il faut qu'on sache où se place le code que vous donnez).

Important : pour tous les exercices, placez les classes dans les paquetages adéquats.

Exercice 1 – Passage d'élèves : redondance de code *(environ 3 pts)*

Le code suivant calcule le passage d'étudiants de DUT et de Master. Un étudiant passe à l'année suivante si sa moyenne est supérieure ou égale à 10. Ce seuil de passage de 10 est toujours le même pour tous les étudiants (DUT et Master). Ce code présente le défaut de redondance de code.

```java
package etudiants;

public abstract class Etudiant {

    private double numero;

    public Etudiant(int numero) {
        this.numero = numero;
    }

    public abstract boolean passage();

}
```

```java
package etudiants;
public class EtudiantDUT extends Etudiant {
  private double noteUE1;
  private double noteUE2;
  private static double seuilPassage = 10;
  public EtudiantDUT(int num, double ue1, double ue2){
        super(num);
        this.noteUE1 = ue1; this.noteUE2 = ue2;
  }
  @Override
  public boolean passage () {
        double moyenne = (noteUE1 + noteUE2) / 2 ;
        if (moyenne >= seuilPassage)
              return true;
        return false;
  }
}
```

```java
package etudiants;
public class EtudiantMaster extends Etudiant {
    private double noteUE4;
    private char appreciationUE5; // ses valeurs sont : 'A', 'B' ou 'C'
    private static double seuilPassage = 10;
    public EtudiantMaster(int numero, double noteUE4, char appreciationUE5) {
        super(numero); this.noteUE4 = noteUE4; this.appreciationUE5 = appreciationUE5;
    }
    @Override
    public boolean passage () {
        double coeffMajoration;
        if (appreciationUE5=='A')
              coeffMajoration = 1.1;
        else if (appreciationUE5=='B')
              coeffMajoration = 1;
        else coeffMajoration = 0.9;
        double moyenne = coeffMajoration * noteUE4;
        if (moyenne >= seuilPassage)
              return true;
        return false;
    }
}
```

Scanned by CamScanner

Travail demandé

1. Donner le code redondant.
2. Corriger les redondances de code en donnant le nouveau code des classes modifiées.

Exercice 2 – Sécurisation des ascenseurs : design patterns *(environ 7 pts)*

Le code suivant commande le moteur des ascenseurs d'une entreprise de fabrication d'ascenseurs. Les méthodes de la classe Ascenseur sont appelées par les capteurs qui équipent l'ascenseur. Lorsqu'un bouton « *étage* » est appuyé, le capteur du bouton appelle la méthode `allerVersEtage(numEtage)`. Les étages sont équipés de capteurs d'étage qui appellent la méthode `arriveeEtage(numEtage)` à chaque fois que l'ascenseur passe un étage.

```java
package ascenseur;
public class Ascenseur implements IAscenseur {
  private int etageCourant = 0;
  private int etageDestination = 0;
  private Moteur moteur = new Moteur();
  public Ascenseur(int etageDepart, Moteur moteur){
      this.etageCourant = etageDepart;
      this.moteur = moteur;
  }
  public void allerVersEtage(int numEtage) {
      this.etageDestination = numEtage;
      if (etageCourant < etageDestination)
            moteur.monter ();
      else if (etageCourant > etageDestination)
            moteur.descendre ();
  }
  public void arriveeEtage(int numEtageArrivee) {
      this.etageCourant = numEtageArrivee;
      if (numEtageArrivee == this.etageDestination)
            arreterMoteur();
  }
  public void arreterMoteur() {
      this.moteur.arreter ();
  }
  // méthode ajoutée pour les tests
  public int getEtageCourant() {
      return etageCourant;
  }
}
```

```java
package ascenseur;
public interface IAscenseur {
  void allerVersEtage(int numEtage);
  void arriveeEtage(int numEtageCourant);
  void arreterMoteur();
}
```

```java
package ascenseur;
public class Moteur {
  private boolean moteurEnMouvement =
false;
  private String sensMouvement;
  public void monter() {
    this.sensMouvement = "vers_haut";
    this.moteurEnMouvement = true;
  }
  public void descendre() {
    this.sensMouvement = "vers_bas";
    this.moteurEnMouvement = true;
  }
  public void arreter() {
    this.moteurEnMouvement = false;
  }
  public boolean isMoteurEnMouvement(){
    return moteurEnMouvement;
  }
}
```

```java
package tests.ascenseur;
class TestsAscenseur {
 @Test
 void test() {
   Moteur m = new Moteur();
   Ascenseur a = new Ascenseur(0, m);
   assertEquals(0, a.getEtageCourant());
   assertFalse(m.isMoteurEnMouvement());
   a.allerVersEtage(2);
   assertTrue(m.isMoteurEnMouvement());
   a.arriveeEtage(1);
   a.arriveeEtage(2);
   assertFalse(m.isMoteurEnMouvement());
   assertEquals(2, a.getEtageCourant());
 }
}
```

On voudrait ajouter deux types de sécurité au fonctionnement de ces ascenseurs :

- Sécurité anti-dépassement : si le capteur de l'étage de destination de l'ascenseur est défaillant (il ne fait donc pas appel à `arriveeEtage()`), l'ascenseur ne s'arrêtera jamais (informatiquement en tout cas) et cela se traduira par un accident. On voudrait donc forcer l'arrêt systématique de l'ascenseur aux étages le plus bas et le plus haut (ajout des attributs `minEtage` et `maxEtage`) qu'on définira pour chaque ascenseur. Nous supposons que les capteurs de ces étages aux extrémités sont infaillibles.

Scanned by CamScanner

- Sécurité anti-surpoids : on voudrait équiper les ascenseurs d'un capteur de pesée afin d'empêcher l'ascenseur de démarrer en cas de surpoids. La charge (poids) maximum supportée est définie pour chaque ascenseur. La classe gérant le capteur de pesée CapteurPesee est fournie ci-contre. La charge mesurée en temps réel est transmise par le capteur de pesée via la méthode setChargeMesuree(). Vous pouvez la récupérer avec getChargeMesuree().

```
package ascenseur.securite;
public class CapeturPesee {
  private double chargeMesuree; // en temps réel
  // cette méthode est utilisée par le capteur de pesée
  public double getChargeMesuree() {
      return chargeMesuree;
  }
  // utilisez cette méthode pour sécuriser l'ascenseur
  public void setChargeMesuree (double chargeMesuree){
      this.chargeMesuree = chargeMesuree;
  }
}
```

Travail demandé

1. Quel pattern permettant de mettre en place la sécurité des ascenseurs sans modifier la classe initiale Ascenseur ?
2. Donner le diagramme de classes de l'application incluant le code de la sécurité.
3. Donner le code Java des classes (inutile de donner le code de classes non modifiées).
4. Donner le code de tests pertinents (écrits avec JUnit) validant votre solution.

Exercice 3 – Principe ISP *(environ 6 pts)*

Le code ci-dessous est un embryon d'application de gestion des examens pour une université.

Appliquez le principe ISP (Interface Segregation Principle) sur l'interface IUsager et les classes qui implémentent ou utilisent cette interface. On veut distinguer plus rigoureusement entre étudiants et enseignants pour ne pas avoir à coder des méthodes absurdes sur Etudiant et Enseignant : seuls les enseignants ont une matière et seuls les étudiants sont inscrits en année 1 ou 2 etc. On veut malgré tout que la classe Université ne dépendent pas des classes concrètes qui représentent les divers types d'usagers. Pensez à mettre à jour les tests unitaires si besoin est (mais le moins possible).

On ne vous demande **pas** de supprimer la redondance de code entre les classes Etudiant et Enseignant.

Les tests sur les exceptions qui doivent être levés n'ont volontairement **pas** été indiqués dans le code : ne les ajoutez **pas**.

```
package université;
public interface IUsager {
      String getNom();
      int getAnnée();
      String getMatiere();
}
package université;
public class Université {
      private Map<String,IUsager> usagers = new HashMap<>();
      public void ajouter(IUsager contact) {
            usagers.put(contact.getNom(), contact);
      }
      public String convoquerExamens(String nom, String details) {
            IUsager e;
            if (usagers.containsKey(nom)
               && (e = usagers.get(nom)) instanceof Etudiant)
                  return nom + " est convoqué(e) aux examens de l'année "
```


Scanned by CamScanner

```
                        + e.getAnnée()+" "+ details;
            else throw new IllegalArgumentException(nom + "n'est pas étudiant");
        }
        public String surveillerExamen(String nom, String details) {
            IUsager p;
            if (usagers.containsKey(nom)
               && (p= usagers.get(nom)) instanceof Enseignant)
                return nom + " doit surveiller l'examen de "
                        + p.getMatiere()+ " "+details;
            else   throw new IllegalArgumentException(nom + "n'est pas enseignant");
        }
}
package usagers;
public class Enseignant implements IUsager {
        private String nom;
        private String matière;
        public Enseignant(String nom, String matière) {
            this.nom = nom;
            this.matière = matière;
        }
        @Override
        public String getNom() {
            return nom;
        }
        @Override
        public int getAnnée() {
            return 0;
        }
        @Override
        public String getMatiere() {
            return matière;
        }
}
package usagers;
public class Etudiant implements IUsager {
        private String nom;
        private int année;
        public Etudiant(String nom, int année) {
            this.nom = nom;
            this.année = année;
        }
        @Override
        public String getNom() {
            return nom;
        }
        @Override
        public int getAnnée() {
            return année;
        }
        @Override
```


Scanned by CamScanner

```
        public String getMatiere() {
                return "";
        }
}
package univTests;
public class UniversitéTest {
        private Université univ = new Université();
        private String nomEtudiant = "Tintin";
        private int année = 4;
        private IUsager étudiant = new Etudiant (nomEtudiant, année);
        private String nomProf = "Tournesol";
        private String matière = "physique";
        private IUsager prof = new Enseignant("Tournesol", "physique");
        private String dateLieu = "le 23 juin amphi Poincaré";
        @Test
        void test() {
                univ.ajouter(étudiant);
                assertEquals(nomEtudiant +" est convoqué(e) aux examens de l'année "
                                + année + " " + dateLieu,
                                univ.convoquerExamens(nomEtudiant, dateLieu));

                univ.ajouter(prof);
                assertEquals(nomProf +" doit surveiller l'examen de "
                                + matière + " " + dateLieu,
                                univ.surveillerExamen(nomProf, dateLieu));
        }
}
```

Exercice 4 – Suppression des dépendances nuisibles *(environ 4 pts)*

Le code ci-dessous est un embryon d'application de géométrie qui vise notamment à calculer la surface totale occupée par un ensemble de polygones. Pour le moment seuls les rectangles sont pris en compte. Notez que pour certains polygones le calcul de la surface ne dépend ni de la largeur ni de la hauteur. Enfin, une instance de Géométrie doit toujours contenir au moins un polygone.

1. Ecrivez en français une contrainte de couplage qui précise ce qu'il faut cacher de quoi.
2. Corrigez le code pour faire respecter cette contrainte. Vous supprimerez la création d'instance dans Géométrie par injection de dépendance. Pensez à mettre à jour les tests unitaires si besoin est (mais le moins possible).

```
package géométrie;
public class Rectangle {
        private double largeur;
        private double hauteur;
        public Rectangle(double largeur, double hauteur) {
                this.largeur = largeur;
                this.hauteur = hauteur;
        }
        public double getLargeur() { return largeur; }
        public double getHauteur() { return hauteur; }
}
package géométrie;
```


Scanned by CamScanner

```java
public class Géometrie {
	private List<Rectangle> rectangles = new ArrayList<>();
	public Géometrie(int largeur, int hauteur) {
		ajouter(new Rectangle(largeur, hauteur));
	}
	public void ajouter(Rectangle r) {
		rectangles.add(r);
	}
	public double surfaceTotale() {
		double s = 0.;
		for(Rectangle p : rectangles) {
			s += p.getLargeur() * p.getHauteur();
		}
		return s;
	}
}
package géométrie;
public class GéometrieTest {
	@Test
	void test() {
		Géometrie e = new Géometrie(4,2);
		Rectangle r = new Rectangle(3,3);

		assertEquals(8., e.surfaceTotale(), 0.001);
		e.ajouter(r);
		assertEquals(17., e.surfaceTotale(), 0.001);
	}
}
```

Scanned by CamScanner

UNIVERSITÉ PARIS DESCARTES

IUT

DÉPARTEMENT INFORMATIQUE

DISCIPLINE : CPOA

Date de l'épreuve : 16/01/19

Année : 2 Groupe : 205

Écrire très lisiblement

NOM (en capitales) : LAROSTOLET

Prénom : Anaïne

NOTE DE 0 À 20	APPRÉCIATIONS
17,5	

Ne rien écrire dans cette marge

Exercice 1

1) Les éléments redondants de ce code sont :

- La constante SeuilPassage qui est présente dans les deux sous classes alors qu'elle est toujours le même.
- Le test qui vérifie si la moyenne est supérieure au seuilPassage.

oui 0,5

2)

```
package etudiants;

public abstract class Etudiants
    private static double seuilPassage
    = 10d;
    private int numero;
```

1/15

Scanned by CamScanner

```
public Etudiant (int numero) {
    this.numero = numero;
}

public static int getSeuilPassage(){
    return seuilPassage;
}

private abstract double moyenne();

public boolean passage () {
    return this.moyenne() >=
            Etudiant.getSeuilPassage();
}
}
package etudiants;

public class EtudiantDUT extends Etudiant {
    private double noteUE1;
    private double noteUE2;

    public EtudiantDUT(int num, double
    ue1, double ue2) {
        super(num);
        this.noteUE1=ue1;
        this.noteUE2=ue2;
    }

@Override
private double moyenne () {
    return (noteUE1+noteUE2)/2;
}

}
```

2/15

Scanned by CamScanner

```
package etudiants

public class EtudiantMaster extends Etudiant
    private double noteUE4;
    private char appreciationUE5;

    public EtudiantMaster(int numero,
    double noteUE4, char appreciationUE5){
        super(numero);
        this.noteUE4 = noteUE4;
        this.appreciationUE5 =
        appreciationUE5;
    }

    @Override
    private double moyenne () {
        double coeffMajoration = 0.9;

        if (appreciationUE5 = 'A'){
            coeffMajoration = 1.1;
        }
        else if(appreciationUE5 = 'B'){
            coeffMajoration = 1;
        }
        return coeffMajoration *
        noteUE4;
    }
}
```

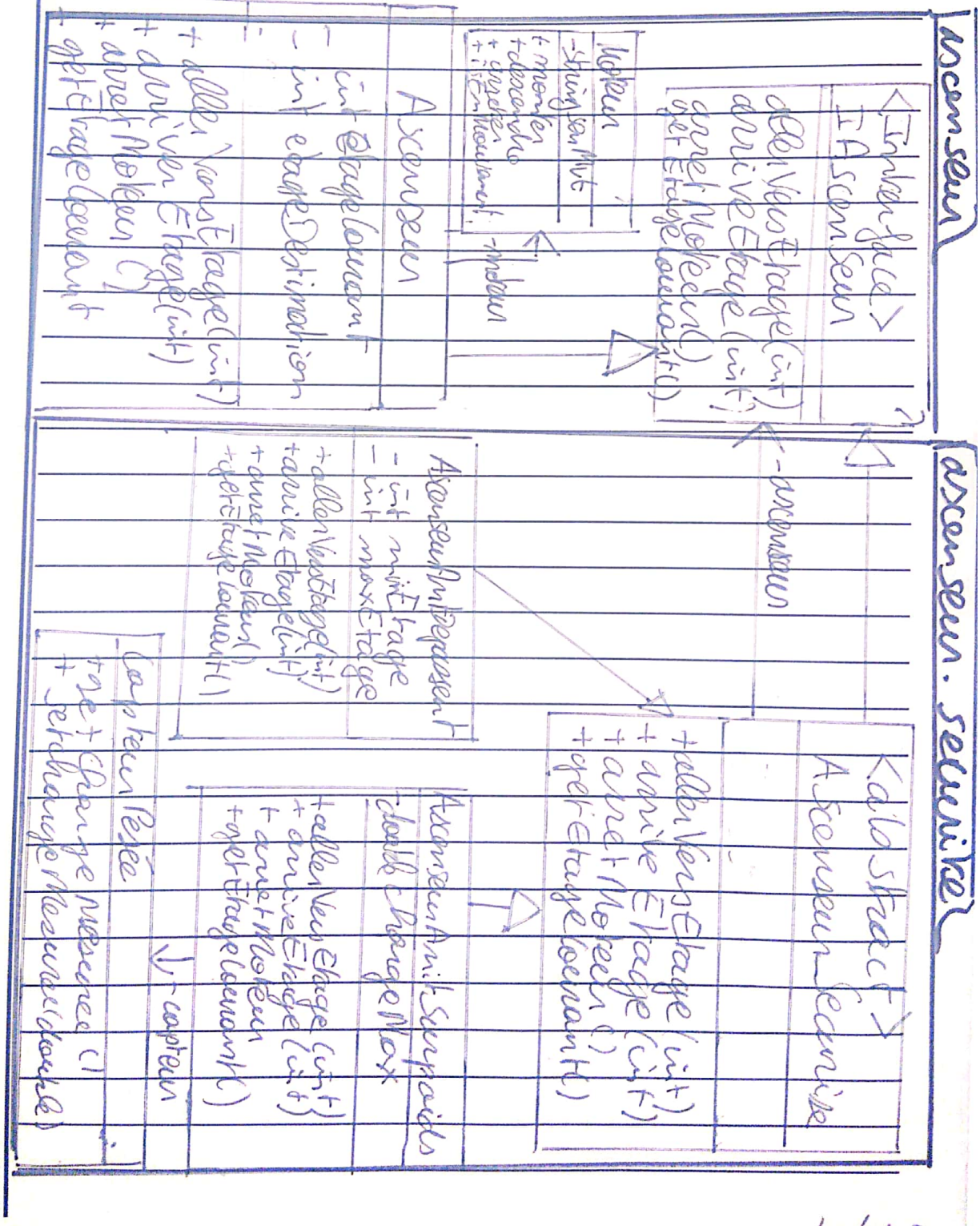
3/15

Scanned by CamScanner

Exercice 2

0.5 1) Le patron permettant de mettre en place la sécurité des ascenseurs sans modifier la classe initiale est le patron Decorateur.

2)



1

4/15

Scanned by CamScanner

UNIVERSITÉ PARIS DESCARTES

IUT

DÉPARTEMENT INFORMATIQUE

DISCIPLINE CPOA

Date de l'épreuve 16/01/19

Année 2 Groupe 205

NOM LAPOSTOLET (en capitales) Écrire très lisiblement

Prénom Axène

NOTE DE 0 À 20

APPRÉCIATIONS

Ne rien écrire dans cette marge

```
package ascenseur.securite

public abstract class AscenseurSecurisé
implements IAscenseur {

    private IAscenseur ascenseur;

    public AscenseurSecurisé(IAscenseur
    ascenseur) {
        this.ascenseur = ascenseur;
    }

    @Override
    public void allerVersEtage(int
    numEtage ) {
        this.ascenseur.allerVersEtage
        (numEtage);
    }

    @Override
    public void arriveeEtage(int
```

5/15

Scanned by CamScanner

```
numEtageCourant){
    this.ascenseur.arriveeEtage
    (numEtageCourant);
}

@Override
public void arreterMoteur(){
    this.ascenseur.arreterMoteur();
}
@Override
public int getEtageCourant{return ascenseur.
  getEtatCourant();}
}

package ascenseur.securite;
public class AscenseurAntiDepassement
extends AscenseurSecurise {

    private int minEtage;
    private int maxEtage;

    public AscenseurAntiDepassement
    (IAscenseur a, int minEtage,
    int maxEtage){
        super(a);
        this.minEtage = minEtage;
        this.maxEtage = maxEtage;
    }

    @Override
    public void allerVersEtage
    (int numEtage){
        if(this.getEtageCourant ==
          this.minEtage ||
          this.getEtageCourant ==
```

6/15

Scanned by CamScanner

```
this. maxEtage) {
        super. arreterMoteur();
    }
    else {
        super. allerVersEtage(numEtage);
    }

}
```

```
package ascenseur. securité
public class AscenseurAntiSurpoids
extens AscenseurSecurisé {
    private double chargeMax;
    private CapteurPesee capteur;

    public AscenseurAntiSurpoids(
    IAscenseur a, CapteurPesee c,
    double chargeMax) {
        super(a);
        this. capteur = c;
        this. chargeMax = chargeMax;
    }

    @Override
    public void allerVersEtage (int
    numEtage) {
        if( capteur. getChargeMesuree
            < this. chargeMax) {
            super. allerVersEtage
            (numEtage);
        }
    }
}
```

else peut être arrêter moteur?

7 /15

Scanned by CamScanner

4)

```
package ascenseur.securite

public class TestAscenseurSecurite

    @Test
    public void test() {
        Moteur m = new Moteur();
        IAscenseur a = new Ascenseur
        (0, m);
        a = new AscenseurAntiDepassement
        (a, 0, 10);
        a.allerVersEtage(8);
        /*Capteur defectueux on appelle
        pas arriveeEtage(8) */
        a.arriveeEtage(10) // capteur infaillible
        assertFalse(m.isMoteurEnMouvement);

    }

        CapteurPesee c = new CapteurPesee
        a = new AscenseurAntiSurpoids(
        a, new CapteurPesee(), 300d);

    @Test
    public void TestPoids() {
        Moteur m = new Moteur();
        IAscenseur a = new Ascenseur
        (0, m);
        CapteurPesee c = new CapteurPesee()
        a = new AscenseurAntiSurpoids
        (a, c, 300d);
        c.setChargeMesure(300d);
        a.allerVersEtage(1);
        assertFalse(m.isMoteurEn
        Mouvement());
    }
}
```

pas clair

0,5

pas assez détaillé

8/15

Scanned by CamScanner

Exercice 3 5

```
package université
public interface IEtudiant extends
 IUsager {
      int getAnnée();
}

package université
public interface IEnseignant
 extends IUsager {
      String getMatiere()
}

package université
public interface IUsager {
      String getNom();
}
```

9/45

Scanned by CamScanner

```
package Université
public class Université {
    private Map<String, IUsagers> usagers
    = new HashMap<String, IUsagers>();

    public void ajouter(IUsagers contact){
        usagers.put(contact.getNom(),
        contact);
    }

    public String consulterExamens
    (String nom, String details){
        IEtudiant e;
        if (usagers.containsKey(nom)){
            try {
                e = (IEtudiant) usagers.
                get(nom);
                return nom + " est
                convoqué aux examens
                de l'année " + e.getAnnee()
                + " " + details;
            } catch (IllegalCastException e){
                throw new IllegalArgument
                Exception(nom +
                " n'est pas étudiant);
            }
            else {
                throw new IllegalArgument
                Exception(nom + " n'est pas
                étudiant);
            }
    }
```

il vaut mieux faire une liste d'étudiants et une d'enseignants

en utilisant des interfaces IEtudiant et IEnseignant votre solution se différencie assez bien fondamentalement avec des avantages et inconvénients

l'idée d'IsP est quand même de faire des interfaces spécifiques donc autant les utiliser ça simplifie le code ensuite en évitant de vérifier sans arrêt le type des usagers.

séparer étudiants et enseignants évite ces tests une fois pour toute

10/15

Scanned by CamScanner

```
public String surveillerExamen
(String nom, String detail){
    IEnseignant e;
    if(usagers.containsKey(nom)){
        try {
            e = (IEnseignant) usagers.
            get(nom);
            return nom+" doit
            surveiller l'examen de "
            + e.getMatiere()+" " +
            details;
        } catch (IllegalCastException e){
            throw new IllegalArgumentException
            (nom+" n'est pas un
            enseignant");
        }
        else {
            throw new IllegalArgument
            Exception(nom+ "n'est
            pas un enseignant");
        }
    }
```

OK acceptable mais pas idéal

```
package usagers
public class Enseignant implements
IEnseignant{
    private String nom;
    private String matiere;

    public Enseignant(String
    nom, String matiere){
        this.nom = nom;
        this.matiere = matiere;
}
```

11/15

Scanned by CamScanner

```
@Override
public String getNom(){
    return nom;
}

@Override
public String getMatiere(){
    return matiere;
}

}

package usagers
public class Etudiant implements
IEtudiant {
    private String nom;
    private int année;

    public Etudiant(String nom,
    int année){
        this.nom = nom;
        this.année = année;
    }

    @Override
    public String getNom(){
        return nom;
    }

    @Override
    public int getAnnée(){
        return année
    }
}
```

Les Tests ne changent pas.

12/15

Scanned by CamScanner

NOTE DE 0 À 20	APPRÉCIATIONS
	il faut cacher les classes qui représentent un polygone de la géométrie (chose pas fait) ? !

Ne rien écrire dans cette marge

Exercice 4

pas clair 0.5

1) Il faut cacher l'implémentation des polygones (et donc la calcule de leur surface qui leur est propre) à la classe géométrie.

0

1

2)

package geometrie.polygones
public interface IPolygones {
    double getSurface();
}

non !! (ne sont pas)

?

package geometrie.polygones
public class Rectangle implements IPolygones {
    private double largeur;
    private double hauteur;
    public Rectangle (double largeur, double hauteur) {
        this.largeur = largeur;
        this.hauteur = hauteur;
    }

13/15

Scanned by CamScanner

```
@Override
public double getSurface() {
    return longueur * largeur;
}

}

package géométrie
public class Géométrie {
    private List<IPolygone> polygones = new
      ArrayList<IPolygone>();

    public Géométrie(IPolygone p){
        ajouter(p);
    }

    public void ajouter(IPolygone p){
        polygones.add(p);
    }

    public double SurfaceTotale(){
        double s = 0d;
        for (Polygone p : polygones){
            st = p.getSurface();
        }
        return s;
    }

package géométrie.polygones
public class GéométrieTest {
    @Test
    void Test() {
        Géométrie = new Géométrie
        (new Rectangle(4, 2));
```

+0.5

oui

14/15

Scanned by CamScanner

```
Rectangle r = new Rectangle
(3, 3);

assertEquals(8d, e.surfaceTotale,
0,001);
e.agrandir(r);
assertEquals(17d, e.surfaceTotale(),
0,0017);
}
}
```

Scanned by CamScanner