Exercices POO

Préambule

1°) Soient les fichiers Java suivants :

fichier : Class2.java

```
package test;
class Class1 {
    int a;
}

public class Class2 {
    int b;
    private int c;
    public int d;
    protected int e;
}
```

fichier Class4.java

```
import test.*;
public class Class4 extends Class2 {
}

class Class5 {
}
```

fichier Class3.java

```
package test;
public class Class3 extends Class2 {
    protected int f;
}
```

fichier Main.java

```
import test.*;
public class Main {
    public static void main(String [] args) {
        …
    }
}
```

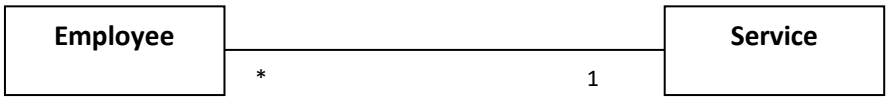
Indiquer pour chacun des champs a, b, c, d, e et f quelles sont les classes des fichiers précédents qui peuvent accéder dans leurs méthodes, au champ en question (sans passer par des accesseurs).

	Classe1	**Classe2**	**Classe3**	**Classe4**	**Classe5**
a					
b					
c					
d					
e					
f					

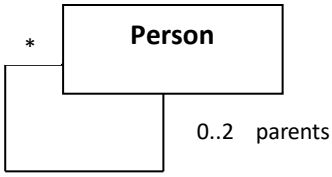
Remarque : extends sert à construire une classe dérivée, il est donc par exemple nécessaire de comprendre que la classe Class4 est une classe dérivée de la classe Class2.

2°) Construire rapidement les classes correspondant aux diagrammes suivants :

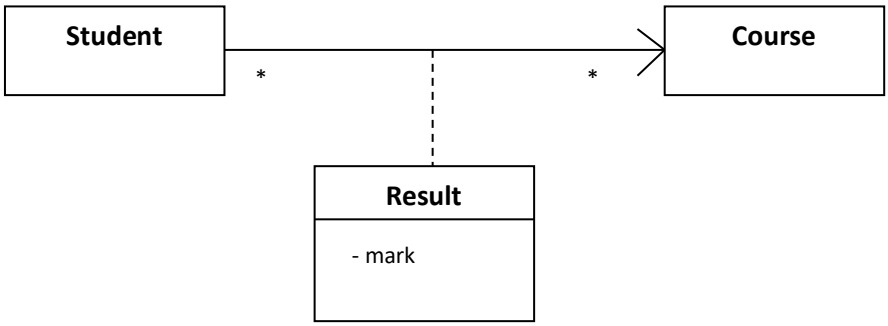
cas a:



cas b:



cas c:



Exercice 1

a) Ecrire la classe De qui doit regrouper les attributs et les méthodes élémentaires permettant de gérer un dé :

- créer un dé (à sa création le dé a une valeur),
- lancer d'un dé et stockage de sa valeur.
  Nb: la classe Random de java.util offre un générateur aléatoire intéressant..
- fournir la valeur du dé

Pour tester cette classe, écrire un programme qui devra :

- créer un dé,
- effectuer un lancer,
- afficher le résultat du lancer,

b) Maintenant que la classe De est créée, il est possible de donner un peu plus d'interactivité au programme de test. Ecrire un programme qui permet de lancer un dé plusieurs fois. Afficher le résultat après chaque lancer. Le nombre de lancer est passé en paramètre de la ligne de commande.

c) Ecrire un programme qui lance deux dés plusieurs fois. Le nombre de lancers est passé en paramètre de la ligne de commande. Le résultat des dés est affiché à chaque lancer; indiquer si le joueur a gagné ou perdu (il gagne si la valeur des deux dés est la même).

Exercice 2 (classes simples avec agrégation)

On souhaite construire une classe de gestion, nommée Application, permettant de gérer et de manipuler des tableaux d'objets Mark, supposés représenter les résultats obtenus par les étudiants d'une classe de l'ESGI, et dont la deuxième dimension est variable. Une représentation graphique de tels tableaux pourrait être :

Etudiant 1	Mark (moyenne matière 1)	Mark (moyenne matière 2)		
Etudiant 2	Mark (moyenne matière 1)	Mark (moyenne matière 2)	Mark (moyenne matière 3)	Mark (moyenne matière 4)
…	Mark (moyenne matière 1)			

Chaque ligne fournit donc les résultats d'un étudiant, chaque colonne la moyenne qu'il a obtenu dans une matière (comme il peut y avoir des absences, le nombre de résultats par étudiant est variable).

Un extrait de la classe Mark est fourni ci-dessous :

```
public class Mark {
        private float value; // désigne la moyenne de la note pour la matière
        private String subject; // désigne le nom de la matière

        public Mark() {
        }

        public Mark(float value, String subject) {
            if (value>=0 && value<=20)
                this.value = value;
            this. subject = subject;
        }

        public float getValue() {
            return value;
        }

        public void setValue(float value) {
            if (value>=0 && value<=20)
            this.value = value;
        }

        public String getSubject () {
            return subject;
        }

        public void setSubject (String subject) {
            this. subject = subject;
        }
        ...
}
```

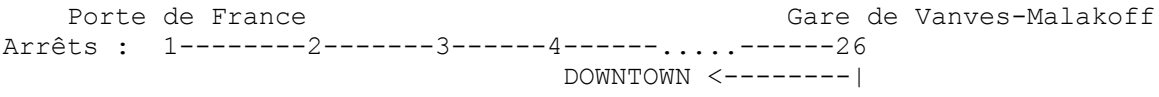
a) Construire la classe Application complète (y compris toString, equals, clone etc ...) et rajouter les fonctions suivantes :

- affichage du tableau (public void print(),
- calcul et retour de la moyenne d'un étudiant de rang donné (public float average(int position),
- rajout d'une note à un étudiant de rang donné (public void add(int position, String subject, float value),
- calcul du plus grand nombre de matières évaluées (public int compute(), à comprendre au sens de "taille max de la deuxième dimension")
- construction d'un tableau des moyennes de tous les étudiants (public float [] avgs(); le tableau devra être ordonné.

b) Cette partie de l'exercice est indépendante de la partie précédente. En utilisant HashSet et Mark, prouver que ne pas définir hashCode peut avoir des conséquences dramatiques en programmation.

Exercice 3 (application objet complète)

L'idée de cette application est de modéliser le trajet d'un bus parisien. Celui-ci exécute toujours le même trajet, depuis un point de départ jusqu'à une certaine destination (par exemple, le bus 89 voyage de la porte de France à la gare de Vanves-Malakoff), en . On supposera pour simplifier que les différents arrêts qu'il effectue sont numérotés de 1 au nombre d'arrêts total. Quand le bus arrive à destination, il effectue le même trajet en sens inverse (là-aussi pour simplifier).



A chaque arrêt, des voyageurs peuvent monter ou descendre du bus, dans une certaine mesure : il y a une capacité maximale à respecter (si elle est atteinte, personne ne peut monter) et il ne peut y avoir moins de 0 voyageurs dans le bus).

On suppose donc qu'un bus est identifié :

- par un numéro de ligne,
- par deux arrêts (départ, destination),
- par une capacité maximale,
- par un nombre de voyageurs,
- par un sens,
- par un nombre d'arrêts,
- par un numéro d'arrêt en cours

a) Construire complètement la classe Bus, en y intégrant, outre les méthodes habituelles, les méthodes suivantes :
   - public void addTravellers(int number) qui permet de faire monter des passagers dans le bus, dans la limite de sa capacité
   - public void removeTravellers(int number) qui permet de faire descendre des passagers dans le bus, dans la limite de sa capacité
   - public void travel() qui permet de faire avancer le bus d'un arrêt

b) Construire une classe de test , possédant un menu, permettant de créer un bus, de le faire avancer d'un arrêt, de faire monter ou descendre des voyageurs.

c) Une amélioration : rajouter un système permettant de gérer les noms des différents arrêts.

Exercice 4 (application objet complète)

Un professeur de mathématiques souhaite initier ses étudiants aux joies de l'arithmétique et décide de leur raconter une petite astuce. Il demande à un de ses étudiants d'écrire sur un papier son âge et l'âge d'un de ses parents (moins de 100 ans). Il lui demande ensuite de montrer le papier au délégué de la classe, qui doit le lire sans dire un mot et calculer le nombre résultat suivant (évidemment toujours sans dire un mot) :

- il doit multiplier l'âge par 2
- ajouter 5 au résultat
- multiplier le résultat par 50
- ajouter l'âge du parent
- enlever 365

Une fois ce calcul effectué, le délégué doit dire le résultat à la classe. Le professeur ajoute alors rapidement 115 au résultat, et annonce l'âge de l'étudiant et celui de son parent (résultat/100 pour l'âge, résultat%100 pour l'âge du parent).

Construire ce cas de la manière suivante :

- définir un package nommé com.esgi.math
- construire 5 classes :
    - une classe application (Application) dont voici le contenu du main :
      Student student = new Student();
      Representative representative = new Representative();
      Teacher teacher = new Teacher();
      student.init();
      teacher.compute(representative,student);
    - une classe Papier (Paper)
    - une classe Etudiant (Student), qui possède un objet Papier
    - une classe Délégué (Representative)
    - et une classe Professeur (Teacher)

Exemple d'exécution
étudiant choisi
quel âge : 22
quel âge parent : 42
délégué arrive
professeur : début du tour
étudiant écrit papier
étudiant montre papier
délégué lit papier
délégué calcule
délégué annonce résultat ...
professeur annonce âge étudiant et âge parent : 22 et 42
avec par exemple :

```
public static void main(String[] args) {
        Student student = new Student();
        Representative representative = new Representative();
        Teacher teacher = new Teacher();
        student.init();
        teacher.compute(representative,student);
        …
}
```