UEF4.3. Programmation Orientée Objet

Contrôle Intermédiaire

(2h- Documents et téléphones interdits)

Questions (4 points)

1) Quelles sont les différences entre les exceptions contrôlées et les exceptions non contrôlées ? Donnez des exemples.
2) Citez les différents types de classes internes en précisant pour chaque type les droits d'accès aux membres de sa classe englobante.
3) Dans quel(s) cas un constructeur sans arguments dans une classe dérivée lève-t-il une erreur ?

Exercice 1 (4 points)

Qu'affichent les programmes suivants ? Donnez la bonne réponse. S'il y a une (des) erreur(s) de compilation ou d'exécution, expliquez la(les) cause(s) puis donnez la correction et le résultat de l'exécution après correction.

A/

```
class A {
public void f(){
System.out.print(" A:f "); }
public void f (int n) {
System.out.print(" A:int:"+n); }
public void f (int n, int q) {
System.out.print(" A:2int:"+q); }
public void f (int n, double y) {
System.out.print(" A:int-
double:"+y);}
}
public class B extends A{
public void f () {
System.out.print(" B:f"); }
public void f (int n) {
System.out.print(" B:int:"+n); }
public void f (int n, int q) {
final int j=1;
```

```
class I{
public int f(int i){
System.out.print("
I:int:"+(i+j));
return (i+j);}
}
I i = new I();
System.out.print
(" B:2int:"+(q+i.f(n)));
}
public static void main(String
args[]){
A a = new A(); a.f(); a.f(0);
a=new B(); a.f(); a.f(1);
a.f(1,1); a.f(3, 4.5);
}
}
```

Ce code :

a. Comporte une erreur qui sera signalée à la compilation
b. Affiche un résultat que vous devez donner
c. Ce code une erreur à l'exécution

B/

```
Package prog1;
public class A{
protected int i=0;
void f(int i) {
System.out.print(i);}
}
```

```
package prog2 ;
import prog1.A;
public class B extends A{
final int i=1;
public void f() {
System.out.print(i+" " +super.i);
super.f(this.i) ;}
public static void main(String
args[]){
A a = new B(); a.f(2018); }
}
```

Ce code:

a. Affiche 2018
b. Affiche 1 0
c. Affiche 0
d. Comporte une erreur qui sera signalée à la compilation

Exercice 2 (5 points)

A. Donner le code java des classes et interfaces suivantes :

1) L'interface `AugmentationSalaire` possède une méthode `double augmenterSalaire(double salaire, double pourcentage)` qui augmente par défaut le salaire d'un certain pourcentage passé en argument et qui lance une exception de type `AugmentationImpossibleException` si le nouveau salaire dépasse un plafond de 200000 DA
2) La classe `Fonctionnaire` implémente l'interface `AugmentationSalaire` et comporte:
   - Trois attributs: nom, prénom et salaire visibles par les classes dérivées
   - Un constructeur exhaustif (qui initialise tous ses attributs)
   - Une surdéfinition de la méthode `equals` de façon à ce que deux fonctionnaires soient considérés égaux s'ils ont le même nom et le même prénom.
3) La classe `ChefDeService` hérite de la classe `Fonctionnaire` et possède:
   - Un attribut `Service` visible seulement par les classes se trouvant dans le même package
   - Un constructeur exhaustif.
   - Une redéfinition de la méthode `augmenterSalaire()` de façon à l'augmenter de 15%.
4) La classe `Stagiaire` hérite de la classe `Fonctionnaire` et possède:
   - Un attribut `dureeDuStage` visible partout
   - Un constructeur exhaustif
   - Une redéfinition de la méthode `augmenterSalaire()` qui lance une exception de type `AugmentationImpossibleException` car un stagiaire ne peut pas bénéficier d'une augmentation.

B. Etant donné qu'un stagiaire n'a pas droit à une augmentation de salaire, comment pourrions-nous corriger la conception proposée ?

Exercice 3 (7 points)

Nous désirons écrire un programme java permettant de vérifier à l'aide d'une pile si une expression arithmétique est correctement parenthésée.

1. Soit la classe `Pile` en figure 1 ci-dessous implémentant une pile en java. Compléter les lignes 5, 9, 13, 14,15 et 19.
2. Ecrire une classe java nommée `ExpressionParenthesee` ayant comme attribut une chaine de caractères représentant une expression arithmétique et nommée `expression`. Pour initialiser ce champ, on utilise un constructeur qui reçoit en entrée une chaine de caractères et utilise une pile pour vérifier si elle est correctement parenthésée. Si l'expression est erronée, le constructeur de la classe `ExpressionParenthesee` doit lancer une exception correspondant au type de l'erreur qui peut être due à une parenthèse ouvrante manquante ou une parenthèse fermante manquante comme le montrent les exemples suivants:

a+b	correcte
(a*b)+ (c*d)	correcte
(a+b)*c))	Parenthèse ouvrante manquante
)a+b(	Parenthèse ouvrante manquante
((a+b)*c)	Parenthèse fermante manquante

N.B: Nous considérons qu'en dehors des parenthèses, le reste de l'expression est correct.

3. Ecrire une classe nommée `Exercice3` qui reçoit une chaine de caractères en argument de la ligne de commande et vérifie si elle est correctement parenthésée à l'aide de la classe `ExpressionParenthesee`. Vous pouvez vous aider de la méthode `charAt (int i)` de la classe `String` qui retourne le caractère qui se trouve à la $i^{ème}$ position en sachant que 0 est l'indice du premier caractère.

```
Import java.util.*;

class Pile {
    private Deque<Character>pile=new .....<Character>(); //ligne 5

    //  empile x au sommet de la pile
    public void empiler(char x){
        ..............................                    //ligne 9
    }
/*retourne l'élément qui est au sommet de la pile en le supprimant de la pile.
Elle lance une exception si la pile est vide*/

    public char depiler() ..............................{      //ligne 13
        if (this.estVide()) ..................................  //ligne 14
        return ..................                              //ligne 15
    }
//retourne true si la pile est vide et false dans le cas contraire
    public boolean estVide() {
        ..............................                    //ligne 19
    }
}
class PileVideException extends Exception{}
```

figure 1: code java des classe Pile et PileVideException