Contrôle Intermédiaire

2h - Documents & Téléphone interdits

Exercice 1(6pts):

A./ Qu'affiche le programme suivant ?

```
class One {
 protected static int x = 5;
 protected int it;
 One(int it) {
  this.it = it; }
protected void addIt(int n) {
  it += n;}
protected void setIt(int it) {
  this.it = it;}
protected int getIt() {
  return it; }
public void display() {
System.out.println("class One: it="
+ it + " x=" + x); }
}
class Two extends One {
 public double y;
 private int it;
 Two(int x, int it) {
    super(x);
    this.it = it; }
protected void addIt(int n){
 it = it + super.it + n;
 n++;}
protected void doubleIt(int n) {
 addIt(2 * n);}
protected void setIt(int it) {
  this.it = it;}
protected int getIt() {
  return it;}
public void display() {
  super.display();
  System.out.println("class Two:
it=" + it + " y=" + y);}
}
public class Ex1 {
 static void f(int p, One a){
  a.setIt(p + One.x);
}
static void f(float x, Two b) {
        b.y = x + b.getIt();
    }
public static void main(String
args[]) {
    One a = new One(1);
    Two b = new Two(2, 3);
    int n = 10;
    float x = 6;
    a.addIt(One.x);
System.out.println("it= " +
a.getIt() + " n= " + n);
    a = b;
    a.addIt(n);
System.out.println("it= " +
a.getIt() + " n= " + n);
    b.doubleIt(Two.x);
System.out.println("it= " +
a.getIt() + " n= " + n);
     One.x = n;
     f(n, a);
     f(x, b);
     a.display();
     b.display();
     }
}
```

B./ Pour chacune des modifications suivantes, dites si le programme compile ou pas. S'il ne compile pas expliquez pourquoi, sinon indiquez qu'affiche le programme.

1.
```
final class One extends Enum
{….//même code question A
```

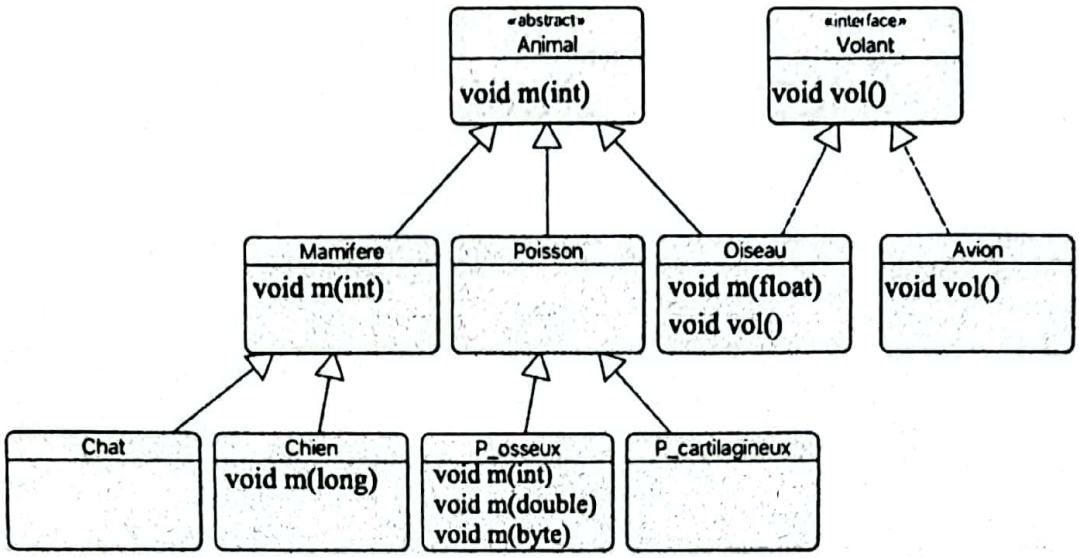
2.
```
class Two extends One {
   public final double y=9;
….//même code question A
```

3.
```
class Two extends One {
….//même code question A
protected void addIt(int n) {
 final int z=n;
 it = it + super.it + z;
 n++; }
….//même code question A
```

4.
```
public class Ex1 {
….//même code question A
static void f(double x, Two b)
throws Exception {
   if (x > 4)
    throw new Exception();
     b.y = x + b.getIt();
  }
public static void main(String
args[]) {
  …//même code question A
 f(n, a);
 try { f(x, b);}
 catch (Exception e){
System.out.println("x>20");
 } finally {
System.out.println("bloc finally");
 }
  …//même code question A
    }
}
```

Exercice 2 (8pts).

Soit la hiérarchie de classes illustrée dans la figure ci-après:



Soit les déclarations suivantes:

```
Animal a1=new Chien(); Animal a2= new Chat();
Volant v1,v2;  Mamifere m=new Mamifere(); Oiseau o=new Oiseau();
Avion av=new Avion();Chat c1=new Chat(); Chien c2=new Chien();
Poisson p=new Poisson(); P_osseux p1=new P_osseux();
P_cartilagineux p2= new P_cartilagineux();
```

Questions :

A/ Dans la suite d'instructions suivantes, indiquez pour chacune d'elles :

1) si elle est correcte ou si elle provoque une erreur à la compilation, en l'expliquant.

2) si on peut résoudre le problème de la compilation, et si oui, s' il reste une erreur à l'exécution ou non.

Instruction	Compilation (ok ou erreur avec explication)	Correction si possible (cast ou autre)	Exécution (OK ou erreur avec explication)
a1= new Animal ();			
v1= new Oiseau();			
v2= new Volant();			
p= new P_cartilagineux();			
m= new P_osseux();			
m= a2;			
a1 = v1;			
a1= m;			
p1=p;			
p2=(P_cartilagineux)p;			
av=(Avion)v1;			

B/ Nous ajoutons les instructions suivantes :

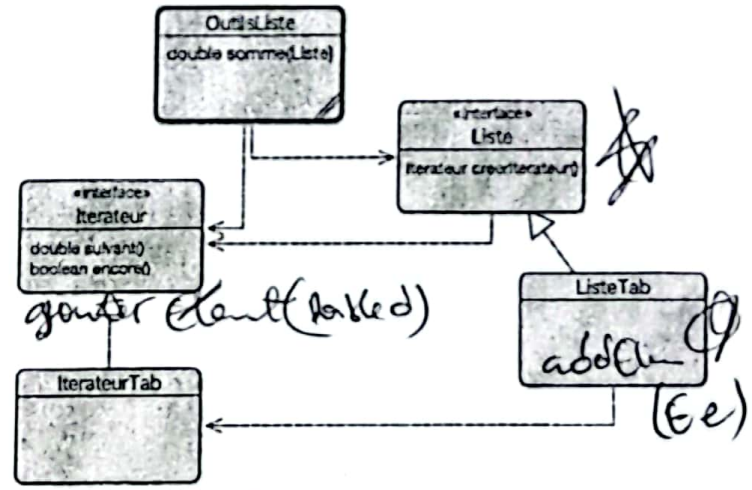
byte b=1; int n=3; long y=5; float x=6; double z=7;

En considérant les déclarations initiales, ainsi que ces variables, indiquez parmi les instructions suivantes celles qui provoquent une erreur à la compilation (en l'expliquant) et précisez pour les instructions correctes la méthode qui sera appelée.

Instruction	**OK/méthode appelée ou *Erreur* avec explication**
o.m(n);	
c1.m(y);	
c2.m(y);	
v1= new Oiseau(); v1.vol();	
v1=av; v1.vol()	
a1=p1; a1.m(z);	
((Poisson)p1).m(b);	
o.m(x);	
a2=o; ((Oiseau)a2).m(x);	

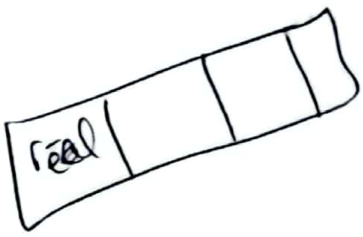
Exercice 3 (6pts)

Soit le diagramme de classe suivant.



La classe OutilsListe permet de générer la somme des éléments d'une liste de réel. Cette liste peut être implémentée sous forme de tableau ou une autre structure de données (exp. liste chainée, file, pile, etc.).
Voici le code de la méthode *somme*:

```
public double somme(Liste liste) {
  Iterateur iterateur = liste.creerIterateur();
  double som=0;
  while (iterateur.encore()) {
    double a = iterateur.suivant();
   som+=a;
  }
return som;
}
```



1. a. Pourquoi a-t-on choisi que les classes *Liste* et *Iterateur* soient des Interfaces et non pas des classes abstraites. Expliquez.

b. La classe IterateurTab doit-elle redéfinir ou surdéfinir la méthode *suivant*? Expliquez.

2. Complétez le code suivant

```
public class ListeTab implements .... {   //1
...[] elements;                            //2
public Iterateur creerIterateur() {
  return new ...(elements); }              //3
 // autres méthodes non demandées
}

public class IterateurTab implements .... {  //4
 Double[] elements;
 int position = 0;
public  IterateurTab(Double[] elements) {
 this.elements = elements; }

public .... suivant() {                      //5
  double a = elements[position];
  position = position + 1;
  return a;
}
public boolean encore() {
  if( (position >= elements.length) ||(
elements[position] == null))  return ...;    //6
  else    return ...;          //7
 }
}
```

3. Nous ajoutons à l'interface *Iterateur* la méthode *ajouterElement(Double d)* qui ajoute un élément au tableau *elements*.

   a. Donnez son implémentation dans la classe *IterateurTab* de sorte à déclencher l'exception *ArrayIndexOutOfBoundsException* si la taille du tableau d'éléments est atteinte.

   b. Compléter le code suivant de la méthode *addElem* de la classe *listeTab*

```
public void addElem(Double d){
      IterateurTab it=.....;          //1
      while(it.encore()){
           it.suivant();
      }
      .....{                          //2
        it.ajouterElement(.....);     //3
        }
      catch(..... e){                 //4
        System.out.println("fin du tableau");
        }
}
```

4. Nous souhaitons que l'interface *Liste* soit une liste d'objets quelconque. Pour cela, nous utilisons le concept de généricité (*La notion de généricité permet de définir des modules paramétrés par le type qu'ils manipulent;* comme vu pour l'interface Comparable):

- Nous modifions l'entête de l'interface pour pouvoir préciser le type d'éléments dans les implémentations: public interface Liste<E>
- Nous modifions le type de paramètre dans la méthode *addElem* pour qu'il soit précisé dans les implémentations *addElem(E e)*

Expliquez quelles modifications doit-on apporter à :

a. l'interface *Iterateur* pour qu'elle soit également générique?
b. la classe *ListeTab* et *IterateurTab* suite à ces modifications?