POO-IG
Programmation Orientée Objet et Interfaces Graphiques
Examen de session 1

9 Janvier 2018

Durée : 2h

Documents autorisés : trois feuilles A4 recto-verso manuscrites ou imprimées. Portables interdits. Le barème est donné à titre indicatif.

IMPORTANT : Dans le questions à choix multiples, plusieurs options peuvent être des réponses valides. Cependant vous devez **entourer au plus une option**. Si plusieurs options sont entourées, la question sera considérée sans réponse.

Question 1 (1 point)

Soit le fragment de code suivant :

```
class Ville {
  private String nom; private double latitude; private double longitude;
  public Ville (String nom, double lat, double lon)
  { this.nom = nom; this.latitude = lat; this.longitude = lon;}
  public double distance (Ville v) {...} //renvoie le nombre de Km entre this et v
  public String getNom () {return nom;}
  public void setNom (String nom) {this.nom = nom;}
  public double getLatitude() {return latitude;}
  public void setLatitude (double l) {latitude = l;}
  public double getLongitude() {return longitude;}
  public void setLongitude (double l) {longitude = l;}
}
class SegmentCourt {
  private Ville depart; private Ville arrivee;
  public SegmentCourt (Ville d, Ville a) {
    if (d.distance(a) > 1000) throw new RuntimeException();
    this.depart = d; this.arrivee = a;
  }
}
```

Et l'utilisation suivante de la classe `SegmentCourt`

```
Ville v1 = new Ville("Paris", 48.85, 2.35);
Ville v2 = new Ville("Berlin", 52.52, 13.40);
SegmentCourt s = new SegmentCourt (v1,  v2);
```

Laquelle des options suivantes viole le principe d'encapsulation de la classe `SegmentCourt` ?

Réponses possibles :

a. `String n = v1.getNom()`,

b. `v1.setNom("Lutece")`,

c. `v2.setLatitude (52.53)`,

d. aucune des options.

Question 2 (1 point)

Etant donnée la définition de classe suivante

```
public class Q {
  int f (int n){
    int a = x;
    for (int x = 1; x < n; x++) { a = a * x; }
    a = a * x;
    return a;
  }
  private int x;
  Q (int x) { this.x = x;}
}
```

Etant données deux variables entières `m` et `n` initialisées, et les instructions suivantes :

```
  Q q = new Q(m); System.out.println (q.f(n));
```

Quelle est la valeur affichée ?

Réponses possibles :

a. Rien. Erreur à la compilation de la classe `Q`

b. $n!\ m$

c. m^{n+1}

d. $(n-1)!\ m^2$

Question 3 (1 point)

Etant donnée la classe `Ville` de la Question 1, laquelle des définitions suivantes est correcte ?

Réponses possibles :

a.
```
class VilleDansRegion extends Ville {
 static final String region;
 VilleDansRegion ( String nom, double lat, double lon, String r )
   { super(nom, lat, lon); region = r; }
}
```

b.
```
class VilleDansRegion extends Ville {
 static final String region;
 {region = "Ile de France";}
 VilleDansRegion ( String nom, double lat, double lon )
   { super(nom, lat, lon); }
}
```

c.
```
class VilleDansRegion extends Ville {
 static final String region = "Ile de France";
 VilleDansRegion ( String nom, double lat, double lon, String r )
   { region = r; super(nom, lat, lon); }
}
```

d. Aucune

Question 4 (3 points)

Etant donné le fragment de code ci-dessous :

```
interface Taggable<T>  {
  void tag(T t);
  T getTag();
}

class Elem implements Taggable<Elem> {...}
class Obj implements Taggable<Obj> {...}
class RedElem extends Elem {...}

class Q {
  static  <XXXX> void idTag (List<T> l) {
    for (T e : l) e.tag (e);
  }
  public static void main (String args[]) {
    ArrayList<Elem> a = new ArrayList<Elem> ();
    ArrayList<RedElem> b = new ArrayList<RedElem> ();
    ArrayList<Obj> c = new ArrayList<Obj> ();
        ...// initialisation de a, b et c
    idTag (a); idTag (b); idTag (c);
  }
}
```

Donner la déclaration du type T qui doit remplacer XXXX dans la méthode `idTag` (donner la réponse dans l'espace ci-dessous)

XXXX :

Question 5 (3 points)

Etant donné le fragment de code ci-dessous :

```
interface Membre <S, R> {
  R action (S s);
}
public class Q {
  public static <XXXX> List<T> apply (List<T> l, S s) {
    List<T> lr = new ArrayList<T> ();
    for ( T elem : l ) lr.add (elem.action (s));
    return lr;
  }
}
```

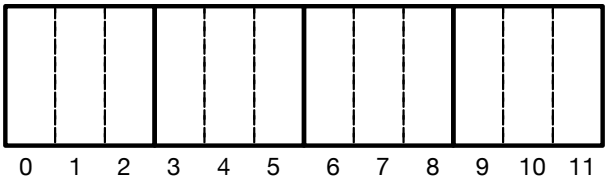
Donner la déclaration de type qui doit remplacer XXXX dans la méthode `apply` (donner la réponse dans l'espace ci-dessous)

XXXX :

Question 6 (12 points)

Le but de cette question est d'écrire **un package `tape`** qui fournit des classes et interfaces pour la simulation (simplifiée) d'une mémoire à accès séquentiel.

FIGURE 1 – Une mémoire comportant 4 blocs de 3 bytes chacun



Le package doit contenir au moins :

1. **Une interface générique** `Moveable` qui intuitivement représente des objets qui peuvent être déplacés le long d'une dimension.
2. **Une exception** `EOTException` (pour *End Of Tape Exception*) de type "checked", qui signale le tentatif de lire en dehors des limites de la mémoire à accès séquentiel.
3. **Une classe** `Tape` qui représente une mémoire à accès séquentiel, avec **une classe interne** `TapePointer` qui décrit des têtes de lecture/écriture.
4. **Une classe** `Q` d'utilité et test.

L'interface `Moveable` doit contenir :
- une méthode `move` censée déplacer l'objet sur laquelle elle est invoquée d'un nombre entier d'unités ;
- une méthode `retrieve` censée retourner un tableau du type variable (intuitivement cela représente des donnés récupérées à la position courante de l'objet).

La classe `Tape` représente une mémoire à accès séquentiel comme un tableau de `Byte` (cf. Figure 1). La mémoire est divisée en blocs consécutifs, tous de la même taille (c'est-à-dire tous comportant le même nombre de cases du tableau).
Une mémoire séquentielle de b blocs se remplit bloc par bloc, de gauche à droite, à partir du bloc 0, jusqu'au bloc b.
En plus de représenter la mémoire et sa subdivision en blocs, la classe `Tape` doit contenir au moins :
- l'indice du dernier bloc utilisé (-1 si aucun bloc n'a été écrit) ;
- une méthode `append` qui écrit un nouveau bloc à la fin de la partie utilisée de la mémoire ;
- une classe interne `TapePointer` qui représente des pointeurs capables de parcourir les blocs de la mémoire ;
- une tête de lecture/écriture `head`, de type `TapePointer`, qui pointe au dernier bloc utilisé (`null` si aucun bloc n'a été écrit) ;
- Une méthode `tapeForwardReader()` qui renvoie un `TapePointer` initialisé sur le bloc 0 (ou `null` si la mémoire est vide).
- Une méthode `tapeReverseReader()` qui renvoie un `TapePointer` initialisé sur le dernier bloc de la mémoire (ou `null` si la mémoire est vide).

La classe interne `TapePointer` implémente l'interface `Moveable` du type approprié, et maintient un pointeur à un bloc (c-à-d l'indice du premier byte du bloc pointé).
En plus des méthodes de l'interface `Moveable`, la classe interne doit fournir au moins une méthode `write`.
- La méthode `move` déplace le pointeur du nombre de blocs passé en paramètre.
- La méthode `retrieve` renvoie le bloc sous le pointeur.
- La méthode `write` écrit sur le bloc pointé un nouveau contenu (de taille correspondante) passé en paramètre.

Contraintes à respecter :

- Au contraire de l'interface `Moveable`, La classe `TapePointer` doit être visible uniquement dans la classe `Tape`.
- Un `TapePointer` peut uniquement pointer à un bloc utilisé (c'est à dire d'indice compris entre 0 et le dernier bloc utilisé). Veiller à vérifier cette contrainte à la fois à la création et à chaque déplacement du pointeur. Toute violation doit soulever une exception `EOTException`.
- La classe `Tape` écrit sur la mémoire uniquement grâce à la méthode `write` de la tête de lecture/écriture `head`.

La classe `Q` doit contenir au moins trois méthodes statiques, 1) `forwardScan`, 2) `backwardScan` et 3) `evenScan` qui affichent sur la console le contenu d'un `Tape` passé en paramètre, respectivement : 1) du début à la fin, 2) de la fin au début, et 3) uniquement les blocs pairs du début à la fin.
Pour parcourir la mémoire, ces méthodes récupèrent et se servent d'un `TapePointer` approprié. De plus elles utilisent l'exception `EOTException` pour détecter que les limites de la mémoire ont été atteints.
La classe `Q` contient enfin une méthode `main` qui initialise le contenu d'un `Tape` avec des appels successifs de `append`, puis teste les trois méthodes statiques ci-dessus.