Examen Final

1 Classes – Héritage

Considérer le programme suivant :

```
1  interface I1{ void f(int p); }
2  interface I2{ void f(); }
3  interface I3{ int f(); }
4  class A{
5          int x;
6          public void f(){ System.out.println("A:f:"+x); }
7  }
8  class B extends A implements I1{ int z;
9          public void f(int p){ System.out.println("B:f:"+x+","+z+","+p);}
10 }
11 class C extends A implements I1,I2{
12         public void f(int p){ System.out.println("C:f:"+x+","+p);}
13         public void f(){ System.out.println("C:f:"+x);}
14 }
15 class D extends C implements I3{
16         public int f(){ System.out.println("D:f:"+x); return 0;}
17 }
18 class Test{
19         public static void t1(A a){ a.f(); }
20         public static void t2(B b){ b.f(); }
21         public static void t3(I1 i){ i.f(); }
22         public static void t4(I2 i){ i.f(); }
23         public static void main(String [] args){
24                 A a = new B();
25                 B b = new C();
26                 B b2 = new A();
27                 I1 ia = a;
28                 I1 ib = new B();
29                 I2 ic = new C();
30                 t1(a);
31                 t1(new B());
32                 t2(a);
33                 t4(ic); } }
```

1. Trouver toutes les erreurs de compilation eventuelles de ce programme. Proposer des corrections.
2. Si on supprime toutes les lignes qui contiennent une erreur de compilation, que va le programme afficher ?

2 Classes Génériques – Exceptions

Considérer le programme suivant :

```
1  class A{
2          void f() { System.out.println("A:f"); }
3          public A() { System.out.println("Constr:A"); }
4  }
5  class B extends A{
6          void f() { System.out.println("B:f"); }
7          void g() { System.out.println("B:g"); }
8  }
9  class G<T> extends A{
10         T t;
11         void f() { System.out.println("G:f:"+t); }
12         public G() { t = new T(); }
13 }
14 class Test{
15         static void t1(A a){ a.f(); }
16         static void t2(B b){ b.g(); }
17         static void t3(G<T> g){ g.f(); }
18         static void t4(G<String> g){ g.f(); }
19         static void t5(A[] tab){ for(A a:tab) a.f(); }
20         static void t6(G<A> ga){ ga.t.f(); }
21         public static void main(String [] args){
22                 t1(new B());
23                 t1(new G<String>());
24                 t2(new G<Integer>());
25                 A ag = new G<String>();
26                 t4(ag);
27                 try{ t5(new B[2]);
28                 }catch(RuntimeException e){ System.out.println("Exc1!");}
29                 try{ t6(new G<B>());
30                 }catch(RuntimeException e){ System.out.println("Exc2!");}
31                 try{
32                         G<B> gb = new G<>();
33                         A ag2 = (A)gb;
34                         t6( (G<A>)ag2 );
35                 }catch(RuntimeException e){ System.out.println("Exc3!");}
36         }
37 }
```

1. Trouver toutes les erreurs eventuelles **et warnings** donnés par le compilateur pour ce programme. Proposer des corrections.
2. Si on supprime toutes les lignes qui contiennent une erreur de compilation (mais pas celles qui donnent un warning), que va le programme afficher ?

3 Expressions Lambda

Considérer le programme (partiel) suivant :

```
1  import java.util.*;
2  import java.util.function.*;
3  class Person{
4    String firstName; String lastName; int age;
5    Person(String f, String l, int a){ firstName = f; lastName = l; age = a; }
6    public String toString(){ return firstName +" "+lastName+" ,"+age; }
7  }
8  class Test{
9    static Person findBest(Person [] p,/*BiPredicate<??> ou Comparator<??>*/ c){
10         /* ... Q1 ... */
11   }
12   public static void main(String [] args){
13     Person [] p = new Person[4];
14     p[0] = new Person("Abe","Lincoln",56);
15     p[1] = new Person("John","Kennedy",32);
16     p[2] = new Person("George","Washongton",44);
17     p[3] = new Person("Franklin","Roosevelt",61);
18     System.out.println("Oldest person is:"+ findBest(p,/* Lambda 1 */));
19     System.out.println("First by last name is:"+ findBest(p,/* Lambda 2 */));
20     System.out.println("First by first name is:"+ findBest(p,/* Lambda 3 */));
21   }
22 }
```

1. Programmer la méthode `findBest`. Cette fonction prend comme premier paramètre un tableau `p` d'objets de la classe `Person`. Elle parcourt le tableau et retourne l'élément "maximum", c'est-à-dire l'élément qui est plus grand que tous les autres. Pour comparer deux objets de type `Person` votre méthode doit utiliser le deuxième paramètre `c` qui implémente une interface fonctionnelle (par exemple, `Comparator` ou `BiPredicate`) qui vous permettra de comparer deux éléments.
2. Remplir les lignes 18-20. Remplacer les commentaires avec des expressions lambda en sorte que le programme affiche respectivement l'objet dont l'age est maximum, l'objet dont le `lastName` est minimum (en ordre alphabetique), et l'objet dont le `firstName` est minimum.

Votre programme doit afficher :

```
Oldest person is:Franklin Roosevelt, 61
First by last name is:John Kennedy, 32
First by first name is:Abe Lincoln, 56
```

NB : Vous pouvez choisir le type du paramètre `c` de la méthode `findBest` comme vous voulez, mais sous la condition que la méthode `main` puisse appeller `findBest` en utilisant des expressions lambda. **Écrivez une seule version de la méthode `findBest`.**

Rappel : pour comparer deux `String` vous pouvez utiliser la méthode `compareTo` (la classe `String` implémente l'interface `Comparable`).

4 Polynômes

Pour cet exercice vous devez programmer une classe `Polynome` qui représente un polynôme. Pour simplifier votre tâche vous pouvez supposer que les coefficients peuvent être représentés par des entiers (`int` ou `Integer`), et que nos polynômes ont une variable unique x. Alors, un objet de la classe `Polynome` doit stocker le coefficient de chaque monôme, en utilisant une structure de données de votre choix (par exemple, un tableau, ou une liste, ou ...).

• Programmer une classe `Polynome` qui implémente au moins les méthodes suivantes :

1. Un constructeur sans arguments, et un constructeur qui prend comme paramètre une liste de `Integer`. La liste contient les coefficients du polynôme qui va être construit en ordre croissant de degré : la position i contient le coefficient du terme x^i.
2. Une méthode `toString`.
3. Une méthode `add` qui prend comme paramètre un deuxième polynôme et retourne un objet qui représente la somme de deux polynômes.
4. Une méthode `mult` qui retourne le produit de deux polynômes.

Exemple : Considérez le programme suivant :

```
1 List<Integer> l1 = new ArrayList<>();
2 l1.add(1); l1.add(2); l1.add(3);
3 Polynome p = new Polynome(l1);
4 System.out.println(p);
5 System.out.println(p.add(p));
6 System.out.println(p.mult(p.add(p)));
```

La ligne 3 initialise un objet qui représente le polynôme $1 + 2x + 3x^2$, puisque la liste donnée comme paramètre est $[1, 2, 3]$. L'affichage du programme doit être :

```
1 + 2x^1 + 3x^2
2 + 4x^1 + 6x^2
2 + 8x^1 + 20x^2 + 24x^3 + 18x^4
```

• Modifiez votre implémentation de la classe `Polynome` pour qu'elle implémente l'interface `UnaryOperator<Integer>`. Le résultat retourné par la méthode `apply` doit être la valeur du polynôme si on remplace x par la valeur donnée. Par exemple, le programme suivant

```
1 List<Integer> l1 = new ArrayList<>();
2 l1.add(1); l1.add(2); l1.add(3);
3 Polynome p = new Polynome(l1);
4 for(int i=0; i<5; i++)
5         System.out.println(p.apply(i));
```

affiche :

```
1
6
17
34
57
```

5 Liste Cyclique et Iterators

Considérez le programme (partiel) suivant, qui implémente une classe de listes chaînées cycliques :

```
1  import java.util.*;
2  class cycleList<T>{
3          T data;
4          cycleList<T> next;
5          public cycleList(T d){ data = d; next = this; }
6          public cycleList(List<T> init){
7                  data = init.get(0);
8                  cycleList<T> tmp = this;
9                  for(int i=1; i<init.size(); i++){
10                         tmp.next = new cycleList<>(init.get(i));
11                         tmp = tmp.next;
12                 }
13                 tmp.next = this;
14         }
15         public String toString(){
16                 String s = data.toString();
17                 cycleList<T> tmp = next;
18                 while(tmp!=this){ s+= " "+tmp.data.toString();
19                         tmp = tmp.next; }
20                 return s;
21         }
22         public Iterator<T> iterator(int step){
23                 /*.. ?? .. */
24         }
25         public static void main(String [] args){
26                 List<Integer> l = new ArrayList<>();
27                 for(int i=0; i<10; i++) l.add(i);
28                 cycleList<Integer> c = new cycleList<>(l);
29                 System.out.println(c);
30                 Iterator<Integer> it = c.iterator(3);
31                 for(int i=0; i<20; i++) System.out.println(it.next());
32         }
33 }
```

Une liste chaînée cyclique est une liste chaînée avec la différence que le dernier pointeur de la liste pointe vers le premier élément. Alors, si on parcourt la liste on arrive encore au début.

• Remplir l'implémentation de la méthode `iterator`. L'objet retourné par cette méthode doit être un Iterator avec les propriétés suivantes : (i) la méthode `next` avance l'iterator `step` positions dans la liste cyclique (ii) la méthode `hasNext` retourne toujours `true` (on n'est jamais au bout d'une liste cyclique!).

Exemple : la boucle de la ligne 31 affiche $0, 3, 6, 9, 2, 5, 8, 1, 4, 7, 0, 3, 6, \ldots$ etc.

Annexe

Ici on a un petit rappel de quelques classes et interfaces de la bibliothèque de Java

```
interface List<T>{
        boolean add(T t);
        void add(int index, T t);
        T get(int index);
        Iterator<T> iterator();
        boolean isEmpty();
        int size();
        T set(int index, T t);
        /*..*/
}
class ArrayList<T> implements List<T>{ /*..*/ }
class LinkedList<T> implements List<T>{ /*..*/ }
interface Iterator<T>{
        boolean hasNext();
        T next();
}
interface Predicate<T>{
        boolean test(T t);
}
interface BiPredicate<T,U>{
        boolean test(T t, U u);
}
interface Comparator<T>{
        int compare(T o1, T o2);
}
interface BinaryOperator<T>{
        T apply(T t1, T t2);
}
interface UnaryOperator<T>{
        T apply(T t);
}
interface Comparable<T>{
        int compareTo(T o);
}
```