Examen Partiel

1 Classes

Considérer le programme suivant :

```
1  class A {
2          private int x;
3          static int y;
4          public A() { this(2); y++; }
5          public A(int x) { this.x = x; }
6          public void f() { System.out.println("A:f:"+x+","+y); }
7          public static void g() { System.out.println("A:g:"+x+","+y); }
8          private void inc() { x++; }
9          private int getY() { return y; }
10 }
11
12 class B{
13         public static void main(String [] args){
14                 A a1 = new A();
15                 A a2 = new A(3);
16                 a1.inc();
17                 a1.f();
18                 a2.f();
19                 A [] at = new A[3];
20                 for(int i=0;i<3;i++)
21                         at[i] = new A();
22                 for(A i:at) i.f();
23         }
24 }
```

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 Héritage

Considérer le programme suivant :

```
1  class A {
2          private int x;
3          static int y;
4          public A() { this(2); y++; }
5          public A(int x) { this.x = x; }
6          A copy() { return new A(x); }
7          public void f() { System.out.println("A:f"+x+","+y); }
8  }
9
10 class B extends A{
11         int z;
12         public B(int z) { super(); this.z = z; }
13         public B copy() { return new B(z); }
14         public void f() { System.out.println("B:f"+x+","+y+","+z); }
15         public void g() { System.out.println("B:g"+z); }
16 }
17
18 class C extends B{
19         int x;
20         public C(int z, int x) { super(z); this.x = x; }
21         C copy() { return new C(z,x); }
22         public void f() { System.out.println("C:f"+x+","+y+","+z); }
23         public int g(int w) { return x+y+z+w; }
24
25         public static void main(String [] args){
26                 A ab = new B();
27                 A ac = new C(6,7);
28                 ac.f();
29                 ac.copy().f();
30                 ac.f();
31                 B bc = new C(8,9);
32                 bc.g();
33                 System.out.println(bc.g(2));
34         }
35 }
```

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 ?

3 Interfaces

Rappel : on a défini les classes `Rational` et `Complex` pour représenter des nombres rationnels et complexes respectivement.

```
1 class Rational{
2         private int num,den;
3         //... Autres methodes/attibuts ...
4 }
5
6 class Complex{
7         private double x,y;
8         //... Autres methodes/attibuts ...
9 }
```

On définit maintenant l'interface `Squareable` comme suit :

```
1 interface Squareable{
2         Squareable newSquare(); //retourne un nouveau objet qui
3                                 //represente le carre de l'objet actuel
4         void Square(); //modifie l'objet actuel x, dont l'etat
5                        //represente desormais x^2
6 }
```

Modifier les définitions des classes `Rational` et `Complex` pour que le programme suivant soit correcte :

```
1         Squareable [] s = new Squareable [2];
2         s[0] = new Rational();
3         s[1] = new Complex();
4         s[0] = s[0].newSquare();
5         s[1].Square();
```

4 Files

Considérer le programme partiel suivant :

```
1  abstract class Fifo {
2          abstract void push(Object o);
3          abstract Object getFirst();
4          abstract Object [] toArray();
5  }
6  class LinkedList {
7          Object data;
8          LinkedList next;
9  }
10 class ListFifo extends Fifo {
11         LinkedList head;
12         /*...*/
13 }
```

La classe abstraite `Fifo` définit une structure First-In-First-Out, c'est-à-dire, une file. La méthode `push` permet d'ajouter un élément. La méthode `getFirst` retire le plus ancien élément (attn : cette méthode modifie l'état de la structure). La méthode `toArray` retourne un tableau avec tous les éléments de la file. (Rappel : On a vu une classe `Stack` représentant une structure LIFO.) **Donner une implémentation concréte** de cette classe, en utilisant des listes chaînées. Pour vous aider, on a déjà défini une classe auxillaire (`LinkedList`) représentant un noeud d'une telle liste. On a aussi commencé la définition d'une class `ListFifo`.

1. Ajouter des implémentations pour les méthodes abstraites.
2. Ajouter une redéfinition de la méthode `toString`, et si necessaire, des constructeurs.

Pour tester votre implémentation, considérez le programme suivant :

```
1          Fifo f = new ListFifo();
2          for(int i=0;i<4;i++) { f.push(i); System.out.println(f); }
3          for(int i=0;i<2;i++) { f.getFirst(); System.out.println(f); }
4          for(int i=0;i<4;i++) { f.push(i); System.out.println(f); }
5          for(int i=0;i<2;i++) { f.getFirst(); System.out.println(f); }
```

Ce programme doit fonctionner correctement et afficher :

```
1  0,
2  0, 1,
3  0, 1, 2,
4  0, 1, 2, 3,
5  1, 2, 3,
6  2, 3,
7  2, 3, 0,
8  2, 3, 0, 1,
9  2, 3, 0, 1, 2,
10 2, 3, 0, 1, 2, 3,
11 3, 0, 1, 2, 3,
12 0, 1, 2, 3,
```