Université Paris Diderot **Programmation Orientée Objet**
L2-L3 **Année 2014-2015**

TD n° 9

Généricité

1 Classe et Interface Génériques

Exercice 1 Voici plusieurs extraits de code. Certains d'entre eux génèrent des erreurs de compilations. Indiquez lesquels en expliquant pourquoi.

1.
```
class <K> A  {
        K a;
        A(K b){ this.a=b; }
}
```

2.
```
class  A  <K>{K a;}
```

3.
```
interface Inter{}
class  Generi  <K,C>{}
class test{
        public static void main(String[] args) {
                Generi<Inter,Integer> b=new Generi<Inter,Integer>();
}       }
```

4.
```
class A <T> { T a; }
public static void main(String[] args) {
        A<int> c=new A<int>();
}
```

`ArrayList` est un type générique défini dans `java.util` qui permet de stocker des éléments. Le type des éléments contenus est un paramètre. Chaque élément stocké possède un indice, cet indice peut aller de zéro à `size()-1`.

La classe `ArrayList` possède les méthodes `T get(int indice)`, `void add(int indice,T element)` et `T remove(int indice)`.

Exercice 2 Écrire une méthode `static` qui prend en entrée un `ArrayList` qui contient des éléments de type `Integer` et qui en retourne la somme.

Exercice 3 On veut écrire une classe `Couple`. La classe `Couple` possède deux attributs `attribut1`, `attribut2`. On initialise ces attributs dans le constructeur. La classe `Couple` possède une méthode `getAttribut1` et une méthode `getAttribut2` qui renvoient respectivement le premier et le second attribut.

1. On veut que les deux attributs d'un `Couple` puissent être n'importe quel type d'objet. Écrivez la classe (sans détailler le corps des méthodes).
2. On veut écrire une classe générale qui, pour n'importe quel type existant `T`, puisse fabriquer un couple d'objets de type `T` et qui n'accepte pas les autres types. Écrivez la classe complètement.

Exercice 4 Voici plusieurs extraits de code, certains d'entre eux génèrent des erreurs de compilations dus à une mauvaise utilisation des types génériques. Indiquez lesquels en expliquant pourquoi.

1.
```
interface Inter{}
class Class1{}
class  Generi  <K>{}
class test{
        public static void main(String[] args) {
```

```
                Generi<Inter> b=new Generi<Inter >();
                Generi<Integer> a=new Generi<Integer >();
                Generi<Generi<Integer>> c= new Generi<Generi<Integer >>();
}       }
```

2.

```
interface Inter<K>{ K method(); }
```

3.

```
class Cla1<K>{
        K a;
        Cla1() {
                a=new K();
}       }
```

4. La classe `Integer` possède la méthode `int intValue`.

```
class Fraction<N>{
        N num;
        N den;
        Fraction(N n,N d) {
                num=n;
                den=d;
        }
        int compareFraction(Fraction<N> f) {
                return num.intValue()*f.den.intValue()-
                f.num.intValue()*den.intValue();
}       }
class test{
        public static void main(String[] args) {
                Fraction <Integer> f=new Fraction<Integer >(3,1);
                Fraction <Integer> g=new Fraction<Integer >(7,5);
                System.out.println(g.compareFraction(f));
}       }
```

5. L'interface `Number` possède la méthode `int intValue` et la classe `Integer` implémente `Number`.

```
class Fraction<N extends Number>{
        N num;
        N den;
        Fraction(N n,N d) {
                num=n;
                den=d;
        }
        int compareFraction(Fraction<N> f) {
                return num.intValue()*f.den.intValue()-
                f.num.intValue()*den.intValue();
}       }
class test{
        public static void main(String[] args) {
                Fraction <Integer> f=new Fraction<Integer >(3,1);
                Fraction <Integer> g=new Fraction<Integer >(7,5);
                System.out.println(g.compareFraction(f));
}       }
```

6.

```
class Generique<N extends Number>{ }
class test{
        public static void main(String[] args) {
                Generique<Object> d= new Generique<Object >();
}       }
```

7.

```
class Cla1{} interface Int1{} interface Int2 {}
class Generique<N extends Cla1&Int1&Int2 >{ }
```

8.

```
class Generique<N> {
        void methode(N a){ System.out.println(a.toString());}
}
```

9.

```
class Classe<K>{
        K[] a;
        Classe(int n) {
                a=new K[n];
}       }
```

10.
```
class Classe<K>{
        K[] a;
        Classe(K[] t) {
                a=t;
}       }
```

2 Classe générique, fonctionnement

La généricité n'intervient que durant la compilation afin de réduire les erreurs de conversions. Par exemple dans la classe `G<A extends TypeExistant>` toute apparition de `A` sera remplacée, après compilation, par `TypeExistant`.

Quelquefois, il est nécessaire de faire, à l'intérieur d'une classe générique, une conversion vers un type paramétré. Ces conversions sont identifiées comme dangereuses par le compilateur (Option : `javac -Xlint`). Cependant on peut spécifier au compilateur de ne pas lancer de message de `Warning`, pour cela il faut spécifier `@SuppressWarnings("unchecked")` avant la méthode qui utilise la conversion.

Exercice 5 On veut écrire une classe (simple) générique `Pile` en java. On utilisera un tableau de taille suffisante pour mémoriser les objets et un attribut de type `int` pour mémoriser le nombre d'éléments stockés. Écrire la classe `Pile`.

3 Référence de type générique

Exercice 6 On suppose que l'on a définit la classe générique `class A <T>{T a;}`. Indiquez, pour chacun des extraits de code suivant, s'il est correcte ou pas.

1.
```
public static void main(String[] args) {
        A<Integer> c=new A<Integer>();
        A<Object> u=c;
}
```

2.
```
public static void main(String[] args) {
        A<Integer> c=new A<Integer>();
        A<?> u=c;
}
```

3.
```
class test{
        public static void main(String[] args) {
                f(new A<Object>());
        }
        public static void f(A<Object> c) {
                c.a=new Object();
}       }
```

4.
```
class test{
        public static void main(String[] args) {
                A<?> c=new A<Object>();
                c.a=new Object();
}       }
```

5.
```
class test{
        public static void main(String[] args) {
                A<Object> c=new A<>();
                c.a=new Object();
                A<?> u=c;
                Object t=u.a;
}       }
```

6.

```
1  class Base { }
2  class Derive extends Base {}
3  class test{
4          public static void main(String[] args) {
5                  A<Derive> c=new A<>();
6                  A<? super Derive> l=c;
7                  l.a= new Derive();
8                  A<? extends Base> u=c;
9                  Base b=u.a;
10 }       }
```

Exercice 7 Soit le code suivant.

```
1  class Base{}
2  class Derive extends Base{}
3  class G<T extends Base,U>{public T a; public U b;}
```

Voici plusieurs types de références sur le type `G`. Le type `G` est défini juste au dessus.

1. Certains ne peuvent exister, dites lesquelles.
2. Des conversions sont autorisées entre les types restants. Quelles sont-elles ? Donnez-les sous forme d'un diagramme.

Voici les types :

`G<Object,Object>`	`G<Object,Base>`
`G<Base,Object>`	`G<Derive,Object>`
`G<? extends Object ,? extends Object>`	`G<? extends Object ,? extends Base>`
`G<?,?>`	`G<? extends Derive,? extends Object>`
`G<? extends Base ,? extends Object>`	`G<? extends Base ,? extends Derive>`
`G<? super Object ,? super Object>`	`G<? super Object ,? super Base>`
`G<? super Base ,? super Object>`	`G<? super Base ,? super Derive>`

4 Méthode générique. (Si vous avez le temps...)

On peut aussi paramétrer les types d'une méthode. Les mêmes mécanismes de compilation sont utilisés que pour les classes. Lorsqu'on invoque une méthode, on est pas obligé de fournir des arguments de type, le compilateur se débrouille.

Exercice 8 Écrire une méthode statique qui retourne le plus grand élément d'une liste d'éléments comparables. `List<T>` et `Comparable<T>` sont des interfaces de `java.util`. `List<T>` implémente `T get(int indice)` et `size()` ; `Comparable<T>` implémente `int compareTo(T o)`.

5 Perfectionnement

Exercice 9 Répondez aux questions suivantes.

1 - Pour un type générique `G<T>` que signifie `G a=G();` ?
2 - Quelle est la valeur de `(new G<String>).getClass()==(new G<Integer>).getClass()` ?
3 - Peut-on écrire `a instanceof G<String>` ?
4 - Peut-on créer une classe générique qui étend une classe non-générique ?
5 - Peut-on étendre une classe générique par une classe non-générique ?
6 - Peut-on étendre la classe `Throwable` par une classe générique ?
7 - Peut-on utiliser `catch` pour attraper un objet d'un type paramétré ?
8 - Peut-on lancer (sans créer) une exception d'un type paramètre ?
9 - Est-ce `G<Sring> a` et `G<Integer> a` engendre des signatures différentes dans une méthode ?
10 - Peut-on écrire `class G<Integer extends String> {}` ?
11 - Si `GD<T>` dérive de `GB<T,A>` quels conversions implicites cela engendre ?
12 - Peut-on utiliser un attribut static d'un paramètre de type dans une classe générique ?
13 - On peut préciser `? super Type` ou `? extends Type` mais quelle syntaxe permet de combiner les deux ?