Plan

Généricité

HMIN215 - Master 1 IPS

Motivation

Forme très simplifiée d'une interface de liste

`Object` est le type des objets stockés dans la liste
comme paramètre de l'opération `add` par exemple.

```
public interface List
{
    void add(Object element);
    Object get(int index);
    int size();
}
```

Motivation

Problèmes posés

- faire une liste de rectangles → contrôler le paramètre de add
  pour protéger `maliste.add(element)`
  `if (element instanceof Rectangle)` test de type
- récupération d'un `element` par un `get` : objet de type statique `Object`
  après `element = maliste.get()`
  `((Rectangle) element).aire()` typecast, coercition

```
public interface List
{
    void add(Object element);
    Object get(int index);
    int size();
}
```

Paramétrage d'une classe ou d'une interface

Classe ou interface générique

- Indiquer un type formel (`E`) derrière le nom de l'interface.
- Remplacer `Object` par `E` partout dans l'interface.

```
public interface List<E>
{
    void add(E element);
    E  get(int index);
    int size();
}
```

Paramétrage d'une classe ou d'une interface

Classe ou interface générique

- <E> **paramètre de généricité**
- on décrit une liste d'éléments de type E, ce type étant précisé plus tard
- E est utilisé dans l'interface (ou la classe) comme si c'était un type ordinaire, connu, même si c'est en réalité un paramètre formel.

```
public interface List<E>
{
    void add(E element);
    E  get(int index);
    int size();
}

public class ArrayList<E>
       implements List<E>{ // vue simplifiée
...
```

Paramétrage d'une classe ou d'une interface

Classe ou interface générique

Créer une liste, se fait par *instanciation* ou *invocation*
Déclarer la liste : `List<E>`
Créer une liste par instanciation de la classe `ArrayList<E>`.

```
public interface Rectangle{ ...}

List<Rectangle> listeRectangles; // invocation de List<E>
listeRectangles = new ArrayList<Rectangle>(); // ArrayList<E>

// ou à partir de Java 1.7

listeRectangles = new ArrayList<>(); // invocation de ArrayList<E>
```

La classe paramétrée Paire (deux paramètres de généricité)

```
1  public class Paire<A,B>
2  {
3          private A fst;
4          private B snd;
5          public Paire(){}
6          public Paire(A f, B s) {fst=f; snd=s;}
7          public A getFst() {return fst;}
8          public B getSnd() {return snd;}
9          public void setFst(A a) {fst=a;}
10         public void setSnd(B b) {snd=b;}
11         public String toString() {return getFst()+''-''+getSnd();}
12 }
```

Instanciation/Invocation

```
Paire<Integer,String> p = new Paire<>(9,"plus grand chiffre");
Integer i=p.getFst(); // pas de typecast !
String s=p.getSnd(); // pas de typecast !
System.out.println(p);
```

Mais pas de paramétrage par un type primitif, on ne peut écrire :

```
Paire<int,String> = new Paire<>(9,new String("neuf"));
```

Le paramétrage des classes

Paramétrage des méthodes d'instance

Cas standard

paramétrage par les paramètres de la classe

```
1  public class Paire<A,B>
2  {
3          private A fst;
4          ...
5          public A getFst() {
6                  return fst;
7          }
8          public void setFst(A a) {
9                  fst=a;
10         }
11         ...
12 }
```

Le paramétrage des classes

Paramétrage des méthodes d'instance

Si besoin

paramétrage par des paramètres supplémentaires

Comparaison des deux premières composantes de deux paires : la deuxième composante n'est pas forcément de même type.

```
1 public class Paire<A,B>
2 {
3         ...
4         public <C> boolean memeFst(Paire<A,C> p) {
5                 return p.getFst()==this.getFst();
6         }
7         ...
8 }
```

Le paramétrage des classes

Paramétrage des méthodes de classe (`static`)

paramétrage obligatoire

```
1  public class Paire<A,B>
2  {
3          ...
4          public static<X,Y> void copieFstTab
5                                  (Paire<X,Y> p,  X[] tableau, int i) {
6
7                  if (i >=0 && i <tableau.length)
8                                    tableau[i]=p.getFst();
9
10         }
11         ...
12 }
```

Le paramétrage des classes

Paramétrage des méthodes
(une utilisation)

```
1 Paire<Integer,String> p5 = new Paire<>(9,"plus grand chiffre");
2
3 Integer[] tab=new Integer[2];
4
5 Paire.copieFstTab(p5,tab,0);
6
7 Paire<Integer,Integer> p2 = new Paire<>(9,10);
8
9 System.out.println(p5.memeFst(p2));
```

Combinaisons de dérivations et d'instanciations

- Classe générique dérivée d'une classe non générique

```
class Graphe{}
class GrapheEtiquete<TypeEtiq> extends Graphe{}
```

- Classe générique dérivée d'une classe générique

```
class TableHash<TK,TV> extends Dictionnaire<TK,TV>{}
```

- Classe dérivée d'une instanciation d'une classe générique

```
class Agenda extends Dictionnaire<Date,String>{}
```

- Classe dérivée d'une instanciation partielle d'une classe générique

```
class Agenda<TypeEvt> extends
                    Dictionnaire<Date,TypeEvt>{}
```

Quelques exemples dans l'API des collections

- public interface Collection<E> extends Iterable<E>
- public class Vector<E> extends AbstractList<E>
- public class HashMap<K,V> extends AbstractMap<K,V>

  K - type des clefs (Keys)
  V - type des valeurs (Values)

Mariage généricité / héritage

Sous-typage des classes pour un paramètre fixé

Stack<String>
est un sous-type de
Vector<String>

```
Vector<String> pi = new Stack<String>();
```

Mariage généricité / héritage

Pas de sous-typage basé sur celui des paramètres

String sous type d'Object
Stack<String> n'est pas un sous type de Stack<Object>

Pourquoi ?

Certaines opérations admises sur une Stack<Object>, telles que `push(Object o)`, ne seraient pas correctes pour une `Pile<String>` (sauf si les types sont immuables), donc on n'autorise pas la partie barrée ci-dessous, qui serait incohérente :

Stack<Object> pi = ~~new Stack<String>()~~ ;
~~pi.push(new Integer(9))~~ ;

Synthèse

Contexte

Java est un langage à typage statique : pour retenir un maximum d'erreurs (de types) à la compilation

Vérification des types par la généricité

- Classes et interfaces génériques
- notation <T>
- niveau de paramétrage : attributs et méthodes non static
- paramétrage complémentaire des méthodes (static ou non)