TELECOM SudParis

IP PARIS

Les classes génériques

CSC 3101

Algorithmique et langage de programmation

Gaël Thomas

Pourquoi la programmation générique

- Bien souvent, une classe manipule un objet de type inconnu

  Exemple : un arbre binaire de recherche générique contient des objets dont le type est inconnu à la conception

⇒ oblige le programmeur à sous-typer explicitement :

```
class Bank {
  Tree root; /* contient des Object */
  public Account lookup(String name) {
    return (Account)root.lookup(name);
  } } }
```

⇒ risque d'erreurs à l'exécution, code difficile à lire

La classe générique

- Classe générique = classe paramétrée par une autre classe

```
class Bank {
  Tree<Account> root; /* Tree stocke des Account */
  Account lookup(String name) {
    /* plus besoin de transtyper le résultat ! */
    return root.lookup(name);
  } } }
```

- Avantages
  - Détection des incohérences de types à la compilation
  - Code plus facile à lire (pas de sous-typage explicite)

Déclaration simple (1/2)

- Déclaration simple d'une classe générique

```
/* la classe Bag est paramétrée par E */
/* ⇔ E inconnu ici, il sera connu à l'instanciation */
class Bag<E> {
  private E[]      elements;
  public E         get(int i) { return element[i]; }
}
```

⇒ `Bag<Potion> b = new Bag<Potion>()`
déclare un `Bag` avec `E` valant `Potion`

`b` est un sac à potions, on peut faire : `Potion p = b.get(0);`

Déclaration simple (2/2)

- Possibilité d'utiliser plusieurs paramètres

```
class Tree<E, F> { /* paramétré par E, F */
  private Tree<E, F> left;
  private Tree<E, F> right;
  private E          key;
  private F          value;
  …
}
```

⇒ `Tree<String, Account>` déclare un `Tree` avec `E` valant `String` et `F` valant `Account`

Déclaration avancée

- Utilisation de `extends` si il faut que le type paramètre hérite d'une classe précise

```
class Bag<E extends Item> {
  private E[] elmts;
  public int getWeight() {…
    for(…) tot += elmts[i].getWeight(); …}
}

class Item {
  public int getWeight();
}
```

D

- U
d

Attention

Si un type paramètre doit mettre en œuvre une **interface** précise, on utilise **quand même** `extends`

```
class Bag<E extends Item> {
  private E[] elmts;
  public int getWeight() {…
    for(…) tot += elmts[i].getWeight(); …}
}

interface Item {
  public int getWeight();
}
```

Utilisation

- On utilise une classe générique en spécifiant les paramètres

```
Truc<String> truc = new Truc<String>();
```

- On peut omettre les paramètres (mais pas recommandé)

```
Truc truc = new Truc();
```

(⇔ `Truc<Object> truc = new Truc<Object>();`)

Tableau de classes génériques

- On **ne peut pas** allouer un tableau de classes génériques

```
Truc<String>[] t = new Truc<String>[10];
```

- Pour créer des tableaux de génériques, il faut omettre les paramètres côté `new` :

```
Truc<String>[] t = new Truc[10]; /* autorisé */
t[0] = new Truc<String>("Hello");
```

Notions clés

- Classe générique = classe paramétrée par d'autres types

```
class Truc<X, Y, Z extends Bidule>
```

- Utilisation en spécifiant les paramètres

```
Truc<A, B, C> truc = new Truc<A, B, C>();
```

- On ne peut pas allouer de tableaux de classes génériques en spécifiant les paramètres

  Allocation avec `Truc<A, B, C>[] trucs = new Truc[10];`