Programmation Orienté Objets en JAVA

Daniele Varacca
Departement d'Informatique
Université Paris Est

2014

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

Sous-typage

En java toute classe est aussi un type
Quand on declare class B extends A

- Je peux implémenter des méthodes abstraites
- Je peux redéfinir des méthodes
- Je peur ajouter champs et méthodes
- Attention au constructeurs!
- super: à utiliser pour le constructeurs, et en cas de redéfinition
- B est un sous-type de A

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

Sous-typage

B est un sous-type de A: un objet de type B peut être utilisé quand un objet de type A est attendu

- Je peux affecter un objet de type B a une variable déclarée de type A

```
class B extends A {...}

A objA = new B(...); //c'est ok!
```

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

Sous-typage

B est un sous-type de A: un objet de type B peut être utilisé quand un objet de type A est attendu

- Je peux passer un objet de type B a une fonction qui attend un argument de type A

```
class B extends A {...}
class C {
    void f (A arg) {...}
}

B objB = new B(...);
C objC = new C(...);

objC.f(objB); //c'est ok!
}
```

Sous-typage

B est un sous-type de A: un objet de type B doit pouvoir respecter le *contrat* promis par A

- Un objet de type B doit avoir toutes les méthodes déclarées par A

```
class A {
    int f(int x) {...}
}
class B extends A {...}

A objA = new B(...);

int y = objA.f(5); //c'est bon!
```

Sous-typage et redéfinition

B est un sous-type de A: un objet de type B doit pouvoir respecter le *contrat* promis par A

- cela ne concerne que la signature: B pourrait faire autre chose

```
class A {
    int f(int x) {return x+1;}
}
class B extends A {
    @Override
    int f(int x) {return 25;}

A objA = new B(...);

int y = objA.f(5);
 //c'est bon!
 //Mais c'est quoi?
```

Sous-typage et polymorphisme

Polymorphisme:

- la méthode appelée est celle de la classe utilisée à la création de l'objet
- ne dépend pas du type de la variable

```
class A {
    int f(int x) {return x+1;}
}
class B extends A {
    @Override
    int f(int x) {return 25;}

A objA = new B(...);

int y = objA.f(5);
 //ça fait 25!
```

Sous-typage et redéfinition

B est un sous-type de A: un objet de type B doit pouvoir respecter le *contrat* promis par A

- cela ne concerne que la signature: B pourrait faire autre chose
- Le compilateur garantit le respect de la signature: pour le comportement, c'est au programmeur de faire gaffe

Plusieurs superclasses

```
class Character {
   ...
   String greeting() {...}
}

class Animal {
   ...
   String noise() {...}
}

class Centaur extends Character, Animal
```

Ceci n'est pas possible en Java

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

Plusieurs superclasses

```
class A {
   ...
   int f() {...}
}

class B {
   ...
   int f() {...}
}

class C extends A,B
```

Quelle est la méhtode f qu'on appelle quand on a un objet de la classe C?

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

Plusieurs superclasses

```
abstract class A {
   ...
   abstract int f();
}

abstract class B {
   ...
   abstract int f();
}

class C extends A,B
```

Le problème ne se présente plus.

Interfaces

Une classe abstraite telle que

- aucune méthode n'est implémentée
- il n'y a aucun champ

est une *interface*.

```
interface A {
   void g(int x);
   int f();
}
```

Interfaces

Une interface

- utilise le mot clé interface
- elle n'a que des méthodes abstraites
- il ne faut pas utiliser abstract
- elle n'a ni champs ni constructeurs
- on déclare les sousclasses avec le mot clé implements

```
interface A {
   void g(int x);
   int f();
}

class B implements A {
   ...
   public void g (int x) {
       ...
    }
```

Interfaces

On peut implementer plusieurs interfaces

```
interface Character {
   String greeting();
}

interface Animal {
   String noise();
}

class Centaur implements Character, Animal
```

Interfaces

- Les interfaces, encore plus que les classes abstraites, déclarent un *contrat*.
- Implémenter une interface c'est promettre qu'on respecte le contrat
- Une interface est aussi un type

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

Interfaces

- Les interfaces, encore plus que les classes abstraites, déclarent un *contrat*.
- Implémenter une interface c'est promettre qu'on respecte le contrat
- Une interface est aussi un type

Quand on implémente une méthode d'une classe abstraite celle ci doit être déclarée public...

Types parametriques

On a vu l'interface List.
Elle a un paramètre:

- List<String>
- List<Character>
- List<List<String>>

Une classe qui implémente List est ArrayList
List<String> x = new ArrayList<String>()

Types parametriques

On a vu l'interface List.
Elle a un paramètre:

- List<String>
- List<Character>
- List<List<String>>

Une classe qui implémente List est ArrayList

```
List<String> x = new ArrayList<>()
```

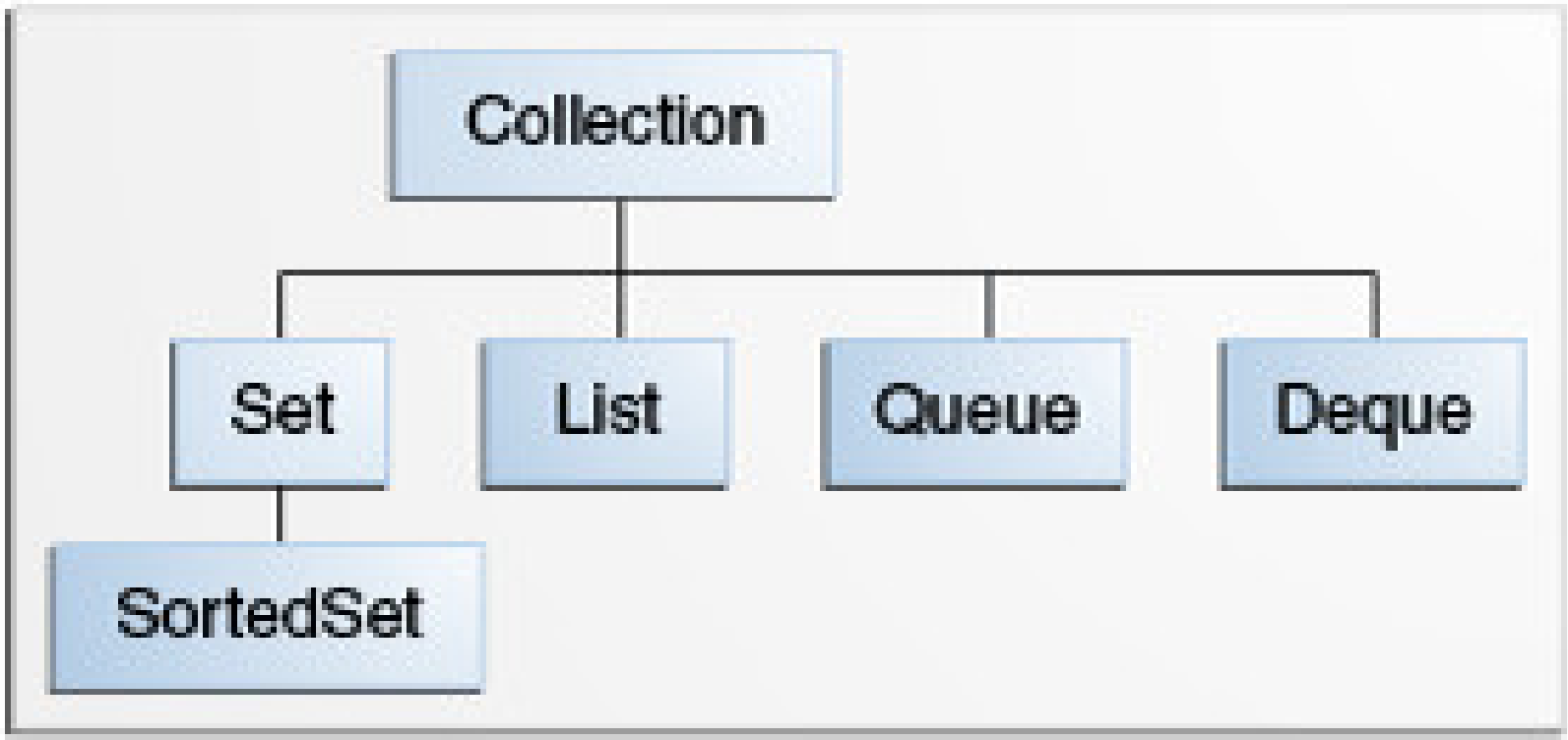
Types paramétriques

Beaucoup d'interfaces sont paramétrées.

- Collections: List<E> et Set<E> (et d'autres)
- Maps: Map<K,V>

En jargon Java on dit qu'il s'agit de types *Génériques*: car il ne peuvent utiliser aucune information particulière sur le type du paramètre.

Les collections



Types Génériques

On peut aussi créer sa propre classe générique.

```
class Pair<E> {
   ...
   E first;
   E second;
   void swap() {
      E temp;
      temp = first;
      first = second;
      second = temp;
   }
}
```

Types Génériques

Les listes de OCAML:

```
abstract class RecursiveList<E>  {
    ...
}
class EmptyList<E> extends RecursiveList<E> {
    ...
}
class NonEmptyList<E> {
    ...
  E head;
   RecursiveList<E> tail;
}
```

Pattern Matching

Comment faire le filtrage?

```
abstract class RecursiveList<E>  {
    ...
    abstract int length();
}
class EmptyList<E> extends RecursiveList<E> {
    ...
    int length() {return 0;}
}
class NonEmptyList<E> {
    ...
  E head;
  RecursiveList<E> tail;
  int length() {return 1+tail.length()}
}
```

Resumé

Concepts importants:

- classes et objets
- classes abstraites et implémentation à la création
- classes concrète qui implémentent classes abstraites
- héritage entre classes concrètes
- polymorphisme
- interfaces
- sous-typage

UPEC
Connaissance - Action
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE