Java Avancé

Sous-Typage

Emerite Neou
emerite.neou@gmail.com

M1 apprentissage

Sous-Typage, Pourquoi ?

- Utiliser un algorithme écrit pour un type général.

Sous-Typage, Pourquoi ?

- Utiliser un algorithme écrit pour un type général.

```
static void runThoughJdbc(Database db, String statement) {/*...*/}

Database db1 = new SQLServer() ;
Database db2 = new Hive() ;
Database db3 = new Cassandra() ;

runThoughJdbc(db1, "UPDATE ...") ;
runThoughJdbc(db2, "UPDATE ...") ;
runThoughJdbc(db3, "UPDATE ...") ;
```

Sous-Typage, Comment ?

- Interface et Heritage.

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un"

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un" ex : un SQLServer est une database, une canette longue est une canette, ...
  - du code en commun.

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un" ex : un SQLServer est une database, une canette longue est une canette, ...
  - du code en commun.
- choisir une interface quand :

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un" ex : un SQLServer est une database, une canette longue est une canette, ...
  - du code en commun.
- choisir une interface quand :
  - décrire un comportement, ie "peut faire"

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un" ex : un SQLServer est une database, une canette longue est une canette, ...
  - du code en commun.
- choisir une interface quand :
  - décrire un comportement, ie "peut faire" ex : une linkedlist peut être itérer, un Integer peut être comparé à un autre ...

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un" ex : un SQLServer est une database, une canette longue est une canette, ...
  - du code en commun.
- choisir une interface quand :
  - décrire un comportement, ie "peut faire" ex : une linkedlist peut être itérer, un Integer peut être comparé à un autre ...
  - aucun rapport entre l'implémentations des classes filles.

Interface vs Heritage

Permettent de faire la même chose mais

- choisir l'héritage quand :
  - relation "est un" ex : un SQLServer est une database, une canette longue est une canette, ...
  - du code en commun.
- choisir une interface quand :
  - décrire un comportement, ie "peut faire" ex : une linkedlist peut être itérer, un Integer peut être comparé à un autre ...
  - aucun rapport entre l'implémentations des classes filles. ex : LinkedList, ArrayList

Interface, Comment ?

- abstract class
- class/class interne

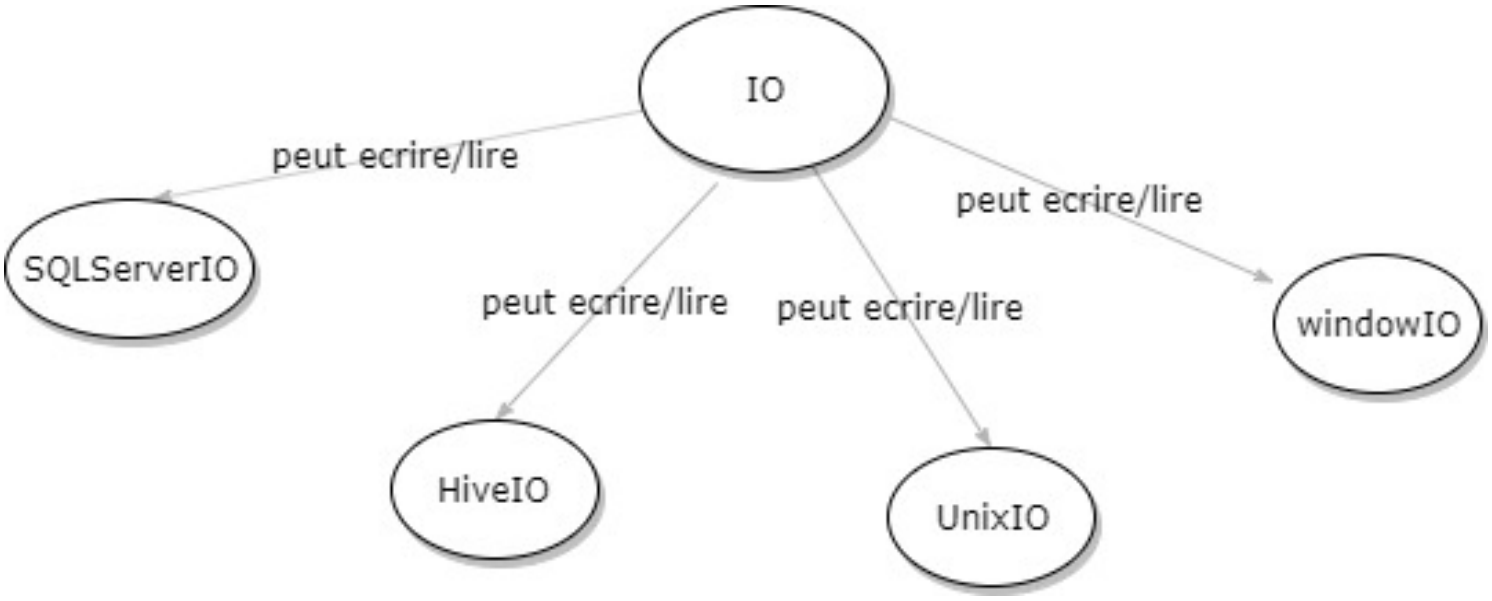
Abstract Class, quand ?

- code en commun entre les classes filles.

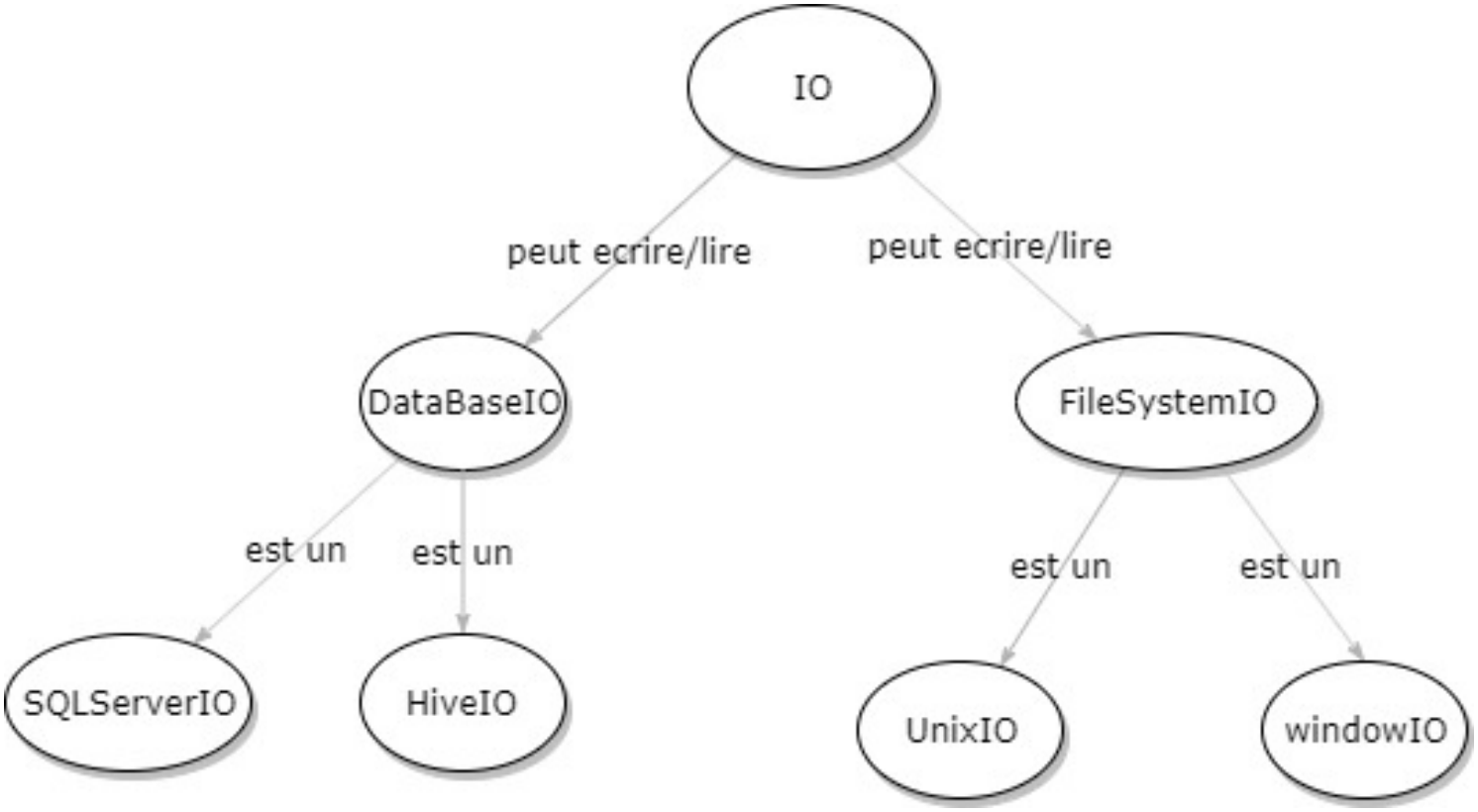
Abstract Class, quand ?

- code en commun entre les classes filles.

Abstract Class, pourquoi ?



Abstract Class, pourquoi ?



class interne, quoi ?

```
public class A {
  public static class B{
  }
}
```

```
public class A {
  public class B {
  }
}
```

```
public class A {
  public void m() {
    class B {
    }
  }
}
```

```
public class A {
  public void m() {
    new Object() {
      ...
    }
  }
}
```

class interne, quand ?

- la classe est unique (pas de code en commun)

class interne, quand ?

- la classe est unique (pas de code en commun)
- seulement instancié dans la class contenante.

class vs class interne

- sucre syntaxique.

class vs class interne

- sucre syntaxique. class interne quand
  - code pas trop long.

Cas spécial des lambdas

si interface fonctionnelle (une seule method dans l'interface et avec @FunctionalInterface)

Cas spécial des lambdas

si interface fonctionnelle (une seule method dans l'interface et avec @FunctionalInterface)

- faire des lambdas

Cas spécial des lambdas

```
@FunctionalInterface
interface Tester {
    public boolean test(String s) ;
}

public static List<String> filtre(List<String> l, Tester t) {...}
```

```
// ne garde que les string de longueur
      sup a 10
filtre(l, new Tester t(){
    @Override
    public boolean test(String s) {
        return s.length() > 10 ;
    }
}) ;
```

```
// ne garde que les string de longueur
      sup a 10
filtre(l, (String s) -> {
    return s.length() > 10 ;
}) ;
```

Cas spécial des lambdas

```
public static List<String> filtre(List<String> l, Tester t) {
    List<String> res = new LinkedList<>() ;
    for (int i = 0  ; i < l.lenght  ; i++) {
        String curr = l.get(i) ;
        if ( t.test(curr ) ) res.append(curr ) ;
    }
    return res ;
}
```