Modularité

Eric Goubault
CEA/Saclay
Eric.Goubault@cea.fr
http://www.di.ens.fr/~goubault

Complément JAVA: les classes

```
class Point {
  int x; int y = 0;
  Point (int x0) { x = x0; }
  void bouge () { x = x + 1; } }
```

La classe Point est constituée de:

- de variables (x), qui peuvent être introduites avec une valeur par défaut.
- de constructeurs (Point), dont le corps est de type `void`; un constructeur est reconnu au fait que son nom est le même que celui de sa classe et que son type de retour n'est pas indiqué.
- de méthodes (bouge) dont le corps est arbitraire.

Les constructeurs

On crée une instance d'une classe avec la construction new suivie d'un constructeur de classe et de ses arguments, par exemple `new Point (3)`.
Le typage ne garantit pas que toutes les variables sont initialisées. Cependant, le compilateur affecte aux variables non initialisées une valeur par défaut du type déclaré.

This

Dans le corps d'une méthode, `this` désigne l'objet en train d'exécuter cette méthode.

```
class Point {
  int x = 0;
  Point () { }
  Point (int x0) { x = x0; }
  void bouge () { x = x + 1; }
  void bouge_bouge () { this.bouge(); this.bouge(); }
}
```

Champs privés/publics

- un champ sans qualificatif est un champ "ami" : autorisation d'accès à toutes les classes du même "paquetage" (voir plus loin)
- un champ `public` est un champ accessible par tous
- un champ `private` est un champ uniquement accessible par les méthodes de la classe en question
- un champ `protected` est un champ accessible par les méthodes de la classe et de ces sous-classes

L'héritage

```
class Bipoint extends Point {  int y;
  Bipoint () { }
  Bipoint (int x0, int y0) { super(x0); y = y0; }
  void bouge () { super.bouge (); y = y + 1; }
}
```

Dans une sous-classe, `super` désigne la classe parente. Un constructeur d'une sous-classe doit faire appel à un constructeur de la classe parente. Cet appel doit être la première instruction. Si ce n'est pas fait, le compilateur ajoute `super()`.

Champs statiques/finaux etc.

- Une méthode ou un champ `static` est unique dans toutes les instances de la classe (exemple: `main` est toujours `static`)
- Une méthode ou un champ `final` est constant dans toutes les instances de la classe

La surcharge

- Les constructeurs de classe et les méthodes sont surchargés.
- Les constructeurs de classes ne sont pas hérités.
- Les méthodes sont héritées. Ainsi, l'ensemble des définitions d'une méthode m dans une classe A est l'ensemble des définitions de m dans la classe A et dans la classe parente.
- Lorsqu'une sous-classe redéfinit une méthode présente dans la classe parente avec le même type, la nouvelle définition cache (override) la précédente. Sinon elle ne fait que la surcharger.

Exercice

```
class C {
  void f() { g(); }
  void g() { System.out.println(1); }

class D extends C {
  void g() { System.out.println(2); }
  public static void main(String[] args) {
    D o = new D();
    o.f();
} }
```


Solution

```
[goubault@cygne13 Tronc-commun]$ javac D.java
[goubault@cygne13 Tronc-commun]$ java D
2
[goubault@cygne13 Tronc-commun]$
```


Classes abstraites (virtuelles)

- Une classe `abstract` est une classe dans laquelle toutes les méthodes ne sont pas implémentées
- On peut hériter d'une telle classe et implémenter les méthodes non-implémenter de la sur-classe, en respectant la signature.

Mais...

- Pas d'héritage multiple en JAVA, donc on ne peut pas hériter de `Thread` et d'une autre classe en même temps,
- On peut néanmoins utiliser les `interfaces` (particulièrement utile quand on parlera de *threads*)

Complément JAVA: Interfaces et implémentations

Les interfaces sont des *spécifications* (un programme pourra en fournir une *implémentation*).

```
interface Printable {
  void print (); }
```

- L'interface `Printable` n'est pas une classe. Elle n'a pas d'implémentation propre. Mais l'interface peut être implémentée (ie. respectée) par différentes classes.
- Une classe peut implémenter plusieurs interfaces.

```
class Printable_point extends Point implements Printable {
  Printable_point () { super(); }
  public void print () { System.out.println(x); } }
```


Utilisation

La déclaration d'une interface I dans la définition d'une classe A:

- effectue des vérifications de typage.
- déclare le type des objets de la classe A un sous-type du type de l'interface I.
- Cela permet par exemple, de voir tous les objets imprimables avec un type commun Printable et de leur envoyer des messages.
- La relation de sous-typage Une classe ne peut hériter que d'une seule classe, mais elle peut simultanément implémenter une ou plusieurs interfaces.
- La relation d'héritage est un arbre, mais la relation de sous-typage est un graphe (treillis)

Le sous-typage

En JAVA, le sous-typage est par nom.

- Une déclaration de classe class A extends B implements B1, ... Bk déclare un nouveau type A, et le déclare sous-type de B et de chaque Bi.
- Une déclaration d'interface interface A extends B1, ... Bk déclare un nouveau type A, et le déclare sous-type de chaque Bi.
- La relation de sous-typage est fermée par reflexivité et transitivité.

Les casts

- Le cast d'un objet change son type statique, mais ne change pas sa représentation (valeur dynamique).
- Un cast se note (A) (v) où A est le type cible et v l'argument.
- Lorsque le type statique de v est un sous-type de A, alors le cast est statique, et il réussit toujours.
- Lorsque le type statique de v est un sur-type de A, alors il est possible que le type dynamique de v soit en fait plus précis et un sous-type de A. Le cast est autorisé, mais il nécessite un test à l'exécution qui peut échouer.
- Tout cast qui peut réussir à l'exécution est permis.

Les conversions

- Un cast (conversions) effectue une conversion de type explicite.
- Certaines conversions de types sont implicites. Par exemple, une affectation d'une valeur à une variable de type A effectue une conversion implicite vers le type A pourvu que la valeur soit un sous-type de B. Les conversions implicites ne peuvent pas échouer à l'exécution.
- Les règles de conversion sont complexes (voir The Java Language Specification pour une description complète...)

Utilisation

- Les conversions statiques, pour changer le comportement d'une programme. En affaiblissant le type statique d'un objet vers le types des objets d'une classe parente, on change la résolution de la surcharge. L'objet se comporte alors comme s'il était de la classe parente.
- Les conversions dynamiques mais sûrs. Pour pallier une faiblesse du système de typage, il y a des situations où le type statique d'un objet est celui d'un classe parente de sa classe réelle. L'utilisateur peut le savoir alors que le compilateur ne sait pas le vérifier statiquement.
- Les tests de classes. Lorsque des objets de plusieurs classes sont mis dans un conteneur, à leur retrait, il faut parfois renforcer leur type à l'aide d'une conversion combinée avec un typecase.

Les packages

- Les packages sont un moyen de grouper plusieurs classes ensembles, en limitant la visibilité de certaines composantes à un package.
- Les composantes des packages sont des classes, des interfaces, ou des sous-packages.
- La fabrication et l'utilisation des packages dépend du système sous-jacent (conventions de nommage, de placement, et d'accès), mais le langage fixe les déclarations correspondantes dans le code source, ainsi que les règles de visibilité.

Utilisation

- Utilisation Pour accéder aux composantes x, y d'un package P on utilise la notation pointée P.x, P.y, etc.
- La déclaration import P.Q.x; rend visible la déclaration x du package P.Q dans l'unité de compilation sans avoir à utiliser la notation P.Q.x.
- La déclaration import P.Q.*; rend visible toutes les déclarations du package P.Q dans l'unité de compilation.
- Fabrication La déclaration package P.Q; rend l'ensemble des déclarations qui suivent comme une partie du package P.Q.
- Tout programme fait partie d'une unité de compilation nommée ou anonyme.

Les règles de visibilité

Classes, champs, méthodes et constructeurs peuvent être:

- public, ie. visible partout;
- non annoté, ie. visible dans le package (nommé ou annomyme) auquel appartient l'unité de compilation.

De plus, champs, méthodes et constructeurs peuvent être:

- private, ie. visible que dans la classe dans laquelle il est défini.
- protected, ie. visible dans le package et dans les sous-classes de C (si la classe est publique).
- La redéfinition dans une sous-classe d'une méthode d'une classe parente doit avoir une visibilité au moins aussi grande.
- Contrainte Une classe A utilisée dans un autre fichier que celui où elle est définie doit être définie dans un fichier du nom A.java.

Compilation et `CLASSPATH`

- `package examples;`

  `class A { ... }`
- `javac -d .  A.java` va produire un sous répertoire du répertoire courant `examples`
- Depuis le répertoire courant (si `CLASSPATH` contient `.`), `java examples.A` exécutera le code
- `java B` cherche B.class dans les répertoires définis par la variable d'environnement `CLASSPATH`. Pensez à la définir correctement, exemple (`csh` ou `tcsh`): `setenv CLASSPATH .:/home/goubault/JAVA`.

Complément JAVA: les exceptions

- Les exceptions sont des objets qui sont créés lors d'erreurs, que l'on peut récupérer et traiter dans une section de code dédiée,
- Quand le programme détecte une situation anormale, il lève une exception, "Throw",
- Qui peut être capturée dans un bloc d'essai "try ... ",
- Et récupérée pour traitement dans un bloc "catch(...) ... ".