IFT1025 été 2010

Miklós Csűrös

18 mai 2010

1 Objets, packages, héritage

Dans le paradigme de programmation orientée objet, on groupe les informations et tâches dans des objets. En Java, chaque objet appartient à une classe. Les classes définissent les *méthodes* par lesquelles on peut accéder à l'information associée avec des objets, ou demander un objet à performer une tâche quelconque. Les classes pour les tableaux (qui sont les objets) sont définies automatiquement, mais pour d'autres objets, il faut écrire le code explicitement.

Notions (IFT1015) :

- ⋆ un objet est l'instance d'une classe, instantiation par `new`
- ⋆ constructeurs
- ⋆ `this`
- ⋆ `null`
- ⋆ variables de l'objet, variables de la classe (variables `static`)
- ⋆ méthodes de l'objet, méthodes de la classe
- ⋆ signature d'une méthode
- ⋆ nom qualifié : `Point.x`

Package. Plusieurs classes sont groupés dans un *package* qui définit l'espace de nom. La classe suivante s'appelle `jeudelavie.Lancer`. La source se trouve dans le fichier `jeudelavie/Lancer.java` (`jeudelavie\Lancer.java` sous Windows), dans le *source path*. La classe compilée est `jeudelavie/Lancer.class` dans le *class path*. Un espace de nom est importé par le mot clé `import`.

```
package jeudelavie;
// importer la classe Random du package java.util
import java.util.Random;
public class Lancer
{
    ...
}
```

Superclasses. En vérité chaque classe est l'extension d'une autre classe, sauf `Object` qui est l'objet de racine. On utilise l'extension d'une classe pour changer l'implémentation.

```
public class Rectangle extends Shape
{
    public double aire()
    {
        // code spécifique à Rectangle
    }
}
```

Règles de visibilité et héritage

Mot clé	Extension	Package	Ailleurs
`private`	non-visible	non-visible	non-visible
[sans spécification]	non-visible	visible	non-visible
`protected`	visible	visible	non-visible
`public`	visible	visible	visible

Des règles analogues s'appliquent à l'héritage de variables et méthodes : celles avec accès `protected` et `public` sont héritées, celles avec accès par défaut sont héritées seulement si la sous-classe appartient au même package, et celles avec accès `private` ne sont pas héritées. Les constructeurs ne sont pas hérités (mais le constructeur sans arguments est là toujours). Accès à la version chez la superclasse par le mot clé `super`.

2 Archives jar

On peut compiler un archive de plusieurs fichiers par le programme `jar`. L'archive résultant, avec extension `.jar` peut être utilisé en Java comme *class path*. Création :

```
% jar -cf Jeudelavie.jar jeudelavie/
```

Utilisation : `java -cp Jeudelavie.jar jeudelavie.Lancer ...`. Extraction : `jar -xf Jeudelavie.jar`.

3 Graphisme

Les packages `java.awt` et `javax.swing` contiennent les classes pour l'interface graphique. Chaque élément graphique est l'extension de la classe `java.awt.Component`. Les éléments de Java Swing sont des sous-classes de `javax.JComponent` (quelques exemples : `JLabel`, `JPanel` (container générique), `JButton`). Un groupe important d'éléments est celui des *containers* : on peut ajouter des éléments à ceux-ci (p.e., y placer un bouton).

JFrame. La classe `javax.swing.JFrame` est un *container* de racine pour les application graphiques. Méthodes importantes :

- ⋆ `setTitle`
- ⋆ `setSize` (la taille du frame doit être spécifiée) et `pack` (calcul automatique de la taille)
- ⋆ `setDefaultCloseOperation`
- ⋆ `setVisible` (le frame est invisible par défaut)
- ⋆ `add`

Graphics. La classe `java.awt.Graphics` donne accès à des méthodes de peinture, dessin et affichage au niveau de pixels sur l'écran graphique. Un `JComponent` est affiché en appellant la méthode

```
public void paintComponent(Graphics g).
```

Pour ses propres éléments graphiques, on écrit une extension d'un `JComponent` avec une nouvelle version de `paintComponent`. Quelques méthodes de la classe `Graphics`

- ⋆ `create` (création d'une copie pour sauvegarder l'état graphique)
- ⋆ `drawLine`
- ⋆ `drawRectangle`,
- ⋆ drawOval
- ⋆ `setColor` (constantes dans la classe `java.awt.Color`)
- ⋆ `drawString`
- ⋆ `setFont` (constantes dans la classe `java.awt.Font`)

Dans Swing, l'argument passé est toujours de type `java.awt.Graphics2D` qui est une extension de `Graphics` avec des méthodes plus avancées (p.e., distortion de l'espace graphique).