Licence MIASHS

INF F5 — Interfaces, paquetages et exceptions

UGA UFR SHS Université Grenoble Alpes

Daniel.Bardou@univ-grenoble-alpes.fr

Sommaire

- Interfaces
  - Notion d'interface
  - Exemples d'interfaces prédéfinies
- Paquetages
  - Notion de paquetage
  - Exemples de paquetages prédéfinis
- Exceptions
  - Motivations
  - Levée d'une exception
  - Traitement d'exceptions
  - Hiérarchie des exceptions prédéfinies
  - Utilisation de **throws**

Interfaces

Notion d'interface

- Une interface Java est la définition d'un type d'objet (comme l'est une classe) limité à une spécification de méthodes (sans en préciser le corps).
- Une interface ne peut détenir que des méthodes abstraites et (plus rarement) des constantes **static**.
- Une interface est destinée à être réalisée (ou « implémentée ») par une ou plusieurs classes.

Interfaces versus classes abstraites

- Dans une interface, toutes les méthodes sont abstraites.
- Une interface ne peut pas détenir d'attributs d'instances.
- Une interface n'a pas de constructeur.
- Une interface peut être implémentée par une ou plusieurs classes.
- Une interface ne peut pas hériter d'une classe.

Héritage et « implémentation »

- Une interface peut hériter d'une ou plusieurs interfaces : l'héritage multiple ne pose pas de problèmes puisque ce qui est hérité est abstrait.
- Une classe peut réaliser (« implémenter ») plusieurs interfaces.
- Une interface peut être implémentée par plusieurs classes, éventuellement dispersées dans la hiérarchie d'héritage.

Syntaxe

- **public interface** Ouvrable {
  public void ouvrir();
  public void fermer();
  public boolean estOuvert();
  }
- **public interface** Entrouvrable **extends** Ouvrable {
  public void entrouvir(**double** angle);
  public double angleOuverture();
  }
- **public interface** Verrouillable {
  public void verrouiller();
  public void deverouiller();
  public boolean estVerouille();
  }

Syntaxe

- **public class** Mairie **extends** ServicePublic **implements** Ouvrable {
  /* obligation de définir ici les méthodes spécifiées dans Ouvrable */
  }
- **public class** Fenetre **implements** Entrouvrable {
  /* obligation de définir ici les méthodes spécifiées dans Entrouvrable (et Ouvrable) */
  }
- **public class** Serrure **implements** Verrouillable {
  /* obligation de définir ici les méthodes spécifiées dans Verrouillable */
  }
- **public class** Porte **implements** Entrouvrable, Verrouillable {
  /* obligation de définir ici les méthodes spécifiées dans Entrouvrable (et Ouvrable) et Verrouillable */
  }

Interface Cloneable

- Cloneable est une des interfaces prédéfinies de Java. Sa définition est très simple :

```
public interface Cloneable {
   }
```

- Cloneable est une interface « balise ».
- La méthode **protected** Object clone() est définie dans la classe Object. Cette méthode lève une exception CloneNotSupportedException si **this** n'est pas du type Cloneable.

Interface Comparable

- Comparable est une autre interface prédéfinie de Java. Sa définition (avant Java 1.5) est :

```
public interface Comparable {
    public int compareTo(Object obj);
}
```

- De nombreuses méthodes prédéfinies en Java utilisent le type **Comparable** (dans leur corps ou pour le typage de leurs paramètres).
- Par exemple, la méthode **public static void** sort(Object[] t) de la classe Arrays réalise un tri par ordre croissant d'un tableau, à condition que tous ses éléments soient des Comparable.

Paquetages

Notion de paquetage

- En Java, les paquetages permettent de « ranger » les classes et les interfaces.
- Un paquetage Java permet de regrouper des classes définies pour opérer ensemble.
- Les paquetages sont aussi des espaces de nommage :
  - plusieurs classes peuvent être définies avec le même nom, mais dans des paquetages différents,
  - le nom complet d'une classe est son « nom court » préfixé de son nom de paquetage (par exemple : `java.lang.Object`).

Sous-paquetage

- Un paquetage peut admettre des sous-paquetages :
  - par exemple, `java.lang` est un sous-paquetage du paquetage `java`.
- Il existe une relation forte entre les paquetages, les classes (et interfaces) et l'espace disque.
  - Pour tout paquetage, il doit exister un répertoire de même nom sur le disque.
  - Le répertoire correspondant à un sous-paquetage doit être un sous-répertoire dans le répertoire correspondant au paquetage englobant.
  - Une classe `C` est définie dans un fichier *C.java* qui doit se trouver dans le répertoire qui correspond à son paquetage.
  - Par exemple, la classe `miashs.fractions.Fraction` doit être définie dans un fichier *Fraction.java* se trouvant dans un répertoire *fractions*, se trouvant dans un répertoire *miashs*.

Anatomie d'un fichier .java

```
package miashs.tm1; // nom du paquetage

   import java.util.Scanner; // importation(s)

   public class Exercice1 {
      // définition de la classe (ou interface)
   }
```

Directives import

- Une directive **import** dans un fichier .java permet d'utiliser les éléments importés en les désignant par leur « nom court ».
  - Par exemple, **import** java.util.Scanner permet d'utiliser la classe Scanner sans la désigner par java.util.Scanner.
- On peut utiliser * pour importer toutes les classes d'un paquetage.
  - Par exemple, **import** java.util.* permet d'utiliser en les désignant par leur nom court la classe Scanner, la classe Arrays, *etc*.

Directives import

- L'utilisation de * ne permet pas d'importer les sous-paquetages.
  - Par exemple, **import** java.util.* importe toutes les classes de java.util mais pas celles du paquetage java.util.regex.
- Il ne peut pas y avoir d'ambiguïté dans les importations (le compilateur veille).
  - Par exemple, **import** java.util.* et **import** java.sql.* nécessite de désigner d'utiliser la classe Date en la désignant par java.util.Date ou java.sql.Date.

Quelques paquetages prédéfinis

- java.lang (importé par défaut)
- java.lang.reflect
- java.util
- java.io
- java.awt
- java.applet
- javax.swing
- …

Exceptions

Motivations

- ```
  public class C {
      private int[] t;
      public C(int[] t) {
          this.t = new int[t.length];
      }
  }
  ```
- ```
  // dans un programme ailleurs
  int[] tab;
  C c = new C(tab);
  ```
- L'endroit où le problème est détecté n'est pas toujours celui où l'erreur est faite. Que faire dans le constructeur de `C` ?

Levée d'exception

- Une levée d'exception se fait en utilisant l'instruction **throw**.
- Exemple :

```
public C(int[] t) {
    if (t == null)
        throw new IllegalArgumentException();
    this.t = new int[t.length];
}
```

- La levée d'exception arrête l'exécution en cours et passe le contrôle au niveau appelant dans la pile d'exécution jusqu'à ce qu'elle soit récupérée.

Traitement d'exceptions

- ```
  int[] tab;
  // initialisation de tab
  for(int i = 0; ; i ++)
      tab[i] += i;
  ```
- Que se passe-t-il ?
- Une ArrayIndexOutOfBoundsException est levée
  ```

Traitement d'exceptions

- ```
  int[] tab;
  // initialisation de tab
  try {
      for(int i = 0; ; i ++)
          tab[i] += i;
  } catch (ArrayIndexOutOfBoundsException e) {
  }
  ```
- Que se passe-t-il ?
- Rien, mais ce n'est pas toujours la meilleure façon de programmer un parcours de tableau.
  ```

Traitement d'exceptions

- ```
  try {
      // instructions
  } catch (Exception e) {
      // traitement de l'exception e
  }
  ```
- Si une exception survient pendant l'exécution des instructions, l'exécution du bloc **try** s'arrête et celle du bloc **catch** commence.
  ```

Traitement d'exceptions

- On peut associer plusieurs **catch** au même **try**.
- ```
  try {
      // instructions
  } catch (Exception1 e) {
      // ...
  } catch (Exception2 e) {
      // ...
  } finally {
      // ...
  }
  ```
- Les types d'exceptions doivent être donnés du plus spécifique au plus général.
- Le bloc **finally** est toujours exécuté.
  ```

Hiérarchie des exceptions

- Throwable
  - Error
    - VirtualMachineError
      - OutOfMemoryError
      - …
    - …
  - Exception
    - RuntimeException
      - ClassCastEcepion
      - NullPointerException
      - ArithmeticException
      - IndexOutOfBoundsException
    - ClassNotFoundException
    - IOException
    - …

Utilisation du throws

- Toutes les `Exception` qui ne sont pas des `RuntimeException` doivent être traitées ou signalées.
- On peut signaler une exception en utilisant le mot réservé **`throws`** dans l'entête d'une méthode.
- Exemple :

```
public void m() throws Exception {
    // ...
}
```

- Toutes les méthodes appelant `m()` doivent :
  - soit signaler une `Exception`,
  - soit faire l'appel de `m()` dans un `try catch` adapté.