digitalent
FORMATION

Le langage Java

Java 5

Sommaire

Introduction

- La version 5 de Java Standard Edition (nom de code : Tiger) propose de nombreuses fonctionnalités en plus.
- Améliorations majeures dans la qualité, la supervision et la gestion, la performance et la faculté de montée en charge, ainsi que la facilité de développement.
- Ce cours se focalise sur les fonctionnalités les plus significatives et les plus attendues du J2SE 5.0

Les génériques

Présentation

- Les Génériques sont très attendus par la communauté Java.
  - Il existait un comité de réflexion sur les génériques depuis plus de 5 ans.
- Les Génériques devraient permettre d'éviter de nombreuses erreurs d'exécution.
- Principe de base : typer les éléments d'une collection.

Exemple sans les génériques

- Il n'est pas possible de spécifier le type d'élément que doit stocker la liste.
- Si un mauvais type est employé lors de la récupération d'un élément de la liste, l'erreur n'est pas détectée en phase de compilation.

```
List voitureList  = new ArrayList();
voitureList.add(new Voiture());
//…
Voiture v1 = (Voiture) voitureList.get(0);
```

Cette méthode était la seule possible avant Java 5.0

Exemple avec les génériques

- La liste est typée lors de sa création.
  - Elle ne peut recevoir qu'un type d'élément donné.
- Il ne faut pas faire de transtypage (cast) lors de la récupération d'un élément.
  - Une éventuelle erreur de type est détectée en phase de compilation.

```
List<Voiture> voitureList  = new ArrayList<Voiture>();
voitureList.add(new Voiture());
//…
Voiture v1 = voitureList.get(0);
```

Exemple de définition

- Démo List
- Démo Map

```
public interface Iterable<T> {
    /**
     * Returns an iterator over elements of type {@code T}.
     *
     * @return an Iterator.
     */
    Iterator<T> iterator();

}
```

Fonctionnalités usuelles

Autoboxing/unboxing (1/4)

- Jusqu'au JDK 1.4, pour insérer un élément de type primitif dans une liste :

```
import java.util.*;


public class TestAutoboxingOld {
	public static void main(String[] args) {
		List liste = new ArrayList();
		int i = 12;
		liste.add(new Integer(i));
	}
}
```

- JDK 5.0 propose l'autoboxing et l'unboxing.
  - L'autoboxing : transformer automatiquement une variable de type primitif en un objet
  - Unboxing : opération inverse

Autoboxing/unboxing (2/4)

- Avec Java 5.0, le compilateur effectue automatiquement la conversion
  - La liste peut être remplie indifféremment par des variables de type primitif (ex:int) ou de type 'wrapper' (ex: Integer').

```
public static List<Integer> buildList() {
                            List<Integer> tempList = new ArrayList<Integer>();
                            int i1 = 1;
                            int i2 = 2;
                            Integer i3 = new Integer(3);
                            tempList.add(i1);
                            tempList.add(i2);
                            tempList.add(i3);

                            return tempList;
              }
```

Exemple de remplissage d'une liste par autoboxing

Autoboxing/unboxing (3/4)

- Un même élément "entier" peut être récupéré sous forme d'un int (type primitif) ou d'un Integer (objet).
  - Aucun cast nécessaire

```
List<Integer> l = new ArrayList<Integer>();
tempList.add(50); // ajout d'un int
tempList.add(new Integer(30))
//…

Integer i1 = tempList.get(0);
int i2 =  tempList.get(0);
```

Autoboxing/unboxing (4/4)

digitalent FORMATION

- Opérations sur des éléments
  - Le 'cast' est fait automatiquement

```
int i = 12;
Integer j = new Integer(15);
int k = i + j;
```

itérations simplifiées (1/2)

- L'itération sur les éléments d'une collection est fastidieuse avec la déclaration d'un objet de type Iterator

```
public static void main(String[] args) {
            List liste = new ArrayList();
            for(int i = 0; i < 10; i++) {
                        liste.add(i);
            }
            for (Iterator iter = liste.iterator();
iter.hasNext(); )
            {
                        System.out.println(iter.next());
            }
}
```

itérations simplifiées (2/2)

digitalent FORMATION

- Dans la nouvelle forme de l'instruction for, l'utilisation d'un Iterator est transparente.

```
public static void main(String[] args) {
            List liste = new ArrayList();
            for(int i = 0; i < 10; i++) {
                        liste.add(i);
            }
            for ( Integer temp:liste )
            {
                        System.out.println( temp );
            }
}
```

Les énumérés (1/3)

- Java 5 propose une nouvelle catégorie d'éléments : les enums
  - Un enum n'est pas une classe
- Écriture simplifiée
  - A la compilation, une classe énumérée implémentant les bonnes pratiques est générée.

Les énumérés (2/3)

```
public enum JourSemaine {
              LUNDI(1), MARDI(2), MERCREDI(3), JEUDI(4),
              VENDREDI(5), SAMEDI(6), DIMANCHE(7);


              private int value;


              private JourSemaine(int value) {
                            this.value = value;
              }


              public int getValue() { return value;    }
}
```

Les énumérés (3/3)

- De très nombreux avantages :
  - La déclaration des valeurs possibles est simplifiée.

```
public static final JourSemaine LUNDI= new JourSemaine(1);
LUNDI(1)
```

  - Les méthodes **equals(...)**et **compareTo(...)** sont automatiquement implémentées
    - Se basent sur l'ordre de déclaration des valeurs possibles :

    Par défaut : MERCREDI > MARDI > LUNDI
  - La méthode **values()** renvoie le tableau des valeurs possibles prises par l'énuméré

Les imports statiques (1/2)

- Jusqu'à la version 1.4 de Java, pour utiliser un membre statique d'une classe, il faut obligatoirement préfixer ce membre par le nom de la classe qui le contient.
  - Par exemple, pour utiliser la constante Pi définie dans la classe java.lang.Math, il est nécessaire d'utiliser Math.PI

```
public class TestStaticImportOld {
            public static void main(String[]
args)        {

System.out.println(Math.PI);
            }
}
```

Les imports statiques (2/2)

- Si une classe est importé statiquement, il n'est plus nécessaire de préciser le type

```
import static java.lang.Math.*;
public class TestStaticImport {
              public static void main(String[] args)
{
                            System.out.println(PI);

              }
}
```

- L'utilisation de l'importation statique s'applique à tous les membres statiques
  - constantes et méthodes statiques de l'élément importé

Attention à ne pas abuser des imports statiques.
Ils peuvent provoquer des conflits d'espaces de nommage.

Nombre variable d'arguments (1/2)

- Cette nouvelle fonctionnalité va permettre de passer un nombre non défini d'arguments d'un même type à une méthode. Ceci va éviter de devoir encapsuler ces données dans une collection.

- Elle implique une nouvelle notation pour préciser la répétition d'un type d'argument. Cette nouvelle notation utilise trois points de suspension : ...

Nombre variable d'arguments (2/2)

digitalent FORMATION

Paramètre traité comme un tableau

```
public class TestVarargs {
            public static void main(String[] args) {
                        int[] valTab = {1,2,3,4};
                        System.out.println("valeur = " + additionner(2,5,6,8,10));
                        System.out.println("valeurTab = " +additionner(valTab) );
            }
            public static int additionner(int... valeurs)
            {
                        int total = 0;
                        for (int val : valeurs) { total += val; }
                        return total;
            }
}
```

Résultat :

valeur = 31
valeurTab = 10

Annotations

Présentation

- Dans les versions précédentes de Java, les métadonnées étaient traitées comme de simples commentaires par le compilateur.
  - Un interpréteur externe lisait les métadonnées.
- Le compilateur de Java 5.0 peut stocker des métadonnées dans des classes.
  - Les métadonnées sont maintenant directement placées dans le code

```
@Override
public String toString()
{ //… }
```

Exemple de métadonnée Java 5.0

Définir / Utiliser une annotation

```
public @interface MonAnnotation {
}

@MonAnnotation
public void crediter(double montant) throws CompteException {
        …
}
```

Annotations existantes

- Le package java.lang définit deux annotations standard pour les méthodes :
  - **@Override** : définit qu'une méthode redéfinit une méthode de la classe mère.
    - Si la méthode 'marquée' ne redéfinit pas une méthode de la classe mère, la compilation échoue.
  - **@Deprecated** : Lance un avertissement si un membre annoté "deprecated" est utilisé.
  - **@SuppressWarnings** : indique au compilateur de ne pas afficher certains warnings.
- Possibilité de créer ses propres annotations

Pour les tags ci-dessus, attention aux majuscules.

Méta-Annotations (1)

- Les méta-annotations sont des annotations destinées à marquer d'autres annotations. Elles appartiennent toutes au package java.lang.annotation.

- Les méta-annotations :
  - **@Documented** : indique à l'outil javadoc que l'annotation doit être présente dans la documentation générée pour tous les éléments marqués.
  - **@Inherit** :indique que l'annotation sera héritée par tous les descendants de l'élément sur lequel elle a été posée. Par défaut, les annotations ne sont pas héritées par les éléments fils.

Méta-Annotations (2)

digitalent FORMATION

- Les méta-annotations (suite) :
  - **@Retention** :indique de quelle manière elle doit être gérée par le compilateur. Elle peut prendre 3 valeurs :
    - **RetentionPolicy.SOURCE** : les annotations ne sont accessibles uniquement dans le fichier source (donc absent des classes compilées .class)
    - **RetentionPolicy.CLASS** (par défaut) : les annotations sont bien enregistrées dans les classes compilées mais ne sont pas prises en compte par la machine virtuelle lors de l⁹exécution.
    - **RetentionPolicy.RUNTIME** : les annotations peuvent être utilisées lors de l⁹exécution

Méta-Annotations (3)

- Les méta-annotations (suite) :
  - **@Target** : permet de limiter le type d⁹éléments sur lesquels l⁹annotation peut être utilisée. Par défaut, une annotation peut être utilisée sur tous les éléments. Valeurs possibles :
    - **ElementType.ANNOTATION_TYPE**
    - **ElementType.CONSTRUCTOR**
    - **ElementType.FIELD**
    - **ElementType.LOCAL_VARIABLE**
    - **ElementType.METHOD**
    - **ElementType.PACKAGE**
    - **ElementType.PARAMETER**
    - **ElementType.TYPE**