4. Avancé

CogniTIC

Building Together Better IT Solutions

Plan du cours

- Chapitre 1 : Les annotations
- Chapitre 2 : Les exceptions
- Chapitre 3 : La généricité
- Chapitre 4 : Les énumérations
- Chapitre 5 : Lecture et écriture de fichiers
- Chapitre 6 : Les interfaces graphiques
- Chapitre 7 : Les threads

Chapitre 1

LES ANNOTATIONS

Les annotations

- Définition :
  - Les annotations sont des « métadonnées », c'est-à-dire un supplément d'information donné au compilateur.
  - Elles sont utilisées à la compilation ou à l'exécution.

- Syntaxe :
  - @Annotation

Les annotations

- Les annotations standards
  - @Override
    - Indique que la méthode annotée est une redéfinition d'une méthode présente dans une superclasse ou dans une interface implémentée. Génère une erreur si cette méthode ne redéfinit aucune méthode existante.
  - @Deprecated
    - Indique que la classe ou la méthode annotée ne devrait plus être utilisée. Génère un avertissement.

Les annotations

- @SuppressWarnings
  - Indique que les avertissements du compilateur ne sont pas indiqués dans la méthode ou la classe annotée.

Les annotations

- Les annotations communes
  - @Generated
    - Indique que ce code a été généré par un outil ou un framework, et pas directement par le développeur. L'attribut obligatoire value permet de préciser l'outil à l'origine de la génération.
  - @Resource/Resources
    - Définit une ressource requise par une classe. Par exemple un composant Java EE de type EJB ou JMS.

Les annotations

- @PostConstruct et @PreDestroy
  - Permettent de désigner des méthodes à exécuter après l'instanciation d'un objet et avant la destruction d'une instance de cet objet. Annotations utiles par exemple en Java EE.

Les annotations

- Les annotations personnalisées
  - Pour créer une annotation personnalisée, il faut créer une « @interface » :

```
public @interface MonAnnotation {
    String valeur1();
    String valeur2();
}
```

```
@MonAnnotation(valeur1 = "val1", valeur2 = "val2")
public void printText(String text) {
    System.out.println(text);
}
```

Chapitre 2

LES EXCEPTIONS

Les exceptions

- Une exception apparaît lorsque le programme rencontre une erreur. Selon le type d'erreur rencontré, le type d'exception sera différent.
- On trouve par exemple :
  - NullPointerException :
    - lorsqu'on cherche à manipuler un objet null
  - ArrayIndexOutOfBoundsException :
    - lorsqu'on tente de retrouver un élément dans un tableau à un index hors-limite

Les exceptions

- Contrairement au C#

(exception vérifiée non vérifiée)

Les exceptions

- On peut « catcher » une exception afin d'empêcher le programme de s'arrêter en cas d'erreur grâce à un Try-Catch :

```
public class Main {

    public static void main(String[] args) {

        int[] tableau = new int[5];

        try {
            int resultat = tableau[6];
            System.out.println(resultat);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Erreur détectée");
        }
        System.out.println("Le programme continue");
    }
}
```

Les exceptions

- On peut faire remonter une exception grâce à un Throws :

```
public int calculAvecErreurPossible(int index) throws ArrayIndexOutOfBoundsException {
    int[] tableau = new int[5];
    return tableau[index];
}
```

- Il faudra alors faire un Try-Catch là où l'erreur sera remontée :

```
try {
    p.calculAvecErreurPossible(10);
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Erreur détectée");
}
System.out.println("Le programme continue");
```

Les exceptions

- On peut également lancer nous-mêmes des exceptions :

```
public void methodePasEncoreImplementee() throws Exception {
    throw new Exception();
}
```

- Mais aussi créer nous-mêmes nos propres types d'exceptions, en créant une classe qui hérite d'Exception. Pourquoi ne pas essayer?

Chapitre 3

LA GÉNÉRICITÉ

La généricité

- Nous utilisons déjà ce principe dans les collections :

```
// Collection non typée
ArrayList listeDObjects = new ArrayList();
listeDObjects.add(new Personne());              // Pas de vérification à l'insertion
Personne p1 = (Personne) listeDObjects.get(0);  // Besoin de cast + risques
p1.agirCommeUnePersonne();

// Collections génériques
ArrayList<Personne> listeDePersonnes = new ArrayList<Personne>();
listeDePersonnes.add(new Personne());               // Vérification du type à l'insertion
listeDePersonnes.get(0).agirCommeUnePersonne();     // Pas besoin de cast
```

La généricité

- Ce principe peut également s'appliquer à des classes. On construit alors une classe générique et des méthodes génériques.

- Le but est :
  - d'éviter la répétition de code (à cause de classes traitant de manière identique des types d'objets différents)
  - d'éviter d'utiliser l'héritage et les conversions.

La généricité

- Exemple de classe générique

```
public class Assurance<T> {

    private T param;

    public Assurance() {
    setParam(null);
    }

    public Assurance(T param) {
        this.setParam(param);
    }

    public T getParam() {
        return param;
    }

    public void setParam(T param) {
        this.param = param;
    }
}
```

Chapitre 4

LES ÉNUMÉRATIONS

Les énumérations

- Une énumération permet d'éviter l'accumulation de champs de type :

```
private static int LANG_FR = 0;
private static int LANG_EN = 1;
```

  - Utilisation :

```
public String getContenu(int lang) {
    if (lang == LANG_FR) {
        return getContenuFr();
    } else if (lang == LANG_EN) {
        return getContenuEn();
    } else {
        return getContenu();
    }
}
```

Les énumérations

- Et de les remplacer par un type énumération :

```
public enum Langues {

    FRANCAIS,
    ENGLISH;
}
```

```
public String getContenu(Langues lang) {
    if (lang == Langues.FRANCAIS) {
        return getContenuFr();
    } else if (lang == Langues.ENGLISH) {
        return getContenuEn();
    } else {
        return getContenu();
    }
}
```

Les énumérations

- En plus de permettre plus de lisibilité, les énumérations peuvent également être plus complexes :

```
public enum Langues {

    FR ("Français", "French"),
    EN ("English", "English"),
    DU ("Neederlands", "Dutch");

    private String or;
    private String en;

    private Langues(String or, String en) {
        this.or = or;
        this.en = en;
    }
```

Chapitre 5

ÉCRITURE ET LECTURE DE FICHIERS

Écriture et lecture de fichiers

- Créer et écrire dans un fichier :

```
PrintWriter out  = new PrintWriter(new FileWriter("file.dat"));

  for (int i = 0; i < 5; i++) {
    out.println("J'écris " + i);
  }

  out.close();
```

Écriture et lecture de fichiers

- Lire un fichier

```
BufferedReader in = new BufferedReader(new FileReader("file.dat"));

String line;

while ((line = in.readLine()) != null) {
    System.out.println(line);
}

in.close();
```

Chapitre 6

LES INTERFACES GRAPHIQUES

Les interfaces graphiques

- JavaFX
  - JavaFX est depuis Java 8 l'outil de création d'interface graphique officiel de Java.
  - Successeur de Swing, qui est lui-même le successeur d'AWT.

Les interfaces graphiques

- Avant de l'utiliser dans Eclipse :
  - Windows/Preferences/Java/Compiler/Errors/Warnings/ Deprecated and restricted API/Forbidden reference (access rules): Warning

Les interfaces graphiques

- Il existe le JavaFX Scene Builder qui permet de créer des interfaces graphiques facilement.

- Après l'avoir installé, il faut également installer dans Eclipse le plugin e(fx)clipse disponible à l'adresse suivante :

http://download.eclipse.org/efxclipse/updates-released/1.2.0/site

Chapitre 7

LES THREADS

Les Threads

- Un thread est une portion de code qui s'exécute en parallèle à d'autres traitements :

```
Thread thread = new Thread() {

public void run() {

        System.out.println("Doing on Background");

        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("Done on Background");

}

};

thread.start();
```

Les Threads

- Le multi-threading (le fait d'utiliser plusieurs threads) est vivement conseillé lorsqu'on travaille avec des interfaces graphiques, afin d'éviter de bloquer le thread principal.