TP2 – Réflexion Java

Programmation par composants

OBJECTIF : l'intérêt de ce TP est de manipuler quelques aspects liés à la **Réflexion Java**. Tout au long de ce TP (et même des suivants), on aura fortement besoin de rechercher certaines informations détaillées sur les classes, méthodes, etc. Pour cela, on peut utiliser l'API Java (http://java.sun.com/javase/6/docs/api/).

Cet énoncé, ainsi que toutes les ressources à télécharger mentionnées ci-dessous, sont disponibles à l'adresse http://www-info.iut2.upmf-grenoble.fr/intranet/enseignements/composants/2010-2011/index_files/Page572.htm.

Partie 1 – Deux exemples simples

Exercice 1

La réflexion permet de découvrir, en exécution, les caractéristiques d'une entité Java (classe, méthode, attribut, etc.) et d'agir même sur cette entité (dans certains cas !). La classe suivante permet, par exemple, de chercher dans la classe **`Serveur`**, une méthode qui s'appelle **`encode(…)`** et de l'invoquer dynamiquement.

```
public class ExempleReflexion
{
	public static void main(String[] args)
	{
		// déterminer l'objet "Class" qui correspond au serveur
		Class<Serveur> clsServ = Serveur.class;

		try
		{
			Method encodeM = clsServ.getDeclaredMethod("encode", new Class[]
									{String.class});

			// appeler la méthode
			if (encodeM != null)
			{
				String message = new String("MON DEUXIEME TP");
				String code = (String) encodeM.invoke(new Serveur(),
								new Object[] {message});

				System.out.println("Résultat : " + code);
			}
		}
		catch (Exception ex)
		{
			ex.printStackTrace();
		}
	}
}
```

- Télécharger le code de la classe **`ExempleReflexion`** et la tester.
- *Note* : Utiliser la classe **`Serveur.java`** développée lors du premier TP (l'importer dans le répertoire **`src`** de votre projet).

Exercice 2

- Écrire une classe qui reçoit en paramètre le nom d'une classe et une commande (**`a`**|**`m`**|**`i`**), et qui permet, selon le choix de l'utilisateur, d'afficher la liste des **a**ttributs, la liste des **m**éthodes ou la liste des **i**nterfaces caractérisant la classe donnée en paramètre. La classe à implémenter est partiellement donnée :

```
public class AfficherInfosClasse
{
	public static void main(String[] args)
	{
		if (args.length != 2)
		{
			System.out.println("Nombre d'arguments incorrect !");
			System.out.println("Usage : java AfficherInfosClasse
                                                    classe [a|m|i]");
			return;
		}
		try
		{
			Class cls = Class.forName(args[0]);

			if (args[1].equals("a")) // attributs
			{
				Field fields[] = cls.getDeclaredFields();
				for (int i = 0; i < fields.length; i++)
					System.out.println("-> " + fields[i]);
			}
			else
				// à compléter
		}
		catch (Exception ex)
		{
			ex.printStackTrace();
		}
	}
}
```

- Tester cette classe sur elle-même par exemple (en donnant comme argument le nom de cette même classe : n'oubliez pas de spécifier le nom qualifié (complet) de la classe). Tester sur la classe **`Serveur`**.
- Tester sur la classe **`Class`** de l'API Java : à vous de vérifier le nom complet de cette classe en regardant dans l'API.

Partie 2 - Interface graphique générique

Le but de cette partie est de créer une application capable de lire sur la console le nom de la classe d'un élément graphique (par exemple **`java.awt.Button`**) et d'ouvrir une fenêtre dans laquelle une instance de cette classe a été ajoutée. Au préalable et après le choix de la classe, l'application affiche à

l'utilisateur une liste de toutes les méthodes de cette classe. L'utilisateur en choisit une. L'application demande alors à l'utilisateur une valeur pour chaque paramètre de la méthode choisie, puis elle invoque la méthode sur l'instance. Ceci permet de modifier une propriété de l'élément graphique, par exemple invoquer la méthode **`setLabel(java.lang.String)`** sur le bouton permet de modifier le texte affiché sur le bouton.

- Télécharger le code de la classe **`DynamicInterface`** qui vous est fournie. Cette classe crée la fenêtre et l'affiche. Elle possède la méthode :

  ```
  public void addElement(Component instance)
  ```

  qui permet d'ajouter l'instance crée par introspection à la fenêtre. Cette instance doit être une instance de la classe `java.awt.Component` (ou d'une de ses sous-classes). Une fois le composant ajouté, appeler la méthode **`public void`** `afficher()` pour afficher la fenêtre. La fermeture de la fenêtre quitte l'application.

- Créer une classe **`Appli`** (par exemple) possédant une méthode **`main`**.
- Demander à l'utilisateur d'entrer le nom (qualifié) de la classe du composant graphique à afficher.
- Vérifier que cette classe hérite de **`java.awt.Component`**, sinon quitter immédiatement le programme.
- Afficher sur la console toutes les méthodes disponibles dans la classe choisie avec un indice. Demander alors à l'utilisateur de sélectionner un indice (prévoir la possibilité de passer directement à l'étape suivante).
  - *Amélioration facultative* : vous constaterez que selon les classes, il peut y avoir un nombre extrêmement important de méthodes (392 pour la classe **`javax.swing.JButton`**). Se contenter des méthodes déclarées peut être trop limitant : par exemple, la classe **`JButton`** ne possède pas de méthode pour modifier le texte du bouton (elle en hérite de **`javax.swing.AbstractButton`**). Proposez une solution.
- Pour la méthode choisie, afficher les types de chaque paramètre et demander à l'utilisateur de fournir une valeur. Pour simplifier, nous nous contenterons des types primitifs et du type **`String`**. Vous pourrez soit mettre les autres paramètres à **`null`**, soit bloquer l'invocation des méthodes demandant d'autres types de paramètres.
  - *Remarque* : consultez la documentation de la classe **`Class`** pour trouver la méthode permettant de distinguer les types primitifs.
- Enfin, invoquer la méthode choisie avec les paramètres saisis sur une instance (que vous aurez crée) de la classe indiquée par l'utilisateur au début.
- Créer une instance de **`DynamicInterface`** et lui passer l'instance d'élément graphique à ajouter, puis afficher la fenêtre.
- *Améliorations facultatives* :
  - Ne pas se limiter à un appel de méthode, mais à autant que l'utilisateur souhaite (lui permettant de modifier plus d'une propriété sur l'élément graphique à afficher).
  - Ne pas se limiter à *un* élément graphique pour l'interface. Il faut alors revoir la classe **`DynamicInterface`** pour pouvoir ajouter plusieurs composants et les disposer correctement.
- *Remarque* : Ce principe se rapproche du principe de fonctionnement des JavaBeans (consulter http://java.sun.com/docs/books/tutorial/javabeans/introspection/index.html pour plus d'information).

La Figure 1 montre un exemple de trace d'exécution de l'application, et la Figure 2 montre la fenêtre générée.

```
Classe de l'élément graphique à ajouter :
javax.swing.JButton
Voici toutes les méthodes de javax.swing.JButton :
...
8 : public boolean
  javax.swing.AbstractButton.imageUpdate(java.awt.Image,int,int,int,int,int)
9 : public void javax.swing.AbstractButton.setEnabled(boolean)
10 : public java.lang.String javax.swing.AbstractButton.getText()
11 : public void javax.swing.AbstractButton.setText(java.lang.String)
12 : public java.lang.String javax.swing.AbstractButton.getActionCommand()
13 : public java.lang.String javax.swing.AbstractButton.getLabel()
...

Choix de la méthode à appeler :
11
Donnez une valeur pour le paramètre de classe java.lang.String
Introspection rules!
```

Figure 1 : Exemple de trace produite par l'application.

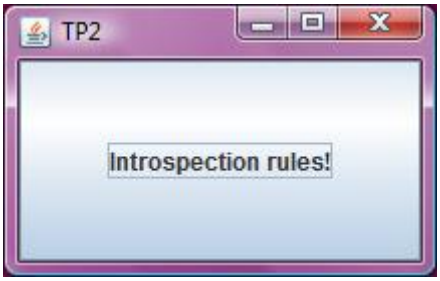

Figure 2 : Exemple d'interface générée.

- Notez que l'avantage de cette méthode est de pouvoir utiliser l'application avec n'importe quels types de composants graphiques. S'il reste tu temps, téléchargez depuis le site web du cours la librairie *SwingX* (http://swinglabs.org), qui fournit des composants graphiques supplémentaires. Ajoutez le jar **`swingx-0.9.6.jar`** au classpath de votre application et testez différentes classes fournies. Par exemple : **`org.jdesktop.swingx.JXGraph`**.