TP3 – Persistance et Sérialisation

Programmation par composants

OBJECTIF : Après les interfaces, les fabriques, la réflexion et les événements, ce 3ème TP est consacré à la **Persistance** et en particulier à la **Sérialisation Java**.

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/Page640.htm.

Bon à savoir

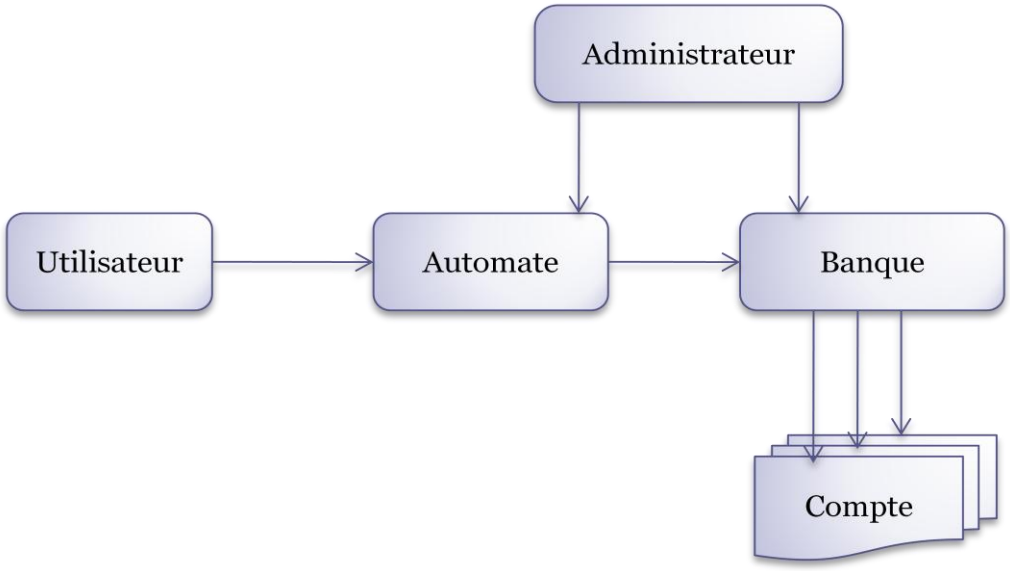
La persistance permet de rendre permanents les objets stockés en mémoire. Ceci est nécessaire vu la nature volatile de la mémoire. En Java, le mécanisme qui permet d'assurer la persistance s'appelle la sérialisation. Sans le mécanisme de sérialisation, le programmeur est obligé d'imaginer une structure de stockage pour ses objets et d'implémenter toutes les procédures qui permettent de réaliser le stockage mais aussi de gérer toutes les dépendances entre les différents objets.

Pour qu'un objet puisse être sérialisé (transformé en une suite séquentielle d'octets), sa classe doit implémenter l'interface **`java.io.Serializable`**. Cette interface ne comporte aucune méthode, elle sert juste d'annotation.

Exercice 1 – Sérialisation Java « binaire »

Partie A)

Soit l'application bancaire représentée par le diagramme ci-dessous.



- Toutes les classes de l'application sont déjà implémentées, cependant, certaines doivent être complétées. Vous pouvez télécharger ces classes à partir du site web.
- Complétez la classe **`Utilisateur`** en implémentant la méthode qui permet de créditer un compte particulier de la banque. Copier le code suivant :

```
public void crediter()
{
	try
	{
		int code = Integer.parseInt(champNumCompte.getText());
		int somme = Integer.parseInt(champSomme.getText());
		champSomme.setText("");
		boolean resultat = automate.crediter(code, somme);
		if (resultat)
			champStatut.setText("Opération bien déroulée.");
		else
			champStatut.setText("Opération mal déroulée.");
	}
	catch (NumberFormatException e)
	{
		champStatut.setText("Attention : saisie incorrecte !");
	}
}
```

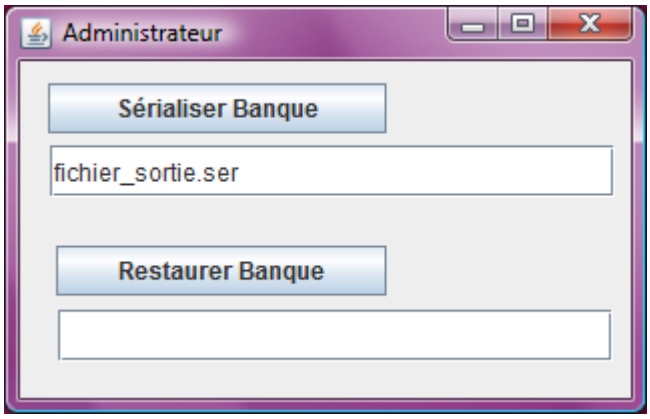
- En vous inspirant du code de cette méthode, complétez la méthode **`debiter()`** qui permet de débiter un compte particulier de la banque.
- Ecrire la classe **`Main`** qui permet de tester cette application. Vous pouvez par exemple utiliser le code suivant :

```
public class Main
{
	public static void main(String[] args)
	{
		// créer une banque
		Banque banque = new Banque("Crédit pour Tous");
		// créer quelques comptes
		banque.ajouterCompte(1001, "client1");
		banque.ajouterCompte(1002, "client2");
		banque.ajouterCompte(1003, "client3");

		// créer un automate et l'associer à la banque
		Automate automate = new Automate(banque);

		// créer l'utilisateur de l'ATM
		Utilisateur user = new Utilisateur(automate);
	}
}
```

Partie B)

Les exploitants de l'application souhaitent la rendre *persistante*. On voudrait plus particulièrement pouvoir sauvegarder l'état de la banque (avec ses comptes bien sûr !) pour pouvoir le retrouver en cas de panne. On voudrait, dans un premier temps, utiliser la sérialisation Java (package **`java.io`**) tel que vous l'avez vue en cours.

- Quelles sont les modifications nécessaires ?
- La classe **`Administrateur`** fournit une interface graphique qui permet de sérialiser ou de restaurer l'état de la banque à un moment donné. Récupérer cette classe à partir du site web.



- Compléter cette classe en implémentant les deux méthodes de sérialisation et de restitution de l'état de la banque (**`serialiserBanque()`** et **`deserialiserBanque()`**).
  - *Important* : lors de la restauration d'un état précédemment sauvegardé, ne pas oublier de mettre à jour la valeur de l'attribut **`banque`** de la classe **`Administrateur`** et aussi celui de la classe **`Automate`**.

```
this.banque = ...;
this.automate.setBanque(...);
```

- Modifier la classe **`Main`** pour pouvoir créer l'administrateur. Ajouter par exemple à la fin de la classe **`Main`** le code suivant :

```
// créer un administrateur de la banque
Administrateur admin = new Administrateur(banque, automate);
```

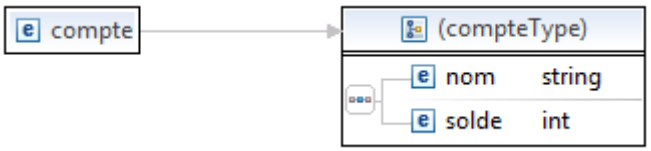
- Tester maintenant l'application. Vérifier surtout la sauvegarde et la restauration.

Exercice 2 – Sérialisation Java XML

- Tenter d'ouvrir (avec un éditeur de texte) l'un des fichiers sérialisés. Que constatez-vous ?

- Pour pouvoir visualiser et manipuler plus facilement les résultats de sérialisation, les concepteurs de l'application ont décidé de faire recours à la sérialisation en format XML.
- Créer dans le dossier **src** un sous répertoire **tp3/xml**. Copier tous les fichiers sources de l'exercice précédent dans ce répertoire.
- Rafraîchir votre projet pour prendre en compte le nouveau répertoire. Renommer ensuite le package **tp3/xml** en **tp3/exo2** pour que Eclipse corrige automatiquement le problème des noms de packages dans les fichiers sources.
- Modifier maintenant, en vous inspirant des exemples vus en cours, l'application pour permettre la sérialisation XML.
  - Que faut-il ajouter dans les deux classes **Banque** et **Compte** ? Exploiter les fonctionnalités d'Eclipse pour réaliser les ajouts d'une façon automatique.
  - *Important* : pour la classe **Banque**, on s'intéresse aux attributs **nom** et **comptes**. Pour la classe **Compte**, on s'intéresse aux attributs **code**, **client**, **d_ouverture** et **solde**.
  - Modifier la classe **Administrateur** (modifier les deux méthodes de sérialisation) pour permettre la sérialisation XML.
- Vérifier les fichiers générés après la sérialisation. Quelle conclusion pouvez-vous en tirer ?

NOTA BENE : il existe des outils simplifiant la sérialisation XML, notamment *Castor*. Il génère automatiquement des classes Java à partir d'un schéma XML (fichier **.xsd**) possédant deux méthodes pour sérialiser et dé sérialiser. Voici un exemple de schéma XML.



La classe générée ressemble à ceci :

```
public class Compte implements java.io.Serializable
{
      private java.lang.String _nom;
      private int _solde;
      ...
      public void marshal(java.io.Writer out) {...}
      public static Compte unmarshal(java.io.Reader reader) {...}
}
```

On sérialise une instance avec :

```
      compte.marshal(new FileWriter("file.xml"));
```

On dé sérialise avec :

```
      Compte compte = Compte.unmarshal(new FileReader("file.xml"));
```