TP 3: Java RMI

1 Exercice 1 : Hello World

1. Reprendre l'exemple du cours et le faire fonctionner : Dans le dossier **`T:\RCPI01\TP_RMI_1`**, créer les 4 fichiers source Java, les éditer, puis ouvrir deux terminaux dans le dossier, compiler, et vérifier le bon fonctionnement. Faites valider par l'enseignant.

2. Modifier **uniquement** la classe **`RmiObjet`** pour que à chaque exécution du client, celui-ci affiche le numéro de l'exécution :

```
$ java RmiClient
Hello World 1
$ java RmiClient
Hello World 2
$ java RmiClient
Hello World 3
```

   Ceci sera fait en ajoutant un attribut entier à la classe, initialisé à 0 dans le constructeur, et en l'incrémentant dans la méthode **`getMessage()`**.

3. Que se passe-t-il si on lance le client alors que le serveur n'est pas actif :

   _____

4. Afin d'éviter ce problème, modifier le programme client en insérant la partie utile dans un bloc **`try { }`** et ajouter dans le **`catch { }`** l'affichage d'un message "impossible de se connecter".

2 Exercice 2 : compte bancaire distant

On souhaite implémenter un compte bancaire distant. Le client pourra créditer ou débiter son compte en précisant le montant et un code opération via la ligne de commande. Par exemple :

`$ java RmiClient + 50` pour ajouter 50 €, ou
`$ java RmiClient - 30` pour débiter 30 €

Le serveur gèrera le compte, caractérisé par son solde. Dans un premier temps, on ne considère que des manipulations avec des entiers. L'objet de type "compte Bancaire" sera donc doté d'un attribut entier "solde" (déclaré de type privé) ainsi que des méthodes suivantes :

- **`void crediter( int montant )`**
- **`void debiter( int montant )`**
- **`int getSolde()`**.

1. Dans le dossier **`T:\RCPI01\TP_RMI_2`**, créer les quatre fichiers nécessaires :
   - **`RmiClient.java`**
   - **`RmiServeur.java`**
   - **`Compte.java`**
   - **`IntfCompte.java`**

   Les compléter conformément au cahier des charges, et vérifier le fonctionnement.

2. Modifier le client pour que le montant puisse être donné avec les centimes, mais **laisser** le transfert et l'attribut solde sous forme entière : la valeur stockée et transmise est simplement exprimée en centimes. L'affichage devra simplement procéder à une division par 100 pour afficher des €. Pour la saisie, il faudra transformer le 2e argument en variable de type "float", puis la multiplier par 100 et la convertir en entier pour transmettre la somme sous forme de centimes.

3. Modifier le serveur pour que celui-ci affiche toutes les secondes le solde du compte. On réalisera ceci via une boucle infinie dans laquelle on placera le code suivant :

```
TimeUnit.SECONDS.sleep(1);
System.out.println( "solde=" + cpte.getSolde() );
```

   (Note : Il faudra aussi ajouter en tête du fichier l'import de
   `java.util.concurrent.TimeUnit`)
   Vérifier le fonctionnement en procédant à quelques transactions via le client.

4. Dans cette boucle infinie, ajouter le code nécessaire pour qu'un taux d'intérêt de 1% par tranche de 10 s. soit appliqué sur le compte. Il faudra compter les périodes de 1 s. et au bout de 10 ajouter au compte un montant calculé sur le solde présent 10 s. auparavant. Il faudra donc **mémoriser** ce montant via une variable du serveur, et le réactualiser à chaque calcul d'intérêts.
   A l'exécution, on devra avoir sur le serveur un affichage du type :

```
...
solde=17.8 compt=8
solde=17.8 compt=9
interets:0.178 arrondi: 0.18
solde=17.98 compt=0
solde=17.98 compt=1
...
```

   L'arrondi est obtenu via la bibliothèque mathématique avec la fonction **`round()`**. Par exemple, pour obtenir les intérêts sous forme entière à partir d'une valeur de type **`Double`**, on écrira :

```
int interets_i = (int)Math.round( interets_d );
```

Annexe : compléments Java

1. **Conversion numérique**
   Pour récuperer les valeurs numériques contenues dans une chaîne de caractères **`str`** de type **`String`**, il faut utiliser les méthodes suivantes :

```
double v1 = Double.parseDouble( str );
int    v2 = Integer.parseInt( str );
```

   Pour convertir un entier en flottant ou inversement, il faut procéder à un *cast* explicite :

```
int   v1_i = 10;
float v1_f = (float)v1_i;  // => 10.0

float v2_f = 3.14;
int   v2_i = (int)v2_f; // => 3
```

2. **Comparaison de chaînes**
   Pour comparer des chaînes de caractères de type **`String`**, on utilise la methode **`equals()`** :

```
String s1,s2;
...
if( s1.equals(s2) ) {
   ...
}
```