TP1 : Java, pas à pas

Dans ce TP, nous allons observer des programmes Java s'exécuter, pas à pas.

1 Quelques dates concernant le projet

2015-03-12 Binômes constitués

2015-03-26 Cahiers des charge prêts

2015-04-30 Soutenances et rendu

2 Recommandations

À l'issue de chaque séance de TP, et au plus tard le vendredi suivant, un compte-rendu doit être transmis par courrier électronique à vincent.laporte@irisa.fr (pas de pièce jointe, merci). Ce compte-rendu prendra généralement la forme du code source écrit pendant la séance accompagné de documents hypertextes produits par `javadoc`[1], accompagnés du code source des programmes que vous aurez écrits.

Pendant tous ces travaux, et les suivants, une attention particulière sera portée à ce que toute classe apparaisse dans un *package* nommé ; en général, chaque classe sera déclarée soit abstraite (`abstract`) soit verrouillée (`final`).

N'hésitez pas à fouiller la bibliothèque standard dont la documentation est accessible sur http://docs.oracle.com/javase/8/docs/api/. Essayez de documenter vos programmes en suivant cet exemple. En particulier, chacun de vos *packages* contiendra un fichier `package-info.java`, point d'entrée de votre compte-rendu.

En général, chaque classe sera codée dans un fichier séparé, dont le nom sera de la forme `Nom.java`. Chaque fichier d'un *package* devra être placé dans un dossier au nom dudit *package*. Les sources seront placées dans un dossier `src`, et l'ensemble des fichier `.class` dans un dossier `bin`. Par exemple la compilation d'un fichier source du *package* `pack` se fera donc depuis le dossier parent de `src` et `bin` par la commande :

```
javac -d bin -cp bin src/pack/Nom.java
```

Pour compiler l'ensemble du *package* `pack`, vous pourrez utiliser :

```
javac -d bin -cp bin src/pack/*.java
```

L'exécution de la fonction `main` de la classe `Nom` du *package* `pack` se fera depuis le dossier principal comme suit :

```
java -cp bin/ pack.Nom
```

1. Plus d'info sur : http://docs.oracle.com/javase/8/docs/technotes/tools/unix/javadoc.html

3 Pas à pas

Les programmes dans ce TP seront écrits sous une forme assez particulière. Ce seront des objets satisfaisant l'interface `Iteration` suivante :

```
public interface Iteration {
  boolean isDone();
  Iteration step();
}
```

Un objet de type `Iteration` représente un programme un cours d'exécution. La méthode `isDone` permet de savoir si l'exécution est terminée, et la méthode `step`, si l'exécution n'est pas terminée, permet d'exécuter un pas dans le programme courant. La méthode `toString` permet de matérialiser l'état de l'exécution sous la forme d'une chaine de caractères.

Définissez cette interface et ajoutez-y des commentaires.

Le programme qui exécute pas à pas un tel programme pourra ressembler à ce qui suit.

```
public static void main(final String[] args) {
  final BufferedReader in = new BufferedReader(
    new InputStreamReader(System.in));
  final Iteration i = select(args);
  try {
    while ( ! i.isDone() ) {
      System.out.println(String.format("%s", i));
      final String msg = in.readLine();
      if (msg == null || "quit".equals(msg)) {
        break;
      }
      i.step();
    }
  } catch (IOException e) { /* nothing */ }
  System.out.println("Au revoir.");
}
```

3.1 Bonjour le monde Programmez, sous la forme d'un objet de type `Iteration`, un programme qui termine en un seul pas après avoir affiché « 你好，世界 ! » ou tout autre message de bienvenue.

3.2 Suite de Fibonacci Programmez de même le calcul des termes de la suite de Fibonacci.

3.3 Tri de tableau Programmez un algorithme de tri de tableau non récursif ; par exemple le tri par sélection.

⋆ **3.4 Tours de Hà Nội** Programmez une solution au problème des tours de Hà Nội. À chaque pas, un seul disque est déplacé.

4 Interface graphique

Dans cette partie, nous élaborons une interface graphique avec Swing au programme de la partie précédente.

4.1 Interface : fenêtres et boutons

a) Réalisez un programme qui ouvre une fenêtre munie d'un boutton « Quitter ».

b) Ajoutez un bouton « Commencer » qui crée un objet `Iteration` et un bouton « Avancer » qui fait exécuter un pas dans cette itération.

c) Ajoutez une zone dans la fenêtre destinée à accueillir une représentation imagée du programme en cours d'exécution. Pour cela utilisez un `JPanel` dont vous redéfinirez la méthode `paintComponent(Graphics g)`.

4.2 Graphique : dessinons les programmes Pour pouvoir dessiner l'état d'un programme en cours d'exécution, ajoutons le comportement `Renderable` qui peut être défini comme suit.

```
static interface Renderable {
  Renderable render(Graphics g, int w, int h);
}
```

a) Choisissez un des programmes de la partie précédente et ajoutez-lui ce comportement.

Attention : cela ne doit pas casser les programmes de la partie précédente. Pour cela on peut définir une interface qui combine `Renderable` et `Iteration`.

b) Modifiez votre interface graphique pour qu'elle fasse appel quand nécessaire à la méthode `render` du programme en cours d'exécution.

4.3 Pour aller plus loin Voici quelques extensions possibles, par ordre croissant de difficulté.

a) Avance rapide Ajoutez la possibilité d'exécuter plusieurs étapes d'un coup.

b) Choix du programme ā exēcuter Ajoutez la possibilité de choisir quel programme doit être exécuté. Cela peut prendre la forme d'un menu déroulant (`JComboBox`).

⋆**c) Retour en arrière** Ajoutez un bouton « Reculer » et le comportement correspondant. Deux patrons de conception peuvent ētre utiles : mēmento et commande.