**Université Paris Diderot** **Programmation Orientée Objet**
**L2** **Année 2016-2017**

# TP n°1

# Révisions - Évaluation

N'oubliez pas de vous inscrire sur DidEL ! Les groupes sont créés, pensez par conséquent à vous inscrire dans *VOTRE* groupe. Si vous n'avez pas le même groupe de TD et TP, inscrivez-vous dans les deux.

Le TP est censé occuper tout le monde toutes les 3 heures, ne soyez donc pas surpris de ne pas pouvoir le finir. Si les exercices 1 à 5 sont fait correctement - code indenté, commenté et testé - c'est déjà très bien.

## Révisions sur les structures de bases

**Exercice 1 [boucle while]**
**Aide :** La méthode

```
public static double Math.random()
```

retourne une valeur flottante pseudo-aléatoire supérieure à 0 et strictement inférieure à 1.

Pour obtenir la partie entière d'un flottant, on utilisera le cast (conversion de type) : `(int)unflottant`.

Écrivez un programme qui choisit deux nombres entre 1 et 9 et demande à l'utilisateur le résultat de leur multiplication, si la réponse est mauvaise le programme repose la question. (Pour la lecture au clavier, reportez-vous à **Annexe : lecture au clavier**)
**Exemple d'exécution :**

```
> java TableMultiplication
Quel est le resultat de 2 X 6 ? 10
Faux !
Recommencez:2 X 6 =  ? 18
Faux !
Recommencez:2 X 6 =  ? 12
Bravo !!!
```

**Exercice 2 [boucle for, switch case]** Une séquence d'ADN est représentée par un mot sur l'alphabet constitué des 4 lettres A, C, G et T. Par exemple, ACCTGTTA.

Écrivez un programme qui **affiche** au hasard une séquence d'ADN. La longueur de la séquence est donnée par l'utilisateur (cf. **Annexe : lecture au clavier**) et on suppose que la probabilité de chaque lettre est identique.

Pour cela, vous écrirez d'abord une méthode

```
public static char baseADN()
```

qui tirera une lettre au hasard de la manière suivante : on engendrera un nombre au hasard entre 1 et 4 et, à l'aide d'un `switch case`, on en déduira la lettre.

**Exercice 3 [String, lecture de la ligne de commande]** Écrivez un programme qui réécrit les chaînes de caractères données en argument de la ligne de commande en les tronquant de leur première et dernière lettre.

**Exemple d'exécution :**

```
>java Tronque Il fait beau aujourd'hui
ai ea ujourd'hu
```

**Aide :** La méthode `s.substring(i,j)` retournera la sous-chaîne constituée des caractères de `i` à `j-1`. Voir l'annexe pour l'explication de la lecture de la ligne de commande.

## Un peu d'objet

**Exercice 4 [Tamagotchis]** Le Tamagotchi est un animal de compagnie virtuel japonais. Ce nom est un mot-valise créé à partir des mots tamago ("œuf") et de l'abréviation de watchi qui vient du mot anglais watch ("montre"). Le jeu consiste à simuler l'éducation d'un animal à l'aide d'une petite console miniature, de la taille d'une montre, dotée d'un programme informatique.

Un tamagotchi va donc être un objet qui vit (perd de l'énergie) et qui doit être alimenté pour survivre (augmenter son énergie). Le but est donc ici de réaliser un petit programme qui consiste à faire interagir l'utilisateur dans le but de maintenir en vie une petite colonie de tamagotchis ayant un comportement basique.

1. Écrivez une classe `Tamagotchi` qui contient les attributs **privés** suivants : `maxEnergie`, `energie` de type entier et `nom` de type `String`. Le constructeur prend comme paramètre une chaîne de caractères pour le nom du tamagotchi et initialise les autres attributs, de façon aléatoire : `maxEnergie` entre 5 et 9 et `energie` entre 2 et 5.
2. Écrivez une méthode de signature `public void parler()` qui écrit à l'écran le nom du tamagotchi et son état de forme : "heureux", si l'attribut `energie` est supérieur à 5, ou "affamé" dans le cas contraire.
3. Écrivez une méthode de signature `public bool manger()` qui augmente d'une valeur aléatoire, comprise entre 1 et 2, la valeur de l'attribut `energie`. Si `energie` est déjà à son niveau maximum (`maxEnergie`) alors le tamagotchi n'a pas faim et la méthode devra renvoyer `false`, et `true` dans le cas où il a bien été nourri.

4. Écrivez une méthode de signature `public boolean vivre()` qui fonctionne suivant deux cas :
   (a) si `energie <= 0`, `false`.
   (b) si `energie > 0`, réduit `energie` et retourne `true`.
5. Écrivez un programme (méthode `main` située dans une classe `SimulTamagotchis`) qui crée `n` tamagotchis, la valeur `n` étant fournie par l'utilisateur. Le programme exécute en boucle les actions suivantes :
   — la méthode `vivre()` est appelée sur tous les tamagotchis. Si elle renvoie false, afficher un message de fin ("MON NOM : je meurs !!!").
   — La méthode `parler()` est appelée sur tous les tamagotchis.
   — L'utilisateur a la possibilité de choisir le tamagotchi qu'il souhaite nourrir pour ce cycle. Si la méthode `nourrir()` du tamagotchi choisi renvoie `true`, on affichera un message de satisfaction, et un message de mécontentement dans le cas contraire.

   On sort de la boucle uniquement si un tamagotchi est mort de faim.

   Exemples d'exécution

```
>java SimulTamagotchis 3
Quel nom pour le nouveau tamagotchi : Pierre
Quel nom pour le nouveau tamagotchi : Paul
Quel nom pour le nouveau tamagotchi : Jacques

 -------Cycle no 1 -------
 Pierre : je suis affamé !
 Paul : tout va bien !
 Jacques : je suis affamé !

 (0) Pierre   (1) Paul   (2) Jacques
 Nourrir quel tamagotchi ?  1

 Paul : je n'ai pas faim !!

 -------Cycle no 2 -------
 Pierre : je suis affamé !
 Paul : tout va bien !
 Jacques : je suis affamé !

 (0) Pierre   (1) Paul   (2) Jacques
 Nourrir quel tamagotchi ?  0

 Pierre : Merci !
```

```
-------Cycle no 3 -------
Pierre : tout va bien !
Paul : je suis affamé !
Jacques : je suis affamé !

(0) Pierre   (1) Paul   (2) Jacques
Nourrir quel tamagotchi ?  1

Paul : Merci !

-------Cycle no 4 -------
Pierre : je suis affamé !
Paul : tout va bien !
Jacques : je suis affamé !

(0) Pierre  (1) Paul  (2) Jacques
Nourrir quel tamagotchi ?  2

Jacques : Merci !

-------Cycle no 5 -------

Pierre : je meurs... Arrrggh !
```

## Ensemble de rationnels (facultatif)

**Exercice 5** On va créer un classe `Ratio` qui implémentera les nombres rationnels. Pour représenter un nombre rationnel on utilise deux attributs entiers `numera` et `denomi` qui seront respectivement le numérateur et le dénominateur d'un rationnel. De plus le dénominateur est toujours strictement positif.

Implémenter les méthodes publiques :

— `Ratio(int numera,int denomi)`
— `Ratio produit(Ratio a)`
— `Ratio addition(Ratio a)`
— `boolean egale(Ratio a)`
— `boolean plusGrand(Ratio a)`
— `String toString()`

Les méthodes produit et addition renvoient respectivement le produit et la somme de deux rationnels. La méthode `egale` renvoie `true` si les deux rationnels sont égaux (et renvoie `false` sinon) et la méthode `plusGrand` renvoie `true` si le rationel `a` est strictement plus grand que this. La méthode `toString` renvoie une représentation du rationnel, par exemple `13/7`.

Nous rappelons que $\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$, $\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd}$, que $\frac{a}{b} < \frac{c}{d}$ si et seulement si $ad < bc$ et $\frac{a}{b} = \frac{c}{d}$ si et seulement si $ad = bc$.

**Exercice 6** Dans cet exercice, on veut construire une classe `Ensemble` qui représente un ensemble de rationnels. C'est à vous de choisir votre implémentation.

En sachant que :

— Il ne peut y avoir deux fois la même valeur dans l'ensemble (c'est à dire pas de doublon).

— L'ensemble doit pouvoir contenir au minimum 1000 rationnels.

— Il faut que cette implémentation soit relativement efficace. (on peut, par exemple uitliser un tableau trié ou une liste chaînée (LinkedList) trié, pour les plus courageeux et ceux qui connaisent, il est possible d'utiliser un arbre de recherche)

1. Écrire la classe `Ensemble` qui contiendra les méthodes publiques
   — `Ensemble()`
   — `void ajoute(Ratio a)`
   — `boolean retire(Ratio a)`
   — `boolean est_dans(Ratio a)`
   — `String toString()`

   La fonction `toString()` renvoie une chaîne de caractères qui affiche les éléments d'une instance de type `Ensemble`.

   **Exemple d'utilisation :**

```
Ensemble E = new Ensemble();
System.out.prinln(E.toString())
>>>>  {}

E.ajoute(new Ratio(1,4));
System.out.prinln(E.toString())
>>>> {1/4}

E.ajoute(new Ratio(3,1));
E.ajoute(new Ratio(3568,6));
System.out.prinln(E.toString())
>>>> {1/4,3/1,1789/3}

E.ajoute(new Ratio(3,1));
System.out.prinln(E.toString())
>>>> {1/4,3/1,1789/3}

System.out.prinln(E.est_dans(new Ratio(3,1)))
>>>> true

System.out.prinln(E.retire(new Ratio(3,1)))
>>>> true

System.out.prinln(E.est_dans(new Ratio(3,1)))
>>>> false
```

2. Implémenter les méthodes
   — `void union(Ensemble A)`

— `void inter(Ensemble A)`
— `void diff(Ensemble A)`

`union` ajoute tous les éléments de $A$ à this. `inter` retire de this tous les éléments qui ne sont pas dans l'ensemble $A$. `diff` retire de this tous les éléments qui sont dans l'ensemble $A$. (Un conseil : Réutilisez du code)

# Annexe

### Documentation

## 1 Documentation Java en ligne

— Tutorial Java d'Oracle *(en anglais)* :
`https://docs.oracle.com/javase/tutorial/`
— Documentation : API `https://docs.oracle.com/javase/8/docs/api/`

### Avec un éditeur (emacs) et la ligne de commande

— Tapez la classe `Machin` dans un fichier appelé `Machin.java`.
— Compilez-le : **javac Machin.java**
Le fichier `Machin.class` est créé.
— Lancez le programme avec **java Machin**.

**Remarque :** Si votre programme comporte plusiseurs classes, vous devrez faire autant de fichiers que de classes publiques.

### Avec Eclipse

1. Depuis un terminal, tapez `eclipse &` ou sélectionnez `Demarrer/Developpement/Eclipse`
`Eclipse` va vous demander de sélectionner votre espace de travail, qui est le dossier qui stockera vos projets.
Il est possible de changer de perspective en sélectionnant `Window/Open perspective`. Pour écrire un programme en Java, choisissez la perspective `Java`.
2. (a) Allez à `File/New/Java Project` (ou `File/New/Projet/Java/Java Project`, puis entrez le nom du projet, par example, "Premiers programmes".
Dans le cadre JRE, choisissez `use a project specific JRE` et `java-6-sun-1.6-0.20`.
   (b) Maintenant, vous pouvez écrire un programme java : allez à `File/New/Class`, entrez le nom de la classe, par exemple, "Multiplicationjava", cochez la case `public static void main(String[] args)` si vous voulez ajouter la fonction `main` dans votre classe.
   (c) Compilation : par défaut, lorsque vous rédigez un programme, Eclipse le compile automatiquement.
   (d) Pour l'exécuter, Allez à `Run/Run As/Java Application` ou utilisez le bouton en forme de flèche blanche sur rond vert.

### Lire au clavier

Voici un exemple de lecture au clavier :

```
import java.util.Scanner; //on va utiliser la classe Scanner
```

```
public class LireEntiers{
   public static void main(String [] args) {

     // initialisation lecture
      Scanner sc = new Scanner(System.in);

      System.out.print("Donner un entier");

      //lecture de la reponse
      int r = sc.nextInt();

      System.out.println("Vous avez donné l'entier "+ r);
   }
}
```

La ligne

```
import java.util.Scanner;
```

doit être mise en début de fichier.

La ligne

```
Scanner sc = new Scanner(System.in);
```

doit être mise dans la méthode qui va utiliser la lecture, elle sert à récupérer un "lecteur sur le clavier".

Pour lire un `double`, on utilisera `sc.nextDouble()`, pour un `boolean`, `sc.nextBoolean()`, etc.

Pour lire un mot `sc.next()`, le retour à la ligne ou un espace étant considéré comme une fin de mot.

### Les arguments de la ligne de comande

Lorsqu'on lance un programme java de la manière suivante :

```
>java Prog Il fait beau
```

Le tableau de `String` donné comme argument du `main` (dans nos exemples, il s'appelle `args`) reçoit le tableau `{"Il", "fait", "beau"}`.

**Sous eclipse :** Pour exécuter, `Run/Run Configurations` et, dans l'onglet `Arguments`, ajoutez les arguments, ici, "Il fait beau".