TP de JAVA – Série 7
Les itérations

Université de Mons-Hainaut
2004 - 2005

1 Résumé du chapitre

- Une instruction `while` entraîne l'exécution d'un bloc de code de façon répétée. La condition de terminaison contrôle combien de fois la boucle est exécutée.
- Erreur classique ("off by one") : une boucle qui s'exécute une fois de trop, ou une fois de moins. Pour l'éviter, pensez à de simples tests.
- Utilisez une boucle `for` lorqu'une variable va d'une valeur à une autre selon un incrément (ou un décrément) fixé.
- Les boucles peuvent être imbriquées. L'exemple typique est l'affichage d'un tableau par lignes et colonnes.
- Vous pouvez «découper» une ligne en mots en utilisant un `StringTokenizer`
- La méthode `charAt` vous permet d'accéder aux caractères d'un `String`
- Utilisez la méthode `Random` du package `java.util.Random` pour générer des nombres aléatoires.

Les packages, classes et méthodes introduites dans ce chapitre :

```
java.lang.String
   indexOf

java.util.Random
   nextDouble
   nextInt

java.util.StringTokenizer
   countTokens
   hasMoreTokens
   nextToken
```

2 Exs. de révision

R7.1 : Quels types de boucles Java supporte ? Donner des règles simples à utiliser avec chaque type de boucle.

R7.2 : Ré-écrire la boucle `do` suivante en une boucle `while`.

```
int n = 1;
double x = 0;
double s;
do
{
  s = 1.0 / (n * n);
  x = x + s;
  n++;
}
while (s > 0.01);
```

R7.3 : Qu'est-ce qu'une boucle infinie ? Sur votre PC, comment pouvez-vous terminer un programme qui exécute une boucle infinie ?

R7.4 : Montrer comment utiliser un `StringTokenizer` pour découper la chaîne de caractères : `"Hello, cruel world!"` en plusieurs «morceaux». Quels sont les «morceaux» résultants ?

R7.5 : Comment découpez-vous la chaîne de l'exercice précédent en caractères ? Quels sont les caractères résultants ?

R7.6 : Que signifie la terminologie "loop and a half" ?

Donnez trois stratégies pour implémenter le "loop and half" décrit ci-après. L'une utilise une variable booléenne, la seconde l'instruction `break`, la dernière une méthode avec de multiples instructions `return`.

Laquelle des 3 approches trouvez-vous la plus claire ?

```
loop
{
lire le nom du pont;
si problème, sortir de la boucle;
lire la longueur du pont (en m);
si problème, sortir de la boucle;
convertir la longueur en pieds;
afficher les données du pont;
}
```

R7.7 : Donnez une stratégie pour lire un input de la forme :

nom du pont longueur du pont

où le nom du pont peut-être un simple mot («Sevres»), ou plusieurs mots («Saint Cloud»). La longueur du pont est un nombre flottant. Contrairement à l'exercice précédent, l'entièreté de l'input est donné sur une seule ligne.

R7.8 : Parfois les étudiants écrivent des programmes avec des instructions telles que : «Entrer les données, 0 pour quitter».
(Une telle valeur (ici 0) est appellée valeur sentinelle).
Expliquer pourquoi ce n'est pas une bonne idée en général.

R7.9 : Comment utiliseriez-vous un générateur de nombres aléatoires pour simuler les trois dessins d'une machine à sous ?

R7.10 : Qu'est-ce qu'une erreur "off by one" ? Donnez un exemple de votre propre expérience de programmation, ou sinon fabriquez-en un.

R7.11 : Donnez deux exemples de boucle `for`. L'un pour lequel il est plus naturel d'utiliser des bornes «symétriques», l'autre pour lequel il est plus naturel d'utiliser des bornes «asymétriques».

R7.12 : Qu'est-ce que les boucles imbriquées ? Donnez un exemple typique où celles-ci sont utilisées.

3 Exs. programmation

P7.1 : *Change.*
Écrire un programme qui demande à l'utilisateur d'entrer la valeur du change (pour la journée en cours)

entre l'Euro et le Dollar U.S., ensuite le programme lit des valeurs entrées en dollars, les convertit en euros, puis les affiche. Le programme stoppe quand l'utilisateur clique "Cancel" dans la boîte de dialogue de `JOptionPane`.

P7.2 : *Course d'un projectile.*
Si un obus de canon est tiré strictement verticalement vers le ciel (à un instant $t = 0$, avec une vitesse initiale v_0), en négligeant les frottements de ce projectile dans l'air, alors (cf. le cours de Mécanique) la position de cet objet à l'instant t (en s), est repérée (sur l'axe vertical d'origine le canon, en m) par :

$$s(t) = -(1/2)gt^2 + v_0 t, \text{ où } g = 9,81 m/s^2$$

L'objet de cet exercice est de simuler le tir (en calculs...) et de vérifier cette loi.
On fait l'approximation que pendant chaque intervalle de temps $\Delta t = 0.01$ (s) la vitesse est constante. Alors dans un tel intervalle, si on note v la vitesse, on a (cf. Méca.) : $v = \dfrac{\Delta s}{\Delta t}$.
Maintenant d'un intervalle de temps à l'autre la vitesse diminue, elle est réduite à cause de la force gravitationnelle g, sa variation (cf. Méca.) est de : $\Delta v = g\Delta t$.
Le programme consistera donc à mettre à jour toutes les 0.01s les valeurs de v et de s, jusqu'à ce que l'obus retombe sur Terre, c'est-à-dire quand s vaudra 0. Ainsi, dans le code on aura :

```
double deltaT = 0.01;
s = s + v * deltaT; //mise à jour
v = v - g * deltaT; //mise à jour
```

Prenez comme paramètre de simulation v_0 comme input (exemple de test avec $100 m/s$), et afficher la valeur de $s(t)$ toutes les secondes, d'une part celle calculée avec la formule exacte, d'autre part celle de la simulation. Utilisez une classe `CannonBall`.
Note : En fait la formule «exacte» ne l'est pas vraiment, car g n'est pas constant : la gravitation (i.e. g) diminue au fur et à mesure que l'on s'éloigne de la Terre.

P7.3 : La suite de Fibonacci est une suite (f_n) d'entiers, définie de proche en proche par la relation de récurrence suivante :

$f_1 = 1$
$f_2 = 1$
$f_n = f_{n-1} + f_{n-2}$ si $n > 2$.

Écrire un programme qui demande à l'utilisateur d'entrer un entier $n > 2$, et qui affiche alors la valeur du n-ième nombre de Fibonacci, i.e. f_n.
Utiliser une classe `FibonacciGenerator` avec une méthode `nextNumber`.

Indication : il est inutile (même non recommandé), de stocker toutes les valeurs f_m (pour $m < n$) : il suffit de conserver les deux précédentes valeurs de la suite pour avoir le terme suivant.

P7.4 : *La moyenne et l'écart-type.*
Écrire un programme qui lit un ensemble de valeurs réelles depuis l'input de JOptionPane, jusqu'au clic de "Cancel". Une fois ce clic atteint, afficher le nombre de valeurs entrées, la moyenne et l'écart-type de ces valeurs.

Rappels : Soit un ensemble $\{x_1, \ldots, x_n\}$ de n réels.

La moyenne : $\bar{x} = \dfrac{x_1 + \ldots + x_n}{n}$

L'écart-type (racine carrée de la variance) : $S = \sqrt{\dfrac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$.

P7.5 : *La factorisation des entiers*
Écrire un programme qui demande à l'utilisateur un entier et affiche alors tous ses facteurs premiers (avec leur multiplicités). Par exemple si l'utilisateur entre le nombre 150, alors le programme affiche : $2, 3, 5, 5$.
Utiliser une classe `FactorGenerator` avec les méthodes `nextFactor` et `hasMoreFactors`.

P7.6 : *Les nombres premiers*
Écrire un programme qui demande à l'utilisateur un entier et affiche alors tous les nombres premiers inférieurs ou égaux à cet entier. Par exemple si l'utilisateur entre le nombre 20, alors le programme affiche : $2, 3, 5, 7, 11, 13, 17, 19$. Notez que 1 n'est pas premier.
Utiliser une classe `PrimeGenerator` avec une méthode
`nextPrime`.

P7.7 : La méthode de *Héron* (connue des grecs depuis l'Antiquité) est une méthode pour calculer les racines carrées. Si x est une approximation de $\sqrt{a}$ alors la moyenne de x et $\frac{a}{x}$ est une meilleure approximation.
Implémenter une classe `RootApproximator` qui démarre avec une valeur approchée de 1, et pour laquelle la méthode `nextGuess` produit une suite d'approximations de plus en plus proches de la valeur exacte.
Ajouter une méthode `hasMoreGuesses` qui retourne `false` si deux approximations successives sont suffisamment proches l'une de l'autre. Alors tester votre classe ainsi :

```
RootApproximator r = new RootApproximator(a);
while (r.hasMoreGuesses()
  System.out.println(r.nextGuess());
```

P7.8 : La meilleure méthode itérative connue pour calculer les solutions d'une équation $f(x) = 0$, avec f une fonction réelle dont on sait qu'elle s'annule sur son domaine de définition et est dérivable, est la méthode de *Newton-Raphson*.
Pour trouver une solution approchée, itérer le calcul suivant jusqu'à obtenir la précision voulue :

$$x_{\text{new}} = x_{\text{old}} - \frac{f(x_{\text{old}})}{f'(x_{\text{old}})}$$

Écrire un programme qui calcule les racines n-ièmes de nombres réels : demander à l'utilisateur un réel $a > 0$ et un entier n, et alors calculer (jusqu'à ce que deux approximations consécutives soient distantes de moins de 10^{-5}) la valeur de $\sqrt[n]{a}$ en utilisant la fonction : $f(x) = x^n - a$.
Suivre la même approche que l'exercice précédent concernant le choix de la classe et des méthodes.

P7.9 : En mathématiques on définit l'exponentionnelle complexe (pour $x \in \mathbb{C}$) comme étant la série :

$$\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

et l'exponentionelle réelle e^x comme la partie réelle de cette fonction. On montre que c'est la même somme pour $x \in \mathbb{R}$.
Écrire un programme qui calcule une valeur approchée de cette somme (on ne peut pas faire une somme infinie...), pour l'entrée d'un réel x.
À chaque étape du calcul, ajouter un terme au précédent total, et pour avoir ce nouveau terme, calculez-le ainsi :

```
term = term * x /n;
```
(Vérifier que c'est une bonne manière de procéder).
Suivre l'approche des deux précédents exercices : implémenter une classe `expApproximator`.

P7.10 : Programmer la simulation suivante : des points sont inscrits aléatoirement dans le carré dont deux extrémités sont (1,1) et (-1,-1). Si le point se trouve dans le cercle unité (de centre (0,0) et de rayon 1) alors le coup est réussi, sinon il est manqué. Exécuter cette simulation et l'utiliser pour déterminer une approximation de Π. Expliquer pourquoi c'est une meilleure méthode d'approximation que le programme "Buffon Needle".