TP de JAVA – Série 4
Types de données fondamentaux

Université de Mons-Hainaut
2004 - 2005

1 HOWTO : mener à bien des calculs

Beaucoup de problèmes de programmation nécessitent l'utilisation de formules pour calculer des valeurs. Il n'est pas toujours trivial de savoir comment transformer l'énoncé d'un problème en une suite de formules mathématiques à calculer, ni en dernier ressort de les programmer en JAVA.

Étape 1 : Comprendre le problème, quelles en sont les entrées et les sorties ?

Prenons un exemple. Supposons que l'on vous demande de simuler le fonctionnement d'une machine qui vend des timbres, selon le procédé suivant : l'utilisateur doit insérer une certaine somme (multiple de 0.10€), et lorsqu'il appuie sur le bouton, la machine lui délivre autant de timbres à 0.40€ que la somme introduite le permet, puis la différence en timbres à 0.10€.

Dans ce problème, il n'y a qu'une seule entrée :

- la somme introduite par l'utilisateur

Et il y a deux sorties à déterminer :

- le nombre de timbres à 0.40€ délivrés
- le nombre de timbres à 0.10€ délivrés.

Étape 2 : Traiter des exemples «à la main».

Étape très importante. Si vous arrivez à le faire, vous serez capable d'automatiser la procédure de calcul.
Ici par exemple pour 1€ inséré, la machine délivrera 2 timbres à 0.40€, et 2 timbres à 0.10€. L'état de la machine lors de la transaction, représenté par un «montant», passe par les 4 valeurs suivantes : O€, 1€, 0.20€, 0€.

Étape 3 : Trouver les équations qui calculent les réponses.

Il faut transformer en variables les valeurs d'exemples que vous avez pris dans les calculs à la main.
Ici on peut prendre : I pour la somme insérée, x, y pour les nombres de timbres à 0.40€, et à 0.10€ délivrés.
L'équation est :

$$I = 0.40x + 0.10y, \text{ avec } x \text{ maximal.}$$

Ainsi x sera la partie entière de $\frac{I}{0.40}$, notation : $x = \lfloor \frac{I}{0.40} \rfloor$
et $y = (I - 0.40x)/0.10$.

Étape 4 : Écrire les les équations sous forme de code en Java.

On se sert des fonctions mathématiques de Java.
Détailler l'état de la machine par une variable `montant`.

Étape 5 : Construire une classe qui va mener à bien ces calculs.

On renvoie au HOWTO précédent «Concevoir et implémenter une classe» pour choisir ses méthodes et variables d'instances. Ici on trouve trois méthodes :

- `void insert(double somme)`
- `int nombreDe04()`
- `int nombreDe01()`

L'état de la machine est représenté par une variable d'instance : `montant`. D'où la classe suivante :

```
public class TimbreMachine
{
   public TimbreMachine()
   {
   montant = 0;
   }
   public void insert(double somme)
   {
      montant = montant + somme;
   }
   public int nombreDe04()
   {
      int x = (int)Math.floor(montant/0.40);
      montant = montant - x * 0.40;
      return x;
   }
   public int nombreDe01()
   {
      int y = (int)Math.round(montant/0.10);
      montant = 0;
      return y;
   }
   private double montant;
}
```

Étape 6 : Tester cette classe.

Vérifier les résultats fournis par les calculs à la main.
Ici :

```
public class TestMachine
{
  public static void main(String [] args)
  {
   TimbreMachine M = new TimbreMachine;
```

```
   M.insert(1);
System.out.println("Nombre de timbres
à 0.40 :" + M.nombreDe04());
System.out.println("Nombre de timbres
à 0.10 :" + M.nombreDe01());
  }
}
```

2 Résumé du chapitre

- Si les deux arguments d'une division / sont des entiers, le résultat est un entier.
- La classe `Math` contient les méthodes `sqrt` et `pow` pour calculer des racines carrées et des puissances.
- Une méthode `static` n'opère pas sur un objet particulier.
- Les chaînes de caractères peuvent être concaténées par le signe +
- Dès que l'un des deux arguments de l'opérateur + est un `String` l'autre est automatiquement converti en un `String`.

3 Exs. de révision

R4.1 : Écrire les expressions mathématiques suivantes en Java:

$s = s_0 + v_0 t + \frac{1}{2} g t^2$

$G = 4\pi^2 \frac{a^3}{P^2(m_1+m_2)}$

$FV = PV \cdot (1 + \frac{INT}{100})^{YRS}$

$c = \sqrt{a^2 + b^2 - 2ab\cos\gamma}$

R4.2 : Écrire les expressions Java suivantes en notation mathématique:

```
dm = m * ((Math.sqrt(1 + v /c) /
Math.sqrt(1 - v / c)) - 1);
volume = Math.PI * r * r * h;
volume = 4 * Math.PI * Math.pow(r, 3) / 3;
p = Math.atan2(z, Math.sqrt(x * x +y * y));
```

R4.3 : Quel est le problème dans cette version de la formule quadratique ?

```
x1 = (-b - Math.sqrt(b * b - 4 * a * c)) / 2 * a;
x2 = (-b + Math.sqrt(b * b - 4 * a * c)) / 2 * a;
```

R4.4 : Donner un exemple de code Java où il y a dépassement de capacité pour un entier. Tester ceci avec un programme. Est-ce que le problème est résolu si vous utilisez le type `float` plutôt que le type `int` ?
Donner un exemple d'erreur d'arrondi pour un calcul en virgule flottante. Tester ceci dans un programme.

R4.5 : Télécharger ici le fichier `Purse.java`. À l'aide de cette classe, écrire un programme de test qui exécute le code suivant:

```
Purse myPurse = new Purse();
myPurse.addNickels(3);
myPurse.addDimes(2);
myPurse.addQuarters(1);
System.out.println(myPurse.getTotal());
```

Le programme affiche le total : 0.6000000000000001. Expliquer pourquoi.
Donner une indication d'amélioration du programme pour que l'utilisateur ne soit pas induit en erreur.

R4.6 : Expliquer la différence entre 2, 2.0, '2', "2" et "2.0".

R4.7 : Expliquer ce que chacun des deux segments de programmes suivants calcule.

```
x = 2;
y = x + x;
```

et

```
s = "2";
t = s + s;
```

R4.8 : Qu'est-ce qu'une variable **final**? Est-il possible de définir une variable **final** sans fournir sa valeur? (Essayer.)

R4.9 : Quelles sont les valeurs des expressions suivantes ?
Pour chaque cas supposer que :

```
double x = 2.5;
double y = -1.5;
int m = 18;      String s = "Hello";
int n = 4;       String t = "World";
```

- `x + n * y - ( x + n ) * y`
- `m / n + m % n`
- `5 * x - n / 5`
- `s + t`
- `s + n`
- `1 - (1 - (1 - (1 - (1 - n))))`

R4.10 : Quelles sont les valeurs de *a*, *b*, *c* et *d* après ces instructions :

```
double a = 1;
double b = a;
a++;
Purse p = new Purse();
Purse q = p;
p.addNickels(5);
double c = p.getTotal();
double d = q.getTotal();
```

4 Exs. programmation

P4.1 : Compléter la classe **Purse** en ajoutant les méthodes **addPennies** et **addDollars**.

P4.2 : Ajouter les méthodes **getDollars** et **getCents** à la classe **Purse**.
La méthode **getDollars** retournera le nombre de dollars entier dans Purse en tant qu'entier.
La méthode **getCents** retournera le nombre de cents, en tant qu'entier.
Par exemple, la quantité d'argent dans purse est de $2.14, **getDollars** retournera 2 et **getCents** retournera 14.

P4.3 : Écrire un programme qui imprime les valeurs :

1
10
100
1000
10000
100000
1000000
10000000
100000000
1000000000
10000000000
100000000000

et ceci à l'aide d'une classe **PowerGenerator** :

```
public class PowerGenerator
{
/**
Construit un générateur de puissance
@param aFactor est le nombre qui sera
multiplié par lui-même
*/
public PowerGenerator(int aFactor){...}
/**
Calcule la puissance suivante
*/
public double nextPower(){...}
...
}
```