TP1 Programmation Java

Il est fortement conseillé d'avoir un dossier (répertoire) sur votre machine qui contiendra vos travaux en Java. Puis rajouter des sous-dossiers par projet ou TP. Ceci afin de bien vous organiser.

Nous allons donc étudier les principales **opérations arithmétiques** de Java, pour les nombres entiers (le type `int`) et les nombres approchés (le type `double`). En Java, le plus grand entier se nomme `Integer.MAX_VALUE`.

On trouve d'abord les opérations élémentaires : l'addition +, la multiplication notée par une étoile * , la soustraction – et la division notée / (on lit **slash**). Mais attention :

a) On ne peut pas écrire 2x comme en maths, il faut bien écrire 2*x

b) La division a/b entre deux int produit un int : c'est la division entière ou « **quotient** ». Par exemple 13/3 aura pour résultat 4 c'est-à-dire en 13 combien de fois 3 … donc 4.

c) Le reste de la division s'écrit `%` qu'on nomme « **modulo** » ou **reste** de la division entière. Par exemple `13 % 3` vaut 1 puisque `13 == 4 * 3 + 1`

d) Comme en maths, la division ou la multiplication est **prioritaire** sur l'addition. Les parenthèses permettent d'éliminer les ambiguïtés[1]. Exemple (2+x)*3 ou 2+x*3. Sur une même ligne, les opérateurs de même priorités auront la même priorité et seront traités de gauche à droite. Donc 1/2/3 vaut (1/2)/3 et non 1/(2/3) (vérifiez, ce n'est pas la même chose !)

e) L'égalité se note `==`, tandis que = est pour l'**affectation.** Nous reviendrons sur cette différence. Attention, on peut écrire (x=4) && (y=5). Cela ne doit pas être lu comme « est-ce que x est égale a 4 ainsi que y égale à 5 ». Il faut lire ici, j'affecte 4 à x, si cela s'est bien déroulé (ce qui souvent le cas mais pas toujours…) la valeur vaut vrai sinon, faux ; Puis j'affecte 5 à y puis je test si les 2 affectations se sont bien déroulées.

f) On met généralement des espaces avant et aprés les égalités. De même, il est de bonne pratique de laisser un espace de part et d'autre d'un opérateur : il est plus facile de lire 2 – x / y que 2-x/y. Mais on il est encore mieux d'avoir 2 – x/y...

Exercice 1.1

a) Que vaut l'expression 3 * (5 / 3) ?

b) Si x vaut 10 , que vaut l'expression 5 – 2 * (x / 3) ?

Exercice 1.2

a) Si n est un entier écrit en décimal, que représentent au niveau des chiffres les opérations n / 10 et n % 10 ?

b) Quelle opération permet de savoir si un entier n est un multiple d'un entier d ?

c) Quelle opération arithmétique permet de savoir si un entier n est pair ?

Exercice 1.3 Traduisez en bon français le morceau de code Java ci-dessous :

```
if (n%2 == 1) {
        p=n;
} else {
        p=n + 1;
}
```

Exercice 1.4

a) Que vaut l'expression 5 / 2 * 3 – 3 % 4 / 3 + 3 / 4 ?

b) Explicitez toutes les parenthèses manquantes rendant inutiles les priorités d'opérateurs.

1 On parle de notation infixée lorsque l'opérateur est situé entre ses arguments, comme dans 1 + 2. Les mathématiques et l'informatique utilisent aussi une notation préfixée f(x,y) avec l'opérateur en tête, et postfixée, comme dans n!, lorsque l'opérateur se place à la fin.

Exercice 1.5

a) Imaginez sur papier un algorithme qui parte d'un nombre **n** de secondes et qui le convertisse en heures/minutes/secondes. Testez-le avec n=4567 secondes.

b) Écrire et testez une méthode `static string convert(int n)` qui suppose que le `n` représente un nombre de secondes, et qui retourne sa conversion en h/m/s sous la forme d'une chaîne de caractères. Par exemple, si `n` vaut 4567 , le résultat sera la chaîne : "1h 16mn 7s"

Le type `double` représente les nombres approchés représentés sur 64 bits. La virgule est notée sous la forme d'un point, comme dans 3.141592653589793 . Attention donc à bien faire la différence entre 5 qui est un entier et 5.0 qui est un nombre approché (une approximation de l'entier exact 5 car des fois on a en mémoire plutôt 5.0000000000001). Les opérations + - * / se font de manière approchée : le résultat sera un double. Par exemple 5/3 vaut 1 mais 5.0/3 vaut 1.666666666666667. Lors d'une opération approchée, si l'un des opérandes est exact, il est automatiquement rendu approché. Par exemple 5.0 / 3 devient 5.0 / 3.0 et de même 2.0 / 2.0 donnera 1.0 et non 1.

On évitera d'utiliser le signe d'égalité == avec les nombres approchés !

Exercice 1.6

a) Testez l'égalité 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.7

b) Puis 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.8

Cela devrait suffire à vous décourager. L'arithmétique des nombres approchés est un domaine complexe et d'imprécision notoire, et les informaticiens essayent toujours de l'éviter si possible. S'il faut vraiment effectuer une comparaison, on utilisera comme en maths une formule :

$x \approx y \Leftrightarrow x - y < \varepsilon$

où ε est une précision donnée, par exemple 10^{-5} ...

Notation scientifique : 10^{-5} se note en Java sous la forme `1e-5` (et non 10e-5). La classe `BigDecimal` permet une manipulation plus fine et plus sûre.

Exercice 1.6 C Écrivez le code d'une méthode qui calcule le volume d'une sphère dont le rayon `R` est donnée en paramètre de la méthode. Rappel : $V = \pi R^3$

Exercice 1.7 Quelles sont les signatures de chacune des méthodes `foo` , `bar` et `gee` de la classe `AucuneImportance` étant donnés les messages valides ci-dessous à l'objet `toto` de cette classe :

`toto.foo(3,"Coka")` → 6.0 `toto.bar()` → "Je ne sais plus"

`toto.foo(3.8,"Coka")` → 7.6 `toto.gee("Coka");` affiche "Coka Light"

Exercice 1.8 (à faire quand on aura vu cette classe)

a) Complétez le source de la classe `Calculette` vue en cours de sorte qu'on puisse demander à une calculette si le nombre entier affiché à l'écran est pair. Indication : rédigez une nouvelle méthode !

b) Supposons que `calc1` soit une instance de Calculette . Quelle expression Java permettrait de savoir si l'écran de calc1 est un nombre pair ?

Le calcul interactif avec BlueJ

Interactif signifie immédiat, sur un mode de dialogue, sans avoir besoin d'écrire une classe ou même une méthode. Il s'agit de tester par exemple le comportement d'une primitive, ou de vérifier que le résultat d'une expression est bien celui attendu. Demandez « Show Code Pad » dans le menu « View » de BlueJ. Dans le panneau en bas à droite de la fenêtre (le « toplevel »).

Exercice 1.9

a) Pour vous convaincre de ne JAMAIS utiliser le signe == d'égalité pour les doubles... entrez successivement :

0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.7

0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 == 0.8

b) De tête : que vaut 25 / 12 ? Et que vaut 25 % 12 ? Vérifiez-le au toplevel de BlueJ.

Attention à l'opérateur == sur les références (référence ⇔ pointeur sur un objet), mais :

Exercice 1.10

a) Avec la même technique au toplevel, quel est le résultat de l'expression :

”Hello ” + ”Toto”

b) Quel est le résultat de la comparaison ”Hello ” + ”Toto” == ”Hello Toto” ?

L'explication est la suivante, et sera propre à la classe String. Le langage Java a été optimisé pour les String et permet de placer les valeurs littérales dans un **pool** (une sorte de mémoire) spécial, afin de limiter les doublons en mémoire. Lors de la déclaration d'une référence de type String, deux situations se présentent : (1) si une valeur littérale y est directement associée, cette dernière sera automatiquement placée dans le pool et la référence pointera dessus. (2) Si une nouvelle instance de String est créée, la valeur qu'elle représente ne sera pas poolée. Il est toutefois possible de demander la mise en pool d'une chaîne de caractères, grâce à la méthode "intern()".

c) Vérifiez par contre que la comparaison new String(”Hello ” + ”Toto”) == ”Hello Toto” donne cette fois comme résultat false ! Plus de détails dans http://thecodersbreakfast.net/index.php?post/2008/02/22/24-comparaison-des-chaines-accentuees-en-java

Le projet « Ticket Machine »

Il s'agit de la classe TicketMachine à télécharger.

Exercice 1.11

a) Lancez BlueJ. Si un projet courant est ouvert (par exemple « Formes »), fermez-le en demandant « Close » dans le menu « Project ». Ne le fermez pas avec la case de fermeture, vous risqueriez de quitter complètement l'application BlueJ !

b) Ouvrez dans BlueJ le projet « ticket-machine » de votre espace de travail. Vous devez voir une seule classe dans la fenêtre BlueJ : TicketMachine .

c) Faites un double-clic sur cette classe. Visualisez sa structure : en-tête de la classe, suivie des champs, puis du constructeur, et enfin des méthodes (nous allons décrire cela plus tard).

Exercice 1.12

a) Créez un distributeur de tickets sncf délivrant des billets à 10 euros.

b) Ouvrez un inspecteur sur les champs de l'objet sncf, vérifiez la valeur du champ price. Gardez l'inspecteur ouvert à côté de la fenêtre du projet.

c) Mettez 12 euros dans la machine sncf . Quel champ a été modifié ?

d) Demandez l'impression d'un billet. Une troisième fenêtre s'ouvre, avec le billet ! Regardez l'évolution des champs.

e) Que se passe-t-il si vous demandez un autre billet ? Vous comprenez pourquoi ce modèle est naïf ? Il ne vérifie rien...

f) Détruisez l'objet sncf (click droit sur l'objet puis « Remove »). Nous allons modifier un peu la classe TicketMachine …

Exercice 1.13

a) Editez le source de la classe TicketMachine.

b) Rajoutez une méthode cancel() qui restitue l'argent actuellement dans le solde (le champ balance), autrement dit qui remet celui-ci à zéro. Placez-la après insertMoney(…). Lorsque vous rajoutez une méthode, veillez à bien respecter l'indentation (distance à la marge)

c) Testez-la en recompilant le projet, puis en créant à nouveau sncf puis en lui envoyant des messages...

Exercice 1.14

Pour importer le package[2] des dates, il faut rajouter « import java.util.date ». Dans la méthode printTicket(), immédiatement avant la seconde ligne System.out.println("####################"), vous allez insérer l'instruction : System.out.println(new Date()); qui affiche sur le billet la nouvelle date courante. Recompilez le projet et testez plusieurs fois de suite la méthode modifiée printTicket() afin observez l'heure changer...

N.B. Nous aurions dû utiliser la méthode toString() qui convertit une date en une chaîne de caractère affichable. En réalité, la méthode System.out.println(…) fonctionne de telle sorte que si on lui passe une autre donnée qu'une chaîne, elle convertit d'abord cette donnée en chaîne, que ce soit un entier, un booléen, une référence vers un objet ou une date…

System.out.println(new Date()) ⇔ System.out.println((new Date()).toString());

Le résultat de new Date() est une référence vers un nouvel objet de type Date. Or dans la classe Date il existe une méthode de conversion vers les chaînes qui doit se nommer OBLIGATOIREMENT toString(), elle est donc utilisée. Sinon, une conversion par défaut s'opérera qui ne donnera peut-être pas entièrement satisfaction ! Il est donc conseillé dans toute classe d'objet

2 Une telle bibliothèque de classes se dit « package » en Java (« paquetage » en français).

de prévoir une méthode toString() qui permettra de convertir un objet en chaîne de caractères, afin de l'afficher par exemple !

Exercice 1.15

Rajoutez en bas de la classe TicketMachine une méthode toString() . On aimerait que la conversion en chaîne d'un distributeur soit par exemple : "TicketMachine[price=10,balance=0,total=12]"

N.B. attention, la signature de cette méthode doit OBLIGATOIREMENT être :

```
public String toString()
```

de manière à modifier celle proposée par défaut par Java et qui est aussi d'accès public.