

TP de JAVA – Série 5

Types de données fondamentaux

Université de Mons-Hainaut
2004 - 2005

Page d'accueil

API Java



Page 1 de 100

Full Screen

Quitter

1. Rappels

1.1. Les Grands Nombres

Chaque type est associé à un intervalle de valeurs possibles.
Les entiers:

- int
 - stocké sur 32 bits
 - valeurs de -2147482648 à 2147483647
- byte
 - stocké sur 8 bits
 - valeurs de -128 à 127

Page d'accueil

API Java



Page 2 de 100

Full Screen

Quitter

- short

- stocké sur 16 bits
- valeurs de -32768 à 32767

- long

- stocké sur 64 bits
- valeurs de
-9223372036854775808 à 9223372036854775807
- déclaration et initialisation:
long price = 3000000000000000L

Page d'accueil

API Java



Page 3 de 100

Full Screen

Quitter

Les réels:

- float
 - stocké avec 7 décimales
 - simple précision
 - valeurs jusqu'à un exposant +38

- double
 - stocké avec 15 décimales
 - double précision
 - valeurs jusqu'à un exposant +308

Page d'accueil

API Java



Page 4 de 100

Full Screen

Quitter

Possibilités de travailler avec des nombres sans limite de valeurs via les objets

- BigDecimal
- BigInteger

Attention toutefois

- les calculs seront plus **lents**
- obligation d'utiliser les méthodes plutôt que les opérateurs
 - add
 - subtract
 - multiply
 - divide

Page d'accueil

API Java



Page 5 de 100

Full Screen

Quitter

1.2. Notion d'Overflow

La notion d'overflow équivaut à un dépassement de la capacité de stockage d'une variable. Tester:

```
int n1=1147483647;  
int n2=2000483647;  
int n3=n1+n2;  
int n4=n1+1;
```

Effectuer des tests sur les autres types de données (double, long, short) menant à des situations d'overflows.

[Page d'accueil](#)

[API Java](#)



Page 6 de 100

[Full Screen](#)

[Quitter](#)

1.3. Affectation

L'affectation d'une variable modifie sa valeur:

```
int n=1147483647;  
System.out.println(n);  
n=2000483647;  
System.out.println(n);
```

Page d'accueil

API Java



Page 7 de 100

Full Screen

Quitter

1.4. Opérations

Effectuer des additions, soustractions, multiplications et divisions entre différents types de données (int, double, float).
Les résultats des calculs sont-ils ceux que vous attendiez?

1.5. API

Documentation en ligne de toutes les classes prédéfinies à l'adresse:
<http://java.sun.com/j2se/1.5.0/docs/api/index.html>

Page d'accueil

API Java



Page 8 de 100

Full Screen

Quitter

1.5.1. Class Math

En utilisant l'API, trouver la documentation de la classe Math, étudier le format, lister les méthodes offertes par la classe Math. Ecrire des tests pour 5 méthodes de cette classe.

1.5.2. Class String

En utilisant l'API, trouver la documentation de la classe String, lister les méthodes offertes par la classe String. Ecrire des tests pour 5 méthodes de cette classe. Utiliser deux des constructeurs dans vos tests.

1.6. Le Formatage des Nombres

Classe NumberFormat: Configurer l'affichage des nombres.

```
int quarters=2;  
int dollars=3;
```

```
final double TAX_RATE = 8.5;
```

```
double total = dollars + quarters * 0.25;  
//price 3.5
```

```
double tax= total * TAX_RATE /100;  
// tax 0.2975
```

```
System.out.println("total:" +total);  
System.out.println("tax:" +tax);
```

Page d'accueil

API Java



Page 9 de 100

Full Screen

Quitter

Ce programme affichera

```
total: 3.5  
tax:0.2975
```

Pour avoir la taxe dans la monnaie existante, on souhaitera arrondir celle-ci à 0.30.

Page d'accueil

API Java



Page **10** de **100**

Full Screen

Quitter

En définissant un objet NumberFormat:

```
NumberFormat formatter=  
    NumberFormat.getNumberInstance();  
    ....//configuration du formatter
```

Et en faisant afficher:

```
System.out.println("tax:" +  
    formatter.format(tax));
```

On obtiendra l'affichage désiré.

Regarder l'API de la classe NumberFormat et écrire un programme qui affichera la taxe avec 2 chiffres après la virgule.

Tester d'autres méthodes de la classe en rapport avec le format d'affichage.

Page d'accueil

API Java



Page 11 de 100

Full Screen

Quitter

1.7. Demander quelque chose à l'utilisateur

```
import javax.swing.JOptionPane;

public class essai
{
    public static void main(String args[])
    {
        String s=
            JOptionPane.showInputDialog("message: ");
        System.out.println(s);
    }
}
```

Page d'accueil

API Java



Page 12 de 100

Full Screen

Quitter

2. Révision exercices

R5.1 : Soit n un entier et x un nombre flottant.

Expliquer la différence entre:

$$n = (\text{int})x;$$

et

$$n = (\text{int})\text{Math.round}(x);$$

R5.2 : Soit n un entier et x un nombre flottant.

Expliquer la différence entre:

$$n = (\text{int}) (x + 0.5);$$

et

$$n = (\text{int})\text{Math.round}(x);$$

Pour quelles valeurs de x ces expressions donnent-elles le même résultat? Pour quelles valeurs de x ces expressions donnent-elles des résultats différents?

Page d'accueil

API Java



Page 13 de 100

Full Screen

Quitter

R5.3 : Ne pas initialiser des variables peut causer de sérieux problèmes. Doit-on toujours initialiser une variable à zéro? Expliquer les avantages et les inconvénients d'une telle stratégie.

R5.4 : Vrai ou Faux? (x est un *int* et s est un *String*)

- `Integer.parseInt("" + x);` est équivalent à `x`
- `"" + Integer.parseInt(s)` est équivalent à `s`
- `s.substring(0, s.length())` est équivalent à `s`

Page d'accueil

API Java



Page 14 de 100

Full Screen

Quitter

R5.5 : Comment récupérer le premier caractère d'une chaîne de caractères? Le dernier caractère?

Comment enlever le premier caractère? Le dernier caractère?

R5.6 : Comment récupérer le dernier chiffre d'un entier? Le premier chiffre?

C'est-à-dire, si n vaut 23456, comment trouver que le premier chiffre est 2 et que le dernier chiffre est 6?

Ne pas convertir les nombres en *string*.

Astuce: Math.log.

R5.7 : Ce chapitre contient plusieurs recommandations en rapport avec les variables et les constantes qui rendent les programmes plus faciles à lire et à maintenir.

Résumer ces recommandations.

Page d'accueil

API Java



Page 15 de 100

Full Screen

Quitter

R5.8 : Quelles sont les valeurs des expressions suivantes?

- `(int)Math.round(x)`
- `(int)Math.round(x) + (int)Math.round(y)`
- `s.substring(1, 3)`
- `s.length() + t.length()`

Pour chaque cas supposer que:

- `double x = 2.5;`
- `double y = -3.5;`
- `String s= "Hello";`
- `String t= "World";`

Page d'accueil

API Java



Page 18 de 100

Full Screen

Quitter

R5.9 : Expliquer les similarités et les différences entre copier des nombres et copier des références vers des objets.

R5.10 : Quand on copie une référence **BankAccount**, l'original et la copie partagent le même objet. Ceci est important car on peut modifier l'état d'un objet à travers chacune des références. Expliquer pourquoi ceci n'est pas un problème avec les références **String**.

Page d'accueil

API Java



Page 17 de 100

Full Screen

Quitter

3. Programmation exercices

P5.1 : Ecrire une classe qui demande à l'utilisateur d'entrer deux entiers (par défaut on vous demande d'utiliser **JOptionPane**) puis affiche

- la somme
- la différence
- le produit
- la moyenne
- la distance (valeur absolue de la différence)
- le maximum (le plus grand des deux)
- le minimum (le plus petit des deux)

Ecrire une classe **Pair**

Page d'accueil

API Java



Page 18 de 100

Full Screen

Quitter

```
public class Pair
{
    /**
     * Construit une paire
     * @param aFirst premiere valeur de la paire
     * @param aSecond deuxieme valeur de la paire
     */
    public Pair(double aFirst, double aSecond){ ... }
    /**
     * Calcule la somme des valeurs de cette paire
     * @return la somme de la premiere et de la deuxieme valeur
     */
    public double getSum(){ ... }
    ...
}
```

Ecrire une classe **PairTest** qui lira deux nombres (en utilisant un **JOptionPane**), construire un objet **Pair**, appeler les méthodes et afficher les résultats.

P5.2 : Ecrire un programme qui lit 4 entiers et affiche leur somme et leur moyenne.

Définir une classe **DataSet** avec les méthodes

- void addValue(int x)
- int getSum()
- double getAverage()

Astuce: Conserver la somme et le nombre de valeur.

Ecrire un programme de test **DataSetTest** qui lira quatre valeurs et appellera **addValue** 4 fois.

Page d'accueil

API Java



Page 20 de 100

Full Screen

Quitter

P5.3 : Ecrire un programme qui lit 4 entiers et affiche la plus grande et la plus petite valeur entrées par l'utilisateur.

Utiliser une classe **DataSet** avec les méthodes

- void addValue(int x)
- int getLargest()
- int getSmallest()

Astuce: Conserver la plus petite et la plus grande valeur trouvées jusque là. Utiliser les méthodes **Math.min** et **Math.max**.

Quelles valeurs doivent être utilisées au départ?

Integer.MIN_VALUE, Integer.MAX_VALUE.

Ecrire un programme de test **DataSetTest** qui lira quatre valeurs et appellera **addValue** 4 fois.

Page d'accueil

API Java



Page 21 de 100

Full Screen

Quitter

P5.4 : Ecrire un programme qui demande à l'utilisateur d'entrer une mesure en metres et la convertit en miles, pieds et pouces. Utiliser une classe:

Page d'accueil

API Java



Page 22 de 100

Full Screen

Quitter

Page d'accueil

API Java



Page 23 de 100

Full Screen

Quitter

```
public class Converter
{
    /** Construit un convertisseur entre deux unités
        @param aConversionFactor le facteur multiplication
            de pour obtenir l'unité destination
    */
    public Converter(double aConversionFactor){...}
    /**
        Convertit la mesure source vers l'unité
            de mesure destination
        @return la valeur d'entrée convertie
            dans l'unité destination
    */
    public double convertTo(double fromMeasurement){...}
    ...
}
```

Construire 3 instances, telles que:

```
final double MILE_TO_KM = 1.609;  
Converter metersToMiles =  
    new Converter(1000 * MILE_TO_KM);
```

Pour information:

- 1 mile = 1.609 kilomètres
- 1 pied = 0.3048 mètre
- 1 pouce = 2.54 centimètres

Page d'accueil

API Java



Page 24 de 100

Full Screen

Quitter

P5.5 : Ecrire un programme qui demande à l'utilisateur d'entrer un rayon x , puis affiche:

- l'air et le périmètre du cercle de rayon x ;
- le volume et la surface de la sphère de rayon x .

Définir les classes **Cercle** et **Sphere**.

P5.6 : Ecrire une classe **SodaCan** dont le constructeur reçoit la hauteur et le diamètre d'une cannette de soda.

Ecrire des méthodes **getVolume** et **getSurfaceArea**.

Ecrire une classe **SodaCanTest** permettant de tester la classe **SodaCan**.

P5.7 : Ecrire un programme qui demande à l'utilisateur d'entrer la longueur x de côté d'un carré, puis affiche:

- l'aire et le périmètre du carré de côté x ;
- la longueur de la diagonale (utiliser le théorème de Pythagore)

Définir la classe **Carre**.

Page d'accueil

API Java



Page 26 de 100

Full Screen

Quitter

P5.8 : Rendre la monnaie.

Ecrire un programme qui indique à un caissier comment rendre la monnaie.

Le programme aura deux entrées: le montant du et le montant reçu du client.

Calculer la différence, et décomposer en pièces de 1 euro, 50 cents, 20 cents, 10 cents et 1 cent... que le client doit recevoir en retour.

Astuce: Transformer d'abord la différence de somme en un entier **balance**, somme exprimée en cents.

Identifier alors le nombre d'euros.

Soustrayer-le de **balance**.

Calculer alors le nombre de pièces de 50 cents nécessaires.

Répéter l'opération pour les pièces de 20 cents, 10 cents. Restera le nombre cents.

Définir une classe **Cashier** avec les méthodes.

- setAmountDue
- receive
- returnEuros
- return50Cents
- return20Cents
- return10Cents
- returnCent

Page d'accueil

API Java



Page 28 de 100

Full Screen

Quitter

Par exemple:

```
Cashier harry = new Cashier();
harry.setAmountDue(9.37);
harry.receive(10);
double euros = harry.returnEuros();
    //retourne 0
double cents50 = harry.return50Cents();
    //retourne 1
double cents20 = harry.return20Cents();
    //retourne 0
double cents10 = harry.return10Cents();
    //retourne 1
double cents = harry.returnCent();
    //retourne 3
```

Page d'accueil

API Java



Page 29 de 100

Full Screen

Quitter

P5.9 : Ecrire un programme qui lit un entier et le décompose en une séquence de ces chiffres pris dans l'ordre inverse.

Par exemple, 16384 sera affiché comme

4
8
3
6
1

Remarque: On supposera que le nombre entré n'a pas plus de 5 chiffres et n'est pas négatif.

Page d'accueil

API Java



Page 30 de 100

Full Screen

Quitter

Ecrire une classe **DigitExtractor**

```
public class DigitExtractor
```

```
{
```

```
    /**
```

```
        Construit un extracteur de chiffres qui prend  
        les chiffres d'un entier dans l'ordre inverse
```

```
        @param anInteger l'entier à décomposer
```

```
    */
```

```
    public DigitExtractor(int anInteger){...}
```

```
    /**
```

```
        Retourne le chiffre suivant
```

```
        @return le chiffre suivant
```

```
    */
```

```
    public int nextDigit(){...}
```

```
    ...
```

```
}
```

Ecrire un programme qui appellera `System.out.println(myExtractor.n`
5 fois.

Page d'accueil

API Java



Page 31 de 100

Full Screen

Quitter

Page d'accueil

API Java



Page 32 de 100

Full Screen

Quitter

P5.10 : Ecrire une classe **QuadraticEquation** dont le constructeur reçoit les coefficients a, b et c d'une équation quadratique $ax^2 + bx + c = 0$.

Ecrire les méthodes **getSolution1** et **getSolution2** qui fournissent les solutions en utilisant la formule quadratique.

Ecrire une classe de test **QuadraticEquationTest** qui demande à l'utilisateur les valeurs de a, b et c, construit un objet **QuadraticEquation**, et affiche les deux solutions.

P5.11 : Ecrire un programme qui lit deux heures en format militaire (0900, 1730) et affiche le nombre d'heures et de minutes entre les deux heures.

Voici un exemple d'exécution:

```
Entrer la premiere heure:
    0900
Entrer la deuxieme heure:
    1730
8 heures 30 minutes
```

Essayer de gérer le cas où la premiere heure est après la deuxieme.

```
Entrer la premiere heure:
    1730
Entrer la deuxieme heure:
    0900
15 heures 30 minutes
```

Page d'accueil

API Java



Page 33 de 100

Full Screen

Quitter

Ecrire une classe **TimeInterval** dont le constructeur prend deux heures militaires en entrée. La classe devra avoir deux méthodes **getHours** et **getMinutes**.

[Page d'accueil](#)

[API Java](#)



Page **34** de **100**

[Full Screen](#)

[Quitter](#)

P5.12 : Ecrire en grandes lettres.

Une grande lettre H peut être affichée comme suit:

```
*  *  
*  *  
*****  
*  *  
*  *
```

Définir une classe **LetterH** avec une méthode:

```
String getLetter()  
{  
    return "* *\n* *\n*****\n* *\n* *\n";  
}
```

Page d'accueil

API Java



Page 35 de 100

Full Screen

Quitter

Faire de même pour les lettres E, L et O. Ecrire le message

H

E

L

L

O

en grandes lettres.

[Page d'accueil](#)

[API Java](#)



Page 36 de 100

[Full Screen](#)

[Quitter](#)

P5.13 : Ecrire un programme qui transforme les numéros 1, 2, 3,..., 12 en le nom du mois correspondant Janvier, Fevrier, Mars, ...,Decembre.

Astuce: Créez une très longue chaîne de caractères "Janvier, Fevrier, Mars, ..." dans laquelle vous placerez des espaces d emanière à ce que chaque mois occupe *la même longueur*. Utilisez **substring**pour extraire le mois souhaité.

Ecrire une classe **Month** dont le paramètre du constructeur est le numéro du mois et dont la méthode **getName** retourne le nom du mois.

P5.14 : Dimanche de pâques

Ecrire un programme qui calcule la date du dimanche de Pâques. Le dimanche de Pâques est le premier dimanche après la première pleine lune de printemps.

Utiliser l'algorithme suivant, inventé par le mathématicien Carl Friedrich Gauss en 1800:

- 1. soit y l'année (telle que 1800 ou 2001)
- 2. diviser y par 19 et considérer le reste a (ignorer le quotient)
- 3. diviser y par 100 pour obtenir un quotient b et un reste c
- 4. diviser b par 4 pour obtenir un quotient d et un reste e
- 5. diviser $8 * b + 13$ par 25 pour obtenir un quotient g (ignorer le reste)

Page d'accueil

API Java



Page 38 de 100

Full Screen

Quitter



- 6. diviser $19 * a + b - d - g + 15$ par 30 pour obtenir un reste h (ignorer le quotient)
- 7. diviser c par 4 pour obtenir un quotient j et un reste k
- 8. diviser $a + 11 * h$ par 319 pour obtenir un quotient m (ignorer le reste)
- 9. diviser $2 * e + 2 * j - k - h + m + 32$ par 7 pour obtenir un reste r (ignorer le quotient)
- 10. diviser $h - m + r + 90$ par 25 pour obtenir un quotient n (ignorer le reste)
- 11. diviser $h - m + r + n + 19$ par 32 pour obtenir un reste p (ignorer le quotient).

Sommaire

Page d'accueil

API Java



Page 40 de 100

Full Screen

Quitter

Pâques tombe le jour **p** du mois **n**. Par exemple, pour 2001:

```
a = 6
b = 20
c = 1
d = 5, e = 0
g = 6
h = 18
j = 0, k = 1
m = 0
r = 6
n = 4
p = 15
```

Donc, en 2001, le dimanche de Pâques est tombé le 15 Avril.

Ecrire une classe **Year** avec les méthodes **getEasterSundayDay** et **getEasterSundayMonth**.