Chapitre 6

Les fonctions mathématiques en Java

Sources

Titre	Auteur	Date	Licence
Développons en Java	Jean-Michel Doudoux	2016	GNU-FDL
Apprenez à programmer en Java	Cyrille Herby	2018	CC-BY-NC-SA

6. Les fonctions mathématiques

Niveau : Elémentaire

La classe java.lang.Math contient une série de méthodes et variables mathématiques. Comme la classe Math fait partie du package java.lang, elle est automatiquement importée. De plus, il n'est pas nécessaire de déclarer un objet de type Math car les méthodes sont toutes static.

Exemple (code Java 1.1) : Calculer et afficher la racine carrée de 3

```
public class Math1 {
  public static void main(java.lang.String[] args) {
    System.out.println(" = " + Math.sqrt(3.0));
  }
}
```

Ce chapitre contient plusieurs sections :

6.1. Les variables de classe

PI représente pi dans le type double (3,14159265358979323846)

E représente e dans le type double (2,7182818284590452354)

Exemple (code Java 1.1) :

```
public class Math2 {
  public static void main(java.lang.String[] args) {
    System.out.println(" PI = "+Math.PI);
    System.out.println(" E = "+Math.E);
  }
}
```

6.2. Les fonctions trigonométriques

Les méthodes sin(), cos(), tan(), asin(), acos(), atan() sont déclarées : public static double fonctiontrigo(double angle)

Les angles doivent être exprimés en radians. Pour convertir des degrés en radian, il suffit de les multiplier par PI/180

6.3. Les fonctions de comparaisons

max (n1, n2)
min (n1, n2)

Ces méthodes existent pour les types int, long, float et double : elles déterminent respectivement les valeurs maximales et minimales des deux paramètres.

Exemple (code Java 1.1) :

```
public class Math1 {

  public static void main(String[] args) {
    System.out.println(" le plus grand = " + Math.max(5, 10));
    System.out.println(" le plus petit = " + Math.min(7, 14));
  }
}
```

Résultat :

```
le plus grand = 10
le plus petit = 7
```

6.4. Les arrondis

La classe Math propose plusieurs méthodes pour réaliser différents arrondis.

6.4.1. La méthode round(n)

Pour les types float et double, cette méthode ajoute 0,5 à l'argument et restitue la plus grande valeur entière (int) inférieure ou égale au résultat.

Exemple (code Java 1.1) :

```
public class Arrondis1 {
  static double[] valeur = {-5.7, -5.5, -5.2, -5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i < valeur.length; i++) {
      System.out.println("round("+valeur[i]+") = "+Math.round(valeur[i]));
    }
  }
}
```

Résultat :

```
round(-5.7) = -6
round(-5.5) = -5
round(-5.2) = -5
round(-5.0) = -5
round(5.0) = 5
round(5.2) = 5
round(5.5) = 6
round(5.7) = 6
```

6.4.2. La méthode rint(double)

Cette méthode effectue la même opération mais renvoie un type double.

Exemple (code Java 1.1) :

```
public class Arrondis2 {
  static double[] valeur = {-5.7, -5.5, -5.2, -5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i < valeur.length; i++) {
    System.out.println("rint("+valeur[i]+") = "+Math.rint(valeur[i]));
    }
  }
}
```

Résultat :

```
rint(-5.7) = -6.0
rint(-5.5) = -6.0
rint(-5.2) = -5.0
rint(-5.0) = -5.0
rint(5.0) = 5.0
rint(5.2) = 5.0
rint(5.5) = 6.0
rint(5.7) = 6.0
```

6.4.3. La méthode floor(double)

Cette méthode renvoie l'entier le plus proche inférieur ou égal à l'argument.

Exemple (code Java 1.1) :

```
public class Arrondis3 {
  static double[] valeur = {-5.7, -5.5, -5.2, -5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i < valeur.length; i++) {
      System.out.println("floor("+valeur[i]+") = "+Math.floor(valeur[i]));
    }
  }
}
```

Résultat :

```
floor(-5.7) = -6.0
floor(-5.5) = -6.0
floor(-5.2) = -6.0
floor(-5.0) = -5.0
floor(5.0) = 5.0
floor(5.2) = 5.0
floor(5.5) = 5.0
floor(5.7) = 5.0
```

6.4.4. La méthode ceil(double)

Cette méthode renvoie l'entier le plus proche supérieur ou égal à l'argument

Exemple (code Java 1.1) :

```
public class Arrondis4 {
  static double[] valeur = {-5.7, -5.5, -5.2, -5.0, 5.0, 5.2, 5.5, 5.7 };

  public static void main(String[] args) {
    for (int i = 0; i < valeur.length; i++) {
```

```
      System.out.println("ceil("+valeur[i]+") = "+Math.ceil(valeur[i]));
    }
  }
}
```

Résultat :

```
ceil(-5.7) = -5.0
ceil(-5.5) = -5.0
ceil(-5.2) = -5.0
ceil(-5.0) = -5.0
ceil(5.0) = 5.0
ceil(5.2) = 6.0
ceil(5.5) = 6.0
ceil(5.7) = 6.0
```

6.4.5. La méthode abs(x)

Cette méthode donne la valeur absolue de x (les nombres négatifs sont convertis en leur opposé). La méthode est définie pour les types int, long, float et double.

Exemple (code Java 1.1) :

```
public class Math1 {
  public static void main(String[] args) {
    System.out.println(" abs(-5.7) = "+Math.abs(-5.7));
  }
}
```

Résultat :

```
abs(-5.7) = 5.7
```

6.5. La méthode IEEEremainder(double, double)

Cette méthode renvoie le reste de la division du premier argument par le deuxième

Exemple (code Java 1.1) :

```
public class Math1 {
  public static void main(String[] args) {
    System.out.println(" reste de la division de 10 par 3 = "
      +Math.IEEEremainder(10.0, 3.0) );
  }
}
```

Résultat :

```
reste de la division de 10 par 3 = 1.0
```

6.6. Les Exponentielles et puissances

6.6.1. La méthode pow(double, double)

Cette méthode élève le premier argument à la puissance indiquée par le second.

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
  System.out.println(" 5 au cube  = "+Math.pow(5.0, 3.0) );
}
```

Résultat :

```
5 au cube  = 125.0
```

6.6.2. La méthode sqrt(double)

Cette méthode calcule la racine carrée de son paramètre.

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
  System.out.println(" racine carrée de 25  = "+Math.sqrt(25.0) );
}
```

Résultat :

```
racine carrée de 25 = 5.0
```

6.6.3. La méthode exp(double)

Cette méthode calcule l'exponentielle de l'argument

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
  System.out.println(" exponentiel de 5  = "+Math.exp(5.0) );
}
```

Résultat :

```
exponentiel de 5  = 148.4131591025766
```

6.6.4. La méthode log(double)

Cette méthode calcule le logarithme naturel de l'argument

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
  System.out.println(" logarithme de 5  = "+Math.log(5.0) );
}
```

Résultat :

```
logarithme de 5  = 1.6094379124341003
```

6.7. La génération de nombres aléatoires

La méthode random() renvoie un nombre aléatoire compris entre 0.0 et 1.0.

Exemple (code Java 1.1) :

```
public static void main(java.lang.String[] args) {
  System.out.println(" un nombre aléatoire  = "+Math.random() );
```

```
}
```

Résultat :

```
un nombre aléatoire  = 0.8178819778125899
```

6.8. La classe BigDecimal

La classe java.math.BigDecimal est incluse dans l'API Java depuis la version 5.0.

La classe BigDecimal qui hérite de la classe java.lang.Number permet de réaliser des calculs en virgule flottante avec une précision dans les résultats similaire à celle de l'arithmétique scolaire.

La classe BigDecimal permet ainsi une représentation exacte des valeurs ce que ne peuvent garantir les données primitives de type numérique flottant (float ou double). Les calculs en virgule flottante privilégient en effet la vitesse de calcul plutôt que la précision.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

public class CalculDouble {

  public static void main(String[] args) {
    double valeur = 10*0.09;
    System.out.println(valeur);
  }
}
```

Résultat :

```
0.8999999999999999
```

Cependant certains calculs, notamment ceux relatifs à des aspects financiers par exemple, requièrent une précision particulière : ces calculs utilisent généralement une précision de deux chiffres.

La classe BigDecimal permet de réaliser de tels calculs en permettant d'avoir le contrôle sur la précision (nombre de décimales significatives après la virgule) et la façon dont l'arrondi est réalisé.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal {

  public static void main(
      String[] args) {
    BigDecimal valeur1 = new BigDecimal("10");
    BigDecimal valeur2 = new BigDecimal("0.09");

    BigDecimal valeur = valeur1.multiply(valeur2);

    System.out.println(valeur);
  }
}
```

Résultat :

```
0.90
```

De plus, la classe BigDecimal peut gérer des valeurs possédant plus de 16 chiffres significatifs après la virgule.

La classe BigDecimal propose de nombreux constructeurs qui attendent en paramètre la valeur en différents types.

Remarque : il est préférable d'utiliser le constructeur attendant en paramètre la valeur sous forme de chaîne de caractères.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal3 {

  public static void main(String[] args) {
    BigDecimal valeur1 = new BigDecimal(2.8);
    BigDecimal valeur2 = new BigDecimal("2.8");

    System.out.println("valeur1="+valeur1);
    System.out.println("valeur2="+valeur2);
  }
}
```

Résultat :

```
valeur1=2.79999999999999982236431605997495353221893310546875
valeur2=2.8
```

Avec cette classe, il est parfois nécessaire de devoir créer une nouvelle instance de BigDecimal à partir de la valeur d'une autre instance de BigDecimal. Aucun constructeur de la classe BigDecimal n'attend en paramètre un objet de type BigDecimal : il est nécessaire d'utiliser le constructeur qui attend en paramètre la valeur sous la forme d'une chaîne de caractères et de lui passer en paramètre le résultat de l'appel de la méthode toString() de l'instance de BigDecimal encapsulant la valeur.

La classe BigDecimal propose de nombreuses méthodes pour réaliser des opérations arithmétiques sur la valeur qu'elle encapsule telles que add(), substract(), multiply(), divide(), min(), max(), pow(), remainder(), divideToIntegralValue(), ...

Le classe BigDecimal est immuable : la valeur qu'elle encapsule ne peut pas être modifiée. Toutes les méthodes qui effectuent une opération sur la valeur encapsulée retournent un nouvel objet de type BigDecimal qui encapsule le résultat de l'opération.

Une erreur courante est d'invoquer la méthode mais de ne pas exploiter le résultat de son exécution.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal7 {
  public static void main(String[] args) {
    BigDecimal valeur = new BigDecimal("10.5");
    BigDecimal bonus = new BigDecimal("4.2");

    valeur.add(bonus);
    System.out.println("valeur=" + valeur);

    valeur = valeur.add(bonus);
    System.out.println("valeur=" + valeur);
  }
}
```

Résultat :

```
valeur=10.5
valeur=14.7
```

La méthode setScale() permet de spécifier la précision de la valeur et éventuellement le mode d'arrondi à appliquer. Elle retourne un objet de type BigDecimal correspondant aux caractéristiques fournies puisque l'objet BigDecimal est immuable.

C'est une bonne pratique de toujours préciser le mode d'arrondi car si un arrondi est nécessaire et que le mode d'arrondi n'est pas précisé alors une exception de type ArithmeticException est levée.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal4 {

  public static void main(String[] args) {
    BigDecimal valeur1 = new BigDecimal(2.8);
    valeur1.setScale(1);
    System.out.println("valeur1="+valeur1);
  }
}
```

Résultat :

```
Exception in thread "main" java.lang.ArithmeticException: Rounding necessary
        at java.math.BigDecimal.divide(BigDecimal.java:1346)
        at java.math.BigDecimal.setScale(BigDecimal.java:2310)
        at java.math.BigDecimal.setScale(BigDecimal.java:2350)
        at com.jmdoudoux.test.bigdecimal.CalculBigDecimal4.main(CalculBigDecimal4.java:10)
```

La classe BigDecimal propose plusieurs modes d'arrondis : ROUND_CEILING, ROUND_DOWN, ROUND_FLOOR, ROUND_HALF_UP, ROUND_HALF_DOWN, ROUND_HALF_EVEN, ROUND_UNNECESSARY et ROUND_UP

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal5 {

  public static void main(String[] args) {
    BigDecimal valeur = null;
    String strValeur = null;

    strValeur = "0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_CEILING);
    System.out.println("ROUND_CEILING    "+strValeur+" :  "+valeur.toString());

    strValeur = "-0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_CEILING);
    System.out.println("ROUND_CEILING   "+strValeur+" : "+valeur.toString());

    strValeur = "0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_DOWN);
    System.out.println("ROUND_DOWN       "+strValeur+" :  "+valeur.toString());

    strValeur = "0.228";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_DOWN);
    System.out.println("ROUND_DOWN       "+strValeur+" :  "+valeur.toString());

    strValeur = "-0.228";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_DOWN);
    System.out.println("ROUND_DOWN      "+strValeur+" : "+valeur.toString());

    strValeur = "0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_FLOOR);
```

```
    System.out.println("ROUND_FLOOR      "+strValeur+" :  "+valeur.toString());

    strValeur = "-0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_FLOOR);
    System.out.println("ROUND_FLOOR     "+strValeur+" : "+valeur.toString());

    strValeur = "0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_HALF_UP);
    System.out.println("ROUND_HALF_UP    "+strValeur+" :  "+valeur.toString());

    strValeur = "0.225";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_HALF_UP);
    System.out.println("ROUND_HALF_UP    "+strValeur+" :  "+valeur.toString());

    strValeur = "0.225";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_HALF_DOWN);
    System.out.println("ROUND_HALF_DOWN  "+strValeur+" :  "+valeur.toString());

    strValeur = "0.226";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_HALF_DOWN);
    System.out.println("ROUND_HALF_DOWN  "+strValeur+" :  "+valeur.toString());

    strValeur = "0.215";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_HALF_EVEN);
    System.out.println("ROUND_HALF_EVEN  "+strValeur+" :  "+valeur.toString());

    strValeur = "0.225";

    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_HALF_EVEN);
    System.out.println("ROUND_HALF_EVEN  "+strValeur+" :  "+valeur.toString());

    strValeur = "0.222";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_UP);
    System.out.println("ROUND_UP         "+strValeur+" :  "+valeur.toString());

    strValeur = "0.226";
    valeur = (new BigDecimal(strValeur)).setScale(2, BigDecimal.ROUND_UP);
    System.out.println("ROUND_UP         "+strValeur+" :  "+valeur.toString());
  }
}
```

Résultat :

```
ROUND_CEILING    0.222 :  0.23
ROUND_CEILING   -0.222 : -0.22
ROUND_DOWN       0.222 :  0.22
ROUND_DOWN       0.228 :  0.22
ROUND_DOWN      -0.228 : -0.22
ROUND_FLOOR      0.222 :  0.22
ROUND_FLOOR     -0.222 : -0.23
ROUND_HALF_UP    0.222 :  0.22
ROUND_HALF_UP    0.225 :  0.23
ROUND_HALF_DOWN  0.225 :  0.22
ROUND_HALF_DOWN  0.226 :  0.23
ROUND_HALF_EVEN  0.215 :  0.22
ROUND_HALF_EVEN  0.225 :  0.22
ROUND_UP         0.222 :  0.23
ROUND_UP         0.226 :  0.23
```

Le mode d'arrondi doit aussi être précisé lors de l'utilisation de la méthode divide().

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal6 {
  public static void main(String[] args) {
    BigDecimal valeur = new BigDecimal("1");
```

```
    System.out.println(valeur.divide(new BigDecimal("3")));
  }
}
```

Résultat :

```
Exception in thread "main" java.lang.ArithmeticException:
Non-terminating decimal expansion; no exact representable decimal result.
        at java.math.BigDecimal.divide(BigDecimal.java:1514)
        at com.jmdoudoux.test.bigdecimal.CalculBigDecimal6.main(CalculBigDecimal6.java:9)
```

Le même exemple en précisant le mode d'arrondi fonctionne parfaitement.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal6 {
  public static void main(String[] args) {
    BigDecimal valeur = new BigDecimal("1");
    System.out.println(valeur.divide(new BigDecimal("3"),4,BigDecimal.ROUND_HALF_DOWN));
  }
}
```

Résultat :

```
0.3333
```

La précision et le mode d'arrondi doivent être choisis avec attention parce que leur choix peut avoir de grandes conséquences sur les résultats de calculs notamment si le résultat final est constitué de multiples opérations. Dans ce cas, il est préférable de garder la plus grande précision durant les calculs et de n'effectuer l'arrondi qu'à la fin.

Il faut être vigilent lors de la comparaison entre deux objets de type BigDecimal. La méthode equals() compare les valeurs mais en tenant compte de la précision. Ainsi, il est préférable d'utiliser la méthode compareTo() qui n'effectue la comparaison que sur la valeur.

Exemple :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.BigDecimal;

public class CalculBigDecimal8 {

  public static void main(String[] args) {
    BigDecimal valeur1 = new BigDecimal("10.00");
    BigDecimal valeur2 = new BigDecimal("10.0");

    System.out.println("valeur1.equals(valeur2) = "+valeur1.equals(valeur2));
    System.out.println("valeur1.compareTo(valeur2) = "+(valeur1.compareTo(valeur2)==0));
  }
}
```

Résultat :

```
valeur1.equals(valeur2) = false
valeur1.compareTo(valeur2) = true
```

La méthode compareTo() renvoie 0 si les deux valeurs sont égales, renvoie -1 si la valeur de l'objet fourni en paramètre est plus petite et renvoie 1 si la valeur de l'objet fourni en paramètre est plus grande.

Il est possible de passer en paramètre de la méthode format() de la classe NumberFormat un objet de type BigDecimal : attention dans ce cas, le nombre de décimales est limité à 16.

Exemple formatage d'un BigDecimal avec un format monétaire :

```
package com.jmdoudoux.test.bigdecimal;

import java.math.*;
import java.text.*;
import java.util.*;

public class CalculBigDecimal9 {

  public static void main(String[] args) {
    BigDecimal payment = new BigDecimal("1234.567");
    NumberFormat n = NumberFormat.getCurrencyInstance(Locale.FRANCE);
    String s = n.format(payment);
    System.out.println(s);
  }
}
```

Résultat :

```
1 234,57 €
```

La mise en oeuvre de la classe BigDecimal est plutôt fastidieuse comparée à d'autres langages qui proposent un support natif d'un type de données décimal mais elle permet d'effectuer des calculs précis.

L'utilisation de la classe BigDecimal n'est recommandée que si une précision particulière est nécessaire car sa mise en oeuvre est coûteuse.

TP : conversion Celsius - Fahrenheit

Voilà un petit TP qui va vous permettre de mettre en œuvre toutes les notions que vous avez vues jusqu'ici :

- les variables ;
- les conditions ;
- les boucles ;
- votre génial cerveau.

Accrochez-vous, car je vais vous demander de penser à des tonnes de choses, et vous serez tout seuls. Lâchés dans la nature… Mais non je plaisante, je vais vous guider un peu.

Élaboration

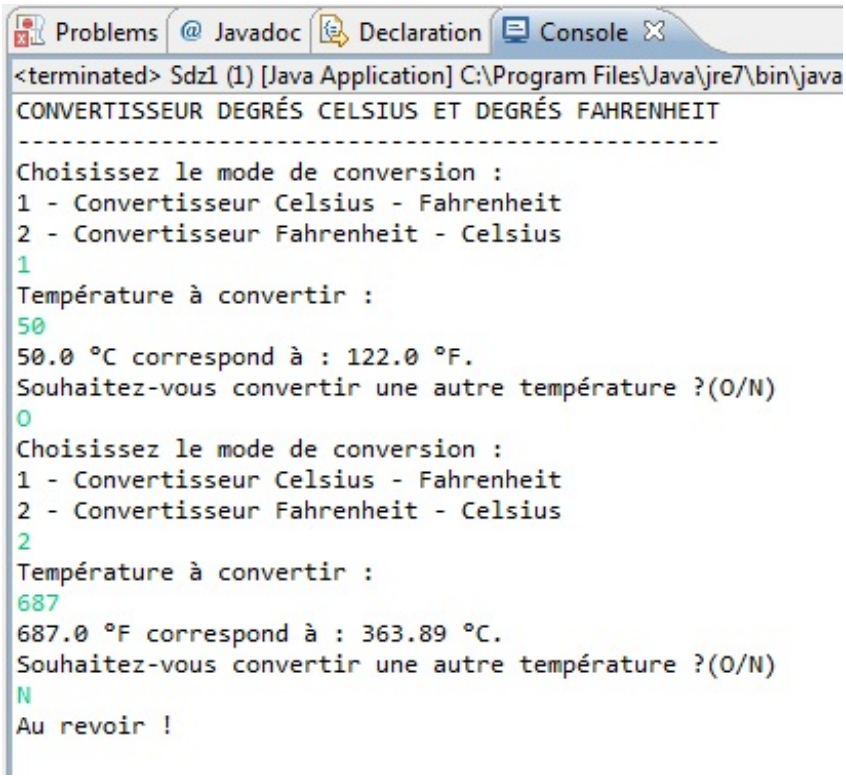
Voici les caractéristiques du programme que nous allons devoir réaliser :

- le programme demande quelle conversion nous souhaitons effectuer, Celsius vers Fahrenheit ou l'inverse ;
- on n'autorise que les modes de conversion définis dans le programme (un simple contrôle sur la saisie fera l'affaire) ;
- enfin, on demande à la fin à l'utilisateur s'il veut faire une nouvelle conversion, ce qui signifie que l'on doit pouvoir revenir au début du programme !

Avant de vous lancer dans la programmation à proprement parler, je vous conseille fortement de réfléchir à votre code… sur papier. Réfléchissez à ce qu'il vous faut comme nombre de variables, les types de variables, comment va se dérouler le programme, les conditions et les boucles utilisées.

À toutes fins utiles, voici la formule de conversion pour passer des degrés Celsius en degrés Fahrenheit : $F = \frac{9}{5} \times C + 32$; pour l'opération inverse, c'est comme ceci : $C = \frac{(F - 32) \times 5}{9}$.

La figure suivante est un aperçu de ce que je vous demande.



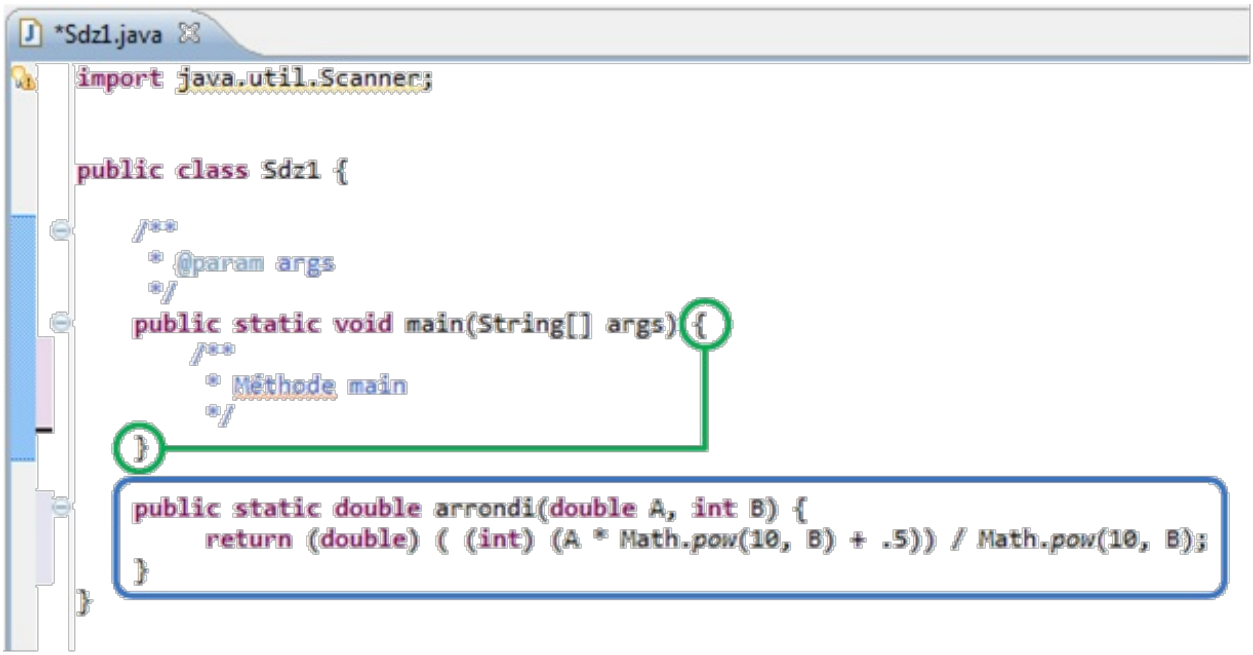
Rendu du TP

Je vais également vous donner une fonction toute faite qui vous permettra éventuellement d'arrondir vos résultats. Vous pouvez très bien ne pas vous en servir. Pour ceux qui souhaitent tout de même l'utiliser, la voici :

Code : Java

```
public static double arrondi(double A, int B) {
  return (double) ( (int) (A * Math.pow(10, B) + .5)) / Math.pow(10,
B);
}
```

Elle est à placer entre les deux accolades fermantes de votre classe, comme à la figure suivante.



Emplacement de la fonction

Voici comment utiliser cette fonction : imaginez que vous avez la variable `faren` à arrondir, et que le résultat obtenu est enregistré dans une variable `arrondFaren` ; vous procéderez comme suit :

Code : Java

```
arrondFaren = arrondi(faren,1); //Pour un chiffre après la virgule
arrondFaren = arrondi(faren, 2);//Pour deux chiffres après la
virgule, etc.
```

Quelques dernières recommandations : essayez de bien indenter votre code ! Prenez votre temps. Essayez de penser à tous les cas de figure. Maintenant à vos papiers, crayons, neurones, claviers… et bon courage !

Correction

STOP ! C'est fini ! Il est temps de passer à la correction de ce premier TP. Ça va ? Pas trop mal à la tête ? Je me doute qu'il a dû y avoir quelques tubes d'aspirine vidés. Mais vous allez voir qu'en définitive, ce TP n'était pas si compliqué que ça.

Surtout, n'allez pas croire que ma correction est parole d'évangile. Il y avait différentes manières d'obtenir le même résultat. Voici tout de même une des solutions possibles.

Code : Java

```
import java.util.Scanner;

class Sdz1 {
  public static void main(String[] args) {
    //Notre objet Scanner
    Scanner sc = new Scanner(System.in);

    //initialisation des variables
```

```
    double aConvertir, convertit=0;
    char reponse=' ', mode = ' ';

    System.out.println("CONVERTISSEUR DEGRÉS CELSIUS ET DEGRÉS
FAHRENHEIT");
    System.out.println("-----------------------------------------
-----");

    do{//tant que reponse = O //boucle principale

      do{//tant que reponse n'est pas O ou N
        mode = ' ';
        System.out.println("Choisissez le mode de conversion : ");
        System.out.println("1 - Convertisseur Celsius -
Fahrenheit");
        System.out.println("2 - Convertisseur Fahrenheit - Celsius
");
        mode = sc.nextLine().charAt(0);

        if(mode != '1' && mode != '2')
          System.out.println("Mode inconnu, veuillez réitérer votre
choix.");

      }while (mode != '1' && mode != '2');

      //saisie de la température à convertir
      System.out.println("Température à convertir :");
      aConvertir = sc.nextDouble();
      //Pensez à vider la ligne lue
      sc.nextLine();

      //Selon le mode, on calcule différemment et on affiche le
résultat
      if(mode == '1'){
        convertit = ((9.0/5.0) * aConvertir) + 32.0;
        System.out.print(aConvertir + " °C correspond à : ");
        System.out.println(arrondi(convertit, 2) + " °F.");
      }
      else{
        convertit = ((aConvertir - 32) * 5) / 9;
        System.out.print(aConvertir + " °F correspond à : ");
        System.out.println(arrondi(convertit, 2) + " °C.");
      }

      //On invite l'utilisateur à recommencer ou à quitter
      do{
        System.out.println("Souhaitez-vous convertir une autre
température ?(O/N)");
        reponse = sc.nextLine().charAt(0);

      }while(reponse != 'O' && reponse != 'N');

    }while(reponse == 'O');

  System.out.println("Au revoir !");

  //Fin de programme
  }

  public static double arrondi(double A, int B) {
    return (double) ( (int) (A * Math.pow(10, B) + .5)) /
Math.pow(10, B);
  }
}
```

Explications concernant ce code

- Tout programme commence par une phase de déclaration des variables.
- Nous affichons le titre de notre programme.
- Ensuite, vous voyez deux `do {` consécutifs correspondant à deux conditions à vérifier :
    - la volonté de l'utilisateur d'effectuer une nouvelle conversion ;
    - la vérification du mode de conversion.
- Nous affichons les renseignements à l'écran, et récupérons la température à convertir pour la stocker dans une variable.
- Selon le mode sélectionné, on convertit la température et on affiche le résultat.
- On invite l'utilisateur à recommencer.
- Fin du programme !

Ce programme n'est pas parfait, loin de là. La vocation de celui-ci était de vous faire utiliser ce que vous avez appris, et je pense qu'il remplit bien sa fonction. J'espère que vous avez apprécié ce TP. Je sais qu'il n'était pas facile, mais avouez-le : il vous a bien fait utiliser tout ce que vous avez vu jusqu'ici !