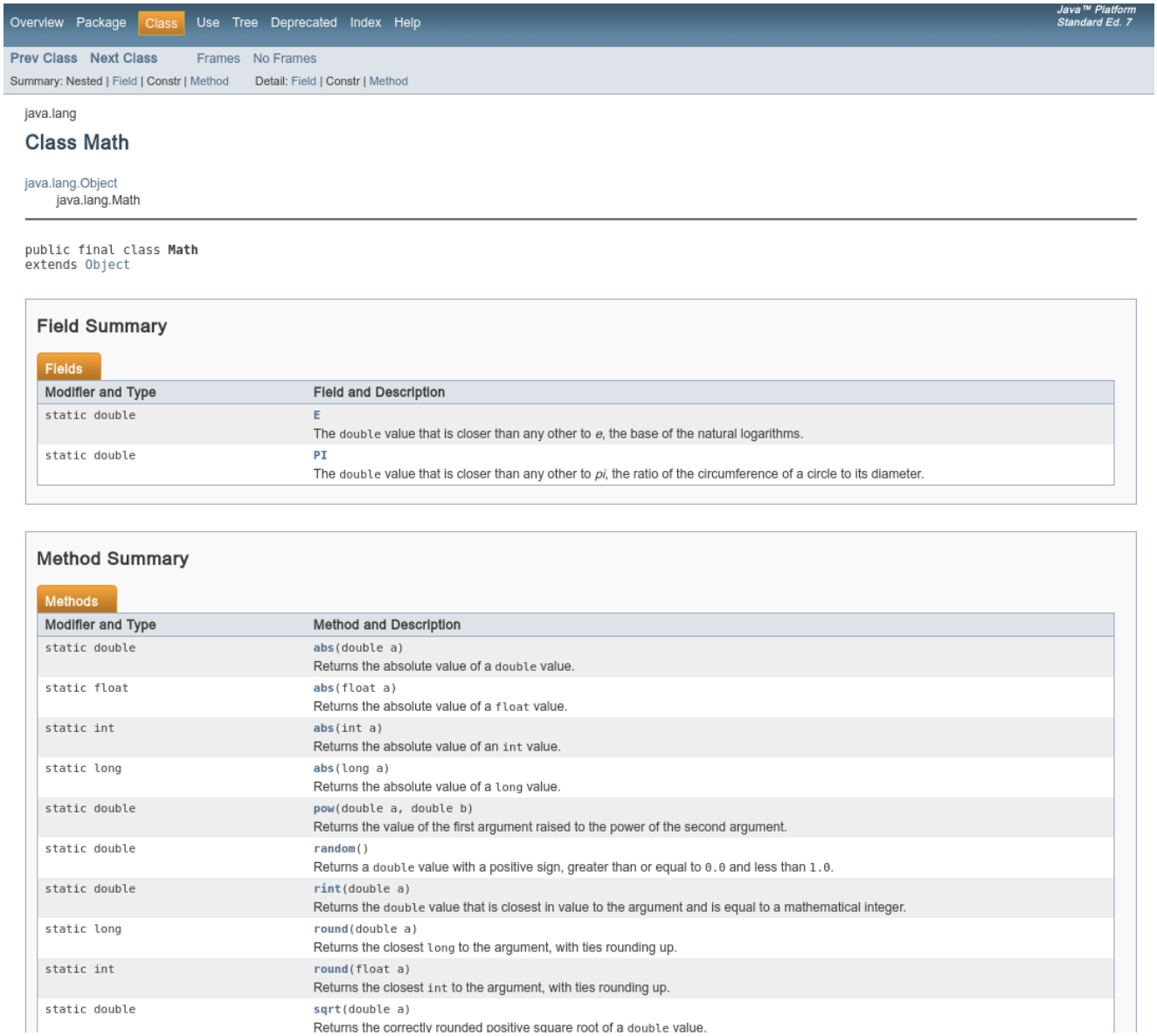
Feuille de TD 4

Utiliser une classe bibliothèque

Voici un extrait de la documentation de la classe `Math`



Overview Package Class Use Tree Deprecated Index Help — Java™ Platform Standard Ed. 7

Prev Class Next Class Frames No Frames

Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method

java.lang

Class Math

java.lang.Object
    java.lang.Math

```
public final class Math
extends Object
```

Field Summary

Fields

Modifier and Type	Field and Description
static double	E The double value that is closer than any other to e, the base of the natural logarithms.
static double	PI The double value that is closer than any other to pi, the ratio of the circumference of a circle to its diameter.

Method Summary

Methods

Modifier and Type	Method and Description
static double	abs(double a) Returns the absolute value of a double value.
static float	abs(float a) Returns the absolute value of a float value.
static int	abs(int a) Returns the absolute value of an int value.
static long	abs(long a) Returns the absolute value of a long value.
static double	pow(double a, double b) Returns the value of the first argument raised to the power of the second argument.
static double	random() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	rint(double a) Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	round(double a) Returns the closest long to the argument, with ties rounding up.
static int	round(float a) Returns the closest int to the argument, with ties rounding up.
static double	sqrt(double a) Returns the correctly rounded positive square root of a double value.

Et voici le code d'une classe `Point`

```java
/*
 * Cette classe permet de modéliser un point du plan à l'aide de ses
 * coordonnées (x, y) dans un repère orthonormé
 */
public class Point{
   private double x;
   private double y;
   public Point(double x, double y){
      this.x = x;
      this.y = y;
   }
   /**
    * Crée un point avec des coordonnées aléatoires
    */
   public Point(){
      // A COMPLETER (1)
   }
   public double getX(){ return this.x;}
   public double getY(){ return this.y;}
   public void setX(double newX){this.x = newX;}
   public void setY(double newY){this.y = newY;}
   public String toString(){ return "("+this.x+", "+this.y+")"; }

   /** Deux points (x1, y1) et (x2, y2) sont égaux s'ils ont les mêmes
    * coordonnées, à 0.0001 près c'est à dire si :
    * |x1-x2| < 0.0001 et |y1-y2| < 0.0001
    */
   public boolean egal(Point autrePoint){
      // A COMPLETER (2)
      return true;
   }
   /** la distance entre deux points (x1, y1) et (x2, y2) est la valeur
    * d = racineCarré( (x1-x2)² + (y1-y2)² )
    * */
   public double distance(Point autrePoint){
      // A COMPLETER (3)
      return 0;
   }
}
```

Complétez le code de cette classe `Point`

Ecrire le code d'une classe bibliothèque

Définissez une classe `BibliothequeTableau` contenant :

1. une méthode `identiques()` qui prend en paramètres deux listes d'entiers et renvoie un booléen indiquant si ces deux listes ont les mêmes éléments, dans le même ordre.

   Par exemple `[2, 3, 3, 5]` et `[2, 3, 3, 5]` sont identiques, mais `[2, 3, 3, 5]` et `[2, 3, 5]` ne le sont pas.
2. une méthode `inverse` qui prend en paramètre une liste d'entiers et renvoie la liste inversée.

   Par exemple, l'inverse de `[2, 3, 3, 5]` est `[5, 3, 3, 5]`.
3. **En bonus** une méthode `fusion` qui prend en paramètre deux listes d'entiers triées et renvoie une nouvelle liste obtenue en effectuant la fusion (triée) de ces deux listes, sans utiliser la méthode *sort() (bonus)*,

   Par exemple, la fusion entre les `[2, 3, 3, 5, 11, 17, 22]` et `[0, 1, 2, 7, 8]` est la liste `[0, 1, 2, 2, 3, 3, 5, 7, 8, 11, 17, 22]`

Un peu d'algo : utiliser la méthode `sort()`

On vous rappelle qu'il est possible de trier (facilement) une ArrayList (entre autres) en utilisant la méthode `sort()` de la façon suivante :

```
import java.util.Collections;
import java.util.ArrayList;
import java.util.List;
class Executable{
     public static void main(String[] args) {
        List<Integer> liste = new ArrayList<>();
        liste.add(4);
        liste.add(-1);
        liste.add(6);
        liste.add(2);
        Collections.sort(liste);
        System.out.println(liste);
        // Affiche [-1, 2, 4, 6]
     }
 }
```

Compléter le code le la classe `BibliothequeTableau` avec les méthodes suivantes. Vous ferez en sorte que vos algorithmes soient efficaces.

1. Ecrire le code d'une méthode `elementsEnPlusieursExemplaires` qui détermine les éléments qui apparaissent plusieurs fois dans une liste.

   Par exemple, les éléments qui apparaissent plusieurs fois dans la liste `[1, 6, 2, -4, 1, 9, 6]` sont `1` et `6`
2. Ecrire le code d'une méthode `mediane` qui détermine l'élément médian de la liste (celui qui a autant d'éléments plus grands que plus petits)

   Par exemple, la médiane de la liste `[1, 6, 2, -4, 1, 9, 6]` est `2` : il y a trois éléments plus petits (`-4` , `1` et `1`) et trois éléments plus grands (`6` , `6` et `9`).

   Dans le cas d'une liste de taille paire, on décide de "privilégier" les petits. Ainsi, la médiane de la liste `[1, 6, 2, -4, 1, 9]` est `2` : il y a trois éléments plus petits (`-4` , `1` et `1`) et deux éléments plus grands (`6` et `9`).
3. **En bonus** Ecrire le code d'une méthode `existeSommeNulle` qui détermine s'il existe dans une liste d'entiers deux entiers dont la somme est nulle.
4. **En bonus** Ecrire le code d'une méthode `plusPetiteDifference` qui trouve dans une liste d'entiers les deux entiers dont la différence est la plus petite possible.
5. **En bonus** Ecrire le code d'une méthode `elementsEnCommun` qui détermine les éléments communs à deux listes.