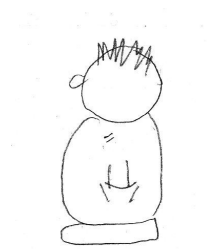


Java à l'école



Proposition de solutions

11TG, Première année de programmation



Version 1.3.4d, Jens Getreu

Table des matières

1 Introduction.....	2
Class Salutation.....	6
2 Permutation.....	8
Class Coordonn?s.....	11
3 Mini-calculatrice.....	14
Class Calculatrice.....	17
4 Nombre malin.....	20
Class NombreMalin.....	25
5.1 Année bissextile.....	29
Variante : structure alternative imbriquée.....	29
Class Ann?.....	32
5.2 Année bissextile.....	34
Variante : opérateurs logiques.....	34
Class Ann?.....	38
6 Équation du second degré.....	40
Class Polyn?e.....	44
7 Vérifier un mot de passe.....	48
Class MotDePasse.....	52
8 Simplifier une fraction.....	55
Class Fraction.....	59
9 Nombre premier.....	62
Class NombreEntier.....	65
10 Jeu « deviner nombre ».....	67
Class NombreSecret.....	71
11 Jeu du loup.....	74
Class Piste.....	79
12 Minimum-Maximum.....	83
Class NombrePositif.....	87
13 Sapin de Noël.....	91
Class Sapin.....	94
14 Décomposition en produit de facteurs _premiers.....	96
Class NombrePremier.....	100

1 Introduction

InterfaceUtilisateur.java

```
1  /**      INTRODUCTION
2      *
3      *
4      * Interface utilisateur pour la classe Salutation
5      *
6      * @author Jens Getreu
7      * @version 1.0.2
8      * */
9  import java.util.*;
10 public class InterfaceUtilisateur
11 {
12     public static void main(String[] args)
13     {
14         // saisie
15         Scanner sc = new Scanner(System.in);
16
17         System.out.println("SALUTATION");
18         System.out.print("Veuillez saisir votre nom : ");
19
20         String nom = sc.nextLine();
21         //traitement
22         Salutation s = new Salutation(nom);
23         // sortie
24         System.out.println(s.saluer() );
25     }
26 }
27
```

```
1  /**      INTRODUCTION
2  *
3  *
4  * Cette classe représente une salutation
5  *
6  * @author Jens Getreu
7  * @version 1.0.2
8  */
9  public class Salutation
10 {
11     /**
12     * Déclaration et initialisation d'une constante: un mot d'accueil.
13     */
14
15     public static final String MOT_ACCUEIL="Bonjour ";
16     //public static final MOT_ACCUEIL="Salut ";
17
18     /**
19     * Le nom de la personne à saluer
20     */
21     protected String nom;
22
23     /**
24     * Constructeur pour objets de la classe salutation
25     * @param n un nom
26     */
27     public Salutation(String n)
28     {
29         nom = n;
30     }
31
32     /**
33     * La méthode saluer() renvoie un bonjour
34     * personnalisé.
35     * @return Bonjour personnalisé
36     */
37     public String saluer()
38     {
39         return MOT_ACCUEIL + nom + ". Comment allez-vous ?";
40     }
41 }
42
```


Class Salutation

java.lang.Object



```
public class Salutation extends java.lang.Object
```

INTRODUCTION Cette classe représente une salutation

Version:

1.0.2

Author:

Jens Getreu

Field Summary

static java.lang.String	<u>MOT_ACCUEIL</u> Déclaration et initialisation d'une constante: un mot d'accueil.
protected java.lang.String	<u>nom</u> Le nom de la personne à saluer

Constructor Summary

<u>Salutation</u> (java.lang.String n) Constructeur pour objets de la classe salutation
--

Method Summary

java.lang.String	<u>saluer</u> () La méthode saluer() renvoie un bonjour personnalisé.
------------------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

MOT_ACCUEIL

```
public static final java.lang.String MOT_ACCUEIL
```

Déclaration et initialisation d'une constante: un mot d'accueil.

See Also:

[Constant Field Values](#)

nom

```
protected java.lang.String nom
```

Le nom de la personne à saluer

Constructor Detail

Salutation

```
public Salutation(java.lang.String n)
```

Constructeur pour objets de la classe salutation

Parameters:

n - un nom

Method Detail

saluer

```
public java.lang.String saluer()
```

La méthode saluer() renvoie un bonjour personnalisé.

Returns:

Bonjour personnalisé

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

2 Permutation

InterfaceUtilisateur.java

```
1  /**      PERMUTATION
2  *
3  * Basé sur Les Générations Pascal Chap. 8.6 Ex. 6
4  *
5  * @author Jens Getreu
6  * @version 1.0.3
7  * */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14         int x;
15         int y;
16
17         //saisie
18         System.out.println("PERMUTATION ");
19         System.out.println();
20
21         System.out.print("Veuillez saisir un nombre entier x : ");
22         x=sc.nextInt();
23
24         System.out.print("Veuillez saisir un nombre entier y : ");
25         y=sc.nextInt();
26         System.out.println();
27
28         Coordonnées c = new Coordonnées(x,y);
29
30         //sortie
31         System.out.println("Les variables avant la permutation : ");
32         System.out.println("x="+c.getCoordX()+" y="+c.getCoordY());
33
34         //traitement
35         c.permuter();
36
37         //sortie
38         System.out.println("Les variables après la permutation : ");
39         System.out.println("x="+c.getCoordX()+" y="+c.getCoordY());
40
41     }
42 }
43
44
```

```

1  /**      PERMUTATION
2  *
3  * Cette classe représente une paire de coordonnées (x,y)
4  *
5  * @author Jens Getreu
6  * @version 1.0.4
7  */
8  public class Coordonnées
9  {
10     /**
11      * La coordonnée x.
12      */
13     protected int coordX;
14     /**
15      * La coordonnée y.
16      */
17     protected int coordY;
18
19     /**
20      * Constructeur pour objets de la classe Coordonnées
21      * @param coordonné x
22      * @param coordonné y
23      */
24     public Coordonnées(int a, int b)
25     {
26         // initialiser
27         coordX=a;
28         coordY=b;
29     }
30
31     /**
32      * La méthode permuter() échange les contenus des
33      * variables x et y.
34      */
35     public void permuter()
36     {
37         int tmp = coordX;
38         coordX = coordY;
39         coordY = tmp;
40     }
41
42     /**
43      * La méthode getX() renvoie la coordonnée x
44      * @return coordonnée x
45      */
46     public int getCoordX()
47     {
48         return coordX;
49     }
50
51     /**
52      * La méthode getY() renvoie la coordonnée y
53      * @return coordonnée y
54      */
55     public int getCoordY()
56     {
57         return coordY;
58     }
59 }
60

```

Class Coordonnées

java.lang.Object



```
public class Coordonnées extends java.lang.Object
```

PERMUTATION Cette classe représente une paire de coordonnées (x,y)

Version:

1.0.4

Author:

Jens Getreu

Field Summary

protected int	coordX La coordonnée x.
protected int	coordY La coordonnée y.

Constructor Summary

Coordonnées (int a, int b) Constructeur pour objets de la classe Coordonnées

Method Summary

int	getCoordX () La méthode getX() renvoie la coordonnée x
int	getCoordY () La méthode getY() renvoie la coordonnée y
void	permuter () La méthode permuter() échange les contenus des variables x et y.

Methods inherited from class java.lang.Object

```
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait
```

Field Detail

coordX

protected int **coordX**

La coordonnée x.

coordY

protected int **coordY**

La coordonnée y.

Constructor Detail

Coordonnées

```
public Coordonnées(inta,  
                    intb)
```

Constructeur pour objets de la classe Coordonnées

Parameters:

coordonné - x
coordonné - y

Method Detail

getCoordX

```
public int getCoordX()
```

La méthode getX() renvoie la coordonnée x

Returns:

coordonnée x

getCoordY

```
public int getCoordY()
```

La méthode getY() renvoie la coordonnée y

Returns:

coordonnée y

permuter

```
public void permuter()
```

La méthode `permuter()` échange les contenus des variables x et y.

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

PREV CLASS [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: NESTED | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

3 Mini-calcolatrice

InterfaceUtilisateur.java

```

1  /**          MINICALCULATRICE
2  *
3  * Basé sur : Les Générations Pascal Chap. 12.6 Ex. 11
4  *
5  * @author Jens Getreu
6  * @version 1.0.1
7  */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14
15         System.out.println("MINICALCULATRICE ");
16
17         System.out.print("Veuillez saisir un nombre réel x : ");
18         float x=sc.nextFloat();
19
20         System.out.print("Veuillez saisir un nombre réel y : ");
21         float y=sc.nextFloat();
22
23         System.out.print("Veuillez saisir un opérateur '+' ou '-': ");
24         String op=sc.next();
25
26         Calculatrice c = new Calculatrice(x);
27
28         if (op.equals("+"))
29         {
30             System.out.println("x+y = "+c.additionner(y));
31         }
32         else
33         {
34             if (op.equals("-"))
35             {
36                 System.out.println("x-y = "+c.soustraire(y));
37             }
38             else
39             {
40                 System.out.println("L'opérateur '"+ op +"' n'est pas défini.");
41             }
42         }
43         System.out.println();
44     }
45 }
46

```

```

1  /**      MINICALCULATRICE
2  *
3  * Réalise des opérations de calcul
4  *
5  * @author Jens Getreu
6  * @version 1.0.3
7  */
8  public class Calculatrice
9  {
10     /**
11     * L'opérande.
12     */
13     protected double mémoireInterne;
14
15     /**
16     * Constructeur de la classe Calculatrice initialisant la
17     * mémoire interne à 0
18     */
19     public Calculatrice()
20     {
21         mémoireInterne = 0;
22     }
23
24     /**
25     * Construire de la classe Calculatrice initialisant
26     * la mémoire interne à la valeur
27     * @param valeur valeur initiale de la mémoire interne
28     */
29     public Calculatrice(double a)
30     {
31         mémoireInterne=a;
32     }
33
34     /**
35     * La méthode additionner() ajoute y à la mémoire interne
36     *
37     * @param y    valeur
38     * @return     la somme de la mémoire interne et y
39     */
40     public double additionner(double y)
41     {
42         mémoireInterne = mémoireInterne + y;
43         return mémoireInterne;
44     }
45
46     /**
47     * La méthode soustraire() soustrait y de la mémoire interne
48     *
49     * @param y    valeur
50     * @return     la soustraction de la mémoire interne et y
51     */
52     public double soustraire(double y)
53     {
54         mémoireInterne = mémoireInterne - y;
55         return mémoireInterne;
56     }
57 }
58

```

Class Calculatrice

java.lang.Object



```
public class Calculatrice extends java.lang.Object
```

MINICALCULATRICE Réalise des opérations de calcul

Version:

1.0.3

Author:

Jens Getreu

Field Summary

protected	mémoireInterne
double	L'opérande.

Constructor Summary

[Calculatrice](#) ()

Constructeur de la classe Calculatrice initialisant la mémoire interne à 0

[Calculatrice](#) (double a)

Constructuer de la classe Calculatrice initialisant la mémoire interne à la valeur

Method Summary

double	additionner (double y)
	La méthode additionner() ajoute y à la mémoire interne

double	soustraire (double y)
	La méthode soustraire() soustrait y de la mémoire interne

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

mémoireInterne

protected double **mémoireInterne**

L'opérande.

Constructor Detail

Calculatrice

public **Calculatrice**()

Constructeur de la classe Calculatrice initialisant la mémoire interne à 0

Calculatrice

public **Calculatrice**(double a)

Constructeur de la classe Calculatrice initialisant la mémoire interne à la valeur

Parameters:

valeur - valeur initiale de la mémoire interne

Method Detail

additionner

public double **additionner**(double y)

La méthode additionner() ajoute y à la mémoire interne

Parameters:

y - valeur

Returns:

 la somme de la mémoire interne et y

soustraire

public double **soustraire**(double y)

La méthode soustraire() soustrait y de la mémoire interne

Parameters:

y - valeur

Returns:

 la soustraction de la mémoire interne et y

4 Nombre malin

InterfaceUtilisateur.java

```

1  /**      NOMBRE MALIN
2      *
3      * @author Jens Getreu
4      * @version 1.0.2
5      */
6  import java.util.*;
7  public class InterfaceUtilisateur
8  {
9      public static void main(String[] args)
10     {
11         Scanner sc = new Scanner(System.in);
12
13         System.out.println("NOMBRE MALIN ");
14         System.out.print("Veuillez saisir un nombre réel : ");
15         double x=sc.nextDouble();
16
17         NombreMalin nm = new NombreMalin(x);
18
19         System.out.println("Bonjour, je suis un nombre malin. ");
20         System.out.println("Je me connais bien: ");
21
22         if (nm.isPositif())
23         {
24             System.out.println("Je suis positif.");
25         }
26
27         if (nm.isNégatif())
28         {
29             System.out.println("Je suis négatif.");
30         }
31
32         if (nm.isPair())
33         {
34             System.out.println("Je suis pair.");
35         }
36
37         if (nm.isImpair())
38         {
39             System.out.println("Je suis impair.");
40         }
41
42         if (nm.isJeune())
43         {
44             System.out.println("Je suis jeune (<30).");
45         }
46
47         if (nm.isVieux())
48         {
49             System.out.println("Je suis vieux (>=30).");
50         }
51
52         if (nm.isDécimal())
53         {
54             System.out.println("J'ai une partie fractionnaire.");
55         }
56
57         if (nm.isEntier())
58         {
59             System.out.println("Je suis un nombre entier.");
60         }
61         System.out.println();
62     }

```

63 }
64

```

1  /**      NOMBRE MALIN
2  *
3  * Cette classe représente un nombre
4  * capable de renseigner sur soi-même
5  *
6  * @author Jens Getreu
7  * @version 1.0.3
8  */
9  public class NombreMalin
10 {
11     /**
12     * Le nombre réel.
13     */
14     protected double nombre;
15
16     /**
17     * Constructeur pour objets de la classe
18     * NombreMalin.
19     * @param nombre nombre
20     */
21     public NombreMalin(double a)
22     {
23         // initialiser
24         nombre=a;
25     }
26
27     /**
28     * La méthode isPositif() permet de tester si
29     * le nombre est positif
30     * @return vrai pour un nombre >=0
31     */
32     public boolean isPositif()
33     {
34         return nombre >= 0;
35     }
36
37     /**
38     * La méthode isNégatif() permet de tester si
39     * le nombre est négatif
40     * @return vrai pour un nombre < 0
41     */
42     public boolean isNégatif()
43     {
44         return nombre < 0;
45     }
46
47     /**
48     * La méthode isPair() permet de tester si
49     * le nombre est pair
50     * @return vrai pour un nombre pair
51     */
52     public boolean isPair()
53     {
54         return isEntier() && (nombre%2 == 0);
55     }
56
57
58     /**
59     * La méthode isPair() permet de tester si
60     * le nombre est impair
61     * @return vrai pour un nombre impair
62     */

```

```

63     public boolean isImpair()
64     {
65         return isEntier() && (nombre%2 != 0);
66     }
67
68
69     /**
70      * La méthode isJeune() permet de tester si
71      * le nombre est inférieur à 30
72      * @return vrai pour un nombre < 30
73      */
74     public boolean isJeune()
75     {
76         return nombre<30;
77     }
78
79     /**
80      * La méthode isVieux() permet de tester si
81      * le nombre est supérieur ou égal à 30
82      * @return vrai pour un nombre >= 30
83      */
84     public boolean isVieux()
85     {
86         return nombre>=30;
87     }
88
89     /**
90      * La méthode isRéal() permet de tester si
91      * le nombre est entier
92      * @return vrai pour un nombre entier
93      */
94     public boolean isEntier()
95     {
96         return nombre==(int)nombre;
97     }
98
99     /**
100      * La méthode isDécimal() permet de tester si
101      * le nombre est un nombre décimal
102      * @return vrai pour un nombre décimal
103      */
104     public boolean isDécimal()
105     {
106         return nombre!=(double)(int)nombre;
107     }
108 }
109

```

Class NombreMalin

java.lang.Object



```
public class NombreMalin extends java.lang.Object
```

NOMBRE MALIN Cette classe représente un nombre capable de renseigner sur soi-même

Version:

1.0.3

Author:

Jens Getreu

Field Summary

protected double	nombre Le nombre réel.
---------------------	---

Constructor Summary

NombreMalin (double a) Constructeur pour objets de la classe NombreMalin.	
--	--

Method Summary

boolean	isDécimal () La méthode isDécimal() permet de tester si le nombre est un nombre décimal
boolean	isEntier () La méthode isRéal() permet de tester si le nombre est entier
boolean	isImpair () La méthode isPair() permet de tester si le nombre est impair
boolean	isJeune () La méthode isJeune() permet de tester si le nombre est inférieur à 30
boolean	isNégatif () La méthode isNégatif() permet de tester si le nombre est négatif

boolean	<code>isPair()</code> La méthode <code>isPair()</code> permet de tester si le nombre est pair
boolean	<code>isPositif()</code> La méthode <code>isPositif()</code> permet de tester si le nombre est positif
boolean	<code>isVieux()</code> La méthode <code>isVieux()</code> permet de tester si le nombre est supérieur ou égal à 30

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`

Field Detail

nombre

protected double **nombre**

Le nombre réel.

Constructor Detail

NombreMalin

public **NombreMalin**(double a)

Constructeur pour objets de la classe `NombreMalin`.

Parameters:

nombre - nombre

Method Detail

isDécimal

public boolean **isDécimal()**

La méthode `isDécimal()` permet de tester si le nombre est un nombre décimal

Returns:

vrai pour un nombre décimal

isEntier

public boolean **isEntier()**

La méthode `isRéal()` permet de tester si le nombre est entier

Returns:
vrai pour un nombre entier

isImpair

public boolean **isImpair()**

La méthode isPair() permet de tester si le nombre est impair

Returns:
vrai pour un nombre impair

isJeune

public boolean **isJeune()**

La méthode isJeune() permet de tester si le nombre est inférieur à 30

Returns:
vrai pour un nombre < 30

isNégatif

public boolean **isNégatif()**

La méthode isNégatif() permet de tester si le nombre est négatif

Returns:
vrai pour un nombre < 0

isPair

public boolean **isPair()**

La méthode isPair() permet de tester si le nombre est pair

Returns:
vrai pour un nombre pair

isPositif

public boolean **isPositif()**

La méthode isPositif() permet de tester si le nombre est positif

Returns:
vrai pour un nombre >=0

isVieux

public boolean **isVieux**()

La méthode isVieux() permet de tester si le nombre est supérieur ou égal à 30

Returns:

vrai pour un nombre ≥ 30

Package **Class** Tree Index Help

PREV CLASS NEXT CLASS

FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD

DETAIL: FIELD | CONSTR | METHOD

5.1 Année bissextile

Variante : structure alternative imbriquée

InterfaceUtilisateur.java

```
1  /**      ANNÉE BISSEXTILE
2  *      Variante : structure alternative imbriquée
3  *
4  *  Voir : Les Générations Pascal Chap. 12.6 Ex. 11
5  *
6  *  @author Jens Getreu
7  *  @version 1.0.2
8  *  */
9  import java.util.*;
10 public class InterfaceUtilisateur
11 {
12     public static void main(String[] args)
13     {
14         Scanner sc = new Scanner(System.in);
15
16         System.out.println("ANNÉE BISSEXTILE");
17
18         System.out.print("Veuillez saisir une année : ");
19         int a = sc.nextInt();
20
21         Année an = new Année(a);
22
23         if (an.isAnnéeBissextile() )
24         {
25             System.out.println("Il s'agit d'une année bissextile.");
26         }
27         else
28         {
29             System.out.println("Il ne s'agit pas d'une année bissextile.");
30         }
31         System.out.println();
32     }
33 }
34
```

```

1  /**      ANNÉE BISSEXTILE
2  *      Variante : structure alternative imbriquée
3  *
4  * Cette classe représente une année
5  *
6  * @author Jens Getreu
7  * @version 1.0.2
8  */
9  public class Année
10 {
11     /**
12     * L'an.
13     */
14     protected int an;
15
16     /**
17     * Constructeur.
18     * @param x année
19     */
20     public Année(int x)
21     {
22         an = x;
23     }
24
25     /**
26     * Cette méthode détermine si une année est bissextile ou non
27     *
28     * @return      vrai pour les années bissextiles
29     */
30     public boolean isAnnéeBissextile()
31     {
32         boolean b = false; // true si bissextile
33
34         if (an % 4 == 0)
35         {
36             if (an % 100 == 0)
37             {
38                 if (an % 400 == 0)
39                 {
40                     b = true;
41                 }
42                 else b = false;
43             }
44             else
45             {
46                 b = true;
47             }
48         }
49         else
50         {
51             b = false;
52         }
53         return b;
54     }
55 }
56

```

Class Année

java.lang.Object
└─ **Année**

```
public class Année extends java.lang.Object
```

ANNÉE BISSEXTILE Variante : structure alternative imbriquée Cette classe représente une année

Version:

1.0.2

Author:

Jens Getreu

Field Summary

protected int	an L'an.
------------------	-----------------------------

Constructor Summary

Année (int x) Constructeur.	
--	--

Method Summary

boolean	isAnnéeBissextile () Cette méthode détermine si une année est bissextile ou non
---------	--

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

an

protected int **an**

L'an.

Constructor Detail

Année

public **Année**(int x)

Constructeur.

Parameters:

x - année

Method Detail

isAnnéeBissextile

public boolean **isAnnéeBissextile**()

Cette méthode détermine si une année est bissextile ou non

Returns:

vrai pour les années bissextiles

[Package](#) **Class** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

5.2 Année bissextile

Variante : opérateurs logiques

InterfaceUtilisateur.java

```
1  /**      ANNÉE BISSEXTILE
2  *      Variante : opérateurs logiques
3  *
4  *  Les Générations Pascal Chap. 12.6 Ex. 11
5  *
6  *  @author Jens Getreu
7  *  @version 1.0.1
8  *  */
9  import java.util.*;
10 public class InterfaceUtilisateur
11 {
12     public static void main(String[] args)
13     {
14         Scanner sc = new Scanner(System.in);
15
16         System.out.println("ANNÉE BISSEXTILE");
17
18         System.out.print("Veuillez saisir une année : ");
19         int a = sc.nextInt();
20
21         Année an = new Année(a);
22
23         if (an.isAnnéeBissextile() )
24         {
25             System.out.println("Il s'agit d'une année bissextile.");
26         }
27         else
28         {
29             System.out.println("Il ne s'agit pas d'une année bissextile.");
30         }
31         System.out.println();
32     }
33 }
34
```

```

1  /**      ANNÉE BISSEXTILE
2  *      Variante : opérateurs logiques
3  *
4  * Cette classe représente une année
5  *
6  * @author Jens Getreu
7  * @version 1.0.3
8  */
9  public class Année
10 {
11     /**
12     * L'an.
13     */
14     protected int an;
15
16     /**
17     * Constructeur.
18     * @param x année
19     */
20     public Année(int x)
21     {
22         an = x;
23     }
24
25     /**
26     * Cette méthode détermine si une année est bissextile ou non
27     *
28     * @return      vrai pour les années bissextiles
29     */
30     public boolean isAnnéeBissextile()
31     {
32         boolean b = false;
33
34         if ((an % 4 != 0) ||
35             (an % 4 == 0) && (an % 100 == 0) && (an % 400 != 0) )
36         {
37             b = false;
38         }
39         else
40         {
41             b = true;
42         }
43
44         return b;
45     }
46
47     //solution alternative équivalente
48     public boolean isAnnéeBissextile2()
49     {
50         boolean b = !((an % 4 != 0) ||
51                     (an % 4 == 0) && (an % 100 == 0)
52                     && (an % 400 != 0) ) ;
53
54         return b;
55     }
56
57     //solution alternative équivalente
58     public boolean isAnnéeBissextile3()
59     {
60         return !((an % 4 != 0) ||
61                 (an % 4 == 0) && (an % 100 == 0)
62                 && (an % 400 != 0) ) ;

```

```
63     }  
64  
65  
66 }  
67
```

Class Année

java.lang.Object



```
public class Année extends java.lang.Object
```

ANNÉE BISSEXTILE Variante : opérateurs logiques Cette classe représente une année

Version:

1.0.3

Author:

Jens Getreu

Field Summary

protected int	an L'an.
------------------	-----------------------------

Constructor Summary

Année (int x) Constructeur.	
--	--

Method Summary

boolean	isAnnéeBissextile () Cette méthode détermine si une année est bissextile ou non
boolean	isAnnéeBissextile2 ()
boolean	isAnnéeBissextile3 ()

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

an

protected int **an**

L'an.

Constructor Detail

Année

public **Année**(intx)

Constructeur.

Parameters:

x - année

Method Detail

isAnnéeBissextile

public boolean **isAnnéeBissextile**()

Cette méthode détermine si une année est bissextile ou non

Returns:

vrai pour les années bissextiles

isAnnéeBissextile2

public boolean **isAnnéeBissextile2**()

isAnnéeBissextile3

public boolean **isAnnéeBissextile3**()

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

6 Équation du second degré

InterfaceUtilisateur.java

```
1  /**      RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ
2  *
3  * Basé sur: Les Générations Pascal Chap. 13.4 Ex. 5
4  *
5  * @author Jens Getreu
6  * @version 1.2.0
7  * */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14
15         System.out.println("RÉSOUDRE L'ÉQUATION DE SECOND DEGRÉ ");
16         System.out.println(" a*X^2 + b*X + c = 0 ");
17         System.out.println();
18
19         System.out.print("Veuillez saisir le coefficient a : ");
20         double a=sc.nextDouble();
21
22         System.out.print("Veuillez saisir le coefficient b : ");
23         double b=sc.nextDouble();
24
25         System.out.print("Veuillez saisir le coefficient c : ");
26         double c=sc.nextDouble();
27
28         Polynôme poly = new Polynôme(a,b,c);
29
30         if (poly.getNombreZéros()==0)
31         {
32             System.out.println("Pas de solution.");
33         }
34         else
35         {
36             if (poly.getNombreZéros()==1)
37             {
38                 System.out.println("Une solution : "+poly.getX1());
39             }
40             else
41             {
42                 System.out.println("Deux solutions : "+poly.getX1()+
43                                     " et "+poly.getX2());
44             }
45         }
46         System.out.println();
47     }
48 }
49
```

```

1  import java.math.*;
2  /**   RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ
3      *
4      * Cette classe représente un polynôme de second degré.
5      *  $aX^2 + bX + c$  sous sa forme factorisée.
6      *
7      * @author Jens Getreu
8      * @version 1.2.0
9      */
10 public class Polynôme
11 {
12     /**
13      * Le nombre de racines du polynôme
14      * (égal au nombre de zéros).
15      */
16     protected int nombreZéros;
17     /**
18      * Le coefficient a du polynôme factorisé.
19      */
20     protected double coeffA;
21     /**
22      * La racine X1 du polynôme factorisé.
23      */
24     protected double solX1;
25     /**
26      * La racine X2 du polynôme factorisé.
27      */
28     protected double solX2;
29
30     /**
31      * Constructeur pour objets de la classe Polynôme.
32      * Il recherche une représentation factorisée du
33      * polynôme sous forme  $a(X-X1)(X-X2)$ .
34      *
35      * Remarque: si le nombre de zéros (=racines)
36      * du polinôme est égal à 1 on peut écrire le
37      * polynôme sous forme
38      *  $a(X-X1)^2$  ou bien
39      *  $a(X-X1)(X-X2)$  avec  $X1=X2$ 
40      *
41      * Un objet de la classe Polynom est inéchangeable comme
42      * c'est le cas pour les Strings.
43      *
44      * @param a coefficient a
45      * @param b coefficient b
46      * @param c coefficient c
47      */
48     public Polynôme(double a, double b, double c) {
49         // factoriser le polynôme
50         coeffA = a;
51         double delta = b*b - 4*a*c;
52         if (delta<0)
53         {
54             nombreZéros = 0;
55         }
56         else
57         {
58             if (delta==0)
59             {
60                 solX1 = -b/(2*a);
61                 solX2 = solX1;
62                 nombreZéros = 1;

```

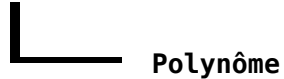
```

63     }
64     else
65     {
66         solX1 = (-b+ Math.sqrt(delta))/(2*a);
67         solX2 = (-b- Math.sqrt(delta))/(2*a);
68         nombreZéros = 2;
69     }
70 }
71 }
72
73
74 /**
75  * La méthode getNombreZéros() renvoie le nombre de zéros
76  * (=racines) du polynôme calculées par le constructeur.
77  * @return nombre de solutions
78  */
79 public int getNombreZéros()
80 {
81     return nombreZéros;
82 }
83
84 /**
85  * La méthode getA() renvoie le paramètre a
86  * @return paramètre a
87  */
88 public double getA()
89 {
90     return coeffA;
91 }
92
93 /**
94  * La méthode getX1() renvoie la première solution calculée par le
95  * constructeur.
96  * @return solution solX1
97  */
98 public double getX1()
99 {
100     return solX1;
101 }
102
103 /**
104  * La méthode getX2 renvoie la deuxième solution calculée par le
105  * constructeur.
106  * @return solution solX2
107  */
108 public double getX2()
109 {
110     return solX2;
111 }
112 }
113

```

Class Polynôme

java.lang.Object



```
public class Polynôme extends java.lang.Object
```

RÉSoudre L'Équation du second degré Cette classe représente un polynôme de second degré. $a \cdot X^2 + b \cdot X + c$ sous sa forme factorisée.

Version:

1.2.0

Author:

Jens Getreu

Field Summary

protected double	<u>coeffA</u> Le coefficient a du polynôme factorisé.
protected int	<u>nombreZéros</u> Le nombre de racines du polynôme (égal au nombre de zéros).
protected double	<u>solX1</u> La racine X1 du polynôme factorisé.
protected double	<u>solX2</u> La racine X2 du polynôme factorisé.

Constructor Summary

[Polynôme](#)(double a, double b, double c)
Constructeur pour objets de la classe Polynôme.

Method Summary

double	<u>getA</u> () La méthode getA() renvoie le paramètre a
--------	--

int	<code>getNombreZéros()</code> La méthode <code>getNombreZéros()</code> renvoie le nombre de zéros (=racines) du polynôme calculées par le constructeur.
double	<code>getX1()</code> La méthode <code>getX1()</code> renvoie la première solution calculée par le constructeur.
double	<code>getX2()</code> La méthode <code>getX2</code> renvoie la deuxième solution calculée par le constructeur.

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`

Field Detail

coeffA

protected double **coeffA**

Le coefficient a du polynôme factorisé.

nombreZéros

protected int **nombreZéros**

Le nombre de racines du polynôme (égal au nombre de zéros).

solX1

protected double **solX1**

La racine X1 du polynôme factorisé.

solX2

protected double **solX2**

La racine X2 du polynôme factorisé.

Constructor Detail

Polynôme

```
public Polynôme(double a,  
                double b,  
                double c)
```

Constructeur pour objets de la classe Polynôme. Il recherche une représentation factorisée du polynôme sous forme $a(X-X_1)(X-X_2)$. Remarque: si le nombre de zéros (=racines) du polynôme est égal à 1 on peut écrire le polynôme sous forme $a(X-X_1)^2$ ou bien $a(X-X_1)(X-X_2)$ avec $X_1=X_2$. Un objet de la classe Polynom est inéchangeable comme c'est le cas pour les Strings.

Parameters:

a - coefficient a
b - coefficient b
c - coefficient c

Method Detail

getA

```
public double getA()
```

La méthode getA() renvoie le paramètre a

Returns:

paramètre a

getNombreZéros

```
public int getNombreZéros()
```

La méthode getNombreZéros() renvoie le nombre de zéros (=racines) du polynôme calculées par le constructeur.

Returns:

nombre de solutions

getX1

```
public double getX1()
```

La méthode getX1() renvoie la première solution calculée par le constructeur.

Returns:

solution solX1

getX2

```
public double getX2()
```

La méthode getX2 renvoie la deuxième solution calculée par le constructeur.

Returns:
solution solX2

[Package](#) **Class** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) **[NO FRAMES](#)** [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

7 Vérifier un mot de passe

InterfaceUtilisateur.java

```
1  /**      VÉRIFIER LE MOT DE PASSE
2  *
3  * Basé sur Les Générations Pascal Chap. 15.2.3 Exemple 1
4  *
5  * @author Jens Getreu
6  * @version 1.0.1
7  * */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14         String s = "";
15
16         MotDePasse m = new MotDePasse ();
17         /*
18         //Extension possible
19         System.out.println("CHANGER LE MOT DE PASSE");
20         System.out.print("Veuillez saisir un nouveau mot de passe : ");
21         m.setMotDePasse(sc.nextLine());
22         */
23
24         System.out.println("VÉRIFIER LE MOT DE PASSE");
25
26
27
28         while (m.getNombreEssais()<3 )
29         {
30             System.out.print("Veuillez saisir votre mot de passe : ");
31             s=sc.nextLine();
32             if (m.isContenuÉgal(s))
33             {
34                 break;
35             }
36             else
37             {
38                 System.out.println("Erreur. Il vous reste "+
39                                     (3-m.getNombreEssais())+" essais.");
40             }
41         }
42
43         if (m.isContenuÉgal(s))
44         {
45             System.out.println("Le mot de passe est correct.");
46         }
47         else
48         {
49             System.out.println("Le mot de passe n'est pas correct.");
50         }
51
52         System.out.println();
53     }
54 }
55
```

```

1  /**      VÉRIFIER LE MOT DE PASSE
2  *
3  * Cette classe représente un mot de passe
4  *
5  * @author Jens Getreu
6  * @version 1.0.3
7  */
8  public class MotDePasse
9  {
10     /**
11      * La chaîne de caractères contenant le
12      * mot de passe.
13      */
14     protected String motSecret;
15
16     /**
17      * Le nombre d'interrogations du
18      * mot de passe.
19      */
20     protected int essais;
21
22     /**
23      * Constructeur
24      */
25     public MotDePasse()
26     {
27         motSecret="hommik";
28         essais=0;
29     }
30
31     /**
32      * La méthode isContenuÉgal() permet de tester si
33      * le contenu d'un String est égal au contenu du
34      * mot de passe secret.
35      * @return      vrai si les deux sont identiques
36      */
37     public boolean isContenuÉgal(String s)
38     {
39         boolean identique = s.equals(motSecret);
40         if (identique)
41         {
42             essais = 0;
43         }
44         else
45         {
46             essais = essais + 1;
47         }
48         return identique;
49     }
50
51
52     /**
53      * La méthode getNombreEssais() renvoie le nombre
54      * d'interrogations du mot de passe secret.
55      * @return      le nombre d'interrogations
56      */
57     public int getNombreEssais()
58     {
59         return essais;
60     }
61
62

```

```
63      /**
64       * La méthode setMotDePasse() permet de changer
65       * le mot de passe .
66       * @param s le nouveau mot de passe
67       */
68      public void setMotDePasse(String s)
69      {
70          essais = 0;
71          motSecret = s;
72      }
73  }
74
```

Class *MotDePasse*

java.lang.Object



```
public class MotDePasse extends java.lang.Object
```

VÉRIFIER LE MOT DE PASSE Cette classe représente un mot de passe

Version:

1.0.3

Author:

Jens Getreu

Field Summary

protected int	<u>essais</u> Le nombre d'interrogations du mot de passe.
protected java.lang.String	<u>motSecret</u> La chaîne de caractères contenant le mot de passe.

Constructor Summary

<u>MotDePasse</u> () Constructeur	
--	--

Method Summary

int	<u>getNombreEssais</u> () La méthode <code>getNombreEssais()</code> renvoie le nombre d'interrogations du mot de passe secret.
boolean	<u>isContenuÉgal</u> (java.lang.String s) La méthode <code>isContenuÉgal()</code> permet de tester si le contenu d'un String est égal au contenu du mot de passe secret.
void	<u>setMotDePasse</u> (java.lang.String s) La méthode <code>setMotDePasse()</code> permet de changer le mot de passe .

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

essais

protected int **essais**

Le nombre d'interrogations du mot de passe.

motSecret

protected java.lang.String **motSecret**

La chaîne de caractères contenant le mot de passe.

Constructor Detail

MotDePasse

public **MotDePasse()**

Constructeur

Method Detail

getNombreEssais

public int **getNombreEssais()**

La méthode getNombreEssais() renvoie le nombre d'interrogations du mot de passe secret.

Returns:

le nombre d'interrogations

isContenuÉgal

public boolean **isContenuÉgal**(java.lang.Strings)

La méthode isContenuÉgal() permet de tester si le contenu d'un String est égal au contenu du mot de passe secret.

Returns:

vrai si les deux sont identiques

setMotDePasse

```
public void setMotDePasse(java.lang.Strings)
```

La méthode setMotDePasse() permet de changer le mot de passe .

Parameters:

s - le nouveau mot de passe

Package	Class	Tree	Index	Help
----------------	--------------	-------------	--------------	-------------

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

8 Simplifier une fraction

InterfaceUtilisateur.java

```
1  /**      SIMPLIFIER UNE FRACTION
2  *
3  * Basé sur Les Générations Pascal Chap. 15.6 Ex 10
4  *
5  * @author Jens Getreu
6  * @version 1.0.2
7  * */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14
15         //saisie
16         System.out.println("SIMPLIFIER UNE FRACTION");
17         System.out.print("Veuillez saisir le numérateur n : ");
18         int n=sc.nextInt();
19
20         System.out.print("Veuillez saisir le dénominateur d : ");
21         int d=sc.nextInt();
22         System.out.println();
23
24         Fraction f = new Fraction(n,d);
25         f.simplifier();
26
27         //Sortie
28         System.out.println("Le numérateur simplifié n = "
29                             +f.getNum() );
30         System.out.println("Le dénominateur simplifié d = "
31                             +f.getDénom() );
32         System.out.println();
33     }
34 }
35
```

```

1  /**      SIMPLIFIER UNE FRACTION
2  *
3  * Cette classe représente une fraction
4  *
5  * @author Jens Getreu
6  * @version 1.0.5
7  */
8  public class Fraction
9  {
10     /**
11      * Le numérateur.
12      */
13     protected int num;
14
15     /**
16      * Le dénominateur.
17      */
18     protected int dénom;
19
20     /**
21      * Constructeur
22      * @param a numérateur
23      * @param b dénominateur
24      */
25     public Fraction(int a, int b)
26     {
27         num = a;
28         dénom = b;
29     }
30
31     /**
32      * La méthode pgcd() calcule le plus grand commun
33      * diviseur du numérateur et du dénominateur.
34      * @ return pgcd
35      */
36     public int pgcd()
37     {
38         int a = num;
39         int b = dénom;
40         int reste = -1;
41
42         //calculer le PGCD
43         //variante 1
44         //la condition est placée avant le bloc d'instructions
45         while (reste != 0)
46         {
47             reste = a % b ;
48             a = b;
49             b = reste;
50         }
51         //le pgcd se trouve dans a
52         return a;
53     }
54
55
56     /**
57      * La méthode pgcd() calcule le plus grand commun
58      * diviseur du numérateur et du dénominateur.
59      * @ return pgcd
60      */
61     /*
62     public int pgcd()

```

```

63     {
64         int a=num;
65         int b=dénom;
66         int reste=-1;
67
68         //calculer le PGCD
69         //variante 2:
70         //la condition est placée après le bloc d'instructions
71         do {
72             reste = a%b;
73             a = b;
74             b = reste;
75         } while (reste != 0);
76         //le pgcd se trouve dans a
77         return a;
78     }
79     */
80
81     /**
82     * La méthode simplifier() simplifie la fraction en recherchant
83     * le PGCD du numérateur et du dénominateur.
84     */
85     public void simplifier()
86     {
87         int x= pgcd();
88         num = num / x;    // int = int / int
89         dénom = dénom / x;
90     }
91
92     /**
93     * La méthode getNum() renvoie le numérateur
94     *
95     * @return numérateur
96     */
97     public int getNum()
98     {
99         return num;
100    }
101
102    /**
103    * La méthode getDénom() renvoie le dénominateur
104    *
105    * @return dénominateur
106    */
107    public int getDénom()
108    {
109        return dénom;
110    }
111 }
112

```

Class Fraction

java.lang.Object



```
public class Fraction extends java.lang.Object
```

SIMPLIFIER UNE FRACTION Cette classe représente une fraction

Version:

1.0.5

Author:

Jens Getreu

Field Summary

protected int	<u>dénom</u> Le dénominateur.
protected int	<u>num</u> Le numérateur.

Constructor Summary

<u>Fraction</u> (int a, int b) Constructeur	
--	--

Method Summary

int	<u>getDénom</u> () La méthode getDénom() renvoie le dénominateur
int	<u>getNum</u> () La méthode getNum() renvoie le numérateur
int	<u>pgcd</u> () La méthode pgcd() calcule le plus grand commun diviseur du numérateur et du dénominateur.

void [simplifier](#)()

La méthode `simplifier()` simplifie la fraction en recherchant le PGCD du numérateur et du dénominateur.

Methods inherited from class `java.lang.Object`

`clone`, `equals`, `finalize`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Field Detail

dénom

protected int **dénom**

Le dénominateur.

num

protected int **num**

Le numérateur.

Constructor Detail

Fraction

public **Fraction**(inta,
 intb)

Constructeur

Parameters:

a - numérateur
b - dénominateur

Method Detail

getDénom

public int **getDénom**()

La méthode `getDénom()` renvoie le dénominateur

Returns:

dénominateur

getNum

```
public int getNum()
```

La méthode getNum() renvoie le numérateur

Returns:

numérateur

pgcd

```
public int pgcd()
```

La méthode pgcd() calcule le plus grand commun diviseur du numérateur et du dénominateur.

simplifier

```
public void simplifier()
```

La méthode simplifier() simplifie la fraction en recherchant le PGCD du numérateur et du dénominateur.

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

9 Nombre premier

InterfaceUtilisateur.java

```

1  /**      NOMBRE PREMIER
2  *
3  *  Les Générations Pascal Chap. 12.6 Ex. 9
4  *
5  *  @author Jens Getreu
6  *  @version 1.0.4
7  *  */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14
15         System.out.println("NOMBRE PREMIER");
16
17         System.out.print("Veuillez saisir un nombre entier p : ");
18         int p=sc.nextInt();
19
20         NombreEntier n = new NombreEntier(p);
21
22         if( n.isNombrePremier() )
23         {
24             System.out.println("p est nombre premier.");
25         }
26         else
27         {
28             System.out.println("p n'est pas nombre premier.");
29         }
30
31         /*
32         // extension possible de l'exercice
33         if( n.isNombrePair() ) {
34             System.out.println("p est pair.");
35         }
36         else {
37             System.out.println("p est impair.");
38         }
39         */
40
41         sc.nextLine();
42     }
43 }
44

```

```

1  /**      NOMBRE PREMIER
2  *
3  * Cette classe représente un nombre entier
4  *
5  * @author Jens Getreu
6  * @version 1.0.4
7  */
8  public class NombreEntier
9  {
10     /**
11      * Le nombre entier.
12      */
13     protected int nombre;
14
15     /**
16      * Constructeur
17      */
18     public NombreEntier(int p)
19     {
20         nombre = p;
21     }
22
23     /**
24      * Cette méthode informe s'il s'agit d'un nombre
25      * premier
26      *
27      * @return vrai si le nombre est premier
28      */
29     public boolean isNombrePremier()
30     {
31         int i=2;
32         while((i < nombre) && (nombre % i != 0))
33         {
34             i=i+1;
35         }
36         return i == nombre;
37     }
38
39     /**
40      * Cette méthode informe s'il s'agit d'un nombre
41      * pair (Extension possible de l'exercice).
42      *
43      * @return vrai si le nombre est pair
44      */
45     public boolean isNombrePair()
46     {
47         return nombre % 2 == 0;
48     }
49 }
50

```

Class *NombreEntier*

java.lang.Object



```
public class NombreEntier extends java.lang.Object
```

NOMBRE PREMIER Cette classe représente un nombre entier

Version:

1.0.4

Author:

Jens Getreu

Field Summary

protected int	nombre Le nombre entier.
------------------	---

Constructor Summary

NombreEntier (int p) Constructeur	
--	--

Method Summary

boolean	isNombrePair () Cette méthode informe s'il s'agit d'un nombre pair (Extension possible de l'exercice).
boolean	isNombrePremier () Cette méthode informe s'il s'agit d'un nombre premier

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

nombre

protected int **nombre**

Le nombre entier.

Constructor Detail

NombreEntier

public **NombreEntier**(intp)

Constructeur

Method Detail

isNombrePair

public boolean **isNombrePair**()

Cette méthode informe s'il s'agit d'un nombre pair (Extension possible de l'exercice).

Returns:

vrai si le nombre est pair

isNombrePremier

public boolean **isNombrePremier**()

Cette méthode informe s'il s'agit d'un nombre premier

Returns:

vrai si le nombre est premier

Package **Class** **Tree** **Index** **Help**

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

10 Jeu « deviner nombre »

InterfaceUtilisateur.java

```

1  /**      DEVINER NOMBRE
2  *
3  * Basé sur Les Générations Pascal Chap. 17.5 Ex. 8
4  *
5  * @author Jens Getreu
6  * @version 1.1.5
7  */
8  import java.util.*;
9  public class InterfaceUtilisateur
10 {
11     public static void main(String[] args)
12     {
13         Scanner sc = new Scanner(System.in);
14
15         System.out.println("DEVINER NOMBRE : ");
16         System.out.print("Limite inférieur : ");
17         int limInf = sc.nextInt();
18         System.out.print("Limite supérieur : ");
19         int limSup = sc.nextInt();
20
21         NombreSecret j = new NombreSecret(limInf,limSup);
22
23         System.out.println("*** Le jeu démarre, vous avez 3 essais ***");
24
25         int estimation=0;
26
27         while (j.getNombreEssais() < 2*3)
28         {
29             System.out.print("Votre estimation : ");
30             estimation=sc.nextInt();
31             if (j.isÉgal(estimation))
32             {
33                 break;
34             }
35             else
36             {
37                 if (j.isTropPetit(estimation))
38                 {
39                     System.out.println("    trop petit ...");
40                 }
41                 else
42                 {
43                     System.out.println("    trop grand...");
44                 }
45             }
46         }
47
48         if (j.isÉgal(estimation))
49         {
50             System.out.println("Vous avez gagné!");
51         }
52         else
53         {
54             System.out.println("Vous avez perdu.");
55         }
56         System.out.println();
57         sc.nextLine();
58     }
59 }
60

```

```

1  /**          DEVINER NOMBRE
2  *
3  * La classe NombreSecret représente un nombre aléatoire et
4  * quelques méthodes pour l'interroger
5  * @author Jens Getreu
6  * @version 1.1.4
7  */
8  public class NombreSecret
9  {
10     /**
11      * Le nombre aléatoire à deviner.
12      */
13     protected int nombre;
14     /**
15      * Le nombre d'essais.
16      */
17     protected int essais;
18
19     /**
20      * Le constructeur pour objets de la classe Jeu initialise
21      * le nombre aléatoire secret.
22      */
23     public NombreSecret(int limInf, int limSup)
24     {
25         essais = 0;
26         nombre = (int)(Math.random()*(limSup-limInf+1)) + limInf;
27     }
28
29     /**
30      * La méthode isTropGrand() compare y avec le nombre secret
31      * et renvoie vrai ou faux.
32      *
33      * @param y    estimation
34      * @return     vrai si y est supérieur au nombre secret
35      */
36     public boolean isTropGrand(int y)
37     {
38         essais = essais + 1;
39         return y > nombre;
40     }
41
42     /**
43      * La méthode isTropPetit() compare y avec le nombre secret
44      * et renvoie vrai ou faux.
45      *
46      * @param y    estimation
47      * @return     vrai si y est inférieur au nombre secret
48      */
49     public boolean isTropPetit(int y)
50     {
51         essais = essais + 1;
52         return y < nombre;
53     }
54
55     /**
56      * La méthode isÉgal() compare y avec le nombre secret
57      * et renvoie vrai ou faux.
58      *
59      * @param y    estimation
60      * @return     vrai si y est égal au nombre secret
61      */
62     public boolean isÉgal(int y)

```

```

63     {
64         essais = essais +1;
65         return y == nombre;
66     }
67
68     /**
69     * La méthode getNombreEssais() renvoie le nombre de questions
70     * posées.
71     *
72     * @return     nombre de questions posées
73     */
74     public int getNombreEssais()
75     {
76         return essais;
77     }
78
79 }
80

```

Class NombreSecret

java.lang.Object



```
public class NombreSecret extends java.lang.Object
```

DEVINER NOMBRE La classe NombreSecret représente un nombre aléatoire et quelques méthodes pour l'interroger

Version:

1.1.4

Author:

Jens Getreu

Field Summary

protected int	<u>essais</u> Le nombre d'essais.
protected int	<u>nombre</u> Le nombre aléatoire à deviner.

Constructor Summary

<u>NombreSecret</u> (int limInf, int limSup) Le constructeur pour objets de la classe Jeu initialise le nombre aléatoire secret.	
---	--

Method Summary

int	<u>getNombreEssais</u> () La méthode getNombreEssais() renvoie le nombre de questions posées.
boolean	<u>isÉgal</u> (int y) La méthode isÉgal() compare y avec le nombre secret et renvoie vrai ou faux.
boolean	<u>isTropGrand</u> (int y) La méthode isTropGrand() compare y avec le nombre secret et renvoie vrai ou faux.

boolean [isTropPetit](#)(int y)

La méthode isTropPetit() compare y avec le nombre secret et renvoie vrai ou faux.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

essais

protected int **essais**

Le nombre d'essais.

nombre

protected int **nombre**

Le nombre aléatoire à deviner.

Constructor Detail

NombreSecret

public **NombreSecret**(intlimInf,
intlimSup)

Le constructeur pour objets de la classe Jeu initialise le nombre aléatoire secret.

Method Detail

getNombreEssais

public int **getNombreEssais**()

La méthode getNombreEssais() renvoie le nombre de questions posées.

Returns:

nombre de questions posées

isÉgal

public boolean **isÉgal**(inty)

La méthode isÉgal() compare y avec le nombre secret et renvoie vrai ou faux.

Parameters:

y - estimation

Returns:

vrai si y est égal au nombre secret

isTropGrand

public boolean **isTropGrand**(inty)

La méthode isTropGrand() compare y avec le nombre secret et renvoie vrai ou faux.

Parameters:

y - estimation

Returns:

vrai si y est supérieur au nombre secret

isTropPetit

public boolean **isTropPetit**(inty)

La méthode isTropPetit() compare y avec le nombre secret et renvoie vrai ou faux.

Parameters:

y - estimation

Returns:

vrai si y est inférieur au nombre secret

Package **Class** Tree Index Help

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

11 Jeu du loup

```

1  /**      JEU DU LOUP
2      *
3      *
4      * @author Jens Getreu
5      * @version 1.0.7
6      */
7  import java.util.*;
8  public class InterfaceUtilisateur
9  {
10     public static void main(String[] args)
11     {
12         Scanner sc = new Scanner(System.in);
13
14         System.out.println("JEU DU LOUP");
15         System.out.print("Veuillez saisir l'avance de 0 sur X : ");
16         int d=sc.nextInt();
17
18         Piste p = new Piste(d);
19
20         // Variante condition est placée avant le bloc d'instructions
21         boolean c =true;
22         while (c)
23         {
24             System.out.print(p.dessiner());
25             System.out.print(" Votre pas : ");
26             c=p.avancer(sc.nextInt());
27         }
28
29
30         System.out.println(p.dessiner());
31         if ((p.getPosX() == p.getPos0()) && (p.getPos0() <= p.POS_MAX) )
32         {
33             System.out.println("Vous avez attrapé le loup!");
34         }
35         else
36         {
37             System.out.println("Le loup vous a échappé.");
38         }
39     }
40
41     /*
42     public static void main(String[] args)
43     {
44         java.util.Scanner sc = new java.util.Scanner(System.in);
45
46         System.out.println("JEU DU LOUP");
47         System.out.print("Veuillez saisir l'avance de 0 sur X : ");
48         int d=sc.nextInt();
49
50         Piste p = new Piste(d);
51
52         // Variante condition est placée après le bloc d'instructions
53         do
54         {
55             System.out.print(p.dessiner());
56             System.out.print(" Votre pas : ");
57         }
58         while (p.avancer(sc.nextInt()))
59         {
60             System.out.println(p.dessiner());
61
62             if ((p.getPosX() == p.getPos0()) && (p.getPos0() <= p.POS_MAX) )

```

```
63         System.out.println("Vous avez attrapé le loup!");
64     }
65     else
66     {
67         System.out.println("Le loup vous a échappé.");
68     }
69 }
70 */
71 }
72
```

```

1  /**      JEU DU LOUP
2  *
3  * Cette classe représente une piste et
4  * deux coureurs 0 et X. X essaie de rattraper
5  * 0 alors que 0 avance aléatoirement.
6  *
7  * @author Jens Getreu
8  * @version 1.0.4
9  */
10 public class Piste
11 {
12     /**
13      * La largeur de la piste.
14      */
15     public static final int POS_MAX = 20;
16     /**
17      * Le nombre de pas maximal.
18      */
19     public static final int PAS_MAX = 5;
20     /**
21      * Le nombre de pas minimal.
22      */
23     public static final int PAS_MIN = 1;
24     /**
25      * La position du pion 0 (loup).
26      */
27     protected int pos0;
28     /**
29      * La position du pion X (joueur).
30      */
31     protected int posX;
32
33     /**
34      * Un constructeur pour objets de la classe
35      * Piste
36      * @param avance0 l'avance initiale de 0 sur X
37      */
38     public Piste(int avance0)
39     {
40         // initialiser la piste
41         pos0 = 1 + avance0;
42         posX = 1;
43     }
44
45     /**
46      * La méthode avancer() avance le pion X
47      * de pasX et le pion 0 aléatoirement
48      * @param pasX pas du pion X
49      * @return faux si le jeu est terminé
50      */
51     public boolean avancer(int pasX)
52     {
53         if ( posX >= PAS_MIN && posX <= PAS_MAX )
54         {
55             posX = posX + pasX;
56         }
57
58         pos0 = pos0 + (int)(Math.random()
59             *(PAS_MAX-PAS_MIN+1))+ PAS_MIN;
60
61         return (posX < POS_MAX)
62             && (pos0 < POS_MAX)

```

```

63         && (pos0 != posX);
64     }
65
66     /**
67     * La méthode dessiner() dessine
68     * la piste avec les deux coureurs 0 et X
69     */
70     public String dessiner()
71     {
72         String d = "";
73         String s;
74         for (int i=0; i<=POS_MAX; i++)
75         {
76             s = " ";
77             if (i==0 || i==POS_MAX)
78             {
79                 s = "|";
80             }
81             if (i==posX)
82             {
83                 s = "X";
84             }
85             if (i==pos0)
86             {
87                 s = "0";
88             }
89             if (i==posX && posX == pos0)
90             {
91                 s = "*";
92             }
93             d = d + s;
94         }
95         return d;
96     }
97     /**
98     * La méthode getPosX() renvoie la
99     * position du coureur X
100    * @return position de X
101    */
102    public int getPosX()
103    {
104        return posX;
105    }
106
107    /**
108    * La méthode getPos0() renvoie la
109    * position du coureur 0
110    * @return position de 0
111    */
112    public int getPos0()
113    {
114        return pos0;
115    }
116 }
117

```

Class Piste

java.lang.Object
└─ **Piste**

```
public class Piste extends java.lang.Object
```

JEU DU LOUP Cette classe représente une piste et deux coureurs O et X. X essaie de rattraper O alors que O avance aléatoirement.

Version:

1.0.4

Author:Jens Getreu

Field Summary

static int	<u>PAS_MAX</u>	Le nombre de pas maximal.
static int	<u>PAS_MIN</u>	Le nombre de pas minimal.
static int	<u>POS_MAX</u>	La largeur de la piste.
protected int	<u>posO</u>	La position du pion O (loup).
protected int	<u>posX</u>	La position du pion X (joueur).

Constructor Summary

<u>Piste</u> (int avanceO) Un constructeur pour objets de la classe Piste	
--	--

Method Summary

boolean	avancer (int pasX) La méthode avancer() avance le pion X de pasX et le pion O aléatoirement
java.lang.String	dessiner () La méthode dessiner() dessine la piste avec les deux coureurs O et X
int	getPosO () La méthode getPosO() renvoie la position du coureur O
int	getPosX () La méthode getPosX() renvoie la position du coureur X

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

PAS_MAX

public static final int **PAS_MAX**

Le nombre de pas maximal.

See Also:

[Constant Field Values](#)

PAS_MIN

public static final int **PAS_MIN**

Le nombre de pas minimal.

See Also:

[Constant Field Values](#)

POS_MAX

public static final int **POS_MAX**

La largeur de la piste.

See Also:

[Constant Field Values](#)

posO

protected int **posO**

La position du pion O (loup).

posX

protected int **posX**

La position du pion X (joueur).

Constructor Detail

Piste

public **Piste**(intavance0)

Un constructeur pour objets de la classe Piste

Parameters:

avance0 - l'avance initiale de O sur X

Method Detail

avancer

public boolean **avancer**(intpasX)

La méthode avancer() avance le pion X de pasX et le pion O aléatoirement

Parameters:

pasX - pas du pion X

Returns:

faux si le jeu est terminé

dessiner

public java.lang.String **dessiner**()

La méthode dessiner() dessine la piste avec les deux coureurs O et X

getPosO

public int **getPosO**()

La méthode getPosO() renvoie la position du coureur O

Returns:

position de O

getPosX

```
public int getPosX()
```

La méthode getPosX() renvoie la position du coureur X

Returns:

position de X

Package **Class** Tree Index Help

PREV CLASS NEXT CLASS

FRAMES **NO FRAMES** [All Classes](#)

SUMMARY: NESTED | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

12 Minimum-Maximum

InterfaceUtilisateur.java

```
1  /**      MIN-MAX
2      *
3      *
4      * @author Jens Getreu
5      * @version 1.0.2
6      */
7  import java.util.*;
8  public class InterfaceUtilisateur
9  {
10     public static void main(String[] args)
11     {
12         Scanner sc = new Scanner(System.in);
13
14         System.out.println("MIN-MAX");
15
16         NombrePositif max = new NombrePositif(NombrePositif.MIN);
17         NombrePositif min = new NombrePositif(NombrePositif.MAX);
18
19         int p=1;
20         while (true)
21         {
22             System.out.print("Veuillez saisir un nombre entier entre 0 et 1000
23             (-1 pour terminer) : ");
24             p=sc.nextInt();
25
26             NombrePositif np = new NombrePositif(p);
27
28             if (!np.isValid())
29             {
30                 break;
31             }
32
33             if (min.isSupérieur(np))
34             {
35                 min=np;
36             }
37
38             if (max.isInférieur(np))
39             {
40                 max=np;
41             }
42             System.out.println("Le minimum = "+min.getValeur());
43             System.out.println("Le maximum = "+max.getValeur());
44         }
45     }
46 }
```

NombrePositif.java

```

1  /**      MIN-MAX
2  *
3  * Cette classe représente un nombre entier positif.
4  *
5  * @author Jens Getreu
6  * @version 1.0.4
7  */
8  public class NombrePositif
9  {
10     /**
11      * La valeur maximale valide d'un NombrePositif.
12      */
13     public static final int MAX = 1000;
14
15     /**
16      * La valeur minimale valide d'un NombrePositif.
17      */
18     public static final int MIN = 0;
19
20
21     /**
22      * La représentation d'un nombre non valide.
23      */
24     public static final int NON_VALIDE = -1;
25
26     /**
27      * Le nombre positif entre MIN et MAX ou non-valide.
28      */
29     protected int nombre;
30
31     /**
32      * Ce constructeur génère un nombre entier positif à partir
33      * du paramètre a.
34      * Un nombre non valide est représenté par -1 par convention.
35      * @param a nombre
36      */
37     public NombrePositif(int a)
38     {
39         if ( (a>=MIN) && (a<=MAX) )
40         {
41             nombre=a;
42         }
43         else
44         {
45             nombre=NON_VALIDE;
46         }
47     }
48
49     /**
50      * Cette méthode renvoie la valeur du NombrePositif
51      * @return valeur
52      */
53     public int getValeur()
54     {
55         return nombre;
56     }
57
58     /**
59      * Cette méthode compare deux "NombrePositif"
60      * @param a NombrePositif à comparer
61      * @return vrai si "nombre" est supérieur à "a"
62      */

```

```

63     public boolean isSupérieur(NombrePositif a)
64     {
65         return nombre > a.getValeur();
66     }
67
68
69     /**
70     * Cette méthode compare deux "NombrePositif"
71     * @param a NombrePositif à comparer
72     * @return vrai si "nombre" est inférieur à "a"
73     */
74     public boolean isInférieur(NombrePositif a)
75     {
76         return nombre < a.getValeur();
77     }
78
79     /**
80     * Cette méthode indique si le NombrePositif
81     * est valide, c'est à dire entre NombrePositif.MIN
82     * et NombrePositif.MAX
83     * @return vrai si valide
84     */
85     public boolean isValid()
86     {
87         return nombre != NON_VALIDE;
88     }
89
90
91 }
92

```

Class NombrePositif

java.lang.Object



```
public class NombrePositif extends java.lang.Object
```

MIN-MAX Cette classe représente un nombre entier positif.

Version:

1.0.4

Author:

Jens Getreu

Field Summary

static int	MAX La valeur maximale valide d'un NombrePositif.
static int	MIN La valeur minimale valide d'un NombrePositif.
protected int	nombre Le nombre positif entre MIN et MAX ou non-valide.
static int	NON_VALIDE La représentation d'un nombre non valide.

Constructor Summary

NombrePositif (int a) Ce constructeur génère un nombre entier positif à partir du paramètre a.	
--	--

Method Summary

int	getValeur () Cette méthode renvoie la valeur du NombrePositif
boolean	isInférieur (NombrePositif a) Cette méthode compare deux "NombrePositif"

boolean	isSupérieur (NombrePositif a) Cette méthode compare deux "NombrePositif"
boolean	isValid () Cette méthode indique si le NombrePositif est valide, c'est à dire entre NombrePositif.MIN et NombrePositif.MAX

Methods inherited from class java.lang.Object

[clone](#), [equals](#), [finalize](#), [getClass](#), [hashCode](#), [notify](#), [notifyAll](#), [toString](#), [wait](#), [wait](#), [wait](#)

Field Detail

MAX

public static final int **MAX**

La valeur maximale valide d'un NombrePositif.

See Also:

[Constant Field Values](#)

MIN

public static final int **MIN**

La valeur minimale valide d'un NombrePositif.

See Also:

[Constant Field Values](#)

nombre

protected int **nombre**

Le nombre positif entre MIN et MAX ou non-valide.

NON_VALIDE

public static final int **NON_VALIDE**

La représentation d'un nombre non valide.

See Also:

[Constant Field Values](#)

Constructor Detail

NombrePositif

```
public NombrePositif(inta)
```

Ce constructeur génère un nombre entier positif à partir du paramètre a. Un nombre non valide est représenté par -1 par convention.

Parameters:

a - nombre

Method Detail

getValeur

```
public int getValeur()
```

Cette méthode renvoie la valeur du NombrePositif

Returns:

valeur

isInférieur

```
public boolean isInférieur(NombrePositifa)
```

Cette méthode compare deux "NombrePositif"

Parameters:

a - NombrePositif à comparer

Returns:

vrai si "nombre" est inférieur à "a"

isSupérieur

```
public boolean isSupérieur(NombrePositifa)
```

Cette méthode compare deux "NombrePositif"

Parameters:

a - NombrePositif à comparer

Returns:

vrai si "nombre" est supérieur à "a"

isValide

```
public boolean isValide()
```

Cette méthode indique si le NombrePositif est valide, c'est à dire entre NombrePositif.MIN et NombrePositif.MAX

Returns:

vrai si valide

[Package](#) **Class** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

13 Sapin de Noël

InterfaceUtilisateur.java

```
1  /**      SAPIN DE NOËL
2      *
3      *
4      * @author Jens Getreu
5      * @version 1.0.2
6      */
7  import java.util.*;
8  public class InterfaceUtilisateur
9  {
10     public static void main(String[] args) {
11         Scanner sc = new Scanner(System.in);
12         System.out.println("SAPIN DE NOËL");
13         System.out.print("Veuillez saisir la hauteur : ");
14         int m=sc.nextInt();
15
16         Sapin s = new Sapin(m);
17
18         System.out.println(s.dessiner());
19
20         sc.nextLine();
21     }
22 }
23
```

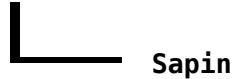
```

1  /**      SAPIN DE NOËL
2  *
3  * Cette classe représente un sapin de Noël
4  *
5  * @author Jens Getreu
6  * @version 1.0.3
7  */
8  public class Sapin
9  {
10     /**
11      * La hauteur du sapin.
12      */
13     protected int hauteur;
14
15     /**
16      * Constructeur
17      * @param h hauteur du sapin
18      */
19     public Sapin(int h)
20     {
21         hauteur=h;
22     }
23
24     /**
25      * La méthode dessiner() renvoie le dessin du sapin
26      *
27      * @return      le dessin du sapin
28      */
29     public String dessiner()
30     {
31         String s="";
32
33         for (int j = 1; j <= hauteur; j++)
34         {
35             for (int i = 1; i <= j; i++)
36             {
37                 s = s + (i + " ");
38             }
39             s = s+"\n";
40         }
41         return s;
42     }
43 }
44

```

Class Sapin

java.lang.Object



```
public class Sapin extends java.lang.Object
```

SAPIN DE NOËL Cette classe représente un sapin de Noël

Version:

1.0.3

Author:

Jens Getreu

Field Summary

protected int	hauteur La hauteur du sapin.
------------------	---

Constructor Summary

Sapin (int h) Constructeur	
---	--

Method Summary

java.lang.String	dessiner () La méthode dessiner() renvoie le dessin du sapin
------------------	---

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

hauteur

protected int **hauteur**

La hauteur du sapin.

Constructor Detail

Sapin

public **Sapin**(inth)

Constructeur

Parameters:

h - hauteur du sapin

Method Detail

dessiner

public java.lang.String **dessiner**()

La méthode dessiner() renvoie le dessin du sapin

Returns:

le dessin du sapin

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

14 Décomposition en produit de facteurs_premiers

InterfaceUtilisateur.java

```
1  /**      DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS
2      *
3      *
4      * @author Jens Getreu
5      * @version 1.0.2
6      */
7  import java.util.*;
8  public class InterfaceUtilisateur
9  {
10     public static void main(String[] args)
11     {
12         Scanner sc = new Scanner(System.in);
13
14         System.out.println("DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS");
15         System.out.print("Veuillez saisir un nombre entier p : ");
16         int p=sc.nextInt();
17
18         System.out.print("p = 1");
19         while (p>1)
20         {
21             NombrePremier np = new NombrePremier(1,p,p);
22             p = p / np.getNombrePremier();
23             System.out.print(" *" +np.getNombrePremier() );
24         }
25         System.out.println();
26     }
27 }
28
```

NombrePremier.java

```

1  /**      DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS
2  *
3  * Cette classe représente un nombre entier
4  *
5  * @author Jens Getreu
6  * @version 1.0.4
7  */
8  public class NombrePremier
9  {
10     /**
11      * Le nombre premier.
12      */
13     protected int np;
14
15     /**
16      * Ce constructeur génère le plus petit nombre premier entre
17      * limInf et limSup.
18      * Si aucun nombre premier existe entre ces deux limites
19      * np est 0.
20      * (Ce constructeur n'est pas utilisé, mais sert comme introduction.)
21      * @param limInf limite inférieur du nombre premier
22      * @param limSup limite supérieur du nombre premier
23      */
24     public NombrePremier(int limInf, int limSup)
25     {
26         np=0;
27         for (int n=limInf; n<=limSup;n++)
28         {
29             int i = 2;      // voir ex. nombre premier
30             while((i < n) && (n%i != 0))
31             {
32                 i=i+1;
33             }
34             if (i==n)
35             {
36                 //trouvé
37                 np = n;
38                 break;
39             }
40         }
41
42         /**
43          * Ce constructeur génère le plus petit nombre premier entre
44          * "limInf" et "limSup" à condition qu'il est diviseur de "dividende".
45          * Si aucun nombre premier vérifie ces conditions
46          * np est 0.
47          * @param limInf limite inférieur du nombre premier
48          * @param limSup limite supérieur du nombre premier
49          */
50
51         public NombrePremier(int limInf, int limSup, int dividende)
52         {
53             np=0;
54             for (int n=limInf; n<=limSup;n++)
55             {
56                 int i=2;      // voir ex. "nombre premier"
57                 while ((i < n) && (n%i != 0))
58                 {
59                     i=i+1;
60                 }
61                 if ((i==n) && (dividende%i == 0))
62                 {
63                     //+ cond. supp.

```

```

63         np=n;
64         break;
65     }
66 }
67 }
68
69 /**
70  * Cette méthode renvoie le nombre premier
71  * @return nombre premier
72  */
73 public int getNombrePremier()
74 {
75     return np;
76 }
77 }
78

```

Class *NombrePremier*

java.lang.Object



```
public class NombrePremier extends java.lang.Object
```

DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS Cette classe représente un nombre entier

Version:

1.0.4

Author:

Jens Getreu

Field Summary

protected int	<u>np</u> Le nombre premier.
------------------	---

Constructor Summary

[NombrePremier](#)(int limInf, int limSup)

Ce constructeur génère le plus petit nombre premier entre limInf et limSup.

[NombrePremier](#)(int limInf, int limSup, int dividende)

Ce constructeur génère le plus petit nombre premier entre "limInf" et "limSup" à condition qu'il est diviseur de "dividende".

Method Summary

int	<u>getNombrePremier</u> () Cette méthode renvoie le nombre premier
-----	---

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

np

protected int np

Le nombre premier.

Constructor Detail

NombrePremier

```
public NombrePremier(intlimInf,  
                     intlimSup)
```

Ce constructeur génère le plus petit nombre premier entre limInf et limSup. Si aucun nombre premier existe entre ces deux limites np est 0. (Ce constructeur n'est pas utilisé, mais sert comme introduction.)

Parameters:

limInf - limite inférieur du nombre premier
limSup - limite supérieur du nombre premier

NombrePremier

```
public NombrePremier(intlimInf,  
                     intlimSup,  
                     intdividende)
```

Ce constructeur génère le plus petit nombre premier entre "limInf" et "limSup" à condition qu'il est diviseur de "dividende". Si aucun nombre premier vérifie ces conditions np est 0.

Parameters:

limInf - limite inférieur du nombre premier
limSup - limite supérieur du nombre premier

Method Detail

getNombrePremier

```
public int getNombrePremier()
```

Cette méthode renvoie le nombre premier

Returns:

nombre premier

[Package](#) **[Class](#)** [Tree](#) [Index](#) [Help](#)

[PREV CLASS](#) [NEXT CLASS](#)

[FRAMES](#) [NO FRAMES](#) [All Classes](#)

SUMMARY: [NESTED](#) | [FIELD](#) | [CONSTR](#) | [METHOD](#)

DETAIL: [FIELD](#) | [CONSTR](#) | [METHOD](#)

