Java à l'école

Proposition de solutions

11TG, Première année de programmation

Version 1.3.4d, Jens Getreu

Table des matières

1 Introduction

InterfaceUtilisateur.java

```java
/**     INTRODUCTION
 *
 *
 * Interface utilisateur pour la classe Salutation
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
        // saisie
        Scanner sc = new Scanner(System.in);

        System.out.println("SALUTATION");
        System.out.print("Veuillez saisir votre nom : ");

        String nom = sc.nextLine();
        //traîtement
        Salutation s = new Salutation(nom);
        // sortie
        System.out.println(s.saluer() );
   }
}

```

Salutation.java

```java
/**        INTRODUCTION
 *
 *
 * Cette classe représente une salutation
 *
 * @author Jens Getreu
 * @version 1.0.2
 */
public class Salutation
{
    /**
     * Déclaration et initialisation d'une constante: un mot d'accueil.
     */

    public static final String MOT_ACCUEIL="Bonjour ";
    //public static final MOT_ACCUEIL="Salut ";

    /**
     * Le nom de la personne à saluer
     */
    protected String nom;

    /**
     * Constructeur pour objets de la classe salutation
     * @param n un nom
     */
    public Salutation(String n)
    {
       nom = n;
    }

    /**
    * La méthode saluer() renvoie un bonjour
    * personnalisé.
    * @return Bonjour personnalisé
    */
    public String saluer()
    {
        return MOT_ACCUEIL + nom +". Comment allez-vous ?";
    }
}

```

Class Salutation

```
java.lang.Object
   |
   +--Salutation
```

```
public class Salutationextends java.lang.Object
```

INTRODUCTION Cette classe représente une salutation

Version:
1.0.2
Author:
Jens Getreu

Field Summary

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

Constructor Summary

Salutation(java.lang.String n) Constructeur pour objets de la classe salutation	

Method Summary

java.lang.String	**saluer**() 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.Stringn)
```

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

---

2 Permutation

InterfaceUtilisateur.java

```java
/**     PERMUTATION
 *
 * Basé sur Les Générations Pascal Chap. 8.6 Ex. 6
 *
 * @author Jens Getreu
 * @version 1.0.3
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);
        int x;
        int y;

        //saisie
        System.out.println("PERMUTATION ");
        System.out.println();

        System.out.print("Veuillez saisir un nombre entier x : ");
        x=sc.nextInt();

        System.out.print("Veuillez saisir un nombre entier y : ");
        y=sc.nextInt();
        System.out.println();

        Coordonnées c = new Coordonnées(x,y);

        //sortie
        System.out.println("Les variables avant la permutation : ");
        System.out.println("x="+c.getCoordX()+" y="+c.getCoordY());

        //traitement
        c.permuter();

        //sortie
        System.out.println("Les variables après la permutation : ");
        System.out.println("x="+c.getCoordX()+" y="+c.getCoordY());


   }
}

```

Coordonnées.java

```java
/**     PERMUTATION
 *
 * Cette classe représente une paire de coordonnées (x,y)
 *
 * @author Jens Getreu
 * @version 1.0.4
 */
public class Coordonnées
{
    /**
     * La coordonnée x.
     */
    protected int coordX;
    /**
     * La coordonnée y.
     */
    protected int coordY;

    /**
     * Constructeur pour objets de la classe Coordonnées
     * @param coordonné x
     * @param coordonné y
     */
    public Coordonnées(int a, int b)
    {
       // initialiser
       coordX=a;
       coordY=b;
    }

    /**
     * La méthode permuter() échange les contenus des
     * variables x et y.
     */
    public void permuter()
    {
        int tmp = coordX;
        coordX  =coordY;
        coordY = tmp;
    }

    /**
    * La méthode getX() renvoie la coordonnée x
    * @return coordonnée x
    */
    public int getCoordX()
    {
        return coordX;
    }

    /**
    * La méthode getY() renvoie la coordonnée y
    * @return coordonnée y
    */
    public int getCoordY()
    {
        return coordY;
    }
}

```

Class Coordonnées

```
java.lang.Object
   └   Coordonnées
```

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.

---

3 Mini-calculatrice

InterfaceUtilisateur.java

```java
/**         MINICALCULATRICE
 *
 * Basé sur : Les Générations Pascal Chap. 12.6 Ex. 11
 *
 * @author Jens Getreu
 * @version 1.0.1
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("MINICALCULATRICE ");

       System.out.print("Veuillez saisir un nombre réel x : ");
       float x=sc.nextFloat();

       System.out.print("Veuillez saisir un nombre réel y : ");
       float y=sc.nextFloat();

       System.out.print("Veuillez saisir un opérateur '+' ou '-': ");
       String op=sc.next();

       Calculatrice c = new Calculatrice(x);

       if (op.equals("+"))
       {
          System.out.println("x+y = "+c.additionner(y));
       }
       else
       {
          if (op.equals("-"))
          {
             System.out.println("x-y = "+c.soustraire(y));
          }
          else
          {
             System.out.println("L'opérateur '"+ op +"' n'est pas défini.");
          }
       }
       System.out.println();
   }
}

```

Calculatrice.java

```java
/**     MINICALCULATRICE
 *
 * Réalise des opérations de calcul
 *
 * @author Jens Getreu
 * @version 1.0.3
 */
public class Calculatrice
{
    /**
     * L'opérande.
     */
    protected double mémoireInterne;

    /**
     * Constructeur de la classe Calculatrice initialisant la
     * mémoire interne à 0
     */
    public Calculatrice()
    {
        mémoireInterne = 0;
    }

    /**
     * Constructuer de la classe Calculatrice initialisant
     * la mémoire interne à la valeur
     * @param valeur valeur initiale de la mémoire interne
     */
    public Calculatrice(double a)
    {
        mémoireInterne=a;
    }

    /**
     * La méthode additioner() ajoute y à la mémoire interne
     *
     * @param  y   valeur
     * @return     la somme de la mémoire interne et y
     */
    public double additionner(double y)
    {
        mémoireInterne = mémoireInterne + y;
        return  mémoireInterne;
    }

    /**
     * La méthode soustraire() soustrait y de la mémoire interne
     *
     * @param  y   valeur
     * @return     la soustraction de la mémoire interne et y
     */
    public double soustraire(double y)
    {
        mémoireInterne = mémoireInterne - y;
        return  mémoireInterne;
    }
}

```

Class Calculatrice

```
java.lang.Object
   |
   +--Calculatrice
```

```
public class Calculatriceextends java.lang.Object
```

MINICALCULATRICE Réalise des opérations de calcul

Version:
1.0.3
Author:
Jens Getreu

Field Summary

protected double	**mémoireInterne** 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 additioner() 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`**`(doublea)`

Constructuer 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`**`(doubley)`

La méthode additioner() ajoute y à la mémoire interne

Parameters:
`y` - valeur
Returns:
la somme de la mémoire interne et y

---

soustraire

`public double` **`soustraire`**`(doubley)`

La méthode soustraire() soustrait y de la mémoire interne

Parameters:
`y` - valeur
Returns:
la soustraction de la mémoire interne et y

4 Nombre malin

```java
/**     NOMBRE MALIN
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);

        System.out.println("NOMBRE MALIN ");
        System.out.print("Veuillez saisir un nombre réel : ");
        double x=sc.nextDouble();

        NombreMalin nm = new NombreMalin(x);

        System.out.println("Bonjour, je suis un nombre malin. ");
        System.out.println("Je me connais bien: ");

        if (nm.isPositif())
        {
             System.out.println("Je suis positif.");
        }

        if (nm.isNégatif())
        {
             System.out.println("Je suis négatif.");
        }

        if (nm.isPair())
        {
             System.out.println("Je suis pair.");
        }

        if (nm.isImpair())
        {
             System.out.println("Je suis impair.");
        }

        if (nm.isJeune())
        {
             System.out.println("Je suis jeune (<30).");
        }

        if (nm.isVieux())
        {
             System.out.println("Je suis vieux (>=30).");
        }

        if (nm.isDécimal())
        {
             System.out.println("J'ai une partie fractionnaire.");
        }

        if (nm.isEntier())
        {
             System.out.println("Je suis un nombre entier.");
        }
        System.out.println();
   }
```

```
63    }
64
```

NombreMalin.java

```java
/**     NOMBRE MALIN
 *
 * Cette classe représente un nombre
 * capable de renseigner sur soi-même
 *
 * @author Jens Getreu
 * @version 1.0.3
 */
public class NombreMalin
{
    /**
     * Le nombre réel.
     */
    protected double nombre;

    /**
     * Constructeur pour objets de la classe
     * NombreMalin.
     * @param nombre nombre
     */
    public NombreMalin(double a)
    {
       // initialiser
       nombre=a;
    }

    /**
     * La méthode isPositif() permet de tester si
     * le nombre est positif
     * @return vrai pour un nombre >=0
     */
    public boolean isPositif()
    {
        return nombre >= 0;
    }

    /**
     * La méthode isNégatif() permet de tester si
     * le nombre est négatif
     * @return vrai pour un nombre < 0
     */
    public boolean isNégatif()
    {
        return nombre < 0;
    }

    /**
     * La méthode isPair() permet de tester si
     * le nombre est pair
     * @return vrai pour un nombre pair
     */
    public boolean isPair()
    {
        return isEntier() && (nombre%2 == 0);
    }


    /**
     * La méthode isPair() permet de tester si
     * le nombre est impair
     * @return vrai pour un nombre impair
     */
```

```
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 isRéel() 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
    └  NombreMalin
```

```
public class NombreMalinextends 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 isRéel() 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	**isPair**() La méthode isPair() permet de tester si le nombre est pair
boolean	**isPositif**() La méthode isPositif() permet de tester si le nombre est positif
boolean	**isVieux**() La méthode isVieux() 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, wait

Field Detail

nombre

```
protected double nombre
```

Le nombre réel.

Constructor Detail

NombreMalin

```
public NombreMalin(doublea)
```

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 isRéel() 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

5.1 Année bissextile

Variante : structure alternative imbriquée

InterfaceUtilisateur.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : structure alternative imbriquée
 *
 * Voir : Les Générations Pascal Chap. 12.6 Ex. 11
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("ANNÉE BISSEXTILE");

       System.out.print("Veuillez saisir une année : ");
       int a = sc.nextInt();

       Année an = new Année(a);

       if (an.isAnnéeBissextile() )
       {
           System.out.println("Il s'agit d'une année bissextile.");
       }
       else
       {
           System.out.println("Il ne s'agit pas d'une année bissextile.");
       }
       System.out.println();
   }
}

```

Année.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : structure alternative imbriquée
 *
 * Cette classe représente une année
 *
 * @author Jens Getreu
 * @version 1.0.2
 */
public class Année
{
    /**
     * L'an.
     */
    protected int an;

    /**
     * Constructeur.
     * @param x année
     */
    public Année(int x)
    {
        an = x;
    }

    /**
     * Cette méthode détermine si une année est bissextile ou non
     *
     * @return     vrai pour les années bissextiles
     */
    public boolean isAnnéeBissextile()
    {
        boolean b = false;  // true si bissextile

        if (an % 4 == 0)
        {
          if (an % 100 == 0)
          {
             if (an % 400 == 0)
             {
                 b = true;
             }
             else b = false;
          }
          else
          {
             b = true;
          }
        }
        else
        {
           b = false;
        }
        return b;
    }
}

```

Class Année

```
java.lang.Object
    └ Année
```

```
public class Annéeextends 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(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

5.2 Année bissextile

Variante : opérateurs logiques

InterfaceUtilisateur.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : opérateurs logiques
 *
 * Les Générations Pascal Chap. 12.6 Ex. 11
 *
 * @author Jens Getreu
 * @version 1.0.1
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("ANNÉE BISSEXTILE");

       System.out.print("Veuillez saisir une année : ");
       int a = sc.nextInt();

       Année an = new Année(a);

       if (an.isAnnéeBissextile() )
       {
           System.out.println("Il s'agit d'une année bissextile.");
       }
       else
       {
           System.out.println("Il ne s'agit pas d'une année bissextile.");
       }
       System.out.println();
   }
}

```

Année.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : opérateurs logiques
 *
 * Cette classe représente une année
 *
 * @author Jens Getreu
 * @version 1.0.3
 */
public class Année
{
    /**
     * L'an.
     */
    protected int an;

    /**
     * Constructeur.
     * @param x année
     */
    public Année(int x)
    {
        an = x;
    }

    /**
     * Cette méthode détermine si une année est bissextile ou non
     *
     * @return     vrai pour les années bissextiles
     */
    public boolean isAnnéeBissextile()
    {
        boolean b = false;

        if ((an % 4 != 0) ||
            (an % 4 == 0) && (an % 100 == 0) && (an % 400 != 0) )
        {
                 b = false;
        }
        else
        {
                 b = true;
        }

        return b;
    }

    //solution alternative équivalente
    public boolean isAnnéeBissextile2()
    {
        boolean b = !((an % 4 != 0) ||
                      (an % 4 == 0) && (an % 100 == 0)
                                    && (an % 400 != 0) ) ;

        return b;
    }

    //solution alternative équilvalente
    public boolean isAnnéeBissextile3()
    {
        return !((an % 4 != 0) ||
                      (an % 4 == 0) && (an % 100 == 0)
                                    && (an % 400 != 0) ) ;
```

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

Class Année

```
java.lang.Object
    └── Année
```

```
public class Annéeextends 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()`

---

6 Équation du second degré

InterfaceUtilisateur.java

```java
/**     RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ
 *
 * Basé sur: Les Générations Pascal Chap. 13.4 Ex. 5
 *
 * @author Jens Getreu
 * @version 1.2.0
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("RÉSOUDRE L'ÉQUATION DE SECOND DEGRÉ ");
       System.out.println(" a*X^2 + b*X + c = 0 ");
       System.out.println();

       System.out.print("Veuillez saisir le coefficient a : ");
       double a=sc.nextDouble();

       System.out.print("Veuillez saisir le coefficient b : ");
       double b=sc.nextDouble();

       System.out.print("Veuillez saisir le coefficient c : ");
       double c=sc.nextDouble();

       Polynôme poly = new Polynôme(a,b,c);

       if (poly.getNombreZéros()==0)
       {
          System.out.println("Pas de solution.");
       }
       else
       {
           if (poly.getNombreZéros()==1)
           {
               System.out.println("Une solution : "+poly.getX1());
           }
           else
           {
               System.out.println("Deux solutions : "+poly.getX1()+
                                  " et "+poly.getX2());
           }
       }
       System.out.println();
   }
}

```

```java
import java.math.*;
/**     RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ
 *
 * Cette classe représente un polynôme de second dégré.
 * a*X^2 + b*X + c sous sa forme factorisée.
 *
 * @author Jens Getreu
 * @version 1.2.0
 */
public class Polynôme
{
    /**
     * Le nombre de racines du polynôme
     * (égal au nombre de zéros).
     */
    protected int nombreZéros;
    /**
     * Le coefficient a du polynôme factorisé.
     */
    protected double coeffA;
    /**
     * La racine X1 du polynôme factorisé.
     */
    protected double solX1;
     /**
     * La racine X2 du polynôme factorisé.
     */
    protected double solX2;

    /**
     * Constructeur pour objets de la classe Polynôme.
     * Il recherche une représentation factorisée du
     * polynôme sous forme a(X-X1)(X-X2).
     *
     * Remarque: si le nombre de zéros (=racines)
     * du polinôme est égal à 1 on peut écrire le
     * polynôme sous forme
     * a(X-X1)^2 ou bien
     * a(X-X1)(X-X2) avec X1=X2
     *
     * Un objet de la classe Polynom est inéchangeable comme
     * c'est le cas pour les Strings.
     *
     * @param a coefficiant a
     * @param b coefficiant b
     * @param c coefficiant c
     */
    public Polynôme(double a, double b, double c) {
        // factoriser le polynôme
        coeffA = a;
        double delta = b*b - 4*a*c;
        if (delta<0)
        {
            nombreZéros = 0;
        }
        else
        {
            if (delta==0)
            {
                solX1 = -b/(2*a);
                solX2 = solX1;
                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
    └  Polynôme
```

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

RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ Cette classe représente un polynôme de second dégré. a*X^2 + b*X + c sous sa forme factorisée.

Version:
1.2.0
Author:
Jens Getreu

Field Summary

protected double	**coeffA** Le coefficient a du polynôme factorisé.
protected int	**nombreZéros** Le nombre de racines du polynôme (égal au nombre de zéros).
protected double	**solX1** La racine X1 du polynôme factorisé.
protected double	**solX2** 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	**getA**() La méthode getA() renvoie le paramètre a

int	**getNombreZéros**() La méthode getNombreZéros() renvoie le nombre de zéros (=racines) du polynôme calculées par le constructeur.
double	**getX1**() La méthode getX1() renvoie la première solution calculée par le constructeur.
double	**getX2**() La méthode getX2 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, 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(doublea,
                doubleb,
                doublec)
```

Constructeur pour objets de la classe Polynôme. Il recherche une représentation factorisée du polynôme sous forme a(X-X1)(X-X2). Remarque: si le nombre de zéros (=racines) du polinôme est égal à 1 on peut écrire le polynôme sous forme a(X-X1)^2 ou bien a(X-X1)(X-X2) avec X1=X2 Un objet de la classe Polynom est inéchangeable comme c'est le cas pour les Strings.

Parameters:

`a` - coefficiant a
`b` - coefficiant b
`c` - coefficiant 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

---

7 Vérifier un mot de passe

InterfaceUtilisateur.java

```java
/**     VÉRIFIER LE MOT DE PASSE
 *
 * Basé sur Les Générations Pascal Chap. 15.2.3 Exemple 1
 *
 * @author Jens Getreu
 * @version 1.0.1
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);
       String s ="";

       MotDePasse m = new MotDePasse ();
       /*
       //Extension possible
       System.out.println("CHANGER LE MOT DE PASSE");
       System.out.print("Veuillez saisir un nouveau mot de passe : ");
       m.setMotDePasse(sc.nextLine());
       */

       System.out.println("VÉRIFIER LE MOT DE PASSE");



       while (m.getNombreEssais()<3 )
       {
           System.out.print("Veuillez saisir votre mot de passe : ");
           s=sc.nextLine();
           if (m.isContenuÉgal(s))
           {
               break;
           }
           else
           {
               System.out.println("Erreur. Il vous reste "+
                           (3-m.getNombreEssais())+" essais.");
           }
       }

       if (m.isContenuÉgal(s))
       {
           System.out.println("Le mot de passe est correct.");
       }
       else
       {
           System.out.println("Le mot de passe n'est pas correct.");
       }

       System.out.println();
   }
}

```

```java
/**     VÉRIFIER LE MOT DE PASSE
 *
 * Cette classe représente un mot de passe
 *
 * @author Jens Getreu
 * @version 1.0.3
 */
public class MotDePasse
{
    /**
     * La chaîne de caractères contenant le
     * mot de passe.
     */
    protected String motSecret;

    /**
     * Le nombre d'interrogations du
     * mot de passe.
     */
    protected int essais;

    /**
     * Constructeur
     */
    public MotDePasse()
    {
        motSecret="hommik";
        essais=0;
    }

    /**
     * La méthode  isContenuÉgal() permet de tester si
     * le contenu d'un String est égal au contenu du
     * mot de passe secret.
     * @return     vrai si les deux sont identiques
     */
    public boolean isContenuÉgal(String s)
    {
        boolean identique =  s.equals(motSecret);
        if (identique)
        {
            essais = 0;
        }
        else
        {
            essais = essais + 1;
        }
        return identique;
    }


    /**
     * La méthode  getNombreEssais() renvoie le nombre
     * d'interrogations du mot de passe secret.
     * @return     le nombre d'interrogations
     */
    public int getNombreEssais()
    {
        return essais;
    }


```

```
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
    └  MotDePasse
```

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	**essais** Le nombre d'interrogations du mot de passe.
protected java.lang.String	**motSecret** La chaîne de caractères contenant le mot de passe.

Constructor Summary

MotDePasse() Constructeur	

Method Summary

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

---

8 Simplifier une fraction

InterfaceUtilisateur.java

```java
/**     SIMPLIFIER UNE FRACTION
 *
 * Basé sur Les Générations Pascal Chap. 15.6 Ex 10
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);

        //saisie
        System.out.println("SIMPLIFIER UNE FRACTION");
        System.out.print("Veuillez saisir le numérateur n : ");
        int n=sc.nextInt();

        System.out.print("Veuillez saisir le dénominateur d : ");
        int d=sc.nextInt();
        System.out.println();

        Fraction f = new Fraction(n,d);
        f.simplifier();

        //Sortie
        System.out.println("Le numérateur simplifié n = "
                                +f.getNum() );
        System.out.println("Le dénominateur simplifié d = "
                                +f.getDénom() );
        System.out.println();
   }
}
```

Fraction.java

```java
/**     SIMPLIFIER UNE FRACTION
 *
 * Cette classe représente une fraction
 *
 * @author Jens Getreu
 * @version 1.0.5
 */
public class Fraction
{
    /**
     *  Le numérateur.
     */
    protected int num;

    /**
     *  Le dénominateur.
     */
    protected int dénom;

    /**
     * Constructeur
     * @param a numérateur
     * @param b dénominateur
     */
    public Fraction(int a, int b)
    {
        num = a;
        dénom = b;
    }

    /**
     * La méthode pgcd() calcule le plus grand commun
     * diviseur du numérateur et du dénominateur.
     * @ return pgcd
     */
    public int pgcd()
    {
        int a = num;
        int b = dénom;
        int reste = -1;

        //calculer le PGCD
        //variante 1
        //la condition est placée avant le bloc d'instructions
        while (reste != 0)
        {
            reste = a %b ;
            a = b;
            b = reste;
        }
        //le pgcd se trouve dans a
        return a;
    }


    /**
     * La méthode pgcd() calcule le plus grand commun
     * diviseur du numérateur et du dénominateur.
     * @ return pgcd
     */
    /*
    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
   |
   +--Fraction
```

```
public class Fractionextends java.lang.Object
```

SIMPLIFIER UNE FRACTION Cette classe représente une fraction

Version:
1.0.5
Author:
Jens Getreu

Field Summary

protected int	**dénom** Le dénominateur.
protected int	**num** Le numérateur.

Constructor Summary

Fraction(int a, int b) Constructeur	

Method Summary

int	**getDénom**() La méthode getDénom() renvoie le dénominateur
int	**getNum**() La méthode getNum() renvoie le numérateur
int	**pgcd**() 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.

---

9 Nombre premier

InterfaceUtilisateur.java

```java
/**     NOMBRE PREMIER
 *
 * Les Générations Pascal Chap. 12.6 Ex. 9
 *
 * @author Jens Getreu
 * @version 1.0.4
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("NOMBRE PREMIER");

       System.out.print("Veuillez saisir un nombre entier p : ");
       int p=sc.nextInt();

       NombreEntier n = new NombreEntier(p);

       if( n.isNombrePremier() )
       {
            System.out.println("p est nombre premier.");
       }
       else
       {
            System.out.println("p n'est pas nombre premier.");
       }

       /*
       // extension possible de l'exercice
       if( n.isNombrePair() ) {
            System.out.println("p est pair.");
       }
       else {
            System.out.println("p est impair.");
       }
       */

       sc.nextLine();
   }
}

```

NombreEntier.java

```java
/**     NOMBRE PREMIER
 *
 * Cette classe représente un nombre entier
 *
 * @author Jens Getreu
 * @version 1.0.4
 */
public class NombreEntier
{
    /**
     * Le nombre entier.
     */
    protected int nombre;

    /**
     * Constructeur
     */
    public NombreEntier(int p)
    {
        nombre = p;
    }

    /**
     * Cette méthode informe s'il s'agit d'un nombre
     * premier
     *
     * @return  vrai si le nombre est premier
     */
    public boolean isNombrePremier()
    {
        int i=2;
        while((i < nombre)  &&  (nombre % i != 0))
        {
            i=i+1;
        }
        return i == nombre;
    }

    /**
     * Cette méthode informe s'il s'agit d'un nombre
     * pair (Extension possible de l'exercice).
     *
     * @return  vrai si le nombre est pair
     */
    public boolean isNombrePair()
    {
        return nombre % 2 == 0;
    }
}

```

Class NombreEntier

```
java.lang.Object
   |
   +--NombreEntier
```

```
public class NombreEntierextends 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

---

10 Jeu « deviner nombre »

InterfaceUtilisateur.java

```java
/**     DEVINER NOMBRE
 *
 * Basé sur Les Générations Pascal Chap. 17.5 Ex. 8
 *
 * @author Jens Getreu
 * @version 1.1.5
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("DEVINER NOMBRE : ");
       System.out.print("Limite inférieur : ");
       int limInf = sc.nextInt();
       System.out.print("Limite supérieur : ");
       int limSup = sc.nextInt();

       NombreSecret j = new NombreSecret(limInf,limSup);

       System.out.println("*** Le jeu démarre, vous avez 3 essais ***");

       int estimation=0;

       while (j.getNombreEssais() < 2*3)
       {
            System.out.print("Votre estimation : ");
            estimation=sc.nextInt();
            if (j.isÉgal(estimation))
            {
               break;
            }
            else
            {
                if (j.isTropPetit(estimation))
                {
                    System.out.println("   trop petit ...");
                }
                else
                {
                    System.out.println("   trop grand...");
                }
            }
       }

       if (j.isÉgal(estimation))
       {
            System.out.println("Vous avez gagné!");
       }
       else
       {
            System.out.println("Vous avez perdu.");
       }
       System.out.println();
       sc.nextLine();
   }
}

```

NombreSecret.java

```java
/**         DEVINER NOMBRE
 *
 * La classe NombreSecret représente un nombre aléatoire et
 * quelques méthodes pour l'interroger
 * @author Jens Getreu
 * @version 1.1.4
 */
public class NombreSecret
{
    /**
     * Le nombre aléatoire à deviner.
     */
    protected int nombre;
    /**
     * Le nombre d'essais.
     */
    protected int essais;

    /**
     * Le constructeur pour objets de la classe Jeu initialise
     * le nombre aléatoire secret.
     */
    public NombreSecret(int limInf, int limSup)
    {
        essais = 0;
        nombre = (int)(Math.random()*(limSup-limInf+1)) + limInf;
    }

    /**
     * La méthode isTropGrand() compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param  y   estimation
     * @return     vrai si y est supérieur au nombre secret
     */
    public boolean isTropGrand(int y)
    {
        essais = essais +1;
        return y > nombre;
    }

    /**
     * La méthode isTropPetit() compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param  y   estimation
     * @return     vrai si y est inférieur au nombre secret
     */
    public boolean isTropPetit(int y)
    {
        essais = essais +1;
        return y < nombre;
    }

    /**
     * La méthode isÉgal() compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param  y   estimation
     * @return     vrai si y est égal au nombre secret
     */
    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
   └   NombreSecret
```

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	**essais** Le nombre d'essais.
protected int	**nombre** Le nombre aléatoire à deviner.

Constructor Summary

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

Method Summary

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

---

11 Jeu du loup

```java
/**     JEU DU LOUP
 *
 *
 * @author Jens Getreu
 * @version 1.0.7
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);

        System.out.println("JEU DU LOUP");
        System.out.print("Veuillez saisir l'avance de O sur X : ");
        int d=sc.nextInt();

        Piste p = new Piste(d);

        // Variante condition est placée avant le bloc d'instructions
        boolean c =true;
        while (c)
        {
            System.out.print(p.dessiner());
            System.out.print(" Votre pas : ");
            c=p.avancer(sc.nextInt());
        }


        System.out.println(p.dessiner());
        if ((p.getPosX() == p.getPosO()) && (p.getPosO() <= p.POS_MAX) )
        {
            System.out.println("Vous avez attrapé le loup!");
        }
        else
        {
            System.out.println("Le loup vous a échappé.");
        }
   }

   /*
   public static void main(String[] args)
   {
        java.util.Scanner sc = new java.util.Scanner(System.in);

        System.out.println("JEU DU LOUP");
        System.out.print("Veuillez saisir l'avance de O sur X : ");
        int d=sc.nextInt();

        Piste p = new Piste(d);

        // Variante condition est placée après le bloc d'instructions
        do
        {
            System.out.print(p.dessiner());
            System.out.print(" Votre pas : ");
        }
        while (p.avancer(sc.nextInt()))
        System.out.println(p.dessiner());

        if ((p.getPosX() == p.getPosO()) && (p.getPosO() <= 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
```

```java
/**     JEU DU LOUP
 *
 * Cette classe représente une piste et
 * deux coureurs O et X. X essaie de rattraper
 * O alors que O avance aléatoirement.
 *
 * @author Jens Getreu
 * @version 1.0.4
 */
public class Piste
{
    /**
     * La largeur de la piste.
     */
    public static final int POS_MAX = 20;
    /**
     * Le nombre de pas maximal.
     */
    public static final int PAS_MAX = 5;
    /**
     * Le nombre de pas minimal.
     */
    public static final int PAS_MIN = 1;
    /**
     * La position du pion O (loup).
     */
    protected int posO;
    /**
     * La position du pion X (joueur).
     */
    protected int posX;

    /**
     * Un constructeur pour objets de la classe
     * Piste
     * @param avanceO l'avance initiale de O sur X
     */
    public Piste(int avanceO)
    {
       // initialiser la piste
       posO = 1 + avanceO;
       posX = 1;
    }

    /**
     * La méthode avancer() avance le pion X
     * de pasX et le pion O aléatoirement
     * @param pasX pas du pion X
     * @return faux si le jeu est terminé
     */
    public boolean avancer(int pasX)
    {
        if ( pasX >= PAS_MIN && pasX <= PAS_MAX )
        {
            posX = posX + pasX;
        }

        posO = posO + (int)(Math.random()
                    *(PAS_MAX-PAS_MIN+1))+ PAS_MIN;

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

Constructor Summary

Piste(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(intavanceO)
```

Un constructeur pour objets de la classe Piste

Parameters:
`avanceO` - 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

---

12 Minimum-Maximum

InterfaceUtilisateur.java

```java
/**     MIN-MAX
 *
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("MIN-MAX");

       NombrePositif max = new NombrePositif(NombrePositif.MIN);
       NombrePositif min = new NombrePositif(NombrePositif.MAX);

       int p=1;
       while (true)
       {
           System.out.print("Veuillez saisir un nombre entier entre 0 et 1000
(-1 pour terminer) : ");
           p=sc.nextInt();

           NombrePositif np = new NombrePositif(p);

           if (!np.isValide())
           {
               break;
           }

           if (min.isSupérieur(np))
           {
               min=np;
           }

           if (max.isInférieur(np))
           {
               max=np;
           }
       }
       System.out.println("Le minimum = "+min.getValeur());
       System.out.println("Le maximum = "+max.getValeur());
   }
}

```

NombrePositif.java

```java
/**     MIN-MAX
 *
 * Cette classe représente un nombre entier positif.
 *
 * @author Jens Getreu
 * @version 1.0.4
 */
public class NombrePositif
{
    /**
     * La valeur maximale valide d'un NombrePositif.
     * */
    public static final int MAX = 1000;

    /**
     * La valeur minimale valide d'un NombrePositif.
     * */
    public static final int MIN = 0;


    /**
     * La représentation d'un nombre non valide.
     * */
    public static final int NON_VALIDE = -1;

    /**
     * Le nombre positif entre MIN et MAX ou non-valide.
     * */
    protected int nombre;

    /**
     * Ce constructeur génère un nombre entier positif à partir
     * du paramètre a.
     * Un nombre non valide est représenté par -1 par convention.
     * @param a nombre
     * */
    public NombrePositif(int a)
    {
        if ( (a>=MIN) && (a<=MAX) )
        {
            nombre=a;
        }
        else
        {
            nombre=NON_VALIDE;
        }
    }

    /**
     * Cette méthode renvoie la valeur du NombrePositif
     * @return  valeur
     */
    public int getValeur()
    {
        return nombre;
    }

    /**
     * Cette méthode compare deux "NombrePositif"
     * @param a NombrePositif à comparer
     * @return  vrai si "nombre" est supérieur à "a"
     */
```

```
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 isValide()
86      {
87          return nombre != NON_VALIDE;
88      }
89
90
91  }
92
```

Class NombrePositif

```
java.lang.Object
    └── NombrePositif
```

```
public class NombrePositifextends 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	**isValide**() 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

---

13 Sapin de Noël

InterfaceUtilisateur.java

```java
/**     SAPIN DE NOËL
 *
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.println("SAPIN DE NOËL");
       System.out.print("Veuillez saisir la hauteur : ");
       int m=sc.nextInt();

       Sapin s = new Sapin(m);

       System.out.println(s.dessiner());

       sc.nextLine();
   }
}

```

Sapin.java

```java
/**     SAPIN DE NOËL
 *
 * Cette classe représente un sapin de noël
 *
 * @author Jens Getreu
 * @version 1.0.3
 */
public class Sapin
{
    /**
     *La hauteur du sapin.
     */
    protected int hauteur;

    /**
     * Constructeur
     *  @param  h hauteur du sapin
     */
    public Sapin(int h)
    {
       hauteur=h;
    }

    /**
     * La méthode dessiner() renvoie le dessin du sapin
     *
     * @return     le dessin du sapin
     */
    public String dessiner()
    {
       String s="";

       for (int j = 1; j <= hauteur; j++)
       {
           for (int i = 1; i <= j; i++)
           {
               s = s + (i + " ");
           }
           s = s+"\n";
       }
       return s;
    }
}

```

Class Sapin

```
java.lang.Object
    └ Sapin
```

```
public class Sapinextends 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

---

14 Décomposition en produit de facteurs_premiers

InterfaceUtilisateur.java

```java
/**     DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS
 *
 *
 * @author Jens Getreu
 * @version 1.0.2
 * */
import java.util.*;
public class InterfaceUtilisateur
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS");
       System.out.print("Veuillez saisir un nombre entier p : ");
       int p=sc.nextInt();

       System.out.print("p = 1");
       while (p>1)
       {
           NombrePremier np = new NombrePremier(1,p,p);
           p = p / np.getNombrePremier();
           System.out.print("*"+np.getNombrePremier() );
       }
       System.out.println();
   }
}

```

NombrePremier.java

```java
/**     DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS
 *
 * Cette classe représente un nombre entier
 *
 * @author Jens Getreu
 * @version 1.0.4
 */
public class NombrePremier
{
    /**
     * Le nombre premier.
     */
    protected int np;

    /**
     * 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.)
     * @param limInf limite inférieur du nombre premier
     * @param limSup limite supérieur du nombre premier
     */
    public NombrePremier(int limInf, int limSup)
    {
        np=0;
        for (int n=limInf; n<=limSup;n++)
        {
            int i = 2;     // voir ex. nombre premier
            while((i < n)  &&  (n%i != 0))
            {
                i=i+1;
            }
            if (i==n)
            {    //trouvé
                np = n;
                break;
            }
        }
    }

   /**
     * 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.
     * @param limInf limite inférieur du nombre premier
     * @param limSup limite supérieur du nombre premier
     */

    public NombrePremier(int limInf, int limSup, int dividende)
    {
        np=0;
        for (int n=limInf; n<=limSup;n++)
        {
            int i=2;                        // voir ex. "nombre premier"
            while ((i < n)  &&  (n%i != 0))
            {
                i=i+1;
            }
            if ((i==n) && (dividende%i == 0))
            {    //+ 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
    └ NombrePremier
```

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	**np** 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	**getNombrePremier**() 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
