Java à l'école

Proposition de solutions

11TG, Première année de programmation

Version 2.4.1, Jens Getreu

Table des matières

1 Introduction

Main.java

```java
/**     INTRODUCTION
 *
 *
 * Interface utilisateur pour la classe Main
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
        // saisie
        Scanner sc = new Scanner(System.in);

        System.out.println("GREETING");
        System.out.print("Please type your name: ");

        String n = sc.nextLine();
        //traîtement
        Salute s = new Salute(n);
        // trâitement + sortie
        System.out.println(s.sayHello() );
   }
}

```

Salute.java

```java
/**        INTRODUCTION
 *
 *
 * Cette classe représente une salutation
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class Salute
{
    /**
     * Déclaration et initialisation d'une constante: un mot d'accueil.
     */

    public static final String GREETING="Hello ";
    //public static final MOT_ACCUEIL="Hi ";

    /**
     * Le nom de la personne à saluer
     */
    private String name;    // accès par this.name

    /**
     * Constructeur pour objets de la classe Salute
     * @param name un nom
     */
    public Salute(String name)
    {
       this.name = name;
    }

    /**
    * La méthode sayHello() renvoie un bonjour
    * personnalisé.
    * @return Bonjour personnalisé
    */
    public String sayHello()
    {
        return GREETING + name +". How are you doing?";
    }
}

```

Class Salute

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

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

INTRODUCTION Cette classe représente une salutation

Version:
2.0.0
Author:
Jens Getreu

Field Summary

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

Constructor Summary

Salute(java.lang.String name) Constructeur pour objets de la classe Salute	

Method Summary

java.lang.String	**sayHello**() La méthode sayHello() renvoie un bonjour personnalisé.

Methods inherited from class java.lang.Object

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

Field Detail

GREETING

`public static final java.lang.String **GREETING**`

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

See Also:
Constant Field Values

---

name

`private java.lang.String **name**`

Le nom de la personne à saluer

Constructor Detail

Salute

`public **Salute**(java.lang.Stringname)`

Constructeur pour objets de la classe Salute

Parameters:
`name` - un nom

Method Detail

sayHello

`public java.lang.String **sayHello**()`

La méthode sayHello() renvoie un bonjour personnalisé.

Returns:
Bonjour personnalisé

---

2 Permutation

Main.java

```java
/**     PERMUTATION
 *
 * Basé sur Les Générations Pascal Chap. 8.6 Ex. 6
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   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("Please enter integer x: ");
        x=sc.nextInt();

        System.out.print("Please enter integer y: ");
        y=sc.nextInt();
        System.out.println();

        Coordinates c = new Coordinates(x,y);

        //sortie
        System.out.print("The variables before permutation: ");
        System.out.println("x="+c.getCoordX()+" y="+c.getCoordY());

        //traitement
        c.swap();

        //sortie
        System.out.print("The variables after permutation:  ");
        System.out.println("x="+c.getCoordX()+" y="+c.getCoordY());


   }
}

```

Coordinates.java

```java
/**     PERMUTATION
 *
 * Cette classe représente une paire de coordonnées (x,y)
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class Coordinates
{
    /**
     * La coordonnée x.
     */
    private int coordX;
    /**
     * La coordonnée y.
     */
    private int coordY;

    /**
     * Constructeur pour objets de la classe Coordonnées
     * @param coordX coordonné X
     * @param coordY coordonné Y
     */
    public Coordinates(int coordX, int coordY)
    {
       // initialiser
       this.coordX = coordX;
       this.coordY = coordY;
    }

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

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

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

```

Class Coordinates

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

```
public class Coordinatesextends java.lang.Object
```

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

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**coordX** La coordonnée x.
private int	**coordY** La coordonnée y.

Constructor Summary

Coordinates(int coordX, int coordY) Constructeur pour objets de la classe Coordonnées	

Method Summary

int	**getCoordX**() La méthode getCoordX() renvoie la coordonnée x
int	**getCoordY**() La méthode getCoordY() renvoie la coordonnée y
void	**swap**() Cette méthode 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

```
private int coordX
```

La coordonnée x.

---

coordY

```
private int coordY
```

La coordonnée y.

Constructor Detail

Coordinates

```
public Coordinates(intcoordX,
                   intcoordY)
```

Constructeur pour objets de la classe Coordonnées

Parameters:

`coordX` - coordonné X
`coordY` - coordonné Y

Method Detail

getCoordX

```
public int getCoordX()
```

La méthode getCoordX() renvoie la coordonnée x

Returns:

coordonnée x

---

getCoordY

```
public int getCoordY()
```

La méthode getCoordY() renvoie la coordonnée y

Returns:

coordonnée y

---

swap

```
public void swap()
```

Cette méthode échange les contenus des variables x et y.

3.1 Mini-calculatrice

Variante : Comparaison de nombres

Main.java

```
1    /**         MINICALCULATRICE
2     *
3     *
4     * @author Jens Getreu
5     * @version 2.0.0
6     * */
7    import java.util.*;
8    public class Main
9    {
10      public static void main(String[] args)
11      {
12          Scanner sc = new Scanner(System.in);
13
14          System.out.println("MINI-CALCULATOR ");
15
16          System.out.print("Please enter real number x: ");
17          float x=sc.nextFloat();
18
19          System.out.print("Please enter real number y: ");
20          float y=sc.nextFloat();
21
22          System.out.print("Please choose operator '0' for 'plus' or '1' for
'minus': ");
23          int op=sc.nextInt();
24
25          DataMemory d = new DataMemory(x);
26
27          if (op == 0)
28          {
29             System.out.println("x+y = "+  d.add(y));
30          }
31          else
32          {
33             if (op == 1)
34             {
35                System.out.println("x-y = "+d.substract(y));
36             }
37             else
38             {
39                System.out.println("Operator '"+ op +"' is not defined.");
40             }
41          }
42          System.out.println();
43      }
44   }
45
```

DataMemory.java

```java
/**     MINICALCULATRICE
 *
 * Réalise des opérations de calcul
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class DataMemory
{
    /**
     * L'opérande.
     */
    private double memory;

    /**
     * Constructeur initialisant la
     * mémoire interne à 0
     */
    public DataMemory()
    {
        memory = 0;
    }

    /**
     * Constructuer initialisant
     * la mémoire interne memory à la valeur
     * @param memory valeur initiale de la mémoire interne
     */
    public DataMemory (double memory)
    {
        this.memory = memory;
    }

    /**
     * Cette méthode ajoute y à la mémoire interne
     * memory
     *
     * @param  y   valeur
     * @return     la somme de la mémoire interne et y
     */
    public double add(double y)
    {
        memory = memory + y;
        return  memory;
    }

    /**
     * Cette méthode soustrait y de la mémoire interne
     *
     * @param  y   valeur
     * @return     la soustraction de la mémoire interne et y
     */
    public double substract(double y)
    {
        memory = memory - y;
        return  memory;
    }
}
```

Class DataMemory

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

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

MINICALCULATRICE Réalise des opérations de calcul

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private double	**memory** L'opérande.

Constructor Summary

DataMemory() Constructeur initialisant la mémoire interne à 0
DataMemory(double memory) Constructuer initialisant la mémoire interne memory à la valeur

Method Summary

double	**add**(double y) Cette méthode ajoute y à la mémoire interne memory
double	**substract**(double y) Cette méthode 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

memory

`private double memory`

L'opérande.

Constructor Detail

DataMemory

`public DataMemory()`

Constructeur initialisant la mémoire interne à 0

DataMemory

`public DataMemory(doublememory)`

Constructuer initialisant la mémoire interne memory à la valeur

Parameters:
`memory` - valeur initiale de la mémoire interne

Method Detail

add

`public double add(doubley)`

Cette méthode ajoute y à la mémoire interne memory

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

---

substract

`public double substract(doubley)`

Cette méthode soustrait y de la mémoire interne

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

3.2 Mini-calculatrice

Variante : Comparaison de chaînes de caractères

Main.java

```java
/**         MINICALCULATRICE
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("MINI-CALCULATOR ");

       System.out.print("Please enter real number x: ");
       float x=sc.nextFloat();

       System.out.print("Please enter real number y: ");
       float y=sc.nextFloat();

       System.out.print("Please choose operator '+' or '-': ");
       String op=sc.next();

       DataMemory d = new DataMemory(x);

       if (op.equals("+"))
       {
          System.out.println("x+y = "+  d.add(y));
       }
       else
       {
          if (op.equals("-"))
          {
             System.out.println("x-y = "+d.substract(y));
          }
          else
          {
             System.out.println("Operator '"+ op +"' is not defined.");
          }
       }
       System.out.println();
   }
}

```

DataMemory.java

```java
/**     MINICALCULATRICE
 *
 * Réalise des opérations de calcul
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class DataMemory
{
    /**
     * L'opérande.
     */
    private double memory;

    /**
     * Constructeur initialisant la
     * mémoire interne à 0
     */
    public DataMemory()
    {
        memory = 0;
    }

    /**
     * Constructuer initialisant
     * la mémoire interne memory à la valeur
     * @param memory valeur initiale de la mémoire interne
     */
    public DataMemory (double memory)
    {
        this.memory = memory;
    }

    /**
     * Cette méthode ajoute y à la mémoire interne
     * memory
     *
     * @param  y   valeur
     * @return     la somme de la mémoire interne et y
     */
    public double add(double y)
    {
        memory = memory + y;
        return  memory;
    }

    /**
     * Cette méthode soustrait y de la mémoire interne
     *
     * @param  y   valeur
     * @return     la soustraction de la mémoire interne et y
     */
    public double substract(double y)
    {
        memory = memory - y;
        return  memory;
    }
}
```

Class DataMemory

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

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

MINICALCULATRICE Réalise des opérations de calcul

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private double	**memory** L'opérande.

Constructor Summary

DataMemory() Constructeur initialisant la mémoire interne à 0
DataMemory(double memory) Constructuer initialisant la mémoire interne memory à la valeur

Method Summary

double	**add**(double y) Cette méthode ajoute y à la mémoire interne memory
double	**substract**(double y) Cette méthode 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

memory

```
private double memory
```

L'opérande.

Constructor Detail

DataMemory

```
public DataMemory()
```

Constructeur initialisant la mémoire interne à 0

DataMemory

```
public DataMemory(doublememory)
```

Constructuer initialisant la mémoire interne memory à la valeur

Parameters:
`memory` - valeur initiale de la mémoire interne

Method Detail

add

```
public double add(doubley)
```

Cette méthode ajoute y à la mémoire interne memory

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

---

substract

```
public double substract(doubley)
```

Cette méthode soustrait y de la mémoire interne

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

4 Nombre malin

Main.java

```java
/**     NOMBRE MALIN
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);

        System.out.println("CLEVER NUMBER ");
        System.out.print("Please enter a rational number: ");
        double x=sc.nextDouble();

        CleverNumber cn = new CleverNumber(x);

        System.out.println("Hello. I am a clever number.");
        System.out.println("I know myself well: ");

        if (cn.isPositive())
        {
            System.out.println("I am positive.");
        }

        if (cn.isNegative())
        {
            System.out.println("I am negative.");
        }

        if (cn.isEven())
        {
            System.out.println("I am even.");
        }

        if (cn.isOdd())
        {
            System.out.println("I am odd.");
        }

        if (cn.isYoung())
        {
            System.out.println("I am young (<30).");
        }

        if (cn.isOld())
        {
            System.out.println("I am old (>=30).");
        }

        if (cn.isDecimal())
        {
            System.out.println("I am decimal (def: with fractional part).");
        }

        if (cn.isInteger())
        {
            System.out.println("I am integer (def: without fractional
part).");
        }
        System.out.println();
```

```
62          }
63      }
64
```

CleverNumber.java

```java
/**     NOMBRE MALIN
 *
 * Cette classe représente un nombre
 * capable de renseigner sur soi-même
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class CleverNumber
{
    /**
     * Le nombre réel.
     */
    private double number;

    /**
     * Constructeur pour objets de la classe
     * CleverNumber.
     * @param number nombre
     */
    public CleverNumber(double number)
    {
       // initialiser
       this.number = number; //this.number(attribut), number(paramètre)
    }

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

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

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


    /**
     * Cette méthode permet de tester si
     * le nombre est impair
     * @return vrai pour un nombre impair
     */
```

```
63        public boolean isOdd()
64        {
65            return isInteger() && (number % 2 != 0);
66        }
67
68
69        /**
70         * Cette méthode permet de tester si
71         * le nombre est inférieur à 30
72         * @return vrai pour un nombre < 30
73         */
74        public boolean isYoung()
75        {
76            return number<30;
77        }
78
79        /**
80         * Cette méthode permet de tester si
81         * le nombre est supérieur ou égal à 30
82         * @return vrai pour un nombre >= 30
83         */
84        public boolean isOld()
85        {
86            return number>=30;
87        }
88
89        /**
90         * Cette méthode permet de tester si
91         * le nombre est entier, c. à d. sans partie fractionnelle.
92         * @return vrai pour un nombre entier
93         */
94        public boolean isInteger()
95        {
96            return number==(double)(int)number; // double == double?
97        }
98
99        /**
100        * Cette méthode permet de tester si
101        * le nombre est un nombre décimal, c. à d. avec
102        * partie fractionnelle.
103        * @return vrai pour un nombre décimal
104        */
105       public boolean isDecimal()
106       {
107           return number!=(double)(int)number; //double == double?
108       }
109   }
110
```

Class CleverNumber

```
java.lang.Object
    └── CleverNumber
```

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

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

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private double	**number** Le nombre réel.

Constructor Summary

CleverNumber(double number) Constructeur pour objets de la classe CleverNumber.

Method Summary

boolean	**isDecimal**() Cette méthode permet de tester si le nombre est un nombre décimal, c.
boolean	**isEven**() Cette méthode permet de tester si le nombre est pair
boolean	**isInteger**() Cette méthode permet de tester si le nombre est entier, c.
boolean	**isNegative**() Cette méthode permet de tester si le nombre est négatif
boolean	**isOdd**() Cette méthode permet de tester si le nombre est impair

boolean	**isOld**() Cette méthode permet de tester si le nombre est supérieur ou égal à 30
boolean	**isPositive**() Cette méthode permet de tester si le nombre est positif
boolean	**isYoung**() Cette méthode permet de tester si le nombre est inférieur à 30

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

number

```
private double number
```

Le nombre réel.

Constructor Detail

CleverNumber

```
public CleverNumber(doublenumber)
```

Constructeur pour objets de la classe CleverNumber.

Parameters:

`number` - nombre

Method Detail

isDecimal

```
public boolean isDecimal()
```

Cette méthode permet de tester si le nombre est un nombre décimal, c. à d. avec partie fractionnelle.

Returns:

vrai pour un nombre décimal

---

isEven

```
public boolean isEven()
```

Cette méthode permet de tester si le nombre est pair

Returns:
vrai pour un nombre pair

---

isInteger

```
public boolean isInteger()
```

Cette méthode permet de tester si le nombre est entier, c. à d. sans partie fractionnelle.

Returns:
vrai pour un nombre entier

---

isNegative

```
public boolean isNegative()
```

Cette méthode permet de tester si le nombre est négatif

Returns:
vrai pour un nombre < 0

---

isOdd

```
public boolean isOdd()
```

Cette méthode permet de tester si le nombre est impair

Returns:
vrai pour un nombre impair

---

isOld

```
public boolean isOld()
```

Cette méthode permet de tester si le nombre est supérieur ou égal à 30

Returns:
vrai pour un nombre >= 30

---

isPositive

```
public boolean isPositive()
```

Cette méthode permet de tester si le nombre est positif

Returns:
vrai pour un nombre >=0

---

isYoung

```
public boolean isYoung()
```

Cette méthode permet de tester si le nombre est inférieur à 30

Returns:

vrai pour un nombre < 30

5.1 Année bissextile

Variante : structure alternative imbriquée

Main.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : structure alternative imbriquée
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("LEAP YEAR");

       System.out.print("Please enter a year: ");
       int a = sc.nextInt();

       Year y = new Year(a);

       if (y.isLeapYear() )
       {
           System.out.println(y.getYear()+" is a leap year.");
       }
       else
       {
           System.out.println(y.getYear()+" is not a leap year.");
       }
       System.out.println();
   }
}

```

Year.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : structure alternative imbriquée
 *
 * Cette classe représente une année
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class Year
{
    /**
     * L'an.
     */
    private int year;

    /**
     * Constructeur.
     * @param year année
     */
    public Year(int year)
    {
        this.year = year;
    }

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

        if (year % 4 == 0)
        {
          if (year % 100 == 0)
          {
             if (year % 400 == 0)
             {
                 b = true;
             }
             else
             {
                 b = false;
             }
          }
          else
          {
             b = true;
          }
        }
        else
        {
           b = false;
        }
        return b;
    }
    /**
     * Cet accesseur (getter) retourne l'attribut year.
     * @return year.
     */
    public int getYear()
```

```
63          {
64              return year;
65          }
66      }
67
```

Class Year

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

```
public class Yearextends java.lang.Object
```

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

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**year** L'an.

Constructor Summary

Year(int year) Constructeur.	

Method Summary

int	**getYear**() Cet accesseur (getter) retourne l'attribut year.
boolean	**isLeapYear**() 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

year

`private int year`

L'an.

Constructor Detail

Year

`public Year(intyear)`

Constructeur.

Parameters:

`year` - année

Method Detail

getYear

`public int getYear()`

Cet accesseur (getter) retourne l'attribut year.

Returns:

year.

---

isLeapYear

`public boolean isLeapYear()`

Cette méthode détermine si une année est bissextile ou non

Returns:

vrai pour les années bissextiles

5.2 Année bissextile

Variante : opérateurs logiques

Main.java

```java
/**     ANNÉE BISSEXTILE
 *      Variante : structure alternative imbriquée
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("LEAP YEAR");

       System.out.print("Please enter a year: ");
       int a = sc.nextInt();

       Year y = new Year(a);

       if (y.isLeapYear() )
       {
           System.out.println(y.getYear()+" is a leap year.");
       }
       else
       {
           System.out.println(y.getYear()+" is not a leap year.");
       }
       System.out.println();
   }
}

```

```java
/**     ANNÉE BISSEXTILE
 *      Variante : structure alternative imbriquée
 *
 * Cette classe représente une année
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class Year
{
    /**
     * L'an.
     */
    private int year;

    /**
     * Constructeur.
     * @param year année
     */
    public Year(int year)
    {
        this.year = year;
    }


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

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

        return b;
    }


    //solution alternative équivalente
    public boolean isleapYear2()
    {
        boolean b = !((year % 4 != 0) ||
                      (year % 4 == 0) && (year % 100 == 0)
                                      && (year % 400 != 0) ) ;

        return b;
    }

    //solution alternative équivalente
    public boolean isLeapYear3()
    {
        return !((year % 4 != 0) ||
```

```
63                          (year % 4 == 0) && (year % 100 == 0)
64                                      && (year % 400 != 0) ) ;
65      }
66
67      /**
68       * Cet accesseur (getter) retourne l'attribut year.
69       * @return year.
70       */
71      public int getYear()
72      {
73          return year;
74      }
75
76  }
77
```

Class Year

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

```
public class Yearextends java.lang.Object
```

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

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**year** L'an.

Constructor Summary

Year(int year) Constructeur.	

Method Summary

int	**getYear**() Cet accesseur (getter) retourne l'attribut year.
boolean	**isLeapYear**() 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

year

`private int year`

L'an.

Constructor Detail

Year

`public Year(intyear)`

Constructeur.

Parameters:

`year` - année

Method Detail

getYear

`public int getYear()`

Cet accesseur (getter) retourne l'attribut year.

Returns:

year.

---

isLeapYear

`public boolean isLeapYear()`

Cette méthode détermine si une année est bissextile ou non

Returns:

vrai pour les années bissextiles

---

6 Équation du second degré

```java
/**     RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("FACTOR QUADRATIC EUQUATION (ÉQUATION DE SECOND
DEGRÉ) ");
       System.out.println(" a*X^2 + b*X + c = 0 ");
       System.out.println();

       System.out.print("Please enter coefficient a: ");
       double a=sc.nextDouble();

       System.out.print("Please enter coefficient b: ");
       double b=sc.nextDouble();

       System.out.print("Please enter coefficient c: ");
       double c=sc.nextDouble();

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

       if (poly.getNZeros()==0)
       {
          System.out.println("No solution.");
       }
       else
       {
           if (poly.getNZeros()==1)
           {
               System.out.println("One solution: "+poly.getX1());
           }
           else
           {
               System.out.println("Two solutions: "+poly.getX1()+
                                  " and "+poly.getX2());
           }
       }
       System.out.println();
   }
}

```

Polynom.java

```
1   import java.math.*;
2   /**     RÉSOUDRE L'ÉQUATION DU SECOND DEGRÉ
3    *
4    * Cette classe représente un polynôme de second dégré.
5    * a*X^2 + b*X + c sous sa forme factorisée.
6    *
7    * @author Jens Getreu
8    * @version 2.0.0
9    */
10  public class Polynom
11  {
12      /**
13       * Le nombre de racines du polynôme
14       * (égal au nombre de zéros).
15       */
16      private int nZeros;
17      /**
18       * Le coefficient a du polynôme factorisé.
19       */
20      private double coeffA;
21      /**
22       * La racine X1 du polynôme factorisé.
23       */
24      private double solX1;
25       /**
26       * La racine X2 du polynôme factorisé.
27       */
28      private 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 coefficiant a
45       * @param b coefficiant b
46       * @param c coefficiant c
47       */
48      public Polynom(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              nZeros = 0;
55          }
56          else
57          {
58              if (delta==0)
59              {
60                  solX1 = -b/(2*a);
61                  solX2 = solX1;
62                  nZeros = 1;
```

```
63                }
64                else
65                {
66                    solX1 = (-b+ Math.sqrt(delta))/(2*a);
67                    solX2 = (-b- Math.sqrt(delta))/(2*a);
68                    nZeros = 2;
69                }
70            }
71        }
72
73
74         /**
75         * Cette méthode renvoie le nombre de zéros
76         * (=racines) du polynôme calculées par le constructeur.
77         * @return nombre de solutions
78         */
79        public int getNZeros()
80        {
81            return nZeros;
82        }
83
84         /**
85         * Cette méthode renvoie le paramètre a
86         * @return paramètre a
87         */
88        public double getCoeffA()
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 Polynom

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

```
public class Polynomextends 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:
2.0.0
Author:
Jens Getreu

Field Summary

private double	**coeffA** Le coefficient a du polynôme factorisé.
private int	**nZeros** Le nombre de racines du polynôme (égal au nombre de zéros).
private double	**solX1** La racine X1 du polynôme factorisé.
private double	**solX2** La racine X2 du polynôme factorisé.

Constructor Summary

Polynom(double a, double b, double c) Constructeur pour objets de la classe Polynôme.

Method Summary

double	**getCoeffA**() Cette méthode renvoie le paramètre a

int	**getNZeros**() Cette méthode 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

```
private double coeffA
```

Le coefficient a du polynôme factorisé.

---

nZeros

```
private int nZeros
```

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

---

solX1

```
private double solX1
```

La racine X1 du polynôme factorisé.

---

solX2

```
private double solX2
```

La racine X2 du polynôme factorisé.

Constructor Detail

Polynom

```
public Polynom(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

getCoeffA

```
public double getCoeffA()
```

Cette méthode renvoie le paramètre a

Returns:

paramètre a

---

getNZeros

```
public int getNZeros()
```

Cette méthode 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

Main.java

```java
/**     VÉRIFIER LE MOT DE PASSE
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);
       String s ="";

       Password m = new Password ();
       /*
       //Extension possible
       System.out.println("CHANGE INITIAL PASSWORD");
       System.out.print("Please enter a new password : ");
       m.setMotDePasse(sc.nextLine());
       */

       System.out.println("CHECK PASSWORD");



       while (m.getNTries()<3 )
       {
           System.out.print("Please enter your password: ");
           s=sc.nextLine();
           if (m.isSecretEqual(s))
           {
               break;
           }
           else
           {
               System.out.println("Error. "+
                           (3 - m.getNTries())+" tries left.");
           }
       }

       if (m.isSecretEqual(s))
       {
           System.out.println("Password is right.");
       }
       else
       {
           System.out.println("Password is wrong.");
       }

       System.out.println();
   }
}

```

Password.java

```java
/**     VÉRIFIER LE MOT DE PASSE
 *
 * Cette classe représente un mot de passe
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class Password
{
    /**
     * La chaîne de caractères contenant le
     * mot de passe.
     */
    private String secretWord;

    /**
     * Le nombre d'interrogations du
     * mot de passe.
     */
    private int nTries;

    /**
     * Constructeur
     */
    public Password()
    {
        secretWord = "hommik";
        nTries = 0;
    }

    /**
     * Cette méthode permet de tester si
     * le contenu d'un String est égal au contenu du
     * mot de passe secret.
     * @param challenge le mot de pass entré par l'utilisateur
     * @return     vrai si les deux sont identiques
     */
    public boolean isSecretEqual(String challenge)
    {
        boolean e =  challenge.equals(secretWord);
        if (e)
        {
            nTries = 0;
        }
        else
        {
            nTries = nTries + 1;
        }
        return e;
    }


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

```

```
63
64      /**
65       * Cette méthode permet de changer
66       * le mot de passe. (Extension optionnelle de l'exercice.)
67       * @param  password   le nouveau mot de passe
68       */
69      public void setPassword(String password)
70      {
71          nTries = 0;
72          secretWord = password;
73      }
74  }
75
```

Class Password

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

```
public class Passwordextends java.lang.Object
```

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

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**nTries** Le nombre d'interrogations du mot de passe.
private java.lang.String	**secretWord** La chaîne de caractères contenant le mot de passe.

Constructor Summary

Password() Constructeur	

Method Summary

int	**getNTries**() Cette méthode renvoie le nombre d'interrogations du mot de passe secret.
boolean	**isSecretEqual**(java.lang.String challenge) Cette méthode permet de tester si le contenu d'un String est égal au contenu du mot de passe secret.
void	**setPassword**(java.lang.String password) Cette méthode 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

nTries

```
private int nTries
```

Le nombre d'interrogations du mot de passe.

---

secretWord

```
private java.lang.String secretWord
```

La chaîne de caractères contenant le mot de passe.

Constructor Detail

Password

```
public Password()
```

Constructeur

Method Detail

getNTries

```
public int getNTries()
```

Cette méthode renvoie le nombre d'interrogations du mot de passe secret.

Returns:

le nombre d'interrogations

---

isSecretEqual

```
public boolean isSecretEqual(java.lang.Stringchallenge)
```

Cette méthode permet de tester si le contenu d'un String est égal au contenu du mot de passe secret.

Parameters:

`challenge` - le mot de pass entré par l'utilisateur

Returns:

vrai si les deux sont identiques

---

setPassword

```
public void setPassword(java.lang.Stringpassword)
```

Cette méthode permet de changer le mot de passe. (Extension optionnelle de l'exercice.)

Parameters:

`password` - le nouveau mot de passe

---

8 Simplifier une fraction

Main.java

```java
/**     SIMPLIFIER UNE FRACTION
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);

        //saisie
        System.out.println("REDUCE FRACTION");
        System.out.print("Please enter numerator n: ");
        int n=sc.nextInt();

        System.out.print("Please enter denominator d: ");
        int d=sc.nextInt();
        System.out.println();

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

        //Sortie
        System.out.println("Reduced numerator n = "
                                +f.getNum() );
        System.out.println("Reduced denominator d = "
                                +f.getDenom() );
        System.out.println();
   }
}

```

Fraction.java

```java
/**     SIMPLIFIER UNE FRACTION
 *
 * Cette classe représente une fraction
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class Fraction
{
    /**
     *  Le numérateur.
     */
    private int num;

    /**
     *  Le dénominateur.
     */
    private int denom;

    /**
     * Constructeur
     * @param num numérateur
     * @param denom dénominateur
     */
    public Fraction(int num, int denom)
    {
        this.num = num;
        this.denom = denom;
    }

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

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


    /**
     * Cette méthode calcule le plus grand commun
     * diviseur du numérateur et du dénominateur.
     * @ return pgcd
     */
    /*
    public int greatestCommonDivisor2()
```

```
63        {
64            int a=num;
65            int b=denom;
66            int remainder=-1;
67
68            //calculer le PGCD
69            //variante 2:
70            //la condition est placée après le bloc d'instructions
71            do {
72                remainder = a % b;
73                a = b;
74                b = remainder;
75            } while (remainder != 0);
76            //le pgcd se trouve dans a
77            return a;
78        }
79        */
80
81        /**
82         * Cette méthode simplifie la fraction en recherchant
83         * le plus grand commun diviseur du numérateur et du dénominateur.
84         */
85        public void reduce()
86        {
87            int x = greatestCommonDivisor(); // ou int x =
this.greatestCommonDivisor();
88            num = num / x;   // int = int / int
89            denom = denom / x;
90        }
91
92
93        /**
94         * La méthode getNum() renvoie le numérateur
95         *
96         * @return numérateur
97         */
98        public int getNum()
99        {
100           return num;
101       }
102
103       /**
104        * La méthode getDenom() renvoie le dénominateur
105        *
106        * @return dénominateur
107        */
108       public int getDenom()
109       {
110           return denom;
111       }
112   }
113
```

Class Fraction

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

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

SIMPLIFIER UNE FRACTION Cette classe représente une fraction

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**denom** Le dénominateur.
private int	**num** Le numérateur.

Constructor Summary

Fraction(int num, int denom) Constructeur	

Method Summary

int	**getDenom**() La méthode getDenom() renvoie le dénominateur
int	**getNum**() La méthode getNum() renvoie le numérateur
int	**greatestCommonDivisor**() Cette méthode calcule le plus grand commun diviseur du numérateur et du dénominateur.

void	**reduce**() Cette méthode simplifie la fraction en recherchant le plus grand commun diviseur 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

denom

```
private int denom
```

Le dénominateur.

---

num

```
private int num
```

Le numérateur.

Constructor Detail

Fraction

```
public Fraction(intnum,
                intdenom)
```

Constructeur

Parameters:

`num` - numérateur
`denom` - dénominateur

Method Detail

getDenom

```
public int getDenom()
```

La méthode getDenom() renvoie le dénominateur

Returns:

dénominateur

---

getNum

`public int getNum()`

La méthode getNum() renvoie le numérateur

Returns:
numérateur

---

greatestCommonDivisor

`public int greatestCommonDivisor()`

Cette méthode calcule le plus grand commun diviseur du numérateur et du dénominateur.

---

reduce

`public void reduce()`

Cette méthode simplifie la fraction en recherchant le plus grand commun diviseur du numérateur et du dénominateur.

---

9 Factorielle

Main.java

```java
/**     FACTORIAL
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       //saisie
       System.out.println("FACTORIAL");
       System.out.print("Please enter a non-negative integer: ");
       int n = sc.nextInt();

       NonNegativeInteger nni = new NonNegativeInteger(n);

       // traîtement + sortie
       System.out.println("The factorial of "+nni.getNumber()+" is:");

       String s = "1";
       for (int i=2; i<=nni.getNumber(); i++)
       {
           s = s + "*" + i;
       }
       s = s + " = " + nni.factorial();
       System.out.println(s);
       System.out.println();
   }
}

```

```java
/**     FACTORIAL
 *
 * Cette classe représente un nombre entier positif.
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class NonNegativeInteger
{
    /**
     * Le nombre positif.
     * */
    private int number;  // accessible par this.number

    /**
     * Ce constructeur génère un nombre entier positif à partir
     * du paramètre number.
     * */
    public NonNegativeInteger(int number)
    {
            this.number = number; // this.number (attribut) = number
(paramètre)
    }

    /**
     * Cet acesseur renvoie la valeur de NonNegativeInteger
     * @return  number
     */
    public int getNumber()
    {
        return number;     // équivalant "return this.number" (pas d'ambiguïté)
    }

    /**
     * Cette méthode renvoie la factorielle de number
     */
    public int factorial()
    {
        int product = 1;
        int i = 1;
        while ( i <= number )
        {
           product = product * i;
           i++;  // abréviation de i = i + 1;
        }
        return product;
    }

    /*
    // Solution alternative
    public int factorial()
    {
        int product = 1;
        for (int i=1; i <= number; i++)
        {
            product = product * i;
        }
        return product;
    }
    */
}
```

Class NonNegativeInteger

```
java.lang.Object
    └  NonNegativeInteger
```

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

FACTORIAL Cette classe représente un nombre entier positif.

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**number** Le nombre positif.

Constructor Summary

NonNegativeInteger(int number) Ce constructeur génère un nombre entier positif à partir du paramètre number.

Method Summary

int	**factorial**() Cette méthode renvoie la factorielle de number
int	**getNumber**() Cet acesseur renvoie la valeur de NonNegativeInteger

Methods inherited from class java.lang.Object

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

Field Detail

number

`private int number`

Le nombre positif.

Constructor Detail

NonNegativeInteger

`public NonNegativeInteger(intnumber)`

Ce constructeur génère un nombre entier positif à partir du paramètre number.

Method Detail

factorial

`public int factorial()`

Cette méthode renvoie la factorielle de number

---

getNumber

`public int getNumber()`

Cet acesseur renvoie la valeur de NonNegativeInteger

Returns:
number

---

10 Nombre premier

Main.java

```java
/**     NOMBRE PREMIER
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("PRIME NUMBER");

       System.out.print("Please enter integer p: ");
       int p=sc.nextInt();

       IntegerNumber n = new IntegerNumber(p);

       if( n.isPrime() )
       {
            System.out.println("p is a prime number.");
       }
       else
       {
            System.out.println("p is not a prime number.");
       }

       /*
       // extension possible de l'exercice
       if( n.isEven() ) {
            System.out.println("p is even.");
       }
       else {
            System.out.println("p is odd.");
       }
       */

       sc.nextLine();
   }
}

```

IntegerNumber.java

```java
/**     NOMBRE PREMIER
 *
 * Cette classe représente un nombre entier
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class IntegerNumber
{
    /**
     * Le nombre entier.
     */
    private int number;

    /**
     * Constructeur
     */
    public IntegerNumber(int number)
    {
        this.number = number;
    }

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

    /*
    public boolean isPrime2()
    {
        int i=2;
        while(i <= number-1)
        {
            if (number % i == 0)
            {
                break;
            }
            i=i+1;
        }
        return i == number;
    }
    */

    /*
    public boolean isPrime3()
    {
        boolean prime= true;
        if (number <= 1)
        {
            prime=false;
        }
```

```
63
64          for(int i=2;i <= number-1;i++)
65          {
66              if (number % i == 0)
67              {
68                  prime=false;
69                  break;
70              }
71          }
72          return prime;
73      }
74      */
75
76      /**
77       * Cette méthode informe s'il s'agit d'un nombre
78       * pair (Extension possible de l'exercice).
79       *
80       * @return  vrai si le nombre est pair
81       */
82      /*
83      public boolean isEven()
84      {
85          return number % 2 == 0;
86      }
87      */
88  }
89
```

Class IntegerNumber

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

```
public class IntegerNumberextends java.lang.Object
```

NOMBRE PREMIER Cette classe représente un nombre entier

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**number** Le nombre entier.

Constructor Summary

IntegerNumber(int number) Constructeur	

Method Summary

boolean	**isPrime**() 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

number

`private int number`

Le nombre entier.

Constructor Detail

IntegerNumber

`public IntegerNumber(intnumber)`

Constructeur

Method Detail

isPrime

`public boolean isPrime()`

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

Returns:
vrai si le nombre est premier

---

11 Jeu « deviner nombre »

Main.java

```java
/**     DEVINER NOMBRE
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("GUESS NUMBER");
       System.out.print("Inferiour limit: ");
       int il = sc.nextInt();
       System.out.print("Superior limit: ");
       int sl = sc.nextInt();

       SecretNumber sn = new SecretNumber(il, sl);

       System.out.println("*** The game starts, you have 3 tries ***");

       int n = 0; // estimation

       while (sn.getNTries() < 2*3)
       {
            System.out.print("Your guess: ");
            n = sc.nextInt();
            if ( sn.equals(n) )
            {
               break;
            }
            else
            {
                if ( sn.isBigger(n) )
                {
                    System.out.println("   too small ...");
                }
                else
                {
                    System.out.println("   too big ...");
                }
            }
       }

       if ( sn.equals(n) )
       {
            System.out.println("You won!");
       }
       else
       {
            System.out.println("You lost.");
       }
       System.out.println();
       sc.nextLine();
   }
}

```

SecretNumber.java

```java
/**         DEVINER NOMBRE
 *
 * La classe SecretNumber représente un nombre aléatoire et
 * quelques méthodes pour l'interroger
 * @author Jens Getreu
 * @version 2.0.1
 */
public class SecretNumber
{
    /**
     * Le nombre aléatoire à deviner.
     */
    private int number;
    /**
     * Le nombre d'essais.
     */
    private int nTries;

    /**
     * Le constructeur pour objets de la classe Jeu initialise
     * le nombre aléatoire secret.
     * @param infLim limite inférieure
     * @param supLim limite supérieure
     */
    public SecretNumber(int infLim, int supLim)
    {
        nTries = 0;
        number = (int)(Math.random()*(supLim-infLim+1)) + infLim;
    }

    /**
     * Cette méthode compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param guess  estimation
     * @return     vrai si number est inférieur à l'estimation.
     */
    public boolean isSmaller(int guess)
    {
        nTries = nTries +1;
        return number < guess;
    }

    /**
     * La méthode isBigger() compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param  guess   estimation
     * @return     vrai si number > guess
     */
    public boolean isBigger(int guess)
    {
        nTries = nTries +1;
        return number > guess;
    }

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

```
63          */
64         public boolean equals(int guess)
65         {
66             nTries = nTries +1;
67             return guess == number;
68         }
69 
70          /**
71          * La méthode getNTries() renvoie le nombre de questions
72          * posées.
73          *
74          * @return     nombre de questions posées
75          */
76         public int getNTries()
77         {
78             return nTries;
79         }
80 
81     }
82 
```

Class SecretNumber

```
java.lang.Object
    └  SecretNumber
```

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

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

Version:
2.0.1
Author:
Jens Getreu

Field Summary

private int	**nTries** Le nombre d'essais.
private int	**number** Le nombre aléatoire à deviner.

Constructor Summary

SecretNumber(int infLim, int supLim) Le constructeur pour objets de la classe Jeu initialise le nombre aléatoire secret.

Method Summary

boolean	**equals**(int guess) La méthode isEqual() compare y avec le nombre secret et renvoie vrai ou faux.
int	**getNTries**() La méthode getNTries() renvoie le nombre de questions posées.
boolean	**isBigger**(int guess) La méthode isBigger() compare y avec le nombre secret et renvoie vrai ou faux.

boolean	**isSmaller**(int guess) Cette méthode 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

nTries

```
private int nTries
```

Le nombre d'essais.

---

number

```
private int number
```

Le nombre aléatoire à deviner.

Constructor Detail

SecretNumber

```
public SecretNumber(intinfLim,
                    intsupLim)
```

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

Parameters:

`infLim` - limite inférieure
`supLim` - limite supérieure

Method Detail

equals

```
public boolean equals(intguess)
```

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

Parameters:

`guess` - estimation

Returns:

vrai si guess est égal au nombre secret

---

getNTries

`public int getNTries()`

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

Returns:
nombre de questions posées

---

isBigger

`public boolean isBigger(intguess)`

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

Parameters:
`guess` - estimation
Returns:
vrai si number > guess

---

isSmaller

`public boolean isSmaller(intguess)`

Cette méthode compare y avec le nombre secret et renvoie vrai ou faux.

Parameters:
`guess` - estimation
Returns:
vrai si number est inférieur à l'estimation.

---

12 Jeu du loup

Main.java

```java
/**     JEU DU LOUP
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
        Scanner sc = new Scanner(System.in);

        System.out.println("TAG GAME");
        System.out.print("Please give O a head start over X. Enter number: ");
        int d=sc.nextInt();

        RunningTrack rt = new RunningTrack(d);

        // Variante condition est placée avant le bloc d'instructions
        boolean c =true;
        while (c)
        {
            System.out.print( rt.draw() );
            System.out.print(" Your step: ");
            c=rt.moveXO( sc.nextInt() );
        }

        System.out.println(rt.draw());
        if ( (rt.getPosX() == rt.getPosO()) && (rt.getPosO() <= rt.POS_MAX) )
        {
            System.out.println("You catched O!");
        }
        else
        {
            System.out.println("O escaped.");
        }
   }

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

        System.out.println("TAG GAME");
        System.out.print("Please give O a head start over X. Enter number : ");
        int d=sc.nextInt();

        RunningTrack rt = new RunningTrack(d);

        // Variante condition est placée après le bloc d'instructions
        do
        {
            System.out.print( rt.draw() );
            System.out.print(" Your step : ");
        }
        while ( rt.moveX( sc.nextInt() ) );

        System.out.println(rt.draw());
        if ((rt.getPosX() == rt.getPosO()) && (rt.getPosO() <= rt.POS_MAX) )
        {
            System.out.println("You catched O!");
```

```
63              }
64              else
65              {
66                  System.out.println("O escaped.");
67              }
68      }
69      */
70  }
71
```

```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 2.0.0
 */
public class RunningTrack
{
    /**
     * La largeur de la piste (running track).
     */
    public static final int POS_MAX = 20;
    /**
     * Le nombre de pas maximal.
     */
    public static final int STEP_MAX = 5;
    /**
     * Le nombre de pas minimal.
     */
    public static final int STEP_MIN = 1;
    /**
     * La position du pion O (loup).
     */
    private int posO;
    /**
     * La position du pion X (joueur).
     */
    private int posX;

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

    /**
     * Cette méthode avance le pion X
     * de stepX et le pion O aléatoirement
     * @param stepX pas du pion X
     * @return faux si le jeu est terminé
     */
    public boolean moveXO(int stepX)
    {
        if ( stepX >= STEP_MIN && stepX <= STEP_MAX )
        {
            posX = posX + stepX;
        }

        posO = posO + (int)(Math.random()
                    *(STEP_MAX-STEP_MIN+1))+ STEP_MIN;

        return (posX < POS_MAX)
               && (posO < POS_MAX)
```

```
63                  && (pos0 != posX);
64      }
65
66      /**
67       * Cette méthode dessine
68       * la piste avec les deux coureurs 0 et X
69       */
70      public String draw()
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       * Cette méthode 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      * Cette méthode 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 RunningTrack

```
java.lang.Object
    └  RunningTrack
```

```
public class RunningTrackextends 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:
2.0.0

Author:
Jens Getreu

Field Summary

static int	**POS_MAX** La largeur de la piste (running track).
private int	**posO** La position du pion O (loup).
private int	**posX** La position du pion X (joueur).
static int	**STEP_MAX** Le nombre de pas maximal.
static int	**STEP_MIN** Le nombre de pas minimal.

Constructor Summary

RunningTrack(int headStartO) Un constructeur pour objets de la classe	

Method Summary

java.lang.String	**draw**() Cette méthode dessine la piste avec les deux coureurs O et X
int	**getPosO**() Cette méthode renvoie la position du coureur O
int	**getPosX**() Cette méthode renvoie la position du coureur X
boolean	**moveXO**(int stepX) Cette méthode avance le pion X de stepX et le pion O aléatoirement

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

POS_MAX

```
public static final int POS_MAX
```

La largeur de la piste (running track).

See Also:
Constant Field Values

---

posO

```
private int posO
```

La position du pion O (loup).

---

posX

```
private int posX
```

La position du pion X (joueur).

STEP_MAX

```
public static final int STEP_MAX
```

Le nombre de pas maximal.

See Also:

Constant Field Values

STEP_MIN

```
public static final int STEP_MIN
```

Le nombre de pas minimal.

See Also:
Constant Field Values

Constructor Detail

RunningTrack

```
public RunningTrack(intheadStart0)
```

Un constructeur pour objets de la classe

Parameters:
`headStart0` - l'avance initiale de O sur X

Method Detail

draw

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

Cette méthode dessine la piste avec les deux coureurs O et X

---

getPosO

```
public int getPosO()
```

Cette méthode renvoie la position du coureur O

Returns:
position de O

---

getPosX

```
public int getPosX()
```

Cette méthode renvoie la position du coureur X

Returns:
position de X

---

moveXO

```
public boolean moveXO(intstepX)
```

Cette méthode avance le pion X de stepX et le pion O aléatoirement

Parameters:

`stepX` - pas du pion X

Returns:

faux si le jeu est terminé

---

13 Minimum-Maximum

Main.java

```
1    /**     MIN-MAX
2     *
3     * @author Jens Getreu
4     * @version 2.1.0
5     *
6     */
7
8    import java.util.*;
9    public class Main
10   {
11      public static void main(String[] args)
12      {
13          Scanner sc = new Scanner(System.in);
14
15          System.out.println("MIN-MAX");
16
17          PositiveNumber np = null;  // reste "null" si aucun np est crée.
18          PositiveNumber max = new PositiveNumber(PositiveNumber.MIN);
19          PositiveNumber min = new PositiveNumber(PositiveNumber.MAX);
20
21          int p=1;
22          while (true)
23          {
24              System.out.print("Please enter an integer from 0 to 1000 (-1 to
quit): ");
25              p=sc.nextInt();
26
27              if ((p >= PositiveNumber.MIN) && (p <= PositiveNumber.MAX))
28              {
29                  np = new PositiveNumber(p);
30              }
31              else
32              {
33                  break;
34              }
35
36              if (min.isBigger(np))
37              {
38                  min=np;
39              }
40
41              if (max.isSmaller(np))
42              {
43                  max=np;
44              }
45          }
46          System.out.println("The minimum is  " + min);  // raccourci de
min.toString()
47          System.out.println("The maximum is  " + max);
48      }
49   }
50
```

PositiveNumber.java

```java
/**     MIN-MAX
 *
 * Cette classe représente un nombre entier positif.
 *
 * @author Jens Getreu
 * @version 2.1.0
 */
public class PositiveNumber
{
    /**
     * La valeur maximale valide d'un PositiveNumber.
     * */
    public static final int MAX = 1000;

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

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

    /**
     * Ce constructeur génère un nombre entier positif à partir
     * du paramètre number.
     *
     * @param number le nombre
     * */
    public PositiveNumber(int number)
    {
            posNum = number;
    }

    /**
     * Cette méthode renvoie la valeur du PositiveNumber
     * @return  valeur
     */
    public int getPosNum()
    {
        return posNum;
    }

    /**
     * Cette méthode compare deux "PositiveNumber"
     * @param number PositiveNumber à comparer
     * @return  vrai si "posNum" est supérieur à "number"
     */
    public boolean isBigger(PositiveNumber number)
    {
        return posNum > number.getPosNum();
    }


    /**
     * Cette méthode compare deux "PositiveNumber"
     * @param number a PositiveNumber à comparer
     * @return  vrai si "posNum" est inférieur à "number"
     */
    public boolean isSmaller(PositiveNumber number)
    {
```

```
63              return posNum < number.getPosNum();
64          }
65
66          /**
67           * @return Représentation textuelle de l'objet.
68           */
69          public String toString()
70          {
71              return String.valueOf(posNum); // int -> String
72          }
73      }
74
```

Class PositiveNumber

```
java.lang.Object
    └  PositiveNumber
```

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

MIN-MAX Cette classe représente un nombre entier positif.

Version:
2.1.0
Author:
Jens Getreu

Field Summary

static int	**MAX** La valeur maximale valide d'un PositiveNumber.
static int	**MIN** La valeur minimale valide d'un PositiveNumber.
private int	**posNum** Le nombre positif entre MIN et MAX ou non-valide.

Constructor Summary

PositiveNumber(int number) Ce constructeur génère un nombre entier positif à partir du paramètre number.

Method Summary

int	**getPosNum**() Cette méthode renvoie la valeur du PositiveNumber
boolean	**isBigger**(PositiveNumber number) Cette méthode compare deux "PositiveNumber"
boolean	**isSmaller**(PositiveNumber number) Cette méthode compare deux "PositiveNumber"

java.lang.String	**toString**()

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Field Detail

MAX

```
public static final int MAX
```

La valeur maximale valide d'un PositiveNumber.

See Also:
Constant Field Values

MIN

```
public static final int MIN
```

La valeur minimale valide d'un PositiveNumber.

See Also:
Constant Field Values

---

posNum

```
private int posNum
```

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

Constructor Detail

PositiveNumber

```
public PositiveNumber(intnumber)
```

Ce constructeur génère un nombre entier positif à partir du paramètre number.

Parameters:
`number` - le nombre

Method Detail

getPosNum

`public int getPosNum()`

Cette méthode renvoie la valeur du PositiveNumber

Returns:
valeur

---

isBigger

`public boolean isBigger(PositiveNumbernumber)`

Cette méthode compare deux "PositiveNumber"

Parameters:
`number` - PositiveNumber à comparer
Returns:
vrai si "posNum" est supérieur à "number"

---

isSmaller

`public boolean isSmaller(PositiveNumbernumber)`

Cette méthode compare deux "PositiveNumber"

Parameters:
`number` - a PositiveNumber à comparer
Returns:
vrai si "posNum" est inférieur à "number"

---

toString

`public java.lang.String toString()`

Overrides:
`toString` in class `java.lang.Object`
Returns:
Représentation textuelle de l'objet.

---

14 Sapin de Noël

Main.java

```java
/**     SAPIN DE NOËL
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       System.out.println("CHRISTMAS TREE");
       System.out.print("Please enter height: ");
       int m=sc.nextInt();

       ChristmasTree ct = new ChristmasTree(m);

       System.out.println(ct.draw());

       sc.nextLine();
   }
}

```

ChristmasTree.java

```java
/**     SAPIN DE NOËL
 *
 * Cette classe représente un sapin de noël
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class ChristmasTree
{
    /**
     *La hauteur du sapin.
     */
    private int height;

    /**
     * Constructeur
     *  @param  height hauteur du sapin
     */
    public ChristmasTree(int height)
    {
       this.height = height;
    }

    /**
     * Cette méthode drenvoie le dessin du sapin
     *
     * @return     le dessin du sapin
     */
    public String draw()
    {
       String s="";

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

```

Class ChristmasTree

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

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

SAPIN DE NOËL Cette classe représente un sapin de noël

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**height** La hauteur du sapin.

Constructor Summary

ChristmasTree(int height) Constructeur

Method Summary

java.lang.String	**draw**() Cette méthode drenvoie le dessin du sapin

Methods inherited from class java.lang.Object

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

Field Detail

height

`private int` **`height`**

La hauteur du sapin.

Constructor Detail

ChristmasTree

`public` **`ChristmasTree`**`(intheight)`

Constructeur

Parameters:

`height` - hauteur du sapin

Method Detail

draw

`public java.lang.String` **`draw`**`()`

Cette méthode drenvoie le dessin du sapin

Returns:

le dessin du sapin

---

15 Décomposition en produit de facteurs_premiers

Main.java

```java
/**     DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS
 *
 *
 * @author Jens Getreu
 * @version 2.0.0
 * */
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("INTEGER FACTORIZATION");
       System.out.print("Please enter integer p: ");
       int p=sc.nextInt();

       System.out.print("p = 1");
       while (p>1)
       {
           PrimeNumber pn = new PrimeNumber(1,p,p);
           p = p / pn.getPrime();
           System.out.print("*"+ pn.getPrime() );
       }
       System.out.println();
   }
}

```

```java
/**     DÉCOMPOSITION EN PRODUIT DE FACTEURS PREMIERS
 *
 * Cette classe représente un nombre entier
 *
 * @author Jens Getreu
 * @version 2.0.0
 */
public class PrimeNumber
{
    /**
     * Le nombre premier.
     */
    private int prime;

    /**
     * Ce constructeur génère le plus petit nombre premier entre
     * infLim et supLim.
     * Si aucun nombre premier existe entre ces deux limites
     * prime est 0.
     * (Ce constructeur n'est pas utilisé, mais sert comme introduction.)
     * @param infLim limite inférieur du nombre premier
     * @param supLim limite supérieur du nombre premier
     */
    public PrimeNumber(int infLim, int supLim)
    {
        prime=0;
        for (int n=infLim; n<=supLim;n++)
        {
            int i = 2;     // voir ex. nombre premier
            while((i < n)  &&  (n%i != 0))
            {
                i=i+1;
            }
            if (i==n)
            {    //trouvé
                prime = n;
                break;
            }
        }
    }

   /**
     * Ce constructeur génère le plus petit nombre premier entre
     * "infLim" et "supLim" à condition qu'il est diviseur de "divisor".
     * Si aucun nombre premier vérifie ces conditions
     * prime est 0.
     * @param infLim limite inférieur du nombre premier
     * @param supLim limite supérieur du nombre premier
     */

    public PrimeNumber(int infLim, int supLim, int divisor)
    {
        prime=0;
        for (int n=infLim; n<=supLim;n++)
        {
            int i=2;                        // voir ex. "nombre premier"
            while ((i < n)  &&  (n%i != 0))
            {
                i=i+1;
            }
            if ((i==n) && (divisor%i == 0))
            {    //+ cond. supp.
```

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

Class PrimeNumber

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

```
public class PrimeNumberextends java.lang.Object
```

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

Version:
2.0.0
Author:
Jens Getreu

Field Summary

private int	**prime** Le nombre premier.

Constructor Summary

PrimeNumber(int infLim, int supLim) Ce constructeur génère le plus petit nombre premier entre infLim et supLim.
PrimeNumber(int infLim, int supLim, int divisor) Ce constructeur génère le plus petit nombre premier entre "infLim" et "supLim" à condition qu'il est diviseur de "divisor".

Method Summary

int	**getPrime**() 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

prime

```
private int prime
```

Le nombre premier.

Constructor Detail

PrimeNumber

```
public PrimeNumber(intinfLim,
                   intsupLim)
```

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

Parameters:

`infLim` - limite inférieur du nombre premier
`supLim` - limite supérieur du nombre premier

PrimeNumber

```
public PrimeNumber(intinfLim,
                   intsupLim,
                   intdivisor)
```

Ce constructeur génère le plus petit nombre premier entre "infLim" et "supLim" à condition qu'il est diviseur de "divisor". Si aucun nombre premier vérifie ces conditions prime est 0.

Parameters:

`infLim` - limite inférieur du nombre premier
`supLim` - limite supérieur du nombre premier

Method Detail

getPrime

```
public int getPrime()
```

Cette méthode renvoie le nombre premier

Returns:

nombre premier

A.1 Hello World

MainFrame.java

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * MainFrame.java
 *
 * Created on 2 sept. 2011, 13:41:03
 */

/**
 *
 * @author getreu
 */
public class MainFrame extends javax.swing.JFrame {

    /** Creates new form MainFrame */
    public MainFrame() {
        initComponents();
    }

    //... folded code ...

    private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
      // saisie
      String n = inputTextField.getText();
      // traîtement
      n = "Hello " + n + ". How are you doing?";
      // sortie
      outputLabel.setText(n);
    }

    /**
    * @param args the command line arguments
    */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new MainFrame().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JTextField inputTextField;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JButton okButton;
    private javax.swing.JLabel outputLabel;
    // End of variables declaration//GEN-END:variables

}

```

A.2 Hello World MCV

MainFrame.java

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * MainFrame.java
 *
 * Created on 2 sept. 2011, 13:41:03
 */

/**
 *
 * @author Jens Getreu
 */
public class MainFrame extends javax.swing.JFrame {
    // Rendre l'objet utilisable dans d'autres méthodes
    private Salute s = null;

    /** Creates new form MainFrame */
    public MainFrame() {
        initComponents();
    }

    //... folded code ...

    private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
        // saisie
        String n = inputTextField.getText();
        // traîtement
        s = new Salute(n);
        // sortie
        outputLabel.setText( s.sayHello() );
    }

    /**
    * @param args the command line arguments
    */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new MainFrame().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JTextField inputTextField;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JButton okButton;
    private javax.swing.JLabel outputLabel;
    // End of variables declaration//GEN-END:variables

}

```

Salute.java

```java
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author getreu
 */
public class Salute {
    public static final String GREETING="Hello ";
    private String name;    // accès par this.name

    public Salute(String name)
    {
       this.name = name;
    }

    public String sayHello()
    {
        return GREETING + name +". How are you doing?";
    }
}

```

A.3 Guessing Game

MainFrame.java

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5
6   /*
7    * MainFrame.java
8    *
9    * Created on 7 sept. 2011, 16:57:45
10   */
11
12  /**
13   *
14   * @author getreu
15   */
16  public class MainFrame extends javax.swing.JFrame {
17      private SecretNumber sn = null;
18
19      /** Creates new form MainFrame */
20      public MainFrame() {
21          initComponents();
22      }
23
29      //... folded code ...
127
128     private void genNumButtonActionPerformed(java.awt.event.ActionEvent evt) {
129         int il = Integer.valueOf( infLimTextField.getText() );
130         int sl = Integer.valueOf( supLimTextField.getText() );
131         sn = new SecretNumber(il, sl);
132         messageLabel.setText("Please enter your guess and press 'Try'.");
133     }
134
135     private void tryButtonActionPerformed(java.awt.event.ActionEvent evt) {
136         int n = Integer.valueOf( guessTextField.getText() );
137
138         if (sn == null)
139         {
140             return;
141         }
142
143         if ( sn.equals(n) )
144         {
145                 messageLabel.setText("You won! You needed "
146                             + (sn.getNTries()+1)/2 + " tries.");
147         }
148         else
149         {
150            if ( sn.isBigger(n) )
151            {
152               messageLabel.setText("Too small! Try again. You had "
153                             + sn.getNTries()/2 + " tries.");
154            }
155            else
156            {
157               messageLabel.setText("Too big! Try again. You had "
158                             + sn.getNTries()/2 + " tries.");
159            }
160         }
161
162     }
163
164     /**
```

```
165      * @param args the command line arguments
166      */
167     public static void main(String args[]) {
168         java.awt.EventQueue.invokeLater(new Runnable() {
169             public void run() {
170                 new MainFrame().setVisible(true);
171             }
172         });
173     }
174 
175     // Variables declaration - do not modify//GEN-BEGIN:variables
176     private javax.swing.JButton genNumButton;
177     private javax.swing.JTextField guessTextField;
178     private javax.swing.JTextField infLimTextField;
179     private javax.swing.JLabel jLabel1;
180     private javax.swing.JLabel jLabel2;
181     private javax.swing.JLabel jLabel3;
182     private javax.swing.JLabel jLabel4;
183     private javax.swing.JLabel messageLabel;
184     private javax.swing.JTextField supLimTextField;
185     private javax.swing.JButton tryButton;
186     // End of variables declaration//GEN-END:variables
187 
188 }
189 
```

SecretNumber.java

```java
/**         DEVINER NOMBRE
 *
 * La classe SecretNumber représente un nombre aléatoire et
 * quelques méthodes pour l'interroger
 * @author Jens Getreu
 * @version 2.0.1
 */
public class SecretNumber
{
    /**
     * Le nombre aléatoire à deviner.
     */
    private int number;
    /**
     * Le nombre d'essais.
     */
    private int nTries;

    /**
     * Le constructeur pour objets de la classe Jeu initialise
     * le nombre aléatoire secret.
     * @param infLim limite inférieure
     * @param supLim limite supérieure
     */
    public SecretNumber(int infLim, int supLim)
    {
        nTries = 0;
        number = (int)(Math.random()*(supLim-infLim+1)) + infLim;
    }

    /**
     * Cette méthode compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param guess  estimation
     * @return     vrai si number est inférieur à l'estimation.
     */
    public boolean isSmaller(int guess)
    {
        nTries = nTries +1;
        return number < guess;
    }

    /**
     * La méthode isBigger() compare y avec le nombre secret
     * et renvoie vrai ou faux.
     *
     * @param  guess   estimation
     * @return     vrai si number > guess
     */
    public boolean isBigger(int guess)
    {
        nTries = nTries +1;
        return number > guess;
    }

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

```
63          */
64         public boolean equals(int guess)
65         {
66             nTries = nTries +1;
67             return guess == number;
68         }
69 
70          /**
71          * La méthode getNTries() renvoie le nombre de questions
72          * posées.
73          *
74          * @return     nombre de questions posées
75          */
76         public int getNTries()
77         {
78             return nTries;
79         }
80 
81     }
82 
```

A.4 Tag Game

MainFrame.java

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5
6   /*
7    * MainFrame.java
8    *
9    * Created on 7 sept. 2011, 14:47:42
10   */
11
12  /**
13   *
14   * @author getreu
15   */
16  public class MainFrame extends javax.swing.JFrame {
17      private RunningTrack rt = null;
18
19      /** Creates new form MainFrame */
20      public MainFrame() {
21          initComponents();
22
23      }
24
30      //... folded code ...
126
127     private void startButtonActionPerformed(java.awt.event.ActionEvent evt) {
128         int d = Integer.valueOf( headStartTextField.getText() );
129
130         rt = new RunningTrack(d);
131         runningTrackLabel.setText( rt.draw() );
132         messageLabel.setText("Please enter step, then run!");
133
134     }
135
136     private void runButtonActionPerformed(java.awt.event.ActionEvent evt) {
137         if (rt== null)
138         {
139             return;
140         }
141
142         int s = Integer.valueOf( stepTextField.getText() );
143         boolean continueGame = rt.moveXO( s );
144         runningTrackLabel.setText( rt.draw() );
145
146         if ( continueGame )
147         {
148             return;
149         }
150
151         if ( (rt.getPosX() == rt.getPosO())
152              && (rt.getPosO() <= rt.POS_MAX)    )
153         {
154             messageLabel.setText("You catched O!");
155         }
156         else
157         {
158             messageLabel.setText("O escaped.");
159         }
160     }
161
162     /**
```

```
163      * @param args the command line arguments
164      */
165     public static void main(String args[]) {
166         java.awt.EventQueue.invokeLater(new Runnable() {
167             public void run() {
168                 new MainFrame().setVisible(true);
169             }
170         });
171     }
172 
173     // Variables declaration - do not modify//GEN-BEGIN:variables
174     private javax.swing.JTextField headStartTextField;
175     private javax.swing.JLabel jLabel1;
176     private javax.swing.JLabel jLabel2;
177     private javax.swing.JLabel jLabel3;
178     private javax.swing.JLabel messageLabel;
179     private javax.swing.JButton runButton;
180     private javax.swing.JLabel runningTrackLabel;
181     private javax.swing.JButton startButton;
182     private javax.swing.JTextField stepTextField;
183     // End of variables declaration//GEN-END:variables
184 
185 }
186 
```

RunningTrack.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 2.0.0
 */
public class RunningTrack
{
    /**
     * La largeur de la piste (running track).
     */
    public static final int POS_MAX = 20;
    /**
     * Le nombre de pas maximal.
     */
    public static final int STEP_MAX = 5;
    /**
     * Le nombre de pas minimal.
     */
    public static final int STEP_MIN = 1;
    /**
     * La position du pion O (loup).
     */
    private int posO;
    /**
     * La position du pion X (joueur).
     */
    private int posX;

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

    /**
     * Cette méthode avance le pion X
     * de stepX et le pion O aléatoirement
     * @param stepX pas du pion X
     * @return faux si le jeu est terminé
     */
    public boolean moveXO(int stepX)
    {
        if ( stepX >= STEP_MIN && stepX <= STEP_MAX )
        {
            posX = posX + stepX;
        }

        posO = posO + (int)(Math.random()
                      *(STEP_MAX-STEP_MIN+1))+ STEP_MIN;

        return (posX < POS_MAX)
               && (posO < POS_MAX)
```

```
63                  && (pos0 != posX);
64      }
65
66      /**
67       * Cette méthode dessine
68       * la piste avec les deux coureurs 0 et X
69       */
70      public String draw()
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       * Cette méthode 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      * Cette méthode renvoie la
109      * position du coureur 0
110      * @return position de 0
111      */
112     public int getPos0()
113     {
114         return pos0;
115     }
116 }
117
```

Version 2.4.1, Getreu