Java à l'école

Exercices et proposition de solutions

11TG, Première année de programmation

Version 2.4.1, Jens Getreu

Table des matières

1 Introduction

Les exercices de ce recueil sont basés sur un concept de programmation appelé « Architecture Modèle/Vue/Contrôleur»[1], consistant entre autres à séparer l'interface utilisateur d'un programme de sa logique interne. Une interface utilisateur est la partie du programme qui assure la communication avec son utilisateur. Le plus simple à réaliser est l'interface textuelle basée sur les deux instructions Java suivantes:
L'instruction `System.out.println(s1)` affiche le texte de la variable `s1` sur l'écran, tandis que l'instruction `String s2 = sc.nextLine()` permet d'assigner un texte saisi par l'utilisateur sur clavier à la variable `s2`. La dernière instruction existe en 3 variantes selon le type de donnée saisie:

- `String s2 = sc.nextLine()` pour les chaînes de caractères,
- `int n1 = sc.nextInt()` pour les nombres entiers et
- `double n2 = sc.nextDouble()` pour les nombre réels.

Toute communication avec l'utilisateur est programmée dans la classe `Main`. Ainsi, les deux instructions ci-dessous ne peuvent apparaître que dans cette classe. Voici le modèle de la classe applicable pour tous les exercices :

```
import java.util.*;
public class Main
{
   public static void main(String[] args)
   {
      Scanner sc = new Scanner(System.in);
        <insérez votre code source ici>

   }
}
```

Bien que la communication avec l'utilisateur soit assurée par la classe `Main`, la logique du programme est réalisée principalement dans une classe à part.

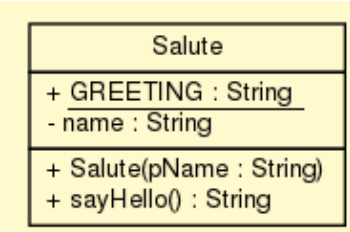
Exercice d'introduction

Nous réalisons une variante du célèbre programme *hello world* : Le programme demande d'abord à l'utilisateur de saisir son nom (par exemple « *Lina* ») et affiche ensuite une salutation personnalisée: « *Hello Lina. How are you doing?* », comme le montre la capture d'écran ci-contre.

```
GREETING
Please type jour name : Lina
Hello Lina. How are you doing?
```

Solution

La logique du programme est codée dans la classe `Salute`.



Comme le montre le diagramme de classe ci-contre, tous les objets de la classe `Salute` possèdent une constante `GREETING` du type String (chaîne de caractères) , un attribut `name` du type String, un constructeur `Salute()` avec un paramètre `pName` du type String et une méthode `sayHello()` renvoyant également un String.

1 Voir : http://fr.wikipedia.org/wiki/Modèle-Vue-Contrôleur

La figure suivante montre le code source de la classe donnant des informations plus détaillées sur son fonctionnement :

```
001   public class Salute
002   {
003       public static final String GREETING="Hello ";
004
005       private String name;
006
007       public Salute(String pName)
008       {
009          name = pName;
010       }
011
012       public String sayHello()
013       {
014           return GREETING + name + ". How are you doing?";
015       }
016   }
```

Linge	Remarque
`001-016`	Définition de la classe `Salute`.
`003`	Déclaration de la constante `GREETING` initialisée à *"Hello "*. `GREETING` est un attribut de la classe `Salute`.
`005`	Déclaration d'une variable propre à l'objet `name`. Une telle variable est appelée « *attribut* »[2] . Il peut être lu ou écrit par l'identificateur `name` ou `this.name` plus tard.
`007-010`	Définition du constructeur de la classe `Salute`: Lors de la création[3] d'un objet[4] de la classe `Salute` , nous devons passer un *paramètre* `pName` au constructeur contenant une chaîne de caractères. Un constructeur, ici `Salute()`, est une méthode spéciale exécutée au moment de la création d'un objet. Un constructeur n'a jamais de valeur de retour.
`009`	La chaîne de caractères dans le paramètre `pName` est ensuite copiée dans l'attribut: `name` (déclaré en ligne `005`).
`012-15`	Définition de la méthode `sayHello()`. Elle utilise l'attribut `name` pour composer un message personnalisé en concaténant le mot "*Hello*", la chaîne de caractères référencée par l'attribut `name` et le texte *"How are you doing?*" Le résultat de la concaténation est renvoyé comme valeur de retour du type `String`.

2 Synonymes : attribut = variable d'instance
3 Synonymes : création = naissance = instanciation
4 Synonymes : objet = instance

Voici le code source de la classe `Main` assurant la communication avec l'utilisateur:

```
001   import java.util.*;
002   public class Main
003   {
004      public static void main(String[] args)
005      {
006           Scanner sc = new Scanner(System.in);
007
008           System.out.println("GREETING");
009           System.out.print("Please type your name : ");
010           String n = sc.nextLine();
011
012           Salute s = new Salute(n);
013           System.out.println( s.sayHello() );
014      }
015   }
```

Linge	Remarque
001–006	Instructions standard et identiques au modèle présenté ci-dessus. [5]
008–009	Les messages « *GREETING* » et « Please type your name: » sont affichés à l'écran.
010	L'instruction `String n = sc.nextLine();` affecte[6] la chaîne de caractères saisie au clavier à la variable `n`.
012	L'instruction `Salute s = new Salute(n);` crée un objet de la classe `Salute` appelée `s` par la suite. L'objet est créé en passant la variable `n` au constructeur.
013	La variable `s` sert à référencer l'objet créé et permet l'accès à sa méthode par l'expression: `s.sayHello()` . Le résultat de la méthode `sayHello()`, c'est-à-dire *"Hello Lina. How are you doing?"*, est affiché par l'instruction `System.out.println(...)`.

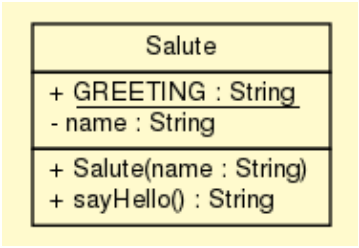
5 De plus amples informations en classe de 12ème.
6 Synonymes dans ce contexte : affecter = copier

Les attributs cachés

Considérons la variante à la classe `Salute`. ci-dessous :

```
1    public class Salute
2    {
3        public static final String GREETING="Hello ";
4
5        private String name;  //accès par this.name
6
7        public Salute(String name)
8        {
9           this.name = name;
10       }
11
12       public String sayHello()
13       {
14           return GREETING + this.name + ". How are you doing?";
15       }
16   }
```

Nous constatons que l'attribut défini dans la ligne `005` et le paramètre défini dans la ligne `007` portent le même nom `name` bien qu'il s'agisse de deux variables différentes ! Comment peut-on accéder à l'une ou l'autre ? S'il y a confusion possible, le nom tout court - ici `name` - désigne le paramètre de la ligne `007`. Le préfixe `this` permet ainsi d'accéder à l'attribut de la ligne `005` en écrivant `this.name`.



Notons que cet ambiguïté n'existe pas dans la ligne `014`, étant donné que le paramètre `name` n'est visible que dans les lignes `008-010`. Nous aurions donc pu écrire également :

```
14            return GREETING + name + ". How are you
doing?";
```

Quel est l'avantage de définir deux variables différentes portant le même nom ? Il arrive – voir ligne 009 - qu'un paramètre serve uniquement de mémoire intermédiaire et que sa valeur soit transmise directement à l'attribut correspondant. Vu que les deux variables ont le même contenu il serait inutile de retenir deux noms de variables différents. Définir deux noms pour la même chose rendrait la lecture du programme plus difficile.

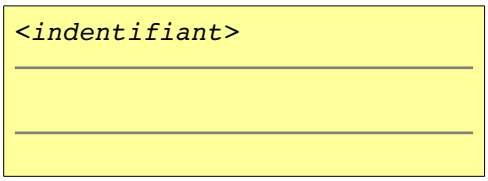
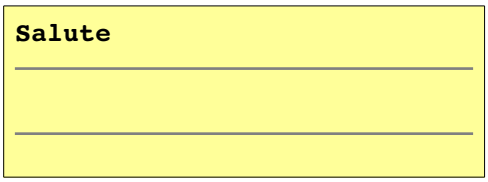
Une solution alternative proposée aux débutants en programmation consiste à éviter deux variables de même nom en faisant précéder tous les noms de paramètres par le lettre `p`. Les lignes 007-010 peuvent s'écrire ainsi :

```
7        public Salute(String pName)
8        {
9           name = pName;
10       }
```

Dans ce recueil d'exercices, nous privilégierons la solution avec le préfixe `this` , car celle-ci permet à l'étudiant de se familiariser dès le départ à la problématique des noms cachés, une source d'erreurs très difficiles à dépister. Cette solution est également conforme aux standards internationaux (voir `http://geosoft.no/development/javastyle.html`). Notons qu'un attribut peut être caché non seulement par un paramètre mais aussi par une variable locale. La notation `pName` ne résout donc pas la problématique des attributs cachées.

Éléments du langage Java

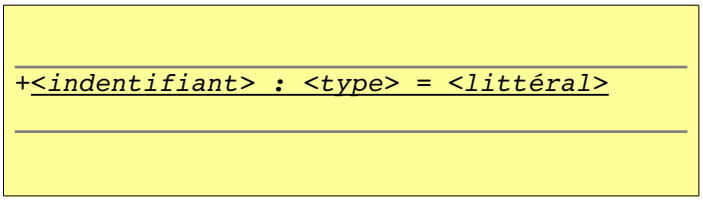
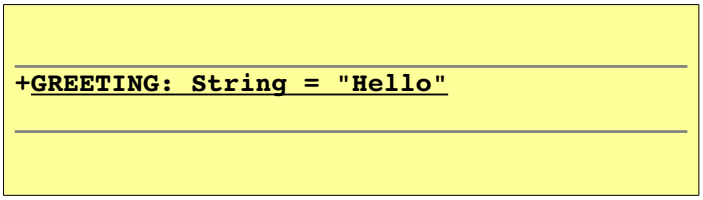
Les tableaux récapitulatifs ci-dessous montrent les conventions utilisées dans ce recueil d'exercices en se référant à l'exercice d'introduction.

Classes	
Définition du terme	Une classe est une description abstraite des données et du comportement d'objets, ces objets étant similaires. Les représentants de la classe sont appelés des instances ou simplement objets.
Contexte	Définition à l'intérieur d'un fichier `.java` portant le même nom que la classe.
Nommage	• 1ère lettre en majuscule • Mélange de minuscules, de majuscules avec la première lettre de chaque mot en majuscule • Donner des noms simples et descriptifs
Exemples de noms (identifiants)	`Main, Salute, NombrePositif, RunningTrack, PrimeNumber`
Syntaxe diagramme de classe UML	
Exemple de diagramme de classe	

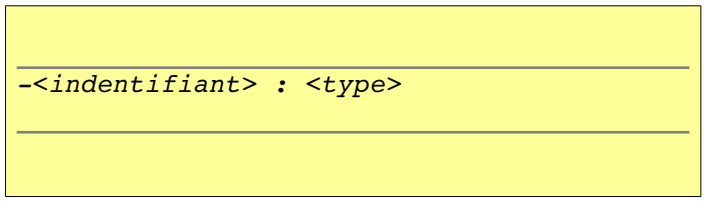
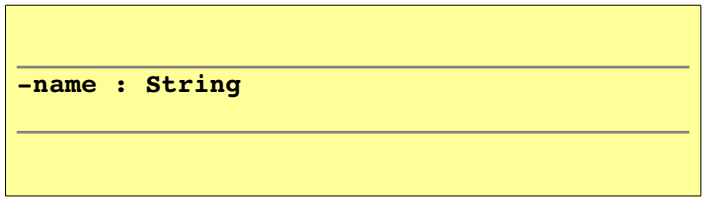
Syntaxe de définition	`public class <indentifiant>`
	`{ ...`
	`}`

Exemples de définition	`public class Salute`
	`{ ...`
	`}`
	
	`public class Main`
	`{ ...`
	`}`

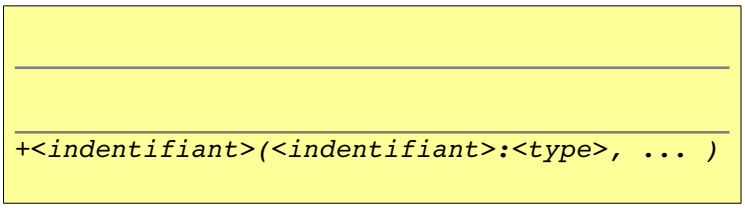
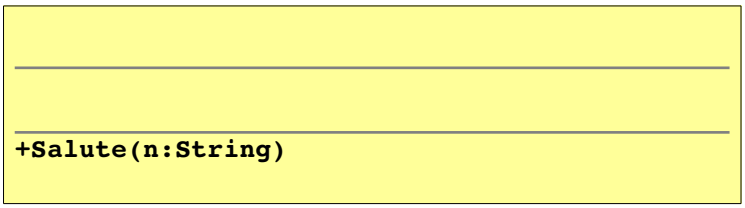
Exemples d'utilisation	`Salute s = new Salute(n);` `System.out.println(s.sayHello() );`

Constantes	
Définition du terme	Un attribut spécial dont la valeur reste invariable après son initialisation.
Contexte	Définition à l'intérieur d'une classe.
Nommage	• Tout en majuscules • Séparer les mots par un souligné (underscore) : _ • Donner des noms simples et descriptifs
Exemples de noms (identifiants)	`GREETING, POS_MAX, POS_MIN, INVALID`
Syntaxe diagramme de classe UML[7]	
Exemple de diagramme de classe	
Syntaxe de déclaration avec initialisation	`public static final <type> <identifiant> = <littéral>;`
Exemples de déclaration avec initialisation	`public static final String GREETING = "Hello";` `public static final int POS_MAX = 20;`
Exemples d'utilisation	`return GREETING + this.name + ". How are you doing?";` `if (i < POS_MAX) ...`

7 La norme UML prévoit que les variables `static` comme les constantes doivent être soulignées dans le diagramme de classe.

Attributs (synonyme : variables d'instances)	
Définition du terme	Une variable d'instance précise l'état d'un objet auquel elle se réfère. Deux objets différents, même appartenant à la même classe, peuvent avoir des valeurs différentes dans leurs variables d'instance respectives. « Ce qu'un objet d'une classe a ou caractérise».
Contexte	Définition à l'intérieur d'une classe.
Nommage	• 1ère lettre en minuscule • Mélange de minuscules, de majuscules avec la première lettre de chaque mot en majuscule • Donner des noms simples et descriptifs • Ne pas faire commencer les noms par '$' ou '_', bien que ce soit possible (toléré). • « Nombre de » se traduit par le préfixe `n`. Par exemple : nombre de zéros devient `nZeros`.
Exemples de noms (identifiants)	`name, coordX, coordY, nZeros, fillLevel`
Syntaxe diagramme de classe UML	
Exemple de diagramme de classe	
Syntaxe de déclaration	`private <type> <identifiant>;`
Exemple de déclaration	`private String name;`
Exemples d'utilisation	`Name = n;` `return GREETING + this.name + ". Comment allez-vous ?";`

Attention ! Si une variable locale ou un paramètre porte le même nom que l'attribut, celui-ci n'est plus accessible directement par son nom ! Pour y remédier, il faut ajouter le mot clef `this.` devant l'attribut. Exemple : soit un attribut et une variable locale avec le nom `length`. L'expression `this.length` désigne l'attribut tandis que `length` désigne la variable locale.

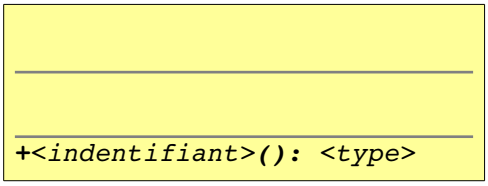
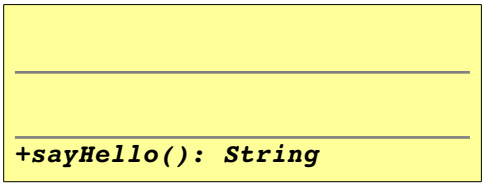
Constructeurs	
Définition du terme	Le constructeur est une méthode spéciale appelée lors de la création de l'objet.
Contexte	Définition à l'intérieur d'une classe.
Nommage	• Les noms de constructeurs doivent être identiques au nom de la classe.
Exemples de noms (identifiants)	`Salute(), PositiveNumber(), RunningTrack(), PrimeNumber()`
Syntaxe diagramme de classe UML	
Exemple de diagramme de classe	

Syntaxe de définition[8]	`public <identifiant>(<identifiant>:<type>, ...)`
	`{`
	`  ...`
	`}`

Exemple de définition	`public Salute(String n)`
	`{`
	`  this.name = name;`
	`}`

Exemples d'utilisation	`Salute s = new Salute(n);`

8 Notons que les constructeurs n'ont jamais de valeur de retour. Ne jamais indiquer cette absence par le mot clé `void` !

Méthodes	
Définition	Descriptif et implémentation des fonctionnalités d'une classe. « Ce qu'un objet d'une classe peut faire ».
Contexte	Définition à l'intérieur d'une classe.
Nommage	• Les noms de méthodes (fonctions) doivent exprimer une action. • Choisir de préférence des verbes. • 1ère lettre en minuscule • Mélange de minuscules, de majuscules avec la première lettre de chaque mot en majuscule • Les méthodes dites *acesseurs* (angl. getter, setter) servant à renseigner ou attribuer une valeur à un attribut portent le nom de l'attribut précédé par le mot clef `get` ou `set`. Exemple : attribut `coordX` correspond aux méthodes `getCoordX()`, `setCoordX()`.
Exemples de noms (identifiants)	`sayHello(); display(); draw(); getCoordX(); setCoordY(); isPrime(); isValid()`
Syntaxe diagramme de classe UML	
Exemple de diagramme de classe	

Syntaxe de définition[9]	`public <type> <identifiant>()`
	`{`
	`  ...`
	`}`

Exemple de définition[10]	`public String sayHello()`
	`{`
	`  return GREETING + this.name + ". Comment allez-vous ?";`
	`}`

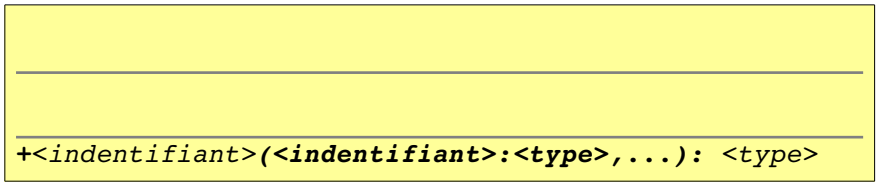
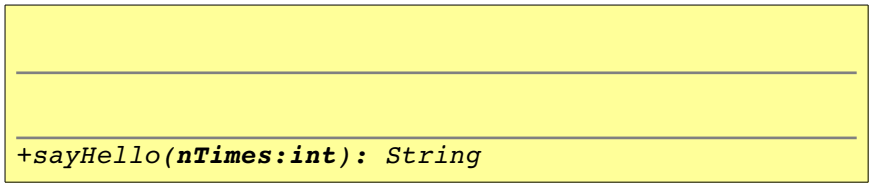
Exemple d'utilisation (appel de méthode)	`System.out.println( s.sayHello() );`

9 Pour réduire la complexité de ce chapitre, les méthodes exposées ici n'ont pas de paramètres. Pour plus d'informations sur le passage des paramètres, voir les pages « Constructeur » et « Variables locales ». Pour définir une méthode avec un paramètre, il suffit d'introduire dans les parenthèses vides la déclaration d'une variable locale, par exemple :
`public String sayHello(int n){...}` et donc `System.out.println(s.sayHello(3))`

10 Dans cet exemple, la méthode retourne une chaîne de caractères composée de trois chaînes. La valeur de retour est du type `String`.

Paramètres	
Définition	Un paramètre est une variable locale d'une méthode servant à passer des valeurs de l'extérieur vers l'intérieur de la méthode lors de son démarrage.
Contexte	Définition à l'intérieur des parenthèses `()` de la définition d'une méthode. Un paramètre ne peut être utilisé que dans la méthode où il est défini.
Nommage	• 1ère lettre en minuscule • Mélange de minuscules, de majuscules avec la première lettre de chaque mot en majuscule • Donner des noms simples et descriptifs • Ne pas faire commencer les noms par '`$`' ou '`_`', bien que ce soit possible. (toléré) • L'expression « Nombre de » se traduit par le préfixe `n`. Par exemple : le nombre de zéros devient `nZeros`.

Exemples de noms (identifiants)	`name, coordX, coordY, nZeros, fillLevel`
	Nommage alternatif voir chapitre Les attributs cachés, page 5 :
	`pName, pCoordX, pCoordY, pNZeros, pFillLevel`

Syntaxe diagramme de classe UML	
Exemple de diagramme de classe	

Syntaxe de définition	`public <type> <identifiant>(<type> <identifiant>, ...)`
	`{`
	`  ...`
	`}`

Exemple de définition	`public String sayHello(int nTimes)`
	`{`
	`  ...`
	`}`

Exemple d'utilisation (appel de méthode)	`int n = 3;` `System.out.println( s.sayHello(n) );`

Les paramètres de type `int`, `double`, `boolean` et `String` se comportent comme s'ils étaient **passés par valeur.** Cela signifie que chaque appel de méthode dispose de sa propre version de paramètres et que l'appel de méthode se fait **par copie** des valeurs passées en argument. Ainsi, à l'intérieur de la méthode, le programmeur manipule seulement la copie des variables venant de l'extérieur. Par conséquent, une modification d'un tel paramètre à l'intérieur d'une méthode ne se répercute jamais à l'extérieur. [11]

11 Pour de plus amples informations voir chapitre "Les mécanismes de passage de paramètres", page 15.

Variables locales	
Définition	Une variable locale est une variable qui ne peut être utilisée que dans la méthode ou le bloc où elle est définie.[12]
Contexte	Définition à l'intérieur d'une méthode.
Nommage	• Voir « Attributs » • Variable courte d'une seule lettre minuscule de préférence.
Exemples de noms (identifiants)	`i, n, x, y, s, sc,` `nombre, fillLevel`
Syntaxe de déclaration	`<type> <identifiant>`
Syntaxe de déclaration avec initialisation	`<type> <identifiant> = <expression>;`
Exemple de déclaration avec initialisation	`String n = sc.nextLine();`
Exemple d'utilisation	`Salute s = new Salute(n)`
Exemple de déclaration	`int i;`
Exemple de déclaration avec initialisation	`int i = 0;`
Exemple d'utilisation	`i = i+1;`

Attention ! Si une variable locale ou un paramètre porte le même nom que l'attribut, celui-ci n'est plus accessible directement par son nom ! Pour y remédier, il faut ajouter le mot clef `this.` devant l'attribut. Exemple : soit un attribut et une variable locale avec le nom `length`. L'expression `this.length` désigne l'attribut tandis que `length` désigne la variable locale.

12 Il n'existe pas de variables globales en Java.

Les expressions

Exercice 1 : La priorité des opérateurs

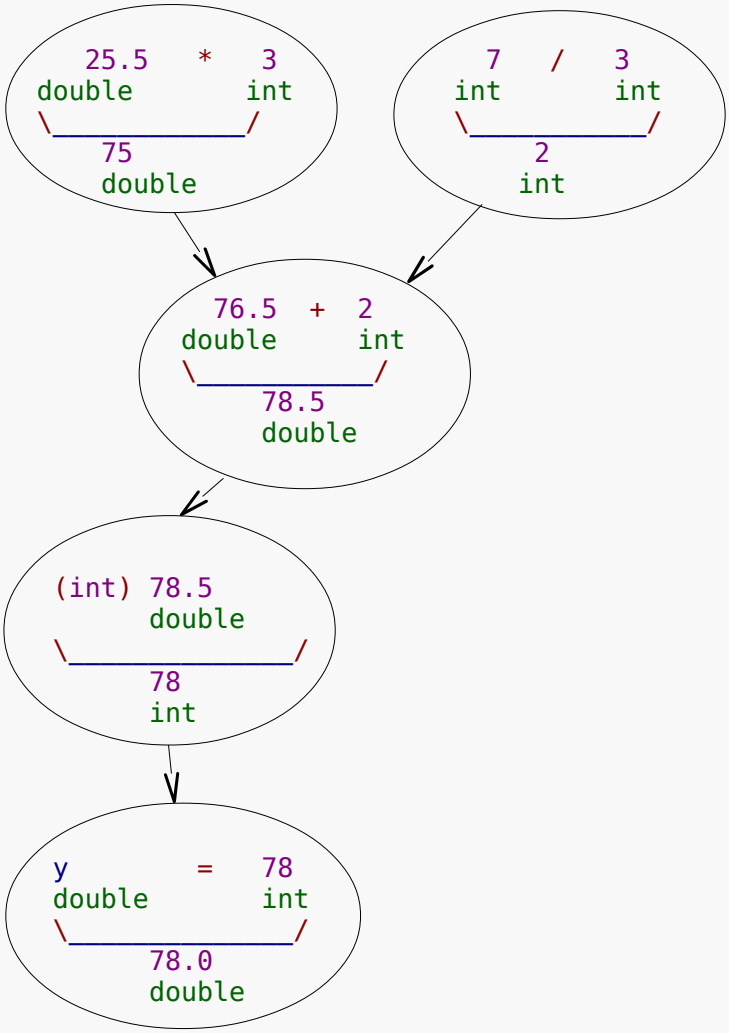
Évaluer successivement l'expression suivante en utilisant la méthode dite "bubbles".[13]

Code source

```
00017:          double x = (int)(25.5 * 3+7 / 3) ;
```

Solution

```
double x =    (int) ( 25.5  *   3     +       7   /   3) ;
```




```
Solution : x est égale 78 et du type double.
```

13 La représentation graphique de la solution proposée aide à bien dissocier les opérations. Je l'ai nommée « bubbles » parce qu'elle fait penser aux bulles dans l'eau.

Exercices 2: Le type boolean

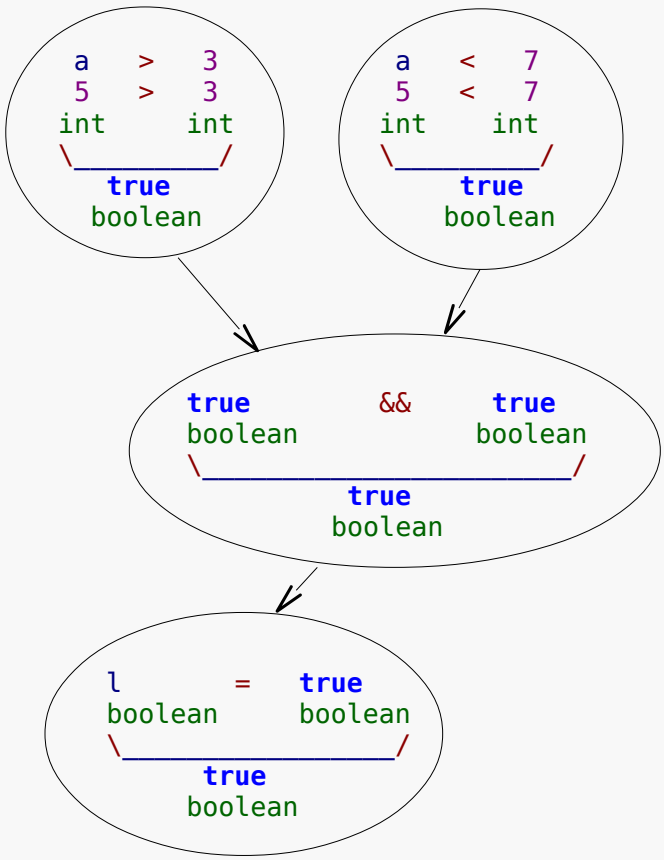
Évaluer successivement l'expression suivante en utilisant la méthode dite "bubbles".

Code source

```
00017:          int a = 5;
00018:          boolean l;
00020:          l = (a > 3) && (a < 7);
```

Solution

```
                a = 5;
                l = (a > 3) && (a < 7);
```




```
Solution : l est égale true; l est du type boolean.
```

Les mécanismes de passage de paramètres

Ce chapitre facultatif est dédié aux lecteurs d'un niveau avancé cherchant à approfondir le sujet.

Main.java

```
1   /**
2    * Classe illustrant le passage de paramètre en Java.
3    *
4    * @author Jens Getreu
5    * @version 1.0.0
6    */
7   public class Main
8   {
9       public static void main(String[] args)
10      {
11          int x = 5;                      // variable primitive
12          int y = 6;                      // variable primitive
13          String  s1 = "a";               // référence sur un objet immutable
14          String  s2 = "A";               // référence sur un autre objet immutable
15          Counter c1 = new Counter(5);    // référence sur un objet mutable
16          Counter c2 = new Counter(55);   // référence sur un autre objet mutable
17
18          System.out.println("Outside change() before modification: \tx="+x
19                             +"\ty="+y+"\ts1="+s1+"\ts2="+s2
20                             +"\tCounter1="+c1.getN()+"\tCounter2="+c2.getN());
21
22          change(x,y,s1,s2,c1,c2);
23
24          System.out.println("Outside change() after modification: \tx="+x
25                             +"\ty="+y+"\ts1="+s1+"\ts2="+s2
26                             +"\tCounter1="+c1.getN()+"\tCounter2="+c2.getN());
27      }
28
29      /**
30       * Cette méthode permet d'illustrer le passage de paramètres en Java.
31       *
32       * @param  xx   une variable primitive
33       * @param  y    une variable primitive
34       * @param  ss1  référence sur un objet immutable
35       * @param  ss2  référence sur un autre objet immutable
36       * @param  cc1  référence sur un objet mutable
37       * @param  cc2  référence sur un objet mutable
38       */
39      public static void change(int xx, int y, String ss1, String ss2,
40                                Counter cc1, Counter cc2)
41      {
42          System.out.println("Inside change() before modification:  \txx="+xx
43                             +"\ty="+y+"\tss1="+ss1+"\tss2="+ss2
44                             +"\tCounter1="+cc1.getN()+"\tCounter2="+cc2.getN());
45
46          // modifier xx
47          xx = 7;                     // xx est une copie de x (passage par valeur).
48                                      // Le changement ne se répercute pas en dehors
49                                      // de change() car x n'est pas modifié.
50          // modifier y
51          y = 8;                      // y (intérieur) est une copie de y (extérieur)
52                                      // (passage par valeur). Bien que les deux
53                                      // identificateurs portent le même noms il s'agit
54                                      // de deux variables différents!
55                                      // (voir: scope). Même comportement que x et xx.
56          // modifier ss1
57          String tmp = "b";           // Les références ss1 et s1 pointent sur ["a"].
58          ss1 = tmp;                  // ss1 pointe sur le nouvel objet ["b"].
59                                      // Le changement de la référence ss1 ne se
60                                      // répercute pas en dehors de change() car la
```

```
                                    // réf. ss1 n'est qu'une copie de la réf. s1
                                    // (passage par référence). La réf s1 - non
                                    // modifiée - pointe toujours sur ["a"].
            // modifier ss2
                                    // ss2 et s2 pointent sur le même objet ["A"].
                                    // La réf ss2 et copie de la réf. s2 (passage
                                    // par référe).
            ss2 = "B";              // Puisque l'objet ["A"] est immutable. Java crée
                                    // un nouvel objet ["B"] en mémoire. ss2 pointe
                                    // sur ["B"], alors que s2 pointe toujours
                                    // sur ["A"]. Le changement de la référence
                                    // ss2 ne se répercute pas en dehors de change().
                                    // car la réf. ss2 était une copie de la réf. s2.
            // modifier cc1
                                    // cc1 et c1 pointent sur le même objet [n=5].
            cc1.setN(7);            // L'objet référencé par c1 [n=5] est modifié,
                                    // car il s'agit du même objet!
                                    // Les références cc1 et c1 n'ont pas changées,
                                    // contrairement à l'objet même.
                                    // LE CHANGEMENT SE RÉPERCUTE EN DEHORS DE
                                    // change() !!!
            // modifier cc2
                                    // La réf. cc2 et la réf c2 pointent sur le même
                                    // objet [n=55]. cc2 et c2 peuvent être différent
                                    // étant donné que cc2 est une copie de c2.
            cc2 = new Counter(77);
                                    // cc2 pointe sur le nouvel objet [n=77] et c2
                                    // sur l'ancien objet [n=55]. cc2 et c2 peuvent
                                    // être différent car cc2 est une copie de c2.
                                    // Le changement ne se répercute pas en dehors
                                    // de change() car la réf c2 pointe toujours sur
                                    // l'objet [n=55].

            System.out.println("Inside change() after modification:   \txx="+xx
                               +"\ty="+y+"\tss1="+ss1+"\tss2="+ss2
                               +"\tCounter1="+cc1.getN()+"\tCounter2="+cc2.getN());
        }
    }
```

Counter.java

```
1
2   /**
3    * Exemple d'une classe définissant des objets mutables.
4    *
5    * @author Jens Getreu
6    * @version 1.0.0
7    */
8   public class Counter
9   {
10      // attribut
11      private int n;
12
13      /**
14       * Constructeur
15       */
16      public Counter(int n)
17      {
18          // initialise l'attribut
19          this.n = n;
20      }
21
22      /**
23       * Acesseur de l'attribut n
24       * @param  n   valeur entier
25       */
26      public void setN(int n)
27
```

```
28      {
29          this.n = n;
30      }
31
32      /**
33       * Acesseur de l'attribut n
35       * @return     valeur de n
36       */
38      public int getN()
39      {
40          return n;
41      }
42  }
```

Sortie

```
Outside change() before modification:   x=5   y=6   s1=a   s2=A   Counter1=5  Counter2=55
Inside change() before modification:    xx=5  y=6   ss1=a  ss2=A  Counter1=5  Counter2=55
Inside change() after modification:     xx=7  y=8   ss1=b  ss2=B  Counter1=7  Counter2=77
Outside change() after modification:    x=5   y=6   s1=a   s2=A   Counter1=7  Counter2=55
```

Conclusion

Question : En java, les paramètres sont-ils passés par référence ou par valeur?

Réponse pratique :

Les *variables primitives* p. ex. `int`, `double`, `boolean` ... et les *objets immutables* p. ex. `String`, `Integer`, `Double`, ... se comportent comme s'ils étaient **passés par valeur**[14], c'est-à-dire. qu'à l'intérieur d'une méthode le programmeur manipule seulement la copie des variables venant de l'extérieur. Ainsi, toute modification d'un tel paramètre à l'intérieur d'une méthode ne se répercute jamais à l'extérieur !

Tous les paramètres de type *objet mutable* (`StringBuffer`, `Salute`, `Counter` etc.[15]) sont passés **par référence,** c'est-à-dire que la modification d'un tel objet à l'intérieur d'une méthode se répercute à l'extérieur !

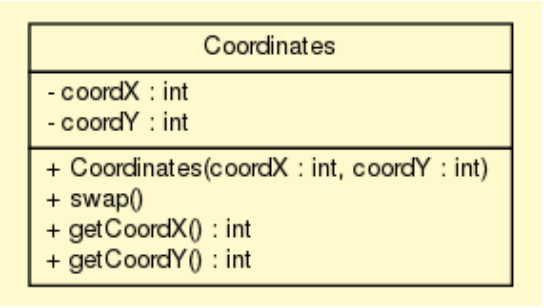
14 En réalité, tous les objets sont passés par référence sachant que Java implémente la notion d'immutabilité, qui signifie qu'un objet ne peut-être modifié après sa création. Contrairement au C++, Java passe une copie de la référence sur l'objet comme paramètre.

15 En principe, toutes les classes écrites dans le contexte scolaire définissent des objets mutables.

2 Permutation

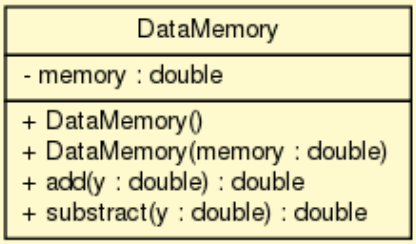
Écrire un programme permettant de lire deux valeurs pour les variables x et y. Permuter ensuite les contenus des deux variables. Afficher les contenus des deux variables avant et après l'opération de permutation.

```
PERMUTATION

Please enter integer x: 3
Please enter integer y: 7

The variables before permutation: x=3 y=7
The variables after permutation:  x=7 y=3
```



3 Mini-calculatrice

L'utilisateur entre deux nombres x et y sur le clavier. Ensuite, un des nombres « 0 » pour l'opération « addition » ou « 1 » pour l'opération «soustraction » est saisi. L'utilisateur est guidé lors de la saisie par des messages adéquats. Le résultat de l'opération s'affiche à l'écran.



Variante : Comparaison de nombres

```
MINI-CALCULATOR
Please enter real number x: 12
Please enter real number y: 7
Please choose operator '0' for 'plus' or '1' for 'minus': 0
x+y = 19.0

MINI-CALCULATOR
Please enter real number x: 12
Please enter real number y: 7
Please choose operator '0' for 'plus' or '1' for 'minus': 1
x-y = 5.0

MINI-CALCULATOR
Please enter real number x: 12
Please enter real number y: 7
Please choose operator '0' for 'plus' or '1' for 'minus': 5
Operator '5' is not defined.
```

Variante : Comparaison de chaînes de caractères

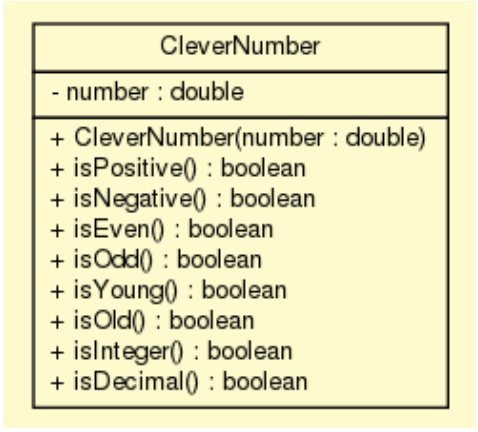
Utilisateur saisie le caractère «+ » pour l'addition ou « - » pour la soustraction. Selon le caractère entré, les nombres x et y sont additionnés ou soustraits

```
MINI-CALCULATOR
Please enter real number x: 3
Please enter real number y: 12
Please choose operator '+' or '-': +
x+y = 15.0

MINI-CALCULATOR
Please enter real number x: 7
Please enter real number y: 2
Please choose operator '+' or '-': -
x-y = 5.0

MINI-CALCULATOR
Please enter real number x: 1
Please enter real number y: 2
Please choose operator '+' ou '-': ?
Operator '?' is not defined.
```

4 Nombre malin

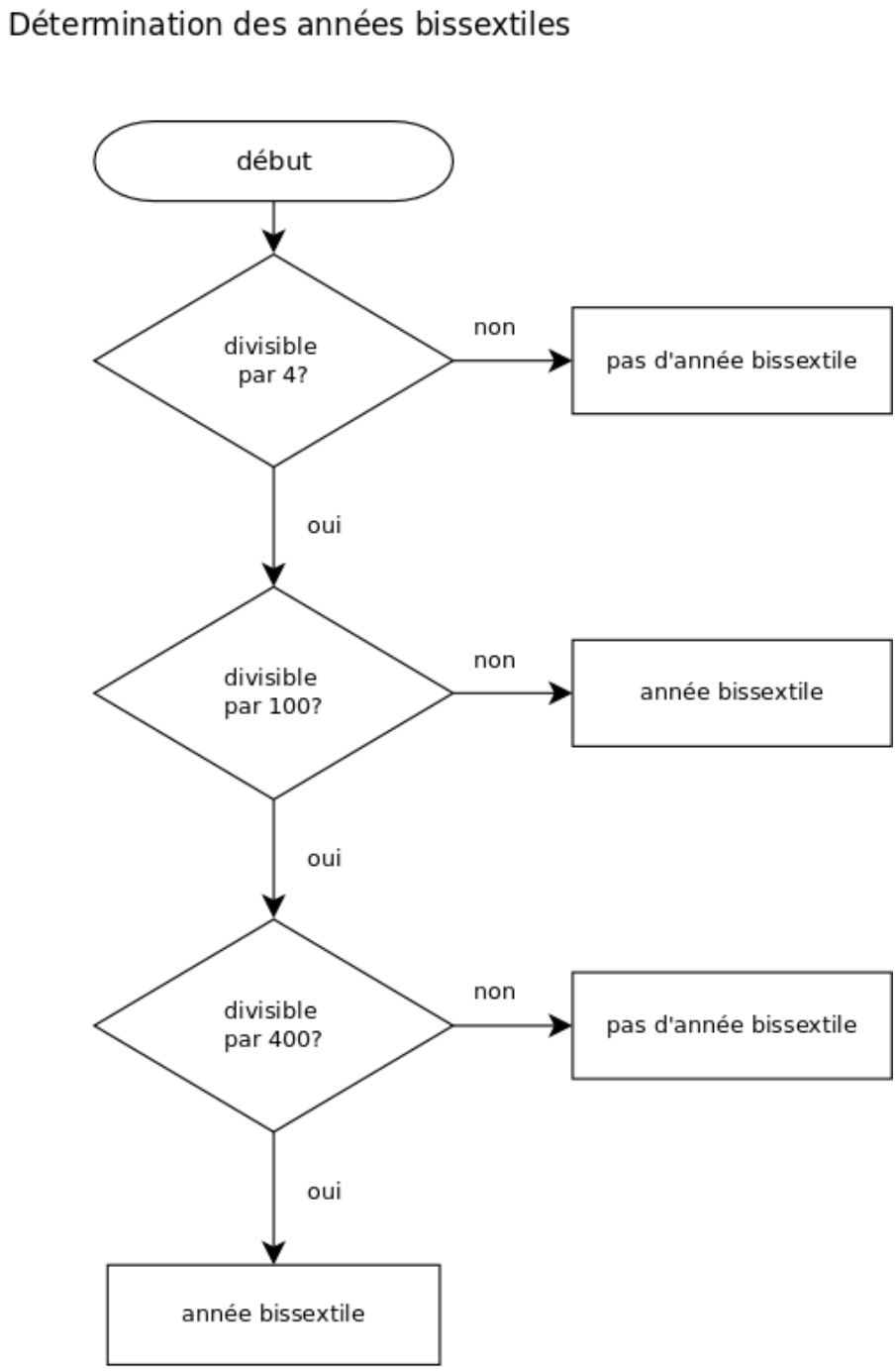
Écrire une classe `NombreMalin` représentant un nombre réel. Développer des méthodes qui renseignent sur ce nombre, par exemple s'il est pair ou impair, positif ou négatif, inférieur ou supérieur à 30, ou s'il est entier ou avec une partie fractionnaire. Le programme à réaliser permet à l'utilisateur de saisir un nombre réel, à partir duquel une instance de la classe est créée et un sous-ensemble des messages suivants est affiché:
Je suis négatif, je suis pair , je suis jeune (<30), je suis un nombre entier, je suis positif , je suis impair, je suis vieux (>=30), j'ai une partie fractionnaire.
Attention: Seuls les nombres entiers sont pairs ou impairs!

```
CLEVER NUMBER
Please enter a rational number: 33.4
Hello. I am a clever number.
I know myself well:
I am positive.
I am old (>=30).
I am decimal (def: with fractional
part).

CLEVER NUMBER
Please enter a rational number: 7
Hello. I am a clever number.
I know myself well:
I am positive.
I am odd.
I am young (<30).
I am integer (def: without fractional
part).
```



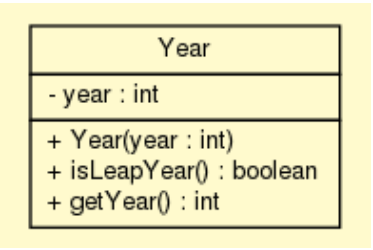
5 Année bissextile

Réaliser un programme permettant à l'utilisateur d'entrer une année sous forme de nombre entier (par exemple 2005). Le programme affiche ensuite sous forme de message si l'année est une année bissextile ou non.



Organigramme de programmation (Programmablaufplan)

```
LEAP YEAR
Please enter a year : 2000
2000 is a leap year.

LEAP YEAR
Please enter a year : 2001
2001 is not a leap year.
```



Variante : structure alternative imbriquée

Réaliser la méthode `isLeapYear()` avec une structure alternative imbriquée.

Variante : opérateurs logiques

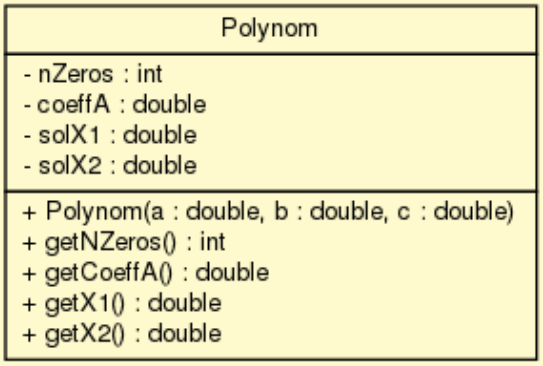
Réaliser la méthode `isLeapYear()` avec des opérateurs logiques par exemple : `&& || !`

6 Équation du second degré

Le programme à réaliser doit résoudre une équation du second degré de la forme: $ax^2+bx+c=0$. Les coefficients a, b et c sont entrés par l'utilisateur. La solution de l'équation s'affiche à l'écran. Rechercher une représentation interne du polynôme initial ax^2+bx+c sous sa forme factorisée $a(x-x1)(x-x2)$. Vérifiez votre programme avec les valeurs ci-dessous.

a	b	c	a	x1	x2	Nombre de racines (=nombre de zéros)
2	4	-30	2	3	-5	2
2	8	8	2	-2	-2	1
2	1	1	2	-	-	0

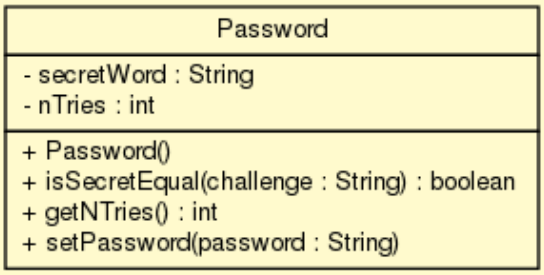
```
FACTOR QUADRATIC EUQUATION (ÉQUATION DE SECOND DEGRÉ)
 a*X^2 + b*X + c = 0

Please enter coefficient a: 1
Please enter coefficient b: 3
Please enter coefficient c: -4
Two solutions : 1.0 and -4.0

FACTOR QUADRATIC EUQUATION (ÉQUATION DE SECOND DEGRÉ)
 a*X^2 + b*X + c = 0

Please enter coefficient a: 1
Please enter coefficient b: -6
Please enter coefficient c: 9
One solution : 3.0
```



7 Vérifier un mot de passe

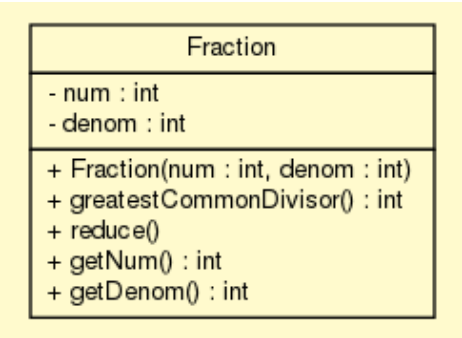
Réaliser un programme permettant de vérifier un mot de passe secret (« hommik »). L'utilisateur a 3 essais pour entrer le mot de passe correct. Il est guidé lors de la saisie par des messages adéquats. Extension possible: L'utilisateur doit d'abord changer son mot de passe.

```
CHECK PASSWORD
Please enter your password: linn
Error. 2 tries left.
Please enter your password: hapu
Error. 1 tries left.
Please enter your password: koor
Error. 0 tries left.
Password is wrong.

CHECK PASSWORD
Please enter your password: karp
Error. 2 tries left.
Please enter your password: hommik
Password is right.
```



8 Simplifier une fraction

Réaliser un programme permettant de saisir le numérateur et le dénominateur d'une fraction. L'utilisateur est guidé lors de la saisie par des messages adéquats. Le programme affiche la fraction simplifiée. Utiliser l'algorithme d'Euclide.



Exemple 1: *pgcd(24,9)*	Exemple 2: *pgcd(12,7)*
24 modulo 9 = 6 9 modulo 6 = 3 6 modulo 3 = 0 *résultat = 3*	12 modulo 7 = 5 7 modulo 5 = 2 5 modulo 2 = 1 2 modulo 1 = 0 *résultat = 1*

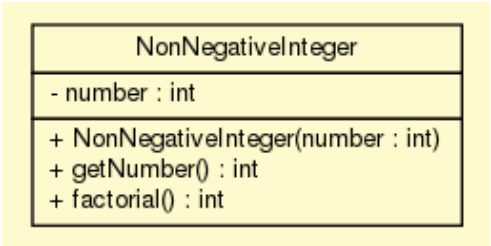
Remarque : réaliser et tester d'abord la méthode `greatestCommonDivisor()`.
Extension possible: ajouter une méthode permettant de porter la fraction à une plus grande expression.

```
REDUCE FRACTION
Please enter numerator n: 24
Please enter denominator d: 18

Reduced numerator n = 4
Reduced denominator d = 3
```

9 Factorielle

Écrire un programme permettant de calculer la factorielle d'un nombre entier positif.

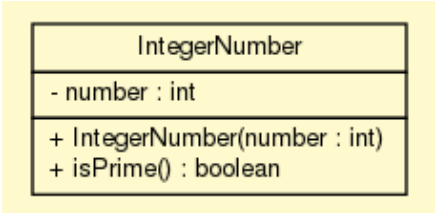
```
FACTORIAL
Please enter a non-negative integer: 5
The factorial of 5 is:
1*2*3*4*5 = 120

FACTORIAL
Please enter a non-negative integer: 7
The factorial of 7 is:
1*2*3*4*5*6*7 = 5040
```



10 Nombre premier

Écrire un programme permettant de tester si un nombre saisi est un nombre premier ou non. Un nombre premier est un nombre entier supérieur à 1 ayant seulement deux diviseurs : 1 et lui-même.
Remarque : cette exercice peut être traité indépendamment ou comme extension de l'exercice « nombre malin ».

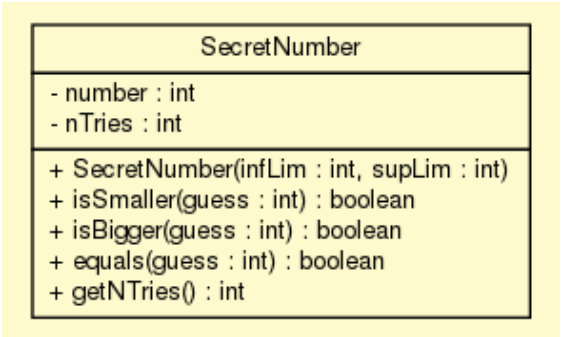
```
PRIME NUMBER
Please enter integer p: 7
p is a prime number.

PRIME NUMBER
Please enter integer p: 24
p is not a prime number.
```



11 Jeu « deviner nombre »

Développer un jeu où l'ordinateur choisit un nombre aléatoire compris entre deux limites entrées par l'utilisateur. L'utilisateur doit deviner ce nombre en trois essais tout en étant guidé lors de la saisie par des messages adéquats. L'ordinateur donne les indications suivantes « trop grand », « trop petit », « vous avez gagné » et « vous avez perdu ».

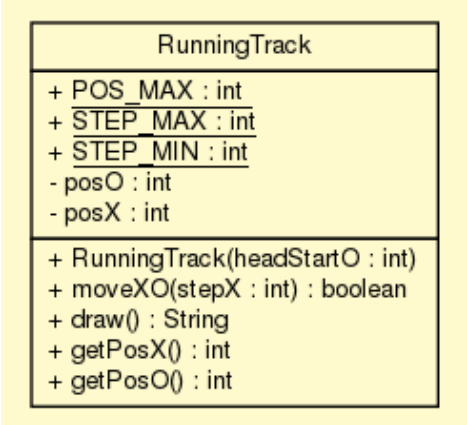
```
GUESS NUMBER
Inferiour limit: 3
Superior limit: 7
*** The game starts, you have 3 tries ***
Your guess: 7
   too big ...
Your guess: 3
   too small ...
Your guess: 5
   too big ...
You lost.

GUESS NUMBER
Inferiour limit: 3
Superior limit: 7
*** The game starts, you have 3 tries ***
Your guess: 3
   too small ...
Your guess: 5
You won!
```



12 Jeu du loup

Le joueur (pion X) essaie d'attraper un loup (pion O). Avant de commencer la course, le joueur donne au loup une certaine avance. A chaque étape, le joueur indique le nombre de pas à faire pour avancer tout en sachant que le loup avance lui-même de 1 à 5 pas aléatoirement. Le jeu se termine avec le message *vous avez rattrapé le loup* ou le message *le loup vous à échappé* si l'un des coureurs à quitté la piste (`RunningTrack`) avant.

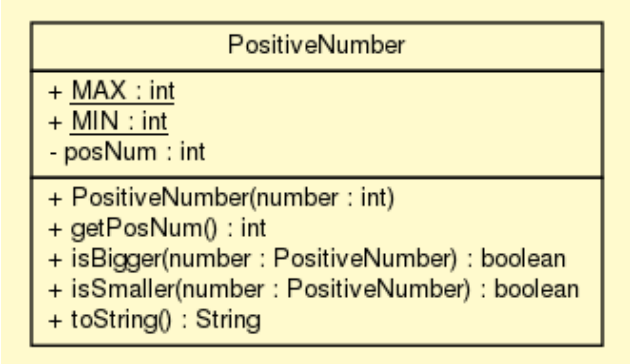
```
TAG GAME
Please give O a head start over X. Enter number: 3
|X__O_________________| Your step: 4
|____X___O____________| Your step: 5
|__________XO_________| Your step: 3
|______________*______|
You catched O!

TAG GAME
Please give O a head start over X. Enter number: 5
|X____O_______________| Your step: 5
|_____X____O__________| Your step: 4
|__________X___O______| Your step: 4
|_______________XO____| Your step: 4
|____________________X_O
O escaped.
```



13 Minimum-Maximum

Écrire un programme affichant le minimum et le maximum d'une série de nombres entiers positifs (comris entre 0 et 1000) que doit saisir l'utilisateur. L'utilisateur indique la fin de la saisie par -1.

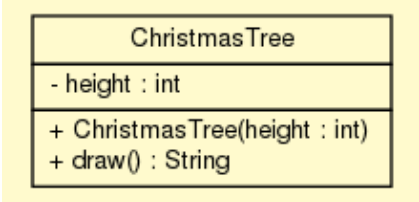
```
MIN-MAX
Please enter an integer from 0 to 1000 (-1 to quit) : 3
Please enter an integer from 0 to 1000 (-1 to quit) : 9
Please enter an integer from 0 to 1000 (-1 to quit) : 2
Please enter an integer from 0 to 1000 (-1 to quit) : -1
The minimum is  2
The maximum is  9
```



14 Sapin de Noël

Écrire un programme affichant la figure suivante à `m` lignes représentant un sapin de Noël. Le paramètre `m` est saisi par l'utilisateur.

```
CHRISTMAS TREE
Please enter height : 5
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

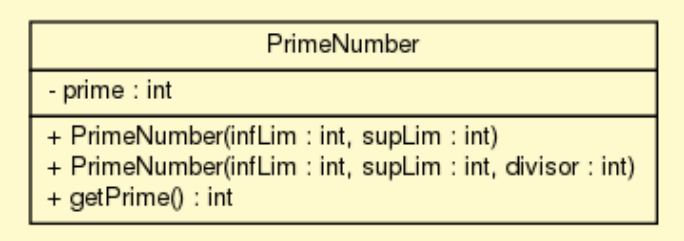


15 Décomposition en produit de facteurs_premiers

L'utilisateur entre un nombre entier au clavier. Le programme affiche sa décomposition en produit de facteurs premiers.

Remarques: Ce programme est l'extension de l'exercice « Nombre Premier ». Réaliser d'abord une classe `PrimeNumber` représentant un nombre premier avec un constructeur `PrimeNumber(int infLim, int supLim)`. Ce constructeur génère le plus petit nombre premier entre `limInf` et `limSup`.Ce constructeur ne sert que comme introduction. Il n'est plus utilisé par la suite. Si aucun nombre premier ne vérifie ces conditions, la valeur par convention est 0.
Un deuxième constructeur `PrimeNumber(int infLim, int supLim, int divisor)`génère le plus petit nombre premier entre `limInf` et `limSup` à condition qu'il soit diviseur de `divisor`. Ce constructeur est appelé par la classe `Main`.

```
INTEGER FACTORIZATION
Please enter integer p : 126
p = 1*2*3*3*7

INTEGER FACTORIZATION
Please enter integer p :
2834688
p = 1*2*2*2*2*2*2*2*2*3*3691
```

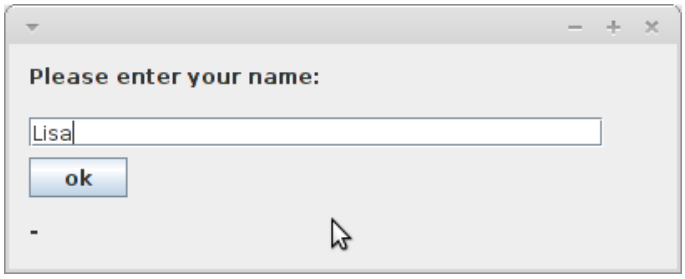


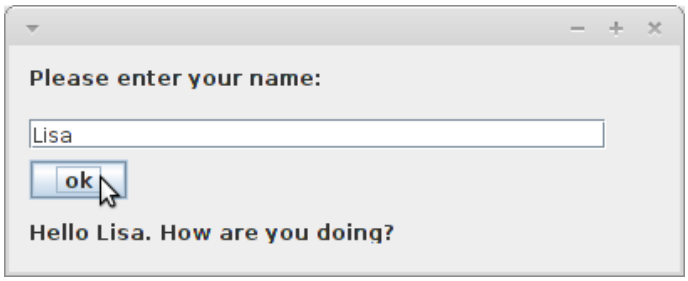
Version 2.4.1, Jens Getreu

Annexe A Netbeans Quickstart

1 Hello World

Programmer une variante de l'application « Hello World » avec l'IDE Netbeans permettant à l'utilisateur de saisir son nom et affichant une salutation personnalisée.





Réaliser l'interface graphique selon les captures d'écran ci-dessus et utiliser les noms d'objets suivants (de la droite vers la gauche et de haut en bas): `inputTextField, okButton, outputLabel.` Suivez les captures d'écran fourni avec ces exercices pour vous familiariser avec Netbeans.

```
public class MainFrame extends javax.swing.JFrame {
...
80   private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
81       // saisie
82       String n = inputTextField.getText();
83       // traîtement
84       n = "Hello " + n + ". How are you doing?";
85       // sortie
86       outputLabel.setText(n);
87   }
}
```

2 Hello World - MCV

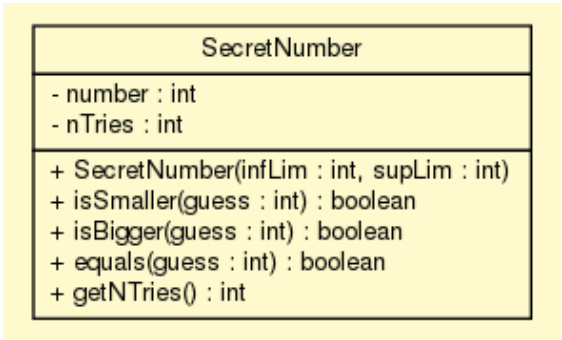
Modifier le programme « Hello World » en ajoutant une deuxième classe. Voici le code de source des deux classes :

```
16 public class MainFrame extends javax.swing.JFrame {
17     // Rendre l'objet utilisable dans d'autres méthodes
18     private Salute s = null;
...
82     private void okButtonActionPerformed(java.awt.event.ActionEvent evt) {
83
84         String n = inputTextField.getText();
85
86         s = new Salute(n);
87
88         outputLabel.setText( s.sayHello() );
89     }
}
```

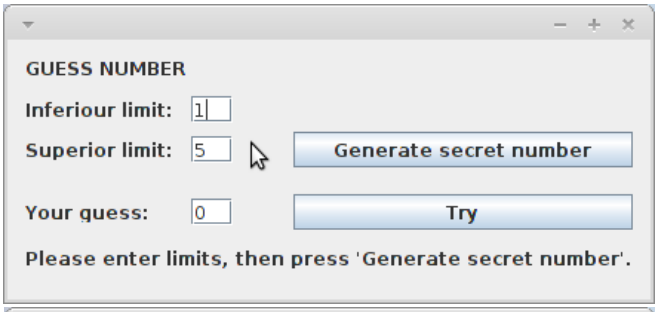
```
10  public class Salute {
11      public static final String GREETING="Hello ";
12      private String name;     // accès par this.name
13
14      public Salute(String name)
15      {
16         this.name = name;
17      }
18
19      public String sayHello()
20      {
21         return GREETING + name +". How are you doing?";
22      }
23  }
```

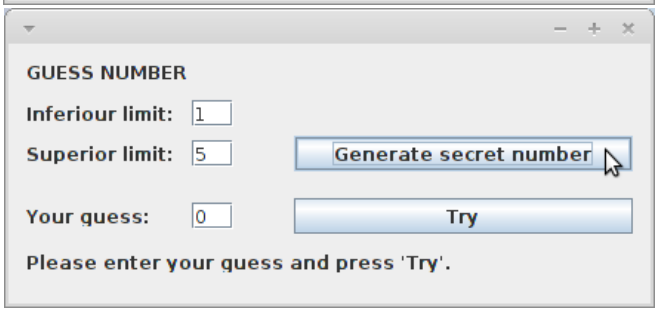
3 Jeu « deviner nombre »

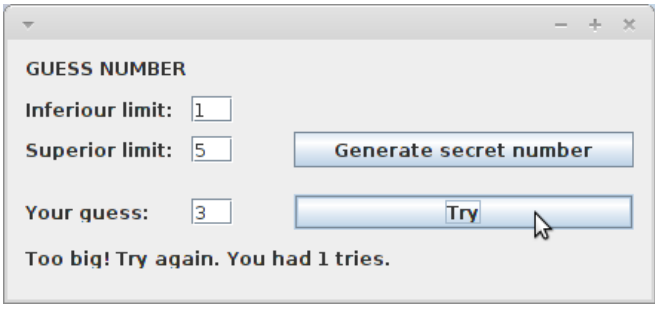
Développer un jeu où l'ordinateur choisit un nombre aléatoire compris entre deux limites entrées par l'utilisateur. L'utilisateur doit deviner ce nombre en trois essais tout en étant guidé lors de la saisie par des messages adéquats. L'ordinateur donne les indications suivantes : « *Too big! Try again. You had X tries.* », « *Too small! ...* » et «*You won! You needed X tries.* ».



Utiliser les noms d'objets suivants (de la droite vers la gauche et de haut en bas) :
`infLimTextField, supLimTextField, genNumButton, guessTextField, tryButton, messageLabel.`

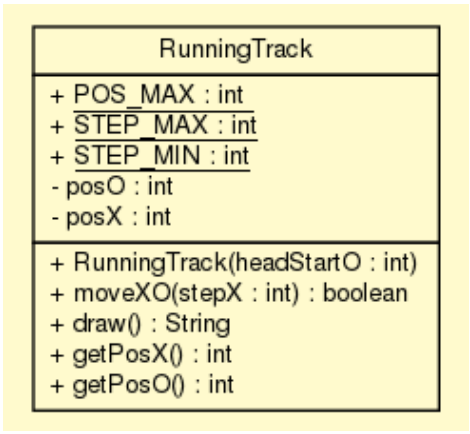




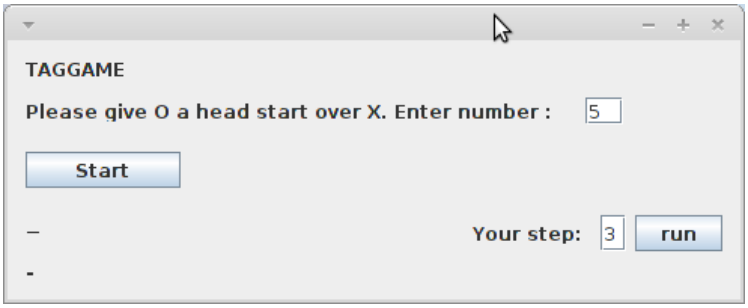


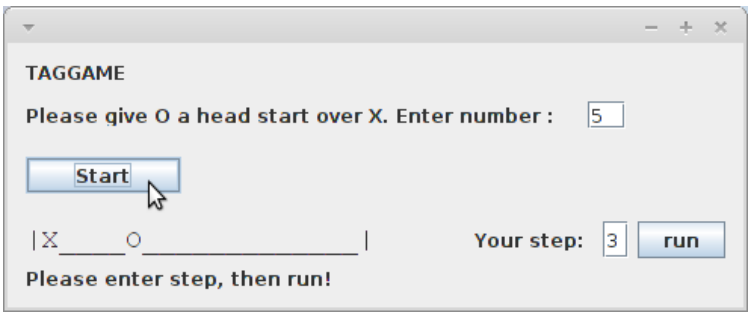
4 Jeu du loup

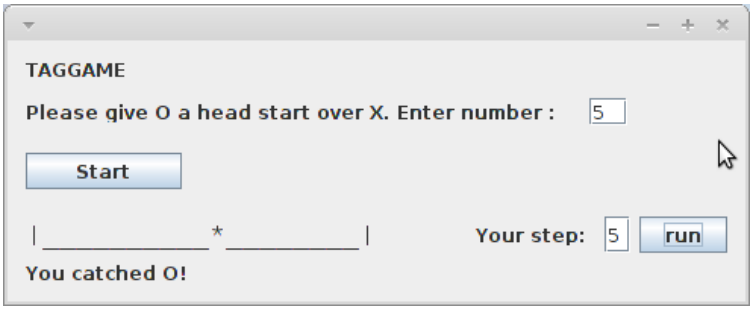
Le joueur (pion X) essaie d'attraper un loup (pion O). Avant de commencer la course, le joueur donne au loup une certaine avance. A chaque étape, le joueur indique le nombre de pas à faire pour avancer tout en sachant que le loup avance lui-même de 1 à 5 pas aléatoirement. Le jeu se termine avec le message *vous avez rattrapé le loup (You catched O)* ou le message *le loup vous à échappé (O escaped)* si l'un des coureurs à quitté la piste (`RunningTrack`) avant.




```
TAG GAME
Please give O a head start over X. Enter number: 3
|X__O_________________| Your step: 4
|____X___O____________| Your step: 5
|__________XO_________| Your step: 3
|____________*________|
You catched O!

TAG GAME
Please give O a head start over X. Enter number: 5
|X____O_______________| Your step: 5
|_____X____O__________| Your step: 4
|__________X___O______| Your step: 4
|________________XO___| Your step: 4
|____________________X_O
O escaped.
```







Utiliser les noms d'objets suivants (de la droite vers la gauche et de haut en bas) : `headStartTextField, startButton, runningTrackLabel, stepTextField, runButton, messageLabel.`