ITI1520 - Lab # 11
Solutionnaire

Adapté de versions antérieures créées par:
Daniel Amyot et Alan Williams

Objectifs

- Classe Ligne
- Examen 2004
  - Questions 1, 2, 6 et 7
  - (les autres à faire par vous-mêmes)

Une classe "Ligne"

- Écrivez une classe **`Ligne`** qui emmagasinera l'information sur une ligne, où une ligne est décrite dans un plan aux coordonnées (*x, y*), et qui fournira des opérations pour manipuler des lignes.
- Chaque objet ligne débute à un point (*xs*, *ys*) et se termine à un point (*xe*, *ye*), où *xs*, *ys*, *xe*, et *ye* sont des nombres réels qui peuvent être positifs ou négatifs.

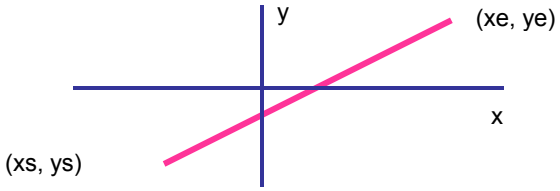


Diagram de classe UML pour `Ligne`

Ligne
- xs : double - ys : double - xe : double - ye : double
+ setPoints (xd: double, yd: double, xa: double, ya: double) : void + longueur () : double + translate (tx: double, ty: double) : void + pente() : double + toString() : String

Description des méthodes

- Fixer les points de départ et d'arrivée de la ligne.
  - Nom de la méthode: **`setPoints(...)`**
  - Paramètres de la méthode: **`xs`**, **`ys`**, **`xe`**, **`ye`**
  - Résultat: (aucun)
  - Modifié: l'objet ligne
- Retourner la longueur de la ligne.
  - La longueur est $\sqrt{(y_e - y_s)^2 + (x_e - x_s)^2}$
  - Nom de la méthode: **`longueur( )`**
  - Paramètres de la méthode: aucun.
  - Résultat: longueur de la ligne (une valeur réelle)

Description des méthodes

- Faire une translation de (*tx*, *ty*), où *tx* et *ty* sont des valeurs réelles positives ou négatives.
  - La *translation* d'une ligne représente le déplacement de la ligne entière dans le plan, comme si on l'avait fait glisser. La valeur *tx* est ajoutée aux coordonnées *x* des points de départ et d'arrivée, et la valeur *ty* est ajoutée aux coordonnées *y* des points de départ et d'arrivée.
    - Nom de la méthode: **`translate(...)`**
    - Paramètres de la méthode: **`tx`**, **`ty`**
    - Résultat: (aucun)
    - Modifié: l'objet ligne

Description des méthodes

- Retourner la pente de la ligne.
  - La pente est définie par (*ye* – *ys*) / (*xe* – *xs*). Attention: il vous faut éviter de faire une division par 0! Pour les lignes verticales (où la pente est indéterminée), retournez `Double.MAX_VALUE` comme résultat.
    - Nom de la méthode: `pente( )`
    - Paramètres de la méthode: aucun.
    - Résultat: la pente de la ligne (une valeur réelle)

Description des méthodes

- Retourner une chaîne `String` contenant l'information sur la ligne.
  - La chaîne pour une ligne qui a (par exemple) comme point de départ (0.0, 1.0) et comme point d'arrivée (3.5, -1.2) devrait être:

    `Ligne de (0.0, 1.0) à (3.5, -1.2)`
  - Le formattage avec un nombre fixe de décimales n'est pas requis.
    - Nom de la méthode: `toString ( )`
    - Paramètres de la méthode: aucun
    - Résultat: une chaîne du format ci-haut

Exercice

- Implémentez la classe **`Ligne`**
- Testez votre classe en utilisant la méthode"main" de **`TestLigne.java`**, disponible sur le site Web du cours.

Examen CSI 1500 de 2004

Attention!

- L'examen de 2004 (le cours a été renommé depuis) présupposait que vous aviez fait les devoirs de 2004.
- L'examen de cette année va supposer que vous avez fait les devoirs de cette année.
- Le nombre de points pour chaque question est indiqué entre crochets []

Question 1A [4]

- Environnement Canada va annoncer une valeur *humidex* dans ses prévisions météo si la température (T) est plus grande ou égale à 30 degrés, si la température est plus grande ou égale à 25 degrés et que le taux d'humidité (H) est plus grand que 35%, ou si la température est plus grande ou égale à 20 degrés et que le taux d'humidité est plus grand ou égal à 65%.
- Écrivez une expression Booléenne qui sera vraie si Environnement Canada va annoncer une valeur *humidex*, et fausse sinon.

Question 1A [4]

- Environnement Canada va annoncer une valeur *humidex* dans ses prévisions météo si la température (T) est plus grande ou égale à 30 degrés, si la température est plus grande ou égale à 25 degrés et que le taux d'humidité (H) est plus grand que 35%, ou si la température est plus grande ou égale à 20 degrés et que le taux d'humidité est plus grand ou égal à 65%.

- Réponse:

  T ≥ 30 OU (T ≥ 25 ET H > 35) OU (T ≥ 20 ET H ≥ 65)

Question 1B [4]

- Observez le programme Java suivant :

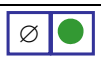
```
MaClasse[] obj;
int index;

obj = new MaClasse [2];
index = 15;
while( index > 2 )
{
   obj[index%2] = new MaClasse ( );
   index = index / 2;
}
// Ligne X
```

- Combien d'instances de **MaClasse** ont été créées pendant l'exécution de ce programme? [2]
- Combien d'instances de **MaClasse** sont encore accessibles à la « Ligne X »? [2]

Question 1B

	obj	index	# objets
Valeurs initiales	?	?	
`obj = new MaClasse[2];`			
`index = 15;`		15	
`while( index > 2 ) : true`			
`obj[index % 2] = new MaClasse( );`			1
`index = index / 2;`		7	
`while( index > 2 ) : true`			

Question 1B

	obj	index	# objets
Page précédente		7	1
`obj[index % 2] = new MaClasse( );`			2
`index = index / 2;`		3	
`while( index > 2 ) : true`			
`obj[index % 2] = new MaClasse( );`			3
`index = index / 2;`		1	
`while( index > 2 ) : false`			

Nombre d'objets créés: 3 Nombre d'objets encore accessibles: 1

Question 1C [4]

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;
  public static int x6()
  {
       ...
  }

  public Foo x7()
  {
       ...
  }
}
```

- Les instructions suivantes (indépendantes les unes des autres) sont utilisées dans la méthode `main()` d'une classe `Test`. Encerclez l'option qui causera une **erreur de compilation**.

```
(a)  Foo[] a = new Foo[5];
     a[4] = new Foo(-1);
(b)  Foo f = Bar.x7();
(c)  Foo.x2 = Bar.x6();
```

```
d)  int k = Foo.x3.x5;

e)  Bar b = new Bar();
    Foo f = b.x7();
```

Question 1C

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;
  public static int x6()
  {
       ...
  }

  public Foo x7()
  {
       ...
  }
}
```

a) OK

`Foo[] a = new Foo[5];` Déclare et crée un tableau de 5 références vers des objets `Foo`. Ces références sont toutes à `null`.

`a[4] = new Foo(-1);` Il y a un constructeur publique `Foo` avec un entier comme paramètre.

Question 1C

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;
  public static int x6()
  {
       ...
  }

  public Foo x7()
  {
       ...
  }
}
```

b) Erreur de compilation

```
Foo f = Bar.x7();
```

La méthode **`x7()`** de la classe **`Bar`** n'est pas **`static`**, alors c'est une méthode d'instance. Les méthodes d'instances ne peuvent pas être invoquées en utilisant le nom de la classe.

Question 1C

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;
  public static int x6()
  {
       ...
  }

  public Foo x7()
  {
       ...
  }
}
```

c) OK

```
Foo.x2 = Bar.x6();
```

La méthode **`x6()`** de la classe **`Bar`** est **`public`**, **`static`**, et retourne une valeur de type **`int`**. Cette méthode peut être invoquée via le nom de la classe.

La valeur **`x2`** de la classe **`Foo`** est **`public`**, **`static`**, et de type **`int`**. Comme la valeur est **`static`**, il s'agit d'une variable de classe et, comme elle est **`public`**, elle peut être accédée de l'extérieur de la classe.

Question 1C

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;
  public static int x6()
  {
       ...
  }

  public Foo x7()
  {
       ...
  }
}
```

d) OK

```
int k = Foo.x3.x5
```

La valeur `x3` de la classe `Foo` est `public`, `static`, et de type `Bar`. Comme la valeur est `public`, `x3` est accessible hors de la classe, et comme elle `static`, elle est accédée via le nom de la classe.

Dans la classe `Bar`, la valeur `x5` est `public` et de type `int`. Ainsi, `x5` est accessible hors de la classe. Elle peut être assignée à une variable de type `int`.

Question 1C

```
class Foo
{
  private int x1;
  public static int x2;
  public static Bar x3;

  public Foo(int x4)
  {
       ...
  }
}
```

```
class Bar
{
  public int x5;
  public static int x6()
  {
       ...
  }

  public Foo x7()
  {
       ...
  }
}
```

e) OK

```
Bar b = new Bar();
```

Déclare et crée un nouvel objet `Bar`. Le constructeur invisible par défaut est utilisé car aucun constructeur explicite n'est défini dans la classe.

```
Foo f = b.x7();
```

La méthode d'instance `x7()` est `public`, donc elle peut être invoquée sur un objet `Bar`. Elle retourne un objet de type `Foo`, qui peut être assigné à `f`.

Question 2 [8]

```
class Football
{
  public static void main(String[ ] args)
  {
    char[] t = {'G', 'e', 'e', '-', 'G', 'e', 'e'};
    Rec(t, t.length - 1);
  }

  public static void Rec(char[] var, int i)
  {
    if (i < 0)
    {
      ; // Ne rien faire
    }
    else
    {
      if ( (var[i] > 'A') && (var[i] < 'Z') )
      {
        System.out.print( (char) (var[i] - 'A' + 'a') );
      }
      else
      {
        if ( (var[i] > 'a') && (var[i] < 'z') )
        {
          System.out.print( (char) (var[i] - 'a' + 'A') );
        }
        else
        {
          ; // Ne rien faire
        }
      }
      Rec(var, i - 1);
    }
  }
}
```

- Voici un programme contenant une méthode récursive
- Que sera-t-il affiché si nous exécutons la méthode **`main`** de cette classe?

Question 2

- Décortiquons ce programme:

```
if ( (var[i] > 'A') && (var[i] < 'Z') )
{
   System.out.print( (char) (var[i] - 'A' + 'a') );
}
```

- Ceci va considérer les lettres majuscules (sauf **`'A'`** et **`'Z'`**) à l'index **`i`** du tableau **`var`**, et afficher leur équivalent en minuscule.

Question 2

```
if ( (var[i] > 'a') && (var[i] < 'z') )
{
   System.out.print( (char) (var[i] - 'a' + 'A') );
}
```

- Ceci va considérer les lettres minuscules (sauf `'a'` et `'z'`) à l'index `i` du tableau `var`, et afficher leur équivalent en majuscule.

Question 2

```
class Football
{
  public static void main(String[ ] args)
  {
    char[] t = {'G', 'e', 'e', '-', 'G', 'e', 'e'};
    Rec(t, t.length - 1);
  }

  public static void Rec(char[] var, int i)
  {
    if (i < 0)
    {
      ; // ne rien faire
    }
    else
    {
      // Convertit majuscules ↔ minuscules sauf pour A et Z
      Rec(var, i - 1);
    }
  }
}
```

Question 2

```
public static void Rec(char[] var, int i)
{
  if (i < 0)          // Cas de base
  {
    ; // ne rien faire
  }
  else
  {
    // Convertit majuscules <-> minuscules sauf pour A et Z

    Rec(var, i - 1); // Appel récursif
  }
}
```

- Traverse le tableau **var** vers l'arrière; l'index **i** va être décrémenté jusqu'à **0**.

Question 2

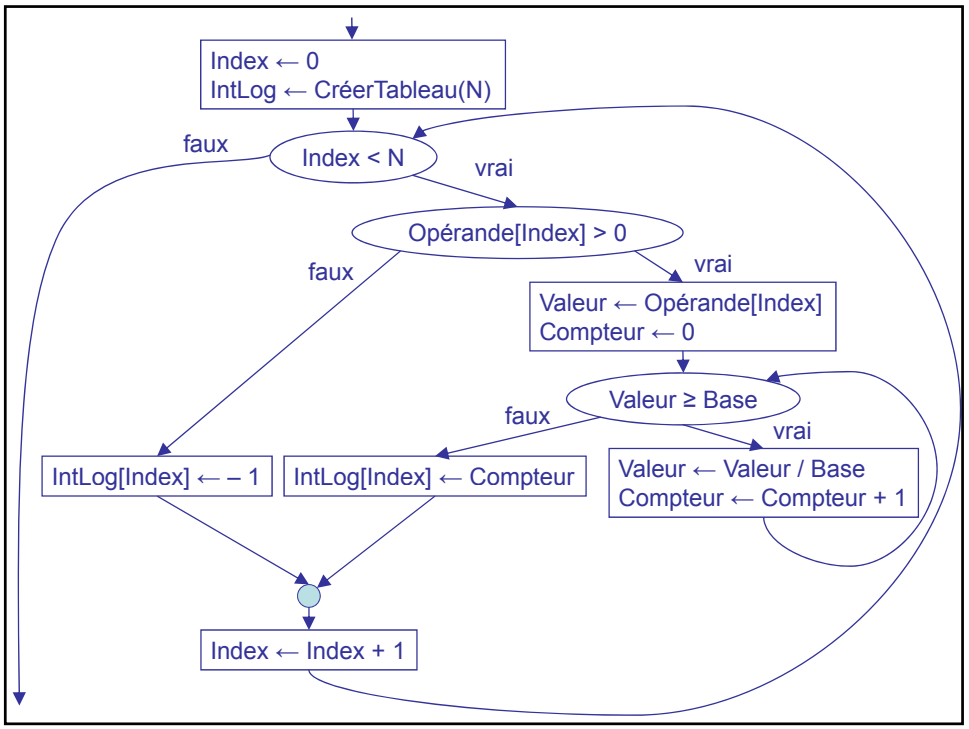
```
public static void main(String[ ] args)
{
  char[] t = {'G', 'e', 'e', '-', 'G', 'e', 'e'};
  Rec(t, t.length - 1);
}
```

- La méthode **Rec** est invoquée avec le tableau **t**. Les caractères qui ne sont pas des lettres sont ignorés.
- Résultat: **EEgEEg**
- Remarque non-pertinente: Pourquoi cette classe était-elle appelée "Football "? ☺

Question 3 [15]

- Traduisez cet algorithme en une méthode Java:

DONNÉES:
Base: *(une base pour logarithme)*
Opérande: *(tableau d'entiers pour lesquels on cherche le log entier)*
N: *(nombre de valeurs dans le tableau Opérande)*
RÉSULTAT:
IntLog: *(tableau de N logarithmes entiers pour les valeurs du tableau Opérande ; la valeur -1 est utilisée si le logarithme n'existe pas)*
INTERMÉDIAIRES:
Index *(index pour tableau)*
Valeur *(utilisée pour divisions successives)*
Compteur *(compte le nombre de fois qu'un opérande peut être divisé par la base)*
EN-TÊTE:
IntLog ← Logarithmes(Base, Opérande, N)



Question 3

```
public static int[] logarithmes(int base, int[] opérande, int n)
 {
   // Déclaration des variables
   int index;    // INTERMÉDIAIRE …
   int valeur;   // INTERMÉDIAIRE …
   int compteur; // INTERMÉDIAIRE …
   int[] intLog; // RÉSULTAT …
   // Module
   index = 0;
   intLog = new int[n];

   while (index < n )
   {
     if ( opérande[index] > 0 )
     {
       valeur = opérande[index];
       compteur = 0;
       while ( valeur >= base )
       {
         valeur = valeur / base;
         compteur = compteur + 1;
       }
       intLog[index] = compteur;
     }
     else
     {
       intLog[index] = -1;
     }
     index = index + 1;
   }
   // Résultat retourné
   return intLog;
 }
```

Commentaires d'usage requis…

Question 4 [15]

- Aux Jeux Olympiques, le pointage pour un plongeon de la plateforme de 10 mètres est calculé de la façon suivante. Chaque plongeon a un « coefficient de difficulté » (CD) basé sur la complexité des éléments techniques inclus (par exemple, un 2 ½ saut périlleux avant en position groupée a un CD de 2.7). Chaque juge observe le plongeon de l'athlète, puis soumet sa note (sur 10 points). La note la plus haute et la note la plus basse sont enlevées, et le pointage pour ce saut est calculé en faisant la somme des notes qui restent et en la multipliant par le coefficient de difficulté.

- Concevez un algorithme qui va calculer le pointage pour le plongeon d'un athlète à partir d'un tableau de notes soumises par N juges, pour un plongeon de coefficient de difficulté CD. Veuillez inclure les commentaires d'usage.

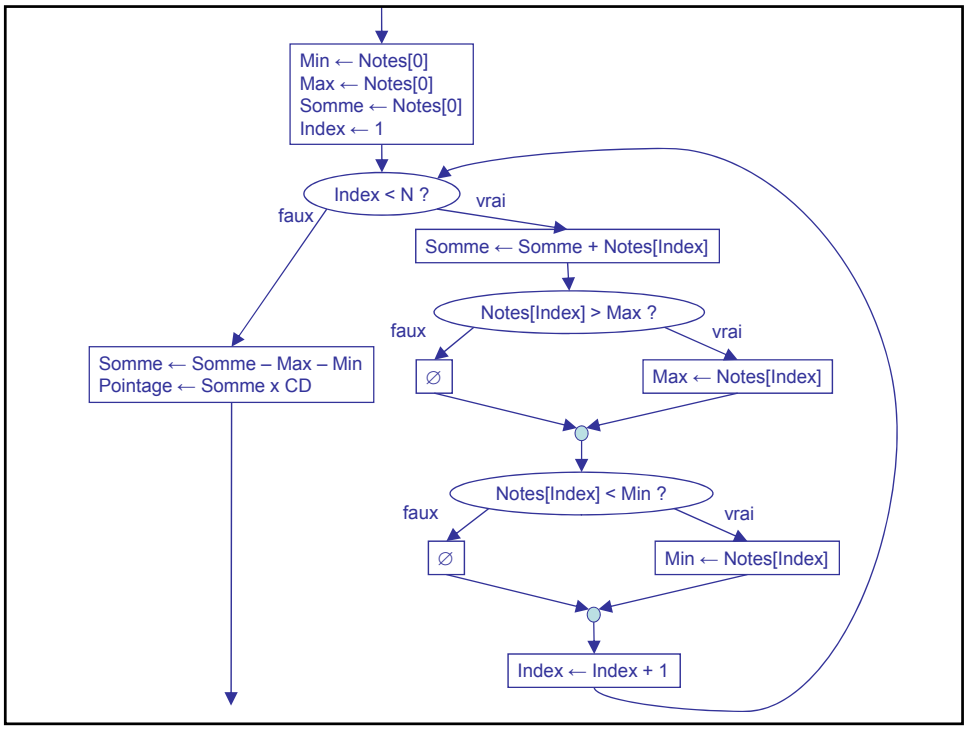
Question 4

DONNÉES:
- CD (*coefficient de difficulté*)
- Notes *(tableau de notes des N juges)*
- N (*nombre de juges*)

INTERMÉDIAIRES:
- Min *(valeur minimale contenue dans Notes)*
- Max *(valeur maximale contenue dans Notes)*
- Index *(index pour parcourir le tableau Notes)*
- Somme *(somme des notes)*

RÉSULTAT: Pointage *(pointage final)*

EN-TÊTE: Pointage ← PointagePlongeon (CD, Notes, N)



Question 5 [15]

- La sous-matrice *bas-droite* d'une matrice est formée en sélectionnant un élément (selon sa ligne et sa colonne) et en excluant tous les éléments qui se trouvent à gauche ou au-dessus de l'élément sélectionné. Par exemple, dans la matrice ***M*** ci-dessous, si nous sélectionnons M_{11} = 5, alors ***S*** est la sous-matrice bas-droite résultante.

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \qquad S = \begin{bmatrix} 5 & 6 \\ 8 & 9 \end{bmatrix}$$

- Écrivez une méthode Java qui prendra en entrée une matrice d'entiers ***M***, de même que des index de ligne et de colonne, et qui retournera une nouvelle matrice qui sera la sous-matrice bas-droite de ***M*** formée à partir de cette position. L'en-tête de cette méthode est la suivante:

```
public static int[][] sousMatrice( int[][] m, int ligne, int col )
```

Question 5

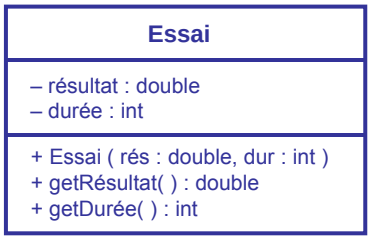
```
public static int[][] sousMatrice( int[][] m, int ligne, int col )
{
   int sLignes; // INTERMÉDIAIRE :  nombre de lignes de s
   int sCols;   // INTERMÉDIAIRE :  nombre de colonnes de s
   int l;       // INTERMÉDIAIRE :  index pour ligne dans s
   int c;       // INTERMÉDIAIRE :  index pour colonne dans s
   int[][] s;   // RÉSULTAT:  la sous-matrice de m voulue

   sLignes = m.length - ligne;
   sCols = m[0].length - col;

   s = new int[sLignes][sCols];
   for ( l = 0; l < sLignes; l = l + 1 )
   {
      for ( c = 0; c < sCols; c = c + 1 )
      {
         s[l][c] = m[l + ligne][c + col] ;
      }
   }
   return s;
}
```

Question 6 [25]

- Pour cette question, vous devez écrire une classe **Expérience** qui représente un enregistrement d'une quelconque expérience scientifique. Afin de s'assurer que les résultats d'une expérience soient répétables, il y a une classe **Essai** qui contient les résultats d'*une* exécution de l'expérience. Une expérience contiendra donc un certain nombre d'objets **Essai**.

Question 6

La classe **Essai** emmagasine un résultat mesuré pendant l'exécution de l'expérience, de même que la durée de cette expérience (en millisecondes). Cette classe a déjà été implémentée; elle contient un constructeur et des accesseurs simples, et son diagramme UML est le suivant :



Question 6

- Dans les pages suivantes, vous devrez développer les méthodes requises pour la classe **Expérience**. Votre classe devra offrir quatre méthodes publiques qui permettront à la méthode **main** de la classe **TestExpérience** d'être exécutée sans problème:

```
class TestExpérience
{
   public static void main (String[] args)
   {
      Expérience uneExpérience;

      uneExpérience = new Expérience( 2 );
      uneExpérience.ajouteEssai( new Essai( 99.1, 10000 ) );
      uneExpérience.ajouteEssai( new Essai( 97.1, 11000 ) );
      uneExpérience.ajouteEssai( new Essai( 94.1, 12000 ) );
      Expérience.setPrédiction( 98.6 );
      uneExpérience.affiche();
   }
}
```

- Résultats affichés :

```
Aucun autre essai ne peut être ajouté à cette expérience.
Essai 0: Résultat 99.1, durée de 10000 (à 0.5 de la prédiction)
Essai 1: Résultat 97.1, durée de 11000 (à 1.5 de la prédiction)
```

Question 6

```
class Expérience
{
 // DÉCLARATIONS DES CHAMPS (4 points)




 // MÉTHODE CONSTRUCTEUR : (5 points)
 // Prend en paramètre un entier représentant le nombre
 // maximum d'essais pour l'expérience.
```

Question 6

```
class Expérience
{
 // DÉCLARATIONS DES CHAMPS (4 points)

  private int nbMax;  // optionnel
  private int nbEssais;
  private Essai[] essais;
  private static double prédiction;

 // MÉTHODE CONSTRUCTEUR : (5 points)
 // Prend en paramètre un entier représentant le nombre
 // maximum d'essais pour l'expérience.

  public Expérience (int max)
  {
    nbMax = max;
    nbEssais = 0;
    essais = new Essai[nbMax];
  }
```

Question 6

```
 // MÉTHODE MODIFICATEUR setPrédiction : (4 points)
 // Paramètres de la méthode: nombre réel représentant le
 // résultat prédit pour l'expérience.
 // Résultat: (aucun)
 // Modifiée: la prédiction pour l'expérience.
```

Question 6

```
// MÉTHODE MODIFICATEUR setPrédiction : (4 points)
// Paramètres de la méthode: nombre réel représentant le
// résultat prédit pour l'expérience.
// Résultat: (aucun)
// Modifiée: la prédiction pour l'expérience.

 public static void setPrédiction (double p)
 {
   prédiction = p;
 }
```

Question 6

```
// MÉTHODE ajouteEssai : (6 points)
// Paramètres de la méthode: un objet Essai à ajouter à cette expérience.
// Résultat: (aucun)
// Modifié:  l'objet Expérience
// Cette méthode doit aussi être robuste: elle affichera un message si
// l'expérience n'a plus de place pour de nouveaux essais (voir tests
// pour le format).
```

Question 6

```
// MÉTHODE ajouteEssai : (6 points)
// Paramètres de la méthode: un objet Essai à ajouter à cette expérience.
// Résultat: (aucun)
// Modifié:  l'objet Expérience
// Cette méthode doit aussi être robuste: elle affichera un message si
// l'expérience n'a plus de place pour de nouveaux essais (voir tests
// pour le format).

 public void ajouteEssai (Essai nouvEssai)
 {
   if (nbEssais < nbMax) // aurait pu être essais.length
   {
     essais[nbEssais] = nouvEssai;
     nbEssais = nbEssais + 1;
   }
   else
   {
     System.out.println ("Aucun autre essai ne peut être ajouté à cette
                          expérience.");
   }
 }
```

Question 6

```
// MÉTHODE affiche : (6 points)
// Paramètres de la méthode: (aucun)
// Résultat: (aucun)
// Cette méthode affiche les résultats et durées de chaque essai, de même
// que la valeur absolue de la différence avec la valeur prédite.
// Voir les résultats de l'affichage pour le format exact
```

```
} // Fin de la classe Expérience
```

Question 6

```
// MÉTHODE affiche : (6 points)
// Paramètres de la méthode: (aucun)
// Résultat: (aucun)
// Cette méthode affiche les résultats et durées de chaque essai, de même
// que la valeur absolue de la différence avec la valeur prédite.
// Voir les résultats de l'affichage pour le format exact

 public void affiche()
 {
   int index;
   double erreur;

   for (index = 0; index < nbEssais; index = index + 1)
   {
     erreur = Math.abs( prédiction - essais[index].getRésultat() );
     System.out.print( "Essai " + index + ": Résultat " +
                       essais[index].getRésultat() );
     System.out.println( ", durée de " + essais[index].getDurée() +
                         " (à " + erreur + " de la prédiction)" );
   }
 }

} // Fin de la classe Expérience
```

Question 7 [10]

- Supposons que nous ayons un tableau d'entiers déjà trié en ordre croissant et qui ne contient pas de valeurs dupliquées. Une *recherche binaire* sur un tel tableau se fait en localisant l'élément au milieu du tableau et en comparant la valeur recherchée avec cet élément. Si nous n'avons pas trouvé la valeur recherchée, alors nous pouvons choisir le sous-intervalle approprié (à gauche ou à droite) pour restreindre la recherche.
- Écrivez une méthode **récursive** qui va faire une recherche binaire de la valeur *trouveMoi* dans un tableau t entre les positions *indexDépart* et *indexFin*. Vous pouvez supposer que ces deux positions sont plus petites que la longueur du tableau. La méthode devra retourner *true* si la valeur se trouve dans le tableau, et *false* sinon.
- L'en-tête de cette méthode est la suivante:

```
public static boolean rechercheBinaire(int[] t, int trouveMoi,
                                       int indexDépart, int indexFin)
```

Question 7

```
/**
 * Méthode récursive pour faire une recherche binaire d'un entier trouveMoi
 * entre les indices gauche et droit d'un tableau d'entiers trié.
 *
 * @param valeurs tableau d'entiers trié.
 * @param trouvemoi valeur à rechercher.
 * @param gauche indice de gauche.
 * @param droite indice de droite.
 * @return Un (<CODE>boolean</CODE>) indiquant si trouveMoi est dans valeurs.
 */
public static boolean rechercheRec( int[] valeurs, int trouveMoi,
                                    int gauche, int droite )
{

 // DÉCLARATION DES VARIABLES / DICTIONNAIRE DE DONNÉES
 int milieu; // INTERMÉDIAIRE: indice du milieu
 boolean trouvé; // RÉSULTAT: indique si on a trouvé trouveMoi

 // MODULE DE L'ALGORITHME
```

Question 7

```
 if ( gauche + 1 >= droite )
 {
   // Cas de base
   trouvé = (     trouveMoi == valeurs[gauche]
              || trouveMoi == valeurs[droite] );
 }
 else
 {
   // On vérifie d'abord si on a trouvé au milieu
   // (permet d'éviter des invocations inutiles et coûteuses)
   milieu = ( gauche + droite ) / 2;
   if ( trouveMoi == valeurs[milieu] )
     { trouvé = true; }
   else
   {
     // Invocation récursive, avec intervalle plus petit
     if ( trouveMoi < valeurs[milieu] )
     {
       trouvé = rechercheRec( valeurs, trouveMoi, gauche, milieu );
     }
     else
     {
       trouvé = rechercheRec( valeurs, trouveMoi, milieu, droite );

     }
   }
 }

 // RÉSULTAT RETOURNÉ
 return trouvé;
}
```