# Interrogation d'informatique
# PCC-ASINSA 2ème année - Janvier 2019

**Durée totale :** *2h*
**Documents autorisés :** *Toutes notes personnelles ou du cours dans une limite de 10 feuilles recto-verso*
***Attention : les téléphones portables sont interdits.***

- Le barème est indicatif et le sujet est sur 14 pages.
- Les exercices sont indépendants.
- Un programme **mal indenté**, **mal commenté** ou avec de **mauvais choix de noms de variables** sera **sanctionné** *(jusqu'à -1 point)*.

**À méditer avant de commencer :**
« Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. »*Martin Golding*

**Barème indicatif :**
- Partie 1 : 3.5 points
- Partie 2 : 2 points
- Partie 3 : 8.5 points
- Partie 4 : 6 points

**Dernières précisions importantes à lire attentivement :**
- les 4 parties sont indépendantes. Vous pouvez les traiter dans l'ordre que vous voulez.
- pour chaque question, vous devrez utiliser (dans la mesure du possible) les méthodes présentées dans les questions précédentes, même si vous ne les avez pas codées ;
- l'efficacité des algorithmes proposés fera l'objet d'une attention particulière ;
- le code java donné dans vos réponses devra être le plus concis possible et donc exploiter le plus possible les structures d'héritage ;
- la propreté de votre copie sera pris en compte dans la notation.

## 1 IHM et écouteurs (3.5 pts)

Soit le code ci dessous :

```
1  import javax.swing.*; import java.awt.*; import java.awt.event.*;
2  public class MonIHM extends JFrame implements ActionListener{
3     private JPanel monPan2;
4     private JButton jb1, jb2, jb3, jb4, jb5;
5     public MonIHM() {
6        new JFrame();
7        setTitle("Ma Fenetre");  setSize(750, 600);
8        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9        // Mon panneau monPan2
10       monPan2 = new JPanel(); monPan2.setLayout(null);
11       monPan2.setBounds(50,40,650,100); monPan2.setBackground(Color.yellow);
12       // Les boutons de monPan2
13       JButton jb1=new JButton("Action1"); jb1.setBounds(25,10,100,80);
14       jb1.addActionListener(this);
15       jb2=new JButton("Action2"); jb2.setBounds(150,10,100,80);
16       jb2.addActionListener(this);
17       jb3=new JButton("Action3"); jb3.setBounds(275,10,100,80);
18       jb3.addActionListener(this);
19       jb4=new JButton("Action4"); jb4.setBounds(400,10,100,80);
20       jb5=new JButton("Action5"); jb5.setBounds(525,10,100,80);
21       jb5.addActionListener(this);
22       monPan2.add(jb1); monPan2.add(jb2); monPan2.add(jb3);
23       monPan2.add(jb4); monPan2.add(jb5);
24       // Mon panneau monPan3
25       JPanel monPan3 = new JPanel();  monPan3.setLayout(null);
26       monPan3.setBounds(0,200,750,400); monPan3.setBackground(Color.blue);
27       JLabel monLab = new JLabel("Coucou ! "); monLab.setBounds(325,300,75,50);
28       monPan3.add(monLab);
29       // Mon panneau monPan1
30       JPanel monPan1 = new JPanel();  monPan1.setLayout(null);
31       monPan1.setBounds(0,0,getWidth(),getHeight());
32       monPan1.setBackground(Color.green);
33       monPan1.add(monPan2);   monPan1.add(monPan3);
34       add(monPan1);
35       setVisible(true);
36    }
37
38    public void actionPerformed(ActionEvent e){
39       monPan2.setBackground(Color.black);
40       if (e.getSource() == jb1) monPan2.setBackground(Color.magenta);
41       if (e.getSource() == jb2) monPan2.setBackground(Color.blue);
42       if (e.getSource() == jb3) monPan2.setBounds(50,300,525,100);
43       if (e.getSource() == jb4) System.out.println("Je suis le bouton jb4");
44    }
45
46    public static void main(String[] Args){
47       new MonIHM();
48    }
49 }
```

**Nom :**
**Prénom :**
**Groupe :**

# A RENDRE AVEC VOTRE COPIE

Répondez aux questions suivantes (une seule bonne réponse par question) :
*Chaque bonne réponse donne 0.5pt et chaque mauvaise réponse donne -0.5pt. Cependant, il n'est pas possible d'avoir de notes négatives sur l'ensemble de cette partie.*

### (Q1.1) Appuyons sur le bouton `jb1`

Si l'utilisateur clique sur le bouton `jb1`, de quelle couleur est le `JPanel monPan2` ?

- ☐ jaune
- ☐ noir
- ☐ magenta
- ☐ noir
- ☐ il y un message d'erreur qui apparaît dans le terminal

- ✓ noir

Le bouton `jb1` a été redéclaré dans le constructeur donc cette variable est différente de l'attribut de la classe. La couleur ne sera donc pas magenta. Par contre il est noir (et non pas jaune) car le programme passe par la première ligne de la méthode `actionPerformed(ActionEvent e)`.

### (Q1.2) Appuyons sur d'autres boutons

Si l'utilisateur clique successivement sur les boutons `jb1`, `jb2` et `jb3`, de quelle couleur est le `JPanel monPan2` ?

- ☐ vert
- ☐ noir
- ☐ magenta
- ☐ noir
- ☐ il y un message d'erreur qui apparaît dans le terminal

- ✓ noir

Le fait de cliquer en dernier sur `jb3` fait repasser le programme dans la méthode `actionPerformed(ActionEvent e)`. La couleur sera donc noire car le programme repasse par la première ligne de cette méthode.

### (Q1.3) Bouton `jb5`

Suite à l'appui sur le bouton `jb5`, se passe t'il quelque chose ?

- ☐ oui
- ☐ non
- ☐ il y un message d'erreur qui apparaît dans le terminal

- ✓ oui

La couleur du `JPanel monPan2` devient noire même si il n'y a aucune action spécifique liée au bouton `jb5`.

### (Q1.4) Bouton `jb4`

Suite à l'appui sur le bouton `jb4`, se passe t'il quelque chose ?

- ☐ rien
- ☐ le message "Je suis le bouton jb4" apparaît dans le terminal.
- ☐ il y un message d'erreur qui apparaît dans le terminal

✓ rien

Il ne se passe rien car l'écouteur n'est pas branché sur `jb4`.

### (Q1.5) Bouton `jb3`

Suite à l'appui sur le bouton `jb3`, que devient `monPan2` ?

- ☐ il ne change pas
- ☐ `monPan2` est déplacé vers le haut et sa largeur est modifiée
- ☐ `monPan2` est déplacé vers le bas et sa hauteur est modifiée
- ☐ `monPan2` est déplacé vers le bas et sa largeur est modifiée
- ☐ `monPan2` est déplacé vers le haut et sa hauteur est modifiée

✓ `monPan2` est déplacé vers le bas et sa largeur est modifiée

### (Q1.6) Constructeur

Que fait la première ligne du constructeur ?

- ☐ elle crée la fenêtre courante
- ☐ elle ne fait rien
- ☐ elle crée une autre fenêtre invisible

✓ elle crée une autre fenêtre invisible

### (Q1.7) Coucou !

Où apparaît le texte "Coucou ! " ?

- ☐ il n'apparaît pas car il dépasse de la fenêtre visible
- ☐ il sera à peu près centré sur la fenêtre
- ☐ il sera au milieu et en bas de la fenêtre
- ☐ il sera au milieu de `monPan3`

✓ il sera au milieu et en bas de la fenêtre

**Nom :**
**Prénom :**
**Groupe :**

# A RENDRE AVEC VOTRE COPIE

## 2 POO et héritage (2 pts)

Répondez aux questions suivantes (une seule bonne réponse par question) :
*Chaque bonne réponse donne 0.5pt et chaque mauvaise réponse donne -0.5pt. Cependant, il n'est pas possible d'avoir de notes négatives sur l'ensemble de cette partie.*

### (Q2.1) Constructeur

Soit un objet de type B descendant d'un type A. Si le constructeur de B s'écrit ainsi :

```
1 public B(int x, int y, char o){
2    super(x,y,3,o);
3 }
```

Quelle est la signature (ou entête) du constructeur de A ?

- ☐ il n'y a pas besoin de constructeur pour A
- ☐ `public A(int x, int y, char o)`
- ☐ `public A(int a, int b, int c, char d)`
- ☐ `public double A(int x, int y, int z, char o)`
- ☐ `public A()`

✓ public A(int a, int b, int c, char d)

### (Q2.2) Utilisation de this

Soit le code suivant :

```
1 public class QCM{
2    private String param1;
3    private int param2;
4
5    public QCM(String param, int param2){
6       this.param1=param;
7       this.param2=param2;
8    }
9 }
```

Dans les 2 lignes du constructeur, où la présence de `this` est nécessaire ?

- ☐ la 1ère ligne
- ☐ la 2ème ligne
- ☐ les 2 lignes
- ☐ sur aucune des 2 lignes

✓ la 2ème ligne

### (Q2.3) Classe Bidule

Soit la classe Bidule défini par le code suivant :

```
1   public class Bidule{
2      private int a;
3      private int b;
4
5      public Bidule(int c, int d){
6         a=c;
7         b=d;
8      }
```

Si les instructions suivantes sont lancées dans le main() de votre programme principal (qui est situé dans un autre fichier .java) :

```
1   Bidule A1;
2   Bidule A2;
3   A1 = new Bidule(1,3);
4   A2 = A1;
5   A2.b=4;
```

Que vaut l'attribut b de A1 ?

- ☐ 4
- ☐ 3
- ☐ 1
- ☐ il y a une erreur à la compilation du programme

✓ il y a une erreur à la compilation du programme
car les attributs sont déclarés en tant que private donc inaccessible en dehors de la classe.

### (Q2.4) Abstrait

Soit une classe A héritant d'une classe abstraite B. Quelle affirmation est vraie ? Veuillez choisir une réponse :

- ☐ La classe A peut être compilée si au moins une méthode abstraite de B est définie dans A
- ☐ La classe A peut être compilée si toutes les méthodes abstraites de B sont définies dans A
- ☐ La classe A ne peut pas être compilée

✓ La classe A peut être compilée si toutes les méthodes abstraites de B sont définies dans A

## 3 Un peu de chimie sous forme de POO (8.5 pts)

En tirant profit de la Programmation Orientée Objet (POO), nous souhaitons modéliser différents concepts issus de la chimie organique : atomes, molécules, matières et mélanges gazeux. La classe `Atome` est définie comme suit :

```
1  public class Atome {
2
3     public String nom;
4     public double masseMolaire;
5
6     public Atome(String anom, double amasse){
7        nom=anom;
8        masseMolaire=amasse;
9     }
10
11    public String toString(){
12       String res = new String();
13       res+=("Atome " +nom+" de masse molaire "+masseMolaire);
14       return res;
15    }
16 }
```

### (Q3.1) Les molécules

La classe `Molecule` est définie selon le diagramme UML ci-dessous (`listeAtomes` est un attribut `public`).

| Molecule |
| --- |
| + Atome[] listeAtomes |
| + Molecule(String [] noms, double [] masses)<br>+ String toString()<br>+ Molecule combiner(Molecule m) |

FIGURE 1 – Diagramme UML de la classe `Molecule`

#### *(Q3.1).1 Écrivez l'implémentation du constructeur*

Ce constructeur prend en paramètres d'entrée les tableaux de noms et de masses molaires des atomes qui composent la molécule.

**1 pt**

```
1     public Molecule(String [] noms, double [] masses){
2        listeAtomes = new Atome[noms.length];
3        for(int i=0; i<listeAtomes.length; i++)
4           listeAtomes[i] = new Atome(noms[i],masses[i]);
5     }
```

*(Q3.1).2 Écrivez l'implémentation de la méthode `toString`*

L'affichage dans le programme principal avec `System.out.println` devra être tel qu'illustré sur la figure 2. Pour rappel, la masse molaire d'une molécule est égale à la somme des masses molaires de ses atomes.

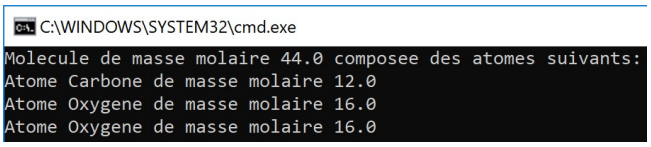


FIGURE 2 – Illustration de l'affichage d'une `Molecule` (CO2).

**2 pt**

```
1   public String toString(){
2      String res = new String();
3      double masse=0;
4      for(int i=0; i<listeAtomes.length; i++)
5         masse += listeAtomes[i].masseMolaire;
6      res+="Molecule de masse molaire "+masse+" composee des atomes suivants:\n";
7      for(int i=0; i<listeAtomes.length; i++)
8         res += listeAtomes[i].toString()+"\n";
9      return res;
10
11  }
```

*(Q3.1).3 Écrivez l'implémentation de la méthode `combiner`*

Cette méthode fabrique une nouvelle molécule qui contient l'union des atomes de deux molécules (la molécule courante et une molécule en paramètre) et renvoie la nouvelle molécule résultante.

**2 pt**

```
1   public Molecule combiner(Molecule m){
2      String []newNoms= new String[m.listeAtomes.length+listeAtomes.length];
3      double []newMasses= new double[m.listeAtomes.length+listeAtomes.length];
4
5      int ind=0;
6      for(int i=0; i<listeAtomes.length; i++){
7         newNoms[ind]=listeAtomes[i].nom;
8         newMasses[ind]=listeAtomes[i].masseMolaire;
9         ind++;
10     }
11     for(int i=0; i<m.listeAtomes.length; i++){
12        newNoms[ind]=m.listeAtomes[i].nom;
13        newMasses[ind]=m.listeAtomes[i].masseMolaire;
14        ind++;
15     }
16
17     return new Molecule(newNoms,newMasses);
18  }
```

### *(Q3.1).4 Ajout des liaisons atomiques*

Les atomes d'une molécule sont reliés entre eux par des *liaisons atomiques*. Chaque liaison atomique relie deux atomes donnés. Proposer une représentation de ces liaisons sous la forme d'un ou plusieurs attribut(s) supplementaire(s) de la classe `Molecule`.
Il vous ait demandé uniquement d'écrire ce/ces attribut(s) supplémentaire(s) et de décrire comment il(s) modélise(nt) les liaisons atomiques. Aucune modification dans le constructeur ou dans les autres méthodes n'est demandé. Dans la suite de l'énoncé nous ne considérerons pas ce/ces attributs supplémentaires.

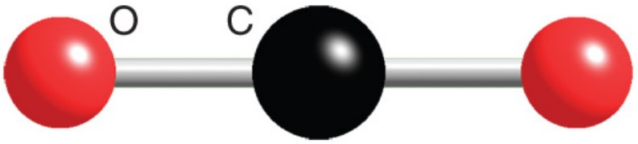


FIGURE 3 – Illustration des liaisons entres atomes pour une `Molecule` (CO2).

**0.5 pt** Plusieurs réponses possibles, par exemples deux tableaux : int [] indicesDepart, int []indicesArrivée ; ou bien un tableau 2D de type matrice d'adjacence.

## (Q3.2) La matière et les mélanges gazeux

La classe `Matiere` définit un composé chimique par un ensemble de molécules et un ensemble de pourcentages correspondant à chaque molécule. Par exemple la matière *Air* est définie comme composée de 80% de diazote (N2) et 20% de dioxygène (O2) (c'est une approximation). La classe `Matiere` est définie selon le diagramme UML ci-dessous. Son implémentation n'est pas demandée. La classe `MelangeGazeux` est une classe fille de la classe `Matiere`, qui définit un

| Matiere |
| --- |
| - Molecule[] listeMolecules<br>- double[] listePourcentages |
| + Matiere(Molecule[] aliste, double[] aPourcentage)<br>+ String toString() |

FIGURE 4 – Diagrammes UML de la classe `Matiere`

composé chimique à l'état gazeux. Elle possède un attribut (public) supplémentaire : la pression (de type `double`).

*(Q3.2).1 Écrivez l'implémentation de la classe `MelangeGazeux` : l'entête de la classe, le/les attribut(s) et le constructeur.*

**1.5 pt**

```
1  public class MelangeGazeux extends Matiere{
2
3    public double pression;
4
5    public MelangeGazeux(Molecule[] aliste, double[] aPourcentage, double
         apression){
6      super(aliste,aPourcentage);
7      pression=apression;
8    }
9
10 }
```

*(Q3.2).2 Est-ce que la classe `Matiere` est abstraite ? justifiez brièvement votre réponse.*

**0.5 pt** Non, car elle ne contient pas de méthode abstraite.

*(Q3.2).3 Ajout de la méthode `pressionsPartielles`*

Cette méthode calcule et retourne la pression partielle de chaque molécule du mélange gazeux. La pression partielle d'une molécule est définie comme le produit de la pression du mélange gazeux et du pourcentage de cette molécule. Donnez l'entête et l'implémentation de cette méthode de la classe `MelangeGazeux`.

**1 pt**

```
1    public double[] pressionsPartielles(){
2      double[] res = new double[listeMolecules.length];
3      for(int i=0; i<listeMolecules.length; i++)
4        res[i]= listePourcentages[i]*pression;
5      return res;
6    }
```

## 4 IHM - Application à la plongée sous-marine (6 pts)

L'objectif est de proposer un programme avec une IHM qui indique si un palier de décompression est nécessaire pendant la remontée d'une plongée, en fonction de la profondeur (en mètre) et de la vitesse de remontée (en mètre/minute). L'IHM est définie comme suit, ce qui donne la fenêtre de la figure 5.

```
1  import javax.swing.*; import java.awt.*;
2  public class OrdiPlongeeIHM extends JFrame{
3      public OrdiPlongeeIHM() {
4              setTitle("Simulateur Plongee");
5              setSize(300, 400);
6              setLocation(200, 100);
7              setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8              JLabel labelData = new JLabel("DonnÃ©es Initiales");
9              labelData.setBounds(50,5,200,30);
10             JTextField textProf = new JTextField("30");
11             textProf.setBounds(30,60,50,40);
12             JTextField textVit = new JTextField("10");
13             textVit.setBounds(30,120,50,40);
14             JLabel labelProf = new JLabel("");
15             labelProf.setBounds(100,65,120,30);
16             labelProf.setText("Profondeur [m]");
17             JLabel labelVit = new JLabel("");
18             labelVit.setBounds(100,125,120,30);
19             labelVit.setText("Vitesse [m/min]");
20             JPanel monPanInit = new JPanel();
21             monPanInit.setLayout(null);
22             monPanInit.setBounds(25,10,250,200);
23             monPanInit.setBackground(Color.yellow);
24             monPanInit.add(labelData);
25             monPanInit.add(textProf);
26             monPanInit.add(labelProf);
27             monPanInit.add(textVit);
28             monPanInit.add(labelVit);
29             JButton calcul = new JButton("Calcul");
30             calcul.setBounds(50,25,100,50);
31             JPanel monPanCalcul = new JPanel();
32             monPanCalcul.setLayout(null);
33             monPanCalcul.setBounds(50,235,200,100);
34             monPanCalcul.setBackground(Color.blue);
35             monPanCalcul.add(calcul);
36             JPanel monPanPpal = new JPanel();
37             monPanPpal.setLayout(null);
38             monPanPpal.setBounds(0,0,getWidth(),getHeight());
39             monPanPpal.setBackground(Color.white);
40             monPanPpal.add(monPanInit);
41             monPanPpal.add(monPanCalcul);
42             add(monPanPpal);
43             setVisible(true);
44     }
45 }
```

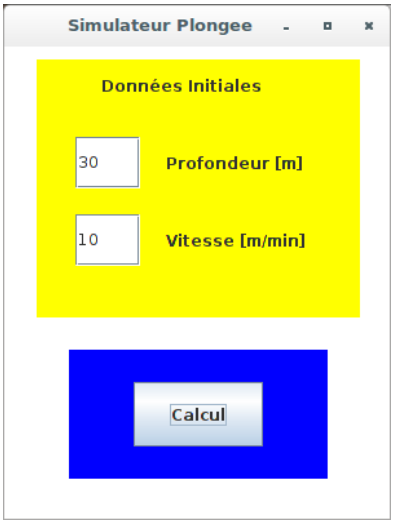


FIGURE 5 – Interface graphique du simulateur de plongée.

Durant son immersion, le plongeur respire un gaz comprimé (en général de l'air). En descendant, l'air se comprime davantage et, pour un volume identique respiré, le taux d'azote absorbé devenant supérieur à celui contenu dans notre organisme, ce gaz se dissout dans le sang par l'intermédiaire des poumons.
À la remontée, la pression diminue et la solubilité de l'azote dans le sang diminue du même fait. Il est alors important que la vitesse de remontée soit suffisamment lente pour permettre à l'azote dans le sang de rester soluble et d'être rejeté par les poumons lors de la respiration. Si la remontée est trop rapide, l'azote dissout dans le sang passera sous forme gazeuse et ne sera pas éliminé par les poumons. Les bulles de gaz chercheront à se déplacer dans le corps provoquant des traumatismes pouvant dans certains cas entraîner la mort.
Il est donc nécessaire d'étudier la quantité de diazote (N2) dissous dans le sang. Cette quantité est appelée tension. Elle est notée $T_{N2}$. La tension du diazote dans le sang n'est pas instantanée. Elle suit la loi de Haldane qui s'énonce de manière simplifiée ainsi :

$$T_{N2}^{n+1} = T_{N2}^{n} + (P_{N2}^{n} - T_{N2}^{n}) * (1 - 0.5^{1/10}) \quad (1)$$

avec :
$T_{N2}^{n}$ la tension courante de N2 $P_{N2}^{n}$ la pression courante du N2 respiré

$T_{N2}^{n+1}$ la tension de N2 après 1 minute

En partant d'une profondeur initiale et en supposant une remontée à une vitesse constante, votre programme (après appui sur le bouton `calcul`) doit déterminer si la tension $T_{N2}^{finale}$, une fois arrivé en surface, est inférieure à un certain seuil (2.38 dans notre cas). Si c'est le cas la remontée peut se faire sans palier, sinon un palier de décompression est nécessaire (c'est à dire que le plongeur devra faire une pause pendant sa remontée). La tension initiale $T_{N2}^{0}$ est égale à la pression partielle initiale $P_{N2}^{0}$ de N2 respiré (qui dépend de la profondeur).
*Pour rappel, la pression du mélange gazeux respiré en surface est de 1 bar et elle augmente linéairement avec la profondeur d'eau. Ainsi à 10 m la pression est à 2 bars, à 20 m elle est à 3 bars et ainsi de suite ...*
Pour simuler cette remontée, il est nécessaire de créer le mélange air contenu dans la bouteille.

Pour cela, en plus du code donné ci-dessus, nous rajoutons ce code dans le constructeur :

```
1       /** Mise en place du mélange gazeux**/
2       // création d'une molecule N2
3       String []nomsN2={new String("Azote"),new String("Azote")};
4       double []massesN2={14.0,14.0};
5       Molecule n2 = new Molecule(nomsN2,massesN2);
6
7       // création d'une molecule O2
8       String []nomsO2={new String("Oxygene"),new String("Oxygene")};
9       double []massesO2={16.0,16.0};
10      Molecule o2 = new Molecule(nomsO2,massesO2);
11
12      // Création du mélange "Air" en surface (1 bar)
13      Molecule []airMol={n2,o2};
14      double []airPourc={0.8,0.2};
15      air=new MelangeGazeux(airMol,airPourc,1);
```

`air` est un `MelangeGazeux` en attribut de notre classe `OrdiPlongeeIHM`.

### (Q4.1) Interaction avec le bouton

Nous souhaitons lancer la simulation après un clic sur le bouton `calcul`. Expliquer en quelques lignes les étapes nécessaires pour permettre une interaction avec le bouton `calcul` ?

**1.5 pt**

- Ajout de la bibliothèque `event`.
- Ajout de `implements ActionListener` dans l'entête de la classe.
- Branchement de l'écouteur au bouton : `bouton.addActionListener(this)`.
- Déclaration du bouton `calcul` en dehors du constructeur en tant qu'attribut de la classe.
- Ajout de la méthode `actionPerformed(ActionEvent e)`.

### (Q4.2) Récupérer les données initiales

Étant donné le code présenté en 4, écrivez le code permettant de récupérer la profondeur et la vitesse (sous forme de `double`). Dans quelle méthode doit se trouver ce code ? Précisez si des modifications sont nécessaires dans le code présenté en 4.
NB : *P*our transformer `maChaine` de type `String` en `double`, il faut utiliser la méthode `Double.parseDouble(maChaine)`.

**1 pt**
Il suffit de rajouter ces lignes dans la méthode `actionPerformed(actionEvent e)` :

```
double profondeur = Double.parseDouble(textProf.getText());
double vitesse = Double.parseDouble(textVit.getText());
```

Il faut aussi penser à déclarer ces 2 `JTextField` en tant qu'attribut de la classe.

### (Q4.3) Simulation d'une remontée

Complétez la méthode précédente avec le code permettant de calculer et d'afficher sur la console, l'évolution de la tension d'azote $T^n_{N2}$ à partir de la profondeur indiquée et jusqu'à l'arrivée à la surface. On partira de la profondeur indiquée par l'utilisateur et de sa vitesse de remontée

supposée constante. Nous précisons qu'aucun `Timer` n'est nécéssaire, une simple boucle suffit, chaque tour de boucle calculant la nouvelle valeur de $T_{N2}^n$ à chaque minute.
Pour vous aider, vous trouverez ci-dessous un exemple du calcul de la pression partielle d'azote pour une pression du mélange gazeux de 4 bars (valeur arbitraire).

```
1         // Exemple du calcul de la pression d'azote à 4 bars (valeur arbitraire).
2         air.pression=4;
3         double pressionN2=air.pressionsPartielles()[0];
```

**2.5 pt**

- Il faut mettre à jour la pression de l'air (et la tension d'azote) en fonction de la profondeur
- Faire la boucle de calcul pour mettre à jour la profondeur, la pression et la tension

```
1  int deltaT=1;
2  air.pression=1+profondeur/10;
3  tension=air.pressionsPartielles()[0];
4    while(profondeur>0){
5      profondeur=profondeur-vitesse*deltaT; // ici deltaT = 1 (pas de calcul de 1min)
6      air.pression=1+profondeur/10;
7      tension=tension+(air.pressionsPartielles()[0]-tension)*(1-Math.pow(0.5,deltaT/10));
8    }
```

### (Q4.4) Détermination de la nécessité d'un palier

Complétez la méthode précédente avec le code nécessaire pour permettre de déterminer si un palier est nécessaire. Pour rappel, le seuil critique de la tension vaut 2.38. Si un palier n'est pas nécessaire, le `JPanel` contenant le bouton `calcul` devient vert et le titre de la fenêtre indique `"GOOOOO !!"`. Dans le cas contraire, le `JPanel` devient rouge et le titre de la fenêtre indique `"NOOOOO !!"`

**1 pt**
Il faut

- comparer la tension à la sortie de boucle à la valeur seuil
- monPanCalcul doit être maintenant être déclaré en tant qu'attribut de la classe
- mettre à jour le setBackground de monPanCalcul
- mettre à jour le titre de la fenêtre avec setTitle

```
1  if (tension<2.38){
2                 monPanCalcul.setBackground(Color.green);
3                 this.setTitle("GOOOOO !!");
4             } else {
5                 monPanCalcul.setBackground(Color.red);
6                 this.setTitle("NOOOOO !!");
7             }
```