Devoir surveillé d'informatique
PCC-ASINSA 2ème année - Janvier 2016

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Durée totale : *3h*
Documents autorisés : *Toutes notes personnelles ou du cours*
Attention : les téléphones portables sont interdits.

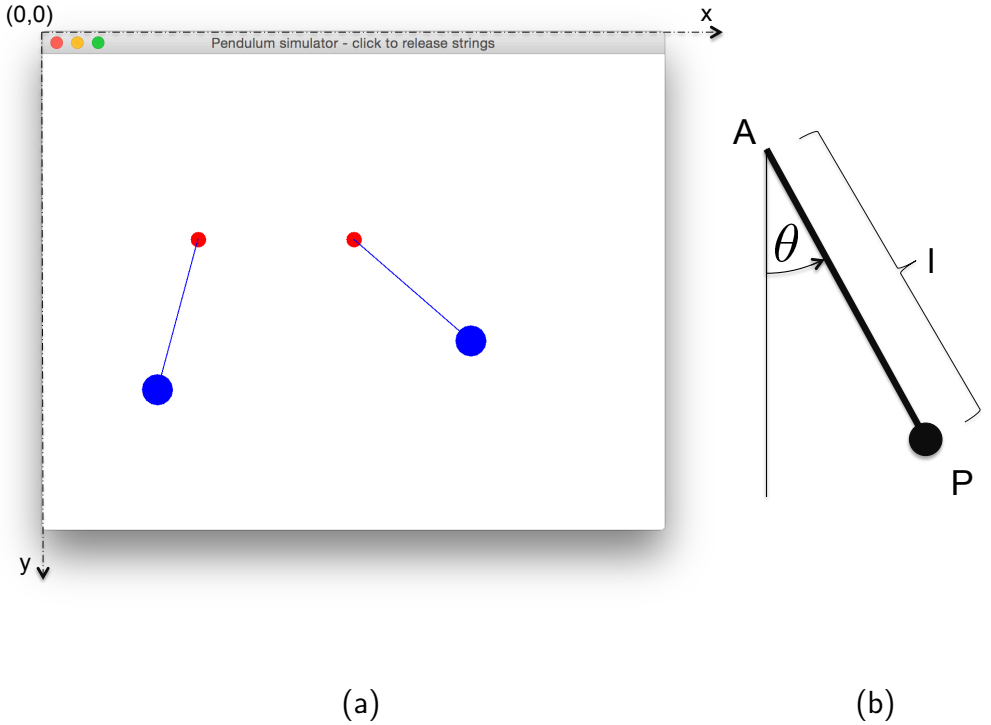
- Le barème est indicatif et le sujet est sur 11 pages.
- Les exercices sont indépendants.
- Un programme **mal indenté**, **mal commenté** ou avec de **mauvais choix de noms de variables** sera **sanctionné** *(entre -1 et +1 point)*.

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*

Dernières précisions importantes :
- 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 implémentées ;
- sauf indications contraires, la visibilité des attributs des objets créés devra être `public`.

1 Exercice 1 — Simulation numérique d'un pendule (12 pts)

On désire développer une application simulant numériquement le comportement d'un pendule simple, comme illustré dans la capture d'écran de l'interface graphique donnée dans la figure 1a.



(a) (b)

FIGURE 1 – L'interface graphique de l'application (a) ; définition de l'angle θ (b)

Notre pendule simple (sans friction) est défini par la longueur l de sa corde. Son état à l'instant t_k est complètement décrit par l'angle θ avec la verticale (voir la figure 1b) et par sa vitesse angulaire. Le poids est la seule force agissant sur le pendule dans notre exemple. Nous allons simuler le comportement dynamique en discrétisant le temps. En supposant que la vitesse angulaire est constante entre deux instants, nous pouvons mettre à jour l'état du pendule d'un instant à l'autre à l'aide des équations suivantes :

$$\begin{aligned} \ddot{\theta}_k &= \frac{-g}{l} \sin \theta_{k-1} \\ \dot{\theta}_k &= \dot{\theta}_{k-1} + \ddot{\theta}_{k-1} \\ \theta_k &= \theta_{k-1} + \dot{\theta}_{k-1} \end{aligned} \tag{1}$$

où g est la gravité (supposé à 1 dans notre cas). La notation θ_k s'interprète comme « θ *à l'instant* t_k »[1].

(Q1.1) Java, c'est la classe

Écrivez la classe `Pendulum` modélisant un pendule simple. Ajoutez tous les attributs nécessaires à la mémorisation de l'état du pendule (g peut être déclaré comme une constante numérique). Pour anticiper un affichage du pendule dans une fenêtre, on ajoutera également les coordonnées cartésiennes du point d'ancrage du pendule (point A dans la figure 1b) comme attribut de la classe. On choisira `double` comme type pour les attributs.
Remarque : il n'est pas nécessaire de mémoriser l'historique de l'état. Seul sa valeur à l'instant t_k suffit.
Ajoutez un constructeur dont les paramètres permettent d'initialiser tous les attributs de la classe.
Réponse : (1pt)

```
class Pendulum {
        public double cx,cy,len;
        public double theta, thetap, thetapp;
        public static final double G=1.0;

        public Pendulum (double cx, double cy, double len, double theta, double thetap) {
             this.cx =cx;
             this.cy =cy;
             this.len=len;
             this.theta=theta;
             this.thetap=thetap;
        }
}
```

0.5 pt pour les attributs
0.5 pt pour le constructeur

(Q1.2)

Écrivez la méthode `public void simulate()` de la classe `Pendulum`, qui met à jour l'état du pendule à partir de l'état de l'instant précédent selon les équations (1). On supposera la gravité g égale à 1.
Réponse : (1pt)

```
public void simulate () {
             thetapp = - 1.0*G*Math.sin(theta)/len;
             thetap += thetapp;
             theta += thetap;
        }
```

1. Les physiciens parmi vous comprenderont que $\dot{\theta}$ n'est pas une vraie dérivée dans cette exemple, mais une différence finie dont la valeur dépend aussi de la fréquence de mise à jour.

1pt pour les affectations correctes

(Q1.3)

Sans ajouter d'attribut, écrivez les méthodes `public double getX()` et `public double getY()` de la classe `Pendulum`, qui renvoient les coordonnées cartésiennes x et y de la position du pendule, c'est-à-dire du point P dans la figure 1b.

Réponse : (0.5 pt)

```
public double getX() {
    return cx + Math.sin(theta) * len;
}

public double getY() {
    return cy + Math.cos(theta) * len;
}
```

0.5pt pour les fonctions correctes

(Q1.4)

Écrivez la méthode `public void draw (Graphics g)` de la classe `Pendulum`, qui va dessiner le pendule via l'instance de `Graphics` donnée en argument. On choisira un rayon de 15 pour le point d'ancrage et de 30 pour la boule. Les couleurs sont à choisir librement. Vous pouvez vous servir des méthodes suivantes pour le dessin :

```
// Choisir la couleur rouge ou la couleur bleu etc.
g.setColor(Color.red); g.setColor(Color.blue);

// Dessiner une ligne de x1,y1 vers x2,y2
g.drawLine(int x1, int y1, int x2, int y2);

// Dessiner un ovale ou rectangle de largeur w et de hauteur h
// dont le point supérieur gauche se trouve à x et y
g.fillOval(int x, int y, int w, int h)
g.fillRect(int x, int y, int w, int h);
```

Attention ! les paramètres des méthodes `drawLine` et `fillOval` sont de type entier. N'oubliez pas de faire les conversions nécessaires.

Réponse : (1pt)

```
public void draw (Graphics g) {
    g.setColor(Color.red);
    double x = getX();
    double y = getY();

    N=7;
    g.fillOval((int)cx-N,(int)cy-N,2*N+1,2*N+1); // (2*N, 2*N) est aussi ok

    g.setColor (Color.blue);

    N=15
    g.fillOval((int)x-N,(int)y-N,2*N+1,2*N+1);  // (2*N, 2*N) est aussi ok

    g.drawLine((int)cx, (int)cy, (int) x, (int) y);
}
```

(Q1.5) Ça se fritte

Pour améliorer le réalisme, nous aimerions ajouter un deuxième type de pendule gérant la friction via l'équation suivante :

$$\ddot{\theta}_k = \frac{-g}{l}\sin\theta_{k-1} - \frac{k}{ml}\dot{\theta}_{k-1} \tag{2}$$

Ici, m est la masse du pendule et k un coefficient de friction. Pour simplifier, nous supposons que ces deux paramètres sont fusionnés en un seul, c'est-à-dire $\kappa = \frac{k}{m}$.
Sans modifier la classe Pendulum, écrivez la classe `PendulumF` modélisant ce nouveau type de pendule, ainsi que son constructeur et toute autre méthode nécessaire. Comme pour `Pendulum`, la nouvelle classe doit permettre la simulation, le dessin, et le renvoi des coordonnées cartésiennes.
Attention ! Implémentez uniquement les méthodes nécessaires. Profitez des avantages de la programmation orientée objet pour limiter la redondance.
Réponse : (2pt)

```
class PendulumF extends Pendulum{
    public double kappa;

    public PendulumF (double cx, double cy, double len, double theta, double thetap, double
         kappa) {
        super(cx,cy,len,theta, thetap);
        this.kappa=kappa;
    }

    public void simulate () {
        thetapp = - 1.0*Math.sin(theta)/len - kappa/len*thetap;
        thetap += thetapp;
        theta += thetap;
    }
}
```

0.5 pt pour l'attribut
1 pt pour le constructeur correcte avec super() 0.5 pt pour simulate

Intégration dans une interface graphique dynamique

(Q1.6)

L'objectif de cette partie est d'intégrer le code de simulation de nos pendules dans une application réelle les affichant sur une interface graphique. Pour cela, veuillez compléter le code source fourni dans la page 5. Ajoutez le code demandé dans les cinq rectangles noirs étiquetés A, B, C, D, E.
Nous souhaitons gérer deux pendules (un pendule simple sans friction, un pendule avec friction), les deux stockés dans un seul tableau. Les deux pendules auront une longueur de l=50 et l'état initial suivant : θ_0=$\frac{3}{4}\pi$ et $\dot{\theta}_0$=0. Les coordonnées des points d'ancrage respectifs sont $(150, 200)$ et $(300, 200)$. Le pendule avec friction est paramétré par κ=$\frac{k}{m}$=$1,8$.
Pensez également de placer l'allocation et le démarrage du Timer dans un des cinq rectangles A,B,C,D ou E. Vous vous servez du code suivant :

```
timer = new Timer(20, XXX);
timer.start();
```

Ici, `XXX` est à remplacer par la référence de l'objet recevant l'appel de la méthode `actionPerformed()`.

CODE A COMPLETER

```
import ...

public class Sim extends JFrame implements ActionListener {
    final int WIDTH = 600;
    final int HEIGHT = 480;
    Timer timer;
```

A

```
    // The constructor of the main window
    public Sim () {
        super ("Pendulum simulator");
        this.setSize(WIDTH, HEIGHT);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

B

```
        this.setVisible(true);
    }
    public void paint (Graphics g) {
```

C

```
    }
    public void actionPerformed(ActionEvent e) {
```

D

```
    }
    public static void main (String [] args) {
```

E

```
    }
}
```

=== Rectangle A === Réponse : (0.5pt)

```
Pendulum objs[];
```

Pas de supression de points s'il s'agit de variables séparées et non pas d'un tableau

=== Rectangle B === Réponse : (1pt)

```
objs = new Pendulum[2];
objs[0] = new Pendulum (150,200,50,Math.PI*3./4.,0);
objs[1] = new PendulumF (300,200,50,Math.PI*3./4.,0,1.8);

timer = new Timer(20, this);
timer.start();
```

0.5pt pour le tableau
0.5pt pour le timer correct

=== Rectangle C === Réponse : (1pt)

```
g.setColor(Color.white);
g.fillRect(0,0,width,height);

for (int i=0; i<objs.length; ++i) {
    objs[i].draw(g);
}
```

0.25 pt pour l'effacement de la fenetre
0.75 pt pour la boucle correcte avec appel de draw

=== Rectangle D === Réponse : (1pt)

```
for (int i=0; i<objs.length; ++i) {
    objs[i].simulate();
}
repaint();
```

0.5 pt pour la boucle correcte avec appel de simulate
0.5 pt pour le repaint
Soyez indulgeant si la simulation est fait dans paint() avec ici uniquement un appel à repaint. On n'enlève pas de points

=== Rectangle E === Réponse : (0.5pt)

```
Sim  s = new Sim ();
```

Couper les cordes

Nous aimerions ajouter une autre fonctionnalité à notre application : à un clic de la souris, les cordes des pendules seront coupées et les pendules se transforment en simples boules suivant des trajectoires de chutes libres (sans friction). Les coordonnées cartésiennes x et y des boules sont soumises aux équations de mise à jour suivantes :

$$\begin{aligned} \ddot{x}_k &= 0 \\ \ddot{y}_k &= 0.2 \\ \dot{x}_k &= \dot{x}_{k-1} + \ddot{x}_{k-1} \\ \dot{y}_k &= \dot{y}_{k-1} + \ddot{y}_{k-1} \\ x_k &= x_{k-1} + \dot{x}_{k-1} \\ y_k &= y_{k-1} + \dot{y}_{k-1} \end{aligned} \tag{3}$$

Dans la suite, les classes existantes `Pendulum` et `PendulumF` ne seront pas modifiées. Le nouveau comportement sera défini par une nouvelle classe.

(Q1.7)

Ecrire une classe `Ball` décrivant le comportement d'une boule, ainsi que son constructeur et toute autre méthode nécessaire. Comme pour `Pendulum`, la nouvelle classe doit permettre la simulation et le dessin.
Inutile d'implémenter les méthodes `getX` et `getY`.
Réponse : (1.5 pt)

```
class Ball  {
        public double x,y,dx,dy;
        public static final double ax=0, ay=0.2;

        public Ball (double x, double y, double dx, double dy) {
            this.x =x;
            this.y =y;
            this.dx=dx;
            this.dy=dy;
        }

         public void simulate () {
            dx += ax;
            dy += ay;
            x += dx;
            y += dy;
        }

        public void draw (Graphics g) {
            g.setColor (Color.blue);
            g.fillOval((int) x-15, (int) y-15, 30, 30);
        }
    }
```

0.25 pt pour les attributs
0.25 pt pour le constructeur
0.5 pt pour simulate
0.5 pt pour draw

(Q1.8)

La coupure des cordes est modélisée par le code java suivant, exécuté à chaque clic de la souris grâce à la méthode suivante :

```
public void mouseClicked( MouseEvent e ) {
    for (int i=0; i<objs.length; ++i) {
        objs[i] = objs[i].returnBall();
    }
}
```

Ici, `obj` est un tableau unique stockant tous les objets en mouvement (pendules de tout type et boules).
Décrivez, dans les grandes lignes, en français et sans écrire du code java, les modifications nécessaires pour que le code ci-dessus fonctionne. Inutile de décrire les modifications nécessaires pour le MouseListener. Inutile de donner des équations numériques précises.

Réponse : (1pt)

0.5 pt pour classe mère commune liant Pendulum, PendulumF, Ball et le changement de type du tableau.
0.5 pt pour la méthode returnBall() décliné en trois versions : une pour ball (renvoie 'this') ; une pour Pendulum (avec calcul de la position et des dérivées) ; Une version abstraite pour la classe mère.)

2 Réseau social (8 pts)

On désire développer une application orientée objet de réseau social qui doit stocker les profils des membres inscrits ainsi que les connections entre les profils (amis), et qui doit fournir quelques fonctions de gestion de ces profils.
Un profil contient le `nom` de la personne, son âge, son `genre` ('m' ou 'f') et une liste (un tableau) de profils d'`amis`. On suppose que le nombre maximal d'amis est 50. La variable statique `MAX_AMIS` dans la classe Profil définit cette constante :
`public static final int MAX_AMIS = 50;`

(Q2.1)

Écrivez la classe `Profil` avec tous ses attributs et son constructeur qui initialise tous les attributs de la classe.
Remarque : Les tableaux doivent être créés avec leur taille maximale. La fin du/des tableau(x) sera détectée par le premier élément « null ».

Réponse : (1 pt)

```
class Profil
{
  public static final int MAX_AMIS = 50;

  public String nom;
  public int age;
  public char genre;
  public Profil[] amis;
  public int nbamis;   // optionnel si on parcours les elements non nuls d'amis

  public Profil(String n, int a, char g)
  {
    nom = n;
    age = a;
    genre = g;
    amis = new Profil[MAX_AMIS];
    nbamis = 0;
  }
}
```

0.5 pt pour les attributs (publics ou privés)
0.5 pt pour le constructeur

(Q2.2)

Rajoutez une méthode `ajouterAmi(...)` qui ajoute un ami à la liste de profils d'amis. La méthode doit renvoyer le nouveau nombre d'amis ou -1 si la liste est complète.

Réponse : (1 pt)

```
public int ajouterAmi(Profil ami)
{
  if (nbamis < MAX_AMIS)
  {
    amis[nbamis] = ami;
    // alternative "deep copy": amis[nbamis] = new Profil(ami.nom, ami.age, ami.genre);
    nbamis++;
    return nbamis;
  }
  else
    return -1;
}
```

(Q2.3)

Surcharger (réécrivez) la méthode `toString(...)` pour qu'elle renvoie une ligne de texte qui contient le nom, l'âge, le genre, et le nombre d'amis d'un profil.

Réponse : (0.5 pt)

```
public String toString()
{
  return "nom : " + nom + "  age : " + age + "  genre : " + genre + "  nombre d'amis : "
      + nbamis;
}
```

(Q2.4)

Rajoutez une méthode `public boolean aAmi(String nom)` qui doit renvoyer `true` si le profil contient un ami avec le nom donné ou `false` sinon.

Réponse : (0.5 pt)

```
public boolean aAmi(String nom_ami)
{
  int i=0;
  while (amis[i]!=null)   // ou : for(i=0; i<nbamis; i++)
  {
    if (amis[i].nom.equals(nom_ami))   // == aussi possible
      return true;
    i++;
  }
  return false;
}
```

(Q2.5)

Rajoutez une méthode `public Profil[] amisCommun(Profil p)` qui renvoie un tableau d'amis communs entre le profil courant et un autre profil `p`. Le tableau renvoyé fera toujours MAX_AMIS cases mais seul les cases non-nulles sont des profils.

Réponse : (1.5 pt)

```
public Profil[] amisCommuns(Profil p)
{
  Profil[] resultat = new Profil[MAX_AMIS];
  int r=0;
  int i=0, j;
  while (i<amis.length && amis[i]!=null)   // ou : for(i=0; i<nbamis; i++)
  {
    j=0;
    while (j<p.amis.length && p.amis[j]!=null)   // ou : for(j=0; j<p.nbamis; j++)
    {
      if (amis[i]==p.amis[j])
        resultat[r++] = amis[i];
      j++;
    }
    i++;
  }
  return resultat;
}
```

ou utilisation aAmi() / ajouterAmi()

(Q2.6)

On souhaite maintenant créer une « base de donnée » avec tous les profils du réseau social. Pour cela, créez une classe `ReseauSocial` (avec ses attributs et un constructeur) qui contient une liste de profils stockée dans un tableau qui est vide au départ. On suppose qu'il existe un nombre maximal de profils dont la valeur est stockée dans cette classe via la constante suivante :
`public static final int MAX_PROFILS = 1000;`

Réponse : (1 pt)

```
public class ReseauSocial
{
  public static final int MAX_PROFILS = 1000;
  private Profil[] profils;
  int nbprofils;

  public ReseauSocial()
  {
    profils = new Profil[MAX_PROFILS];
    nbprofils=0;
  }
}
```

(Q2.7)

Rajoutez à la classe `ReseauSocial` une méthode `public int ajouterProfils(Profil[] p)` qui ajoute le tableau `p` à la liste des profils et qui renvoie le nombre total de profils dans la base.

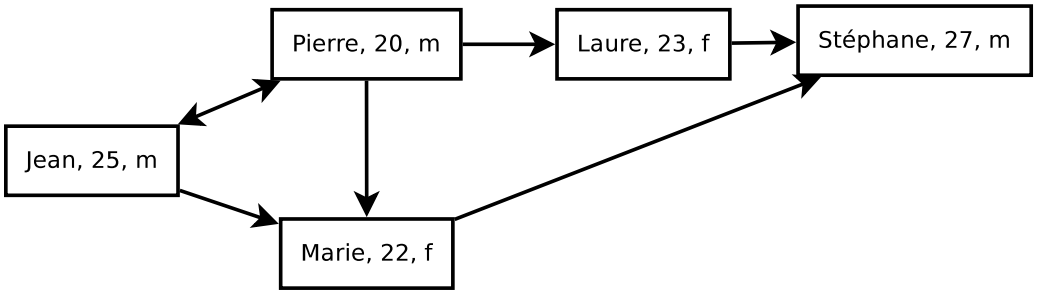
Réponse : (1 pt)

```
public int ajouterProfils(Profil[] p)
{
  for(int i=0; i<p.length; i++)
  {
    if (nbprofils < MAX_PROFILS)  // ou: verification de la taille totale au debut
    {
      profils[nbprofils] = p[i];
      nbprofils++;
    }
  }
  return nbprofils;
}
```

-0.5 points si le dépassement du tableau n'est pas géré.

(Q2.8)

Rajouter une méthode `main` qui crée le réseau social illustré ci-dessous. Les blocs correspondent aux profils avec le nom, l'âge et le genre, et une flèche de A vers B signifie que B est un ami de A.



Réponse : (1 pt)

```
public static void main(String[] a)
{
  ReseauSocial rs = new ReseauSocial();

  Profil jean = new Profil("Jean", 25, 'm');
  Profil pierre = new Profil("Pierre", 20, 'm');
  Profil marie = new Profil("Marie", 22, 'f');
  Profil laure = new Profil("Laure", 23, 'f');
  Profil stephane = new Profil("Stephane", 27, 'm');

  jean.ajouterAmi(pierre);
  jean.ajouterAmi(marie);

  pierre.ajouterAmi(jean);
  pierre.ajouterAmi(marie);
  pierre.ajouterAmi(laure);

  marie.ajouterAmi(stephane);
  laure.ajouterAmi(stephane);

  Profil[] profils = {jean, pierre, marie, laure, stephane};
  rs.ajouterProfils(profils);
}
```

(Q2.9)

Dans la méthode `main` écrivez le code affichant les amis communs entre Jean et Pierre.

Réponse : (0.5 pt)

```
Profil[] m = jean.amisCommuns(pierre);
int i=0;
while (i<m.length && m[i]!=null)
{
  System.out.println(m[i]);
  i++;
}
```