Projet intégrateur : informatique
1ère thématique transversale
Comportement d'un lève-charge

CB1 2008-2009

Objectifs et déroulement de la session informatique

Dans cette première thématique transversale du projet intégrateur, vous avez réalisé le montage expérimental et étudié le modèle mathématique de la trajectoire d'une masse suspendue à un système de lève-charge. L'objectif de la session informatique est de réaliser une application graphique Java de simulation de la trajectoire, qui vous permettra de confronter résultats pratiques et théoriques. Les fonctionnalités attendues de cette application sont :

- calcul de l'instant auquel la trajectoire de la masse coupe théoriquement la barrière lumineuse ;
- calcul de la trajectoire théorique complète de la masse : pendant son élévation puis après l'interruption du lève-charge ;
- importation des données de la trajectoire obtenues expérimentalement ;
- tracé des courbes des trajectoires théorique et expérimentale sur un même graphique.

Remarque : Le développement de l'application (conception des algorithmes et implémentation en Java) est un **travail individuel**. Seul le recueil des données numériques propres au montage sera effectué de manière collective par chacun des groupes-projet.

Organisation de la session et rendu des travaux

Vous disposez de 3 semaines pour la réalisation de ce projet, jalonnées comme suit :

TP1 : 27 février (2h) (*conception, implémentation et test des algorithmes de calcul de la trajectoire théorique*) : compléter le fichier source `SommesEtMatrices.java` et le déposer sur Campus à la fin de la séance.

d'ici la séance suivante : terminer l'implémentation de `SommesEtMatrices.java` ; rassembler les paramètres (données numériques) de calcul de la trajectoire théorique de la masse ; faire le lien entre la formule de calcul obtenue en mathématiques et les algorithmes développés.

TP2 : 6 mars (2h) (*calcul et tracé de la trajectoire théorique*) : compléter le fichier source `TrajectoireTheo.java` et le déposer sur Campus à la fin de la séance.

d'ici la séance suivante : terminer l'implémentation `TrajectoireTheo.java` ; rassembler les résultats numériques obtenus expérimentalement dans un fichier Excel ; enregistrer ce fichier au format texte avec séparateur (CSV).

TP3 : 13 mars (2h) (*lecture du fichier CSV et tracé de la courbe expérimentale*) : compléter le fichier source `TrajectoireExpe.java` et le déposer sur Campus à la fin de la séance.

20 mars – 20h : date **au plus tard** de dépôt de l'archive complète du projet sur Campus.

À chaque séance de TP, pensez à apporter le polycopié de cours de PROGRAMMATION, ainsi que :

- à la 1ère séance : le polycopié de cours d'ALGORITHMIQUE ;
- à la 2ème séance : la formule mathématique et les paramètres numériques du montage ;
- à la 3ème séance : le fichier CSV contenant vos mesures expérimentales de la trajectoire.

Les fichiers source seront téléchargés et déposés sur Campus, dans l'espace intégrateur 1ère année. Le retour des évaluateurs se fera aussi par l'intermédiaire de Campus. Enfin, **jusqu'au 18 mars**, un espace-forum sur Campus sera ouvert pour vous permettre de poser vos questions. Vous pouvez vous y abonner pour recevoir, par mail, les messages de ce forum.

Évaluation

La note de la partie informatique de la thématique comprend :
- l'évaluation des fichiers remis à l'issue de chacune des trois séances de TP (10% chacun)
- l'évaluation de l'application finale (70%)

Ces évaluations reposeront principalement sur les critères suivants :

1. **conformité du programme, justesse de la programmation et des algorithmes** ;
2. **qualité (clarté) du code** : les règles énoncées en cours de programmation et d'algorithmique doivent être respectées.
3. **initiative** : vous êtes vivement encouragés à apporter à votre programme toutes les améliorations qui vous sembleraient judicieuses pour simplifier le programme et son utilisation ou encore pour faciliter l'analyse des résultats théoriques et expérimentaux.

Les conditions suivantes sont quant à elles éliminatoires :
- **remise d'un devoir au-delà des date et heure limites** : *N'attendez pas le dernier moment pour déposer sur Campus : vous pouvez déposer votre projet au fur et à mesure de son avancement. Seule la dernière version déposée sera évaluée.*
- **erreur de compilation ou levée d'exception** : *Les parties de code inachevées ou buguées doivent être placées en commentaire.*
- un rappel enfin :

  la fraude et la copie sont sanctionnées par un **0 pour toute la partie informatique**, sans préjuger d'éventuelles **suites disciplinaires.**

Enseignants.

Gilles Chabert, Philippe David, Romuald Debruyne,
Sophie Demassey, Rémi Douence, Narendra Jussien.

Responsable : Sophie.Demassey@emn.fr, bureau B210.

1 TP1 : conception et implémentation des algorithmes

1.1 Objectifs et déroulement de la séance

Avant de débuter le développement de l'application à proprement dite, vous allez concevoir et implémenter les algorithmes nécessaires au calcul de la trajectoire théorique de la masse. Ces algorithmes consistent principalement en des calculs de sommes partielles et d'expressions polynomiales sur des matrices. Comme ces calculs résultent parfois en de très grands nombres (supérieurs à 10^{300}), cet exercice sera l'occasion d'observer un principe fondamental en informatique : *l'infini ne se calcule pas*. En effet, comme l'espace mémoire d'un ordinateur est limité, il n'est pas possible de manipuler des données numériques arbitrairement grandes. Cependant, vous verrez que choisir un type de données adéquat ou implémenter un algorithme «intelligent» permet de repousser cette limite.

Ce premier programme comprend trois classes :

`Matrice`	fournit un ensemble d'opérations de manipulation des matrices de réels ;
`TestSommesEtMatrices`	fournit un ensemble de tests pour vous aider à valider vos algorithmes ;
`SommesEtMatrices`	est un squelette de classe dans laquelle vous implémenterez les algorithmes.

Seule cette dernière classe `SommesEtMatrices` est à compléter : chaque algorithme sera implémenté dans le corps d'une méthode dont la signature est donnée. Pour vérifier votre implémentation, vous exécuterez `TestSommesEtMatrices`.

1. téléchargez depuis Campus les fichiers source : `Matrice.java`, `TestSommesEtMatrices.java` et `SommesEtMatrices.java` et enregistrez-les dans un nouveau répertoire ;
2. compilez les 3 fichiers puis exécutez `TestSommesEtMatrices` qui devrait afficher : `19 erreurs` ;
3. implémentez chacune des méthodes de la classe `SommesEtMatrices` comme spécifiées en commentaire dans l'en-tête des méthodes ;
4. testez votre implémentation au fur et à mesure en exécutant `TestSommesEtMatrices` ;
5. répondez aux questions `Q1-Q5` posées en commentaires dans le fichier `SommesEtMatrices.java` ;
6. à la fin de la séance : déposez le fichier `SommesEtMatrices.java` sur Campus. Même inachevé, assurez-vous que celui-ci compile et s'exécute correctement.

1.2 La classe `SommesEtMatrices`

Les méthodes de classe (`static`) à implémenter :

```
int factorielle(int)
int sommeFactorielleNaif(int)
long sommeFactorielle(int)
double sommeExposant(double, int)
Matrice sommeExposant(Matrice, int)
Matrice exponentielle(double, Matrice, int)

Matrice combinaisonMatrices(Matrice, Matrice, Matrice, Matrice, double)
Matrice combinaisonExponentielleMatrices(Matrice, Matrice, Matrice, Matrice, double, int)
```

La spécification et la signature de chacune des méthodes sont fournies, par exemple :

```
/** calcul de la factorielle de n   fonction
 * @param n entier positif          noms et types des paramètres en entrée
 * @return 1x2x···xn entier */      valeur et type de la donnée de retour
 public int factorielle(int n) {    signature
       return 0 ;                   instruction à modifier
 }
```

Des questions portant notamment sur la complexité des algorithmes sont aussi posées. Vous pouvez y répondre en complétant le commentaire directement dans le fichier source :

```
// Q1.1 : combien d'opérations environ sont effectuées par factorielle(10) ?
// Q1.2 : quelle est la complexité de factorielle(n) ?
// Q2.1 : combien d'opérations environ sont effectuées par sommeFactorielleNaif(10) ?
// Q2.2 : quelle est la complexité de sommeFactorielleNaif(n) ?
// Q3.1 : combien d'opérations environ sont effectuées par sommeFactorielle(10) ?
// Q3.2 : quelle est la complexité de sommeFactorielle(n) ?
// Q4 : comparez sommeFactorielleNaif(100) et sommeFactorielle(100) ?
// Q5 : que pouvez-vous dire de sommeExposant(10,500) ?
```

1.3 Spécifications de la classe `Matrice`

Cette classe permet de modéliser des matrices de flottants $b = (b_{ij})_{0 \leq i < m, 0 \leq j < n} \in \mathbb{R}^{m \times n}$:

`Matrice(int m, int n)`	constructeur : crée la matrice zéro de $\mathbb{R}^{\mathtt{m} \times \mathtt{n}}$
`Matrice(Matrice a)`	constructeur copie : crée une matrice identique à `a`
`Matrice(double[][])`	constructeur : crée une matrice à partir d'un tableau 2D
`int nbLignes()`	retourne la première dimension de la matrice courante
`int nbColonnes()`	retourne la seconde dimension de la matrice courante
`double getElement(int i, int j)`	retourne l'élément $b_{\mathtt{ij}}$ de la matrice courante
`void setElement(int i, int j, double val)`	affecte `val` à l'élément $b_{\mathtt{ij}}$ de la matrice courante
`static Matrice identite(int n)`	crée et retourne la matrice identité de $\mathbb{R}^{\mathtt{n} \times \mathtt{n}}$
`static Matrice nulle(int n)`	crée et retourne la matrice identiquement nulle de $\mathbb{R}^{\mathtt{n} \times \mathtt{n}}$
`void ajoute(Matrice a)`	ajoute la matrice `a` à la matrice courante
`void retranche(Matrice a)`	soustrait la matrice `a` à la matrice courante
`void multiplie(double l)`	multiplie la matrice courante par un scalaire `l`
`void multiplieDroite(Matrice a)`	multiplie à droite la matrice courante par `a`
`void multiplieGauche(Matrice a)`	multiplie à gauche la matrice courante par `a`
`static Matrice somme(Matrice a, Matrice b)`	crée et retourne la matrice somme `a+b`
`static Matrice produit(double l, Matrice a)`	crée et retourne la matrice produit `l.a`
`static Matrice produit(Matrice a, Matrice b)`	crée et retourne la matrice produit `ab`
`boolean equals(Matrice a)`	retourne vrai ssi la matrice `a` est égale à la matrice courante
`String toString()`	retourne une chaîne représentant la matrice courante

Rappel Java :

`Matrice a = Matrice.somme(b,c) ;`	appel d'une **méthode de classe** (`static`)
`a.ajoute(b) ; int m = a.nbLignes() ;`	appel de **méthodes d'instance** sur **a**
`double[][] t = {{0,1},{1,0},{0,0}} ;`	OK : déclaration+allocation+initialisation
`double[][] t ; t = {{0,1},{1,0},{0,0}} ;`	NON : erreur de syntaxe

1.4 D'ici la semaine prochaine...

1. Terminer l'implémentation de `SommesEtMatrices` ;
2. établir les formules mathématiques de la trajectoire de la masse avant (phase I) et après (phase II) l'arrêt du lève-charge ;
3. faire le lien entre ces formules et les algorithmes de `SommesEtMatrices` ;
4. rassembler les valeurs des paramètres physiques du dispositif (propres à votre montage) entrant dans ces formules.

2 TP2 : calcul et tracé de la courbe théorique

2.1 Objectifs et déroulement de la séance

Il s'agit de développer le programme de calcul et de visualisation de la trajectoire théorique de la masse dans les phases I (avant) et II (après l'interruption du levage), étant donnés :

- la formulation mathématique de la trajectoire $x(t)$ (l'ordonnée de la masse en fonction du temps) ;
- les valeurs des paramètres physiques du dispositif (masse, raideur du ressort,...) entrant dans cette formulation et que vous aurez mesurées sur votre propre montage ;
- les classes `Matrice` et de `SommesEtMatrices` pour le calcul de $x(t)$;
- un programme Java d'affichage de courbes entièrement fourni, comprenant quatre fichiers :

`Graphique.java`	création et affichage de graphiques ;
`Courbe.java`	modélise une courbe par un tableau de points de $\mathbb{R} \times \mathbb{R}$;
`FichierPNG.java`	pour l'enregistrement du graphique au format PNG ;
`marques.png`	pour l'affichage des marques des points d'une courbe. *NB : ce fichier doit être enregistré dans le répertoire d'où le programme est lancé.*

Vous implémenterez les fonctions de calcul et de tracé de la trajectoire théorique dans une classe `TrajectoireTheo` dont le squelette est fourni.

1. téléchargez depuis Campus les fichiers : `TrajectoireTheo.java`, `Graphique.java`, `Courbe.java`, `FichierPNG.java`, `marques.png` et enregistrez-les dans votre répertoire projet (avec `SommesEtMatrices.java` et `Matrice.java`) ;
2. compilez tous les fichiers puis exécutez `Graphique` pour tester votre installation ;
3. implémentez la classe `TrajectoireTheo` comme spécifiée ci-dessous ;
4. à la fin de la séance : déposez le fichier `TrajectoireTheo.java` sur Campus. Même inachevé, assurez-vous que celui-ci compile et s'exécute correctement.

2.2 La classe `TrajectoireTheo`

En mathématiques, vous avez étudié les solutions $Y(t) \in \mathbb{R}^3$ du système d'équations différentielles modélisant le mouvement général du dispositif. Ces solutions s'expriment comme suit :

$$Y(t) = e^{tB}.C + tD + E \quad \text{où } Y = \begin{pmatrix} x(t) \\ \dot{x}(t) \\ X(t) \end{pmatrix}, B \in \mathbb{R}^{3\times3}, C, D, E \in \mathbb{R}^3.$$

Les courbes $x_I(t)$ et $x_{II}(t)$ de trajectoire de la masse dans la phase I et dans la phase II satisfont cette équation pour des valeurs (B_I, C_I, D_I, E_I) et $(B_{II}, C_{II}, D_{II}, E_{II})$ des matrices qui dépendent des paramètres physiques du dispositif. Pour tracer la trajectoire complète, on calculera la position $x(t)$ de la masse à différents instants t variant avec un pas de temps t_0 et compris entre un instant initial $t_{\min} = 0$ et un instant final $t_{\max} = K.t_0 : t = 0, t_0, 2t_0, \ldots, Kt_0$.

Initialement, la masse suit la trajectoire croissante $x_I(t)$ jusqu'à ce que celle-ci rencontre la barrière lumineuse placée à une hauteur x_1. À cet instant, noté $t_1 = \min\{kt_0 \mid 0 \leq k \leq K, x_I(kt_0) > x_1\}$, la masse change de trajectoire $x_{II}(t)$. Afin que les deux trajectoires coincident à l'instant t_1, les matrices $(B_{II}, C_{II}, D_{II}, E_{II})$ sont initialisées en fonction de la valeur de $Y_I(t_1)$.

On enregistrera dans un objet `Courbe` l'ensemble P des points calculés de la trajectoire :

$$P = \{(t, x(t)) \mid t = 0, t_0, 2t_0, \ldots, Kt_0\} \quad \text{avec} \quad x(t) = \begin{cases} x_I(t) & \text{si } t \leq t_1 \\ x_{II}(t) & \text{si } t \geq t_1 \end{cases}$$

Enfin, on affichera la courbe via la classe `Graphique`.

1. Exprimer les matrices B, C, D, E en fonction des paramètres physiques mesurés (masse, raideur du ressort,...) et correspondant aux solutions de la trajectoire dans les phases I et II ;
2. déclarer chacun des paramètres utiles comme une **constante de classe** `TrajectoireTheo` et initialiser (attention à la correspondance des unités ! (g, m, s,...)), par exemple :
```
private final static double R = 0.025 ; // rayon de la poulie (m)
```
3. déclarer chacune des matrices B, C, D, E comme une **variable d'instance** de `TrajectoireTheo` et compléter le constructeur de `TrajectoireTheo` pour réaliser la création de ces objets (les éléments des matrices sont initialisés à 0), par exemple :
```
private Matrice B ; // matrice 3x3
...
B = new Matrice(3,3) ;
```
4. implémenter la méthode `initialiserMatricesPhaseI` pour l'initialisation des matrices B, C, D, E correspondant à la solution théorique de la trajectoire dans la phase I ;
5. implémenter la méthode `initialiserMatricesPhaseII` pour l'initialisation des matrices B, C, D, E correspondant à la solution théorique de la trajectoire dans la phase II.
6. implémenter les méthodes suivantes :

`Matrice Y(double t)`	retourne le vecteur $Y(t) = e^{tB}.C + tD + E$
`double x(double t)`	retourne le premier élément du vecteur $Y(t)$

7. implémenter la méthode `creeCourbe` qui calcule l'ensemble des points P puis initialise et retourne un objet `Courbe` correspondant ;
8. implémenter `main` pour afficher la courbe via un objet `Graphique`. *On pourra prendre exemple sur l'implémentation du* `main` *test de la classe* `Graphique`.

2.3 Spécifications des classes `Courbe` et `Graphique`

La classe `Courbe` permet de modéliser des courbes spécifiées par : une liste de points $(x, y) \in \mathbb{R}^2$, un titre, une couleur, un symbole de marque, un niveau de transparence. Voici une sélection des méthodes les plus utiles, reportez-vous au code `Courbe.java` pour plus de détails :

`Courbe(String)`	constructeur : crée une courbe vide de nom `s`
`int getNbPoints()`	retourne le nombre de points de la courbe
`double getX(int i)`	retourne l'abscisse du `i`-ème point de la courbe
`double getY(int i)`	retourne l'ordonnée du `i`-ème point de la courbe
`void setCouleur(Color c)`	modifie la couleur d'affichage de la courbe
`void ajouter(double x, double y)`	ajoute un point (x, y) à la courbe

La classe `Graphique` permet d'afficher des objets de type `Courbe` :

`Graphique(String t, int l, int h)`	crée un graphique de titre `t` et de dimensions `l`×`h` en pixels
`void ajouter(Courbe c)`	ajoute une courbe `c` au graphique
`void montrer()`	affiche le graphique

2.4 D'ici la semaine prochaine...

1. Terminer l'implémentation de `TrajectoireTheo` ;
2. enregistrer les valeurs de la trajectoire expérimentale au moyen du logiciel *Dynamic* dans un classeur Excel puis le convertir en un fichier texte (CSV) ;
3. revoir les exercices des TP de programmation portant sur la lecture d'un fichier (`FileReader, BufferedReader, readLine(), parseDouble()`,...).

3 TP3 : lecture et tracé de la courbe expérimentale

3.1 Objectifs et déroulement de la séance

Il s'agit de développer le programme de lecture du fichier de données expérimentales que vous aurez préalablement relevées sur votre montage. Cette partie sera implémentée dans une classe `TrajectoireExpe.java` dont un squelette sommaire vous est fourni.

1. téléchargez depuis Campus le fichier `TrajectoireExpe.java` et enregistrez-le dans le répertoire de votre projet ;
2. implémentez la classe `TrajectoireExpe` comme spécifiée ci-dessous ;
3. à la fin de la séance : déposez le fichier `TrajectoireExpe.java` sur Campus. Même inachevé, assurez-vous que celui-ci compile et s'exécute correctement.

3.2 La classe `TrajectoireExpe`

La classe `TrajectoireExpe` possèdera a minima les deux fonctionnalités suivantes :
- la lecture d'un fichier texte comprenant l'ensemble des points $(t, x(t))$, avec t l'instant et $x(t)$ la hauteur de la masse, mesurés expérimentalement ;
- la création d'un objet `Courbe` à partir de cet ensemble de points.

Vous êtes libres d'implémenter ces deux fonctions comme vous l'entendez. À vous également d'intégrer cette nouvelle classe dans votre projet. Si vous ne disposez pas encore de vos valeurs de mesures expérimentales, créez un fichier texte à partir de données fictives.

3.3 Lecture de données numériques dans un fichier texte

Pour la lecture des données, on pourra faire appel à des bibliothèques standards du JDK pour : ouvrir le fichier en lecture (`FileReader`), lire le fichier ligne à ligne (`BufferedReader`) puis mot à mot (`String`), enfin convertir les mots en valeurs numériques (`Double`, `Integer`).

La sélection de méthodes présentée ci-dessous devrait suffire à votre réalisation. Pour une description avancée de ces méthodes et classes, reportez vous à l'API Java :
http://java.sun.com/j2se/1.5.0/docs/api/ ou http://java.sun.com/javase/6/docs/api/.

Classe	Méthode	Description
`FileReader`	`FileReader(String s)`	ouvre le fichier nommé `s` en lecture (converti en flux de caractères)
`BufferedReader`	`BufferedReader(FileReader)`	mise en tampon (buffer) du flux
`BufferedReader`	`String readLine()`	extrait et retourne la prochaine ligne de texte du tampon courant ; retourne `null` si le tampon est vide (ex : fin du fichier)
`String`	`String[] split(String r)`	sépare la chaîne de caractères courante, en un tableau de chaînes, suivant un séparateur décrit par une expression régulière `r` (voir ci-après)
`String`	`String replace(char oc, char nc)`	retourne la chaîne de caractères courante dans laquelle le caractère `oc` est remplacé par le caractère `nc`
`Double`	`static double parseDouble(String s)`	convertit `s` en `double`
`Integer`	`static int parseInt(String s)`	convertit `s` en `int`

Aide à l'implémentation : quelques rappels...

- pensez à *importer* les classes : `FileInputStream`, `BufferedReader`, `FileReader` proviennent de la bibliothèque `java.io` ;
- attention les méthodes de ces classes peuvent lever des exceptions de type `IOException` : il faut les intercepter (`try{...} catch`) ou bien les relancer (`throws`) ;
- vous êtes libres du format de votre fichier de données. Rappelez-vous cependant qu'en java, un fichier texte se lit de haut en bas (ligne par ligne avec la méthode `readLine()` de `BufferedReader`). De plus, si plusieurs données sont contenues dans une même ligne, il faut les séparer par un ou plusieurs caractères tels que point-virgule, espace, tabulation. Attention à ne pas utiliser une virgule ou un point si ces caractères apparaissent dans vos données ! En java, la méthode `String[] split(String r)` de la classe `String` permet de séparer une chaîne de caractères suivant un séparateur quelconque décrit par une *expression régulière* `r`.
  Exemples d'utilisation sur la chaîne `"java 1.5 c'est bien... java 6 c'est mieux !"` :

`r`	liste des sous-chaînes retournée par `split(r)`
`"java"`	`""; " 1.5 c'est bien... "; " 6 c'est mieux !"`
`"\\."`	`"java 1"; "5 c'est bien"; ""; ""; " java 6 c'est mieux !"`
`"[. ]"`	`"java"; "1"; "5"; "c'est"; "bien"; ""; ""; ""; " java"; "6"; "c'est"; "mieux !"`
`"[. ]+"`	`"java"; "1"; "5"; "c'est"; "bien"; "java"; "6"; "c'est"; "mieux !"`
`"[ \t]+"`	`"java"; "1.5"; "c'est"; "bien..."; "java"; "6"; "c'est"; "mieux !"`

  Le dernier exemple `r="[ \t]+"` permet de séparer la chaîne mot à mot suivant les caractères : ESPACE (' ') OU ('[' ']') TABULATION ('\t') CONSÉCUTIFS ('+').
- Il n'est possible de convertir une chaîne de caractères `s` en un double au moyen de la méthode `Double.parseDouble(String s)` que si le format de la chaîne `s` est reconnu, comme par exemple : `"777"; "3.14"; "1.94E-1";...` Notez le point à la place de la virgule, et la spécification de la méthode `replace(char oldChar, char newChar)` de la classe `String`.

3.4 D'ici la semaine prochaine...

1. Terminez l'implémentation de votre projet et n'hésitez pas à apporter votre touche personnelle !
2. Créez une archive **complète** de votre projet comprenant : tous les fichiers source commentés, tout autre fichier nécessaire à la compilation et l'exécution, une image au format `png` de votre graphique final, un fichier pdf ou texte simple contenant les instructions pour l'exécution de votre programme ;
3. Déposez l'archive sur Campus **au plus tard, le vendredi 20 mars 2009 à 20h**.
   Rappels des conditions éliminatoires : **fraude et copie**, retard de remise du projet, erreur de dépôt (archive vide, incomplète ou illisible), erreurs de compilation, exceptions levées à l'exécution.

Pensez à poser vos questions sur le forum de Campus avant mercredi 18 mars inclus.

Bon courage :)