

Sujets de TD et TP JAVA

M2 BBSG

C.Chaouiya

claudine.chaouiya@esil.univ-mrs.fr

H.Garreta

henri.garreta@lif.univ-mrs.fr

2006/2007

Feuille 1

Notre but ici est de commencer à écrire des programmes en Java, mais sans nous occuper pour le moment de programmation orientée objets. On définira donc des classes modules, entièrement faites de membres **static**.

Notes :

1. Rappelons la structure minimale d'un programme java :

```
class NomDeLaClasse {  
    ...  
    public static void main(String[] args) {  
        // code de la méthode principale  
    }  
    ...  
}
```

Immédiatement avant et/ou après la fonction **main**, à l'intérieur de la classe, se trouvent les déclarations des éventuelles autres fonctions et variables requises par le programme. En principe, le texte ci-dessus est écrit dans un fichier **NomDeLaClasse.java**. On le compile en tapant la commande : **javac NomDeLaClasse.java** et on l'exécute par la commande : **java NomDeLaClasse**.

2. Pour afficher à la console la valeur d'une variable notée **x** (quel que soit son type) on écrit une expression du genre de :
System.out.println("valeur de x : " + x);
3. Les arguments de la ligne de commande sont dans le tableau de chaînes, souvent nommé **args** que reçoit la méthode **main** (**args[0]** est le premier argument, **args[1]** le second, etc.). C'est ainsi que nos programme recevrons des valeurs de l'utilisateur, pour le moment. N'essayez pas de lire des valeurs, même simples, au clavier. En Java, ce n'est pas élémentaire...
4. Pour obtenir l'entier qu'une chaîne représente, cherchez parmi les méthodes statiques de la classe **java.lang.Integer**.

Question 1: Quelques boucles salutaires... Ecrivez un programme qui affiche les informations fournies par les méthodes suivantes (**N** est fourni en ligne de commande par l'utilisateur) :

- une méthode qui calcule la somme des **N** premiers entiers **pairs**, (pour **N**= 5, on aura 2+4+6+8+10=30),
- une méthode qui remplit le tableau des coefficients de la décomposition d'un entier **N** en puissances de 10 ($N = 1234 = 4 \times 10^0 + 3 \times 10^1 + 2 \times 10^2 + 1 \times 10^3$),
- une méthode qui indique si un entier **N** est divisible par 3 (en utilisant la règle suivante : "un entier est divisible par 3 lorsque la somme de ses chiffres est divisible par 3").
- une méthode qui affiche **N** lignes d'étoiles selon la figure 1.

Question 2: Ecrivez une méthode **factorielle** qui calcule **n!** en précision infinie, et un programme qui la teste en calculant et affichant la factorielle d'un nombre donné en argument de la ligne de commande. Exemple d'exécution :

```
[port-chaouiya-10:~/COURS/Java] chaouiya% java etoiles 4
*
* *
* * *
* * * *

[port-chaouiya-10:~/COURS/Java] chaouiya% java etoiles 10
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
```

FIG. 1 – afficher un triangle d'étoiles...

```
> java Factorielle 100  
9332621544394415268169923885626670049071596826438162146859296389  
5217599993229915608941463976156518286253697920827223758251185210  
91686400000000000000000000000000  
>
```

Note : Les entiers en précision infinie sont définis dans `java.math.BigInteger` (notez que le paquetage `java.math` n'est pas la classe `java.lang.Math`).

Question 3: Ecrivez une méthode factorielle qui calcule $n!$. On suppose ici que, contrairement à celle de l'exercice précédent, cette méthode est destinée à être appelée de nombreuses fois dans un programme. Ainsi, on se propose de mémoriser les valeurs $k!$ déjà calculées pour minimiser le temps de calcul lors des appels ultérieurs.

Note : Pour la mémorisation des valeurs déjà calculées, explorez la classe `java.util.Vector`.

Question 4: Ecrivez un programme qui tire n nombres au hasard (entre 0 et 1) et calcule la moyenne et l'écart-type de la suite obtenue (vous utiliserez `java.lang.Math.random()` dont vous consulterez la documentation en ligne).

Rappel : la moyenne des termes x_i ($i = 1, \dots, n$) est :

$$m = \frac{\sum_{i=1}^n x_i}{n}$$

et l'écart-type est la racine carrée de la variance :

$$\sigma = \sqrt{\frac{\sum_{i=1}^n x_i^2}{n} - m^2}$$

Notez que la distribution des valeurs étant uniforme, la moyenne doit tendre vers 0.5 et l'écart type vers $\sqrt{1/12} = 0.288675134595$.

Question 5: On appelle nombre de Harshad, un nombre entier divisible par la somme de ses chiffres. Par exemple 12 ($=3*4$), 18, 20,...140, 144, sont de tels nombres. Ecrivez la méthode qui, étant donné un entier n vérifie si n est un nombre de Harshad ou non. Ecrivez un programme d'essai qui indique si les entiers fournis en ligne de commande (en nombre arbitraire) sont des nombre de Hashad ou non. Voici un exemple d'exécution :

```
> java Harshad 12 34 140 165
12 est un nombre de Harshad
34 n'est pas un nombre de Harshad
140 est un nombre de Harshad
165 n'est pas un nombre de Harshad
```

Dans les exercices qui suivent, on illustre essentiellement la notion d'encapsulation, tout en améliorant notre capacité à écrire de nouvelles classes !

Feuille 2

Question 1: [difficile] (une présentation orientée objets de l'inévitable crible d'Erathostène)
Pour obtenir les nombres premiers compris entre 2 et un certain entier N on va construire une liste chaînée d'objets appelés des **MangeNombres**, chacun comportant deux variables d'instance : un nombre premier et une référence sur le *mangenombres* suivant de la liste. Le comportement d'un *mangenombres* se réduit à l'opération manger un nombre : le *mangenombres* m_p associé au nombre p mange les multiples de p : si on lui donne à manger un nombre q qui n'est pas multiple de p , m_p le donne à manger à son *mangenombres* suivant, s'il existe. Si m_p n'a pas de suivant, celui-ci est créé, associé à q . La création d'un *mangenombres* m_p provoque l'affichage de p . Définissez la classe **MangeNombres** et écrivez un programme affichant les nombres premiers entre 2 et N .

Question 2:

A. Définissez une classe **Point** pour représenter les points du plan rapporté à une origine fixée. Les coordonnées d'un point sont ici deux nombres flottants x , y , mémorisés dans deux variables d'instance privées (par exemple de type `double`). La classe **Point** comportera, outre un constructeur recevant en argument les coordonnées (cartésiennes) d'un point, les méthodes d'instance suivantes :

- `double x()`, `double y()` qui renvoient les coordonnées cartésiennes du point,
- `double rho()`, `double theta()`, qui renvoient les coordonnées polaires du point,
- `void homothetie(double k)`, qui applique au point une homothétie de centre $(0; 0)$ et de rapport k ,
- `void translation(double dx, double dy)`, qui applique au point une translation de vecteur (dx, dy) ,
- `void rotation(double a)`, qui applique au point une rotation de centre $(0; 0)$ et d'angle a ,
- `String toString()` qui transforme un point en une chaîne de caractères de la forme " (x,y) ".

B. Une ligne polygonale est définie par une suite de points (les sommets). Ecrivez une classe **LignePol** dont chaque instance représente une ligne polygonale (la suite des sommets étant représentée par un objet `java.util.Vector`), avec

- un constructeur qui construit la ligne vide (aucun sommet),
- une méthode `void ajouter(Point p)` qui ajoute un sommet à la ligne polygonale
- les méthodes `homothetie`, `translation`, `rotation` avec les mêmes arguments que précédemment.

C. Pour tester les lignes polygonales, les points et leurs transformations, écrivez un programme simple qui crée une ligne de 10 sommets aléatoires (utilisez la méthode `java.lang.Math.random()`), l'affiche, lui applique une ou plusieurs transformations et l'affiche à nouveau.

D. Imaginons qu'une enquête effectuée auprès des utilisateurs de notre classe **Point** a montré qu'en pratique l'opération `rotation` est utilisée beaucoup plus souvent que l'opération `translation`. Du point de vue des performances il y a donc intérêt à représenter les points par leurs coordonnées polaires plutôt que cartésiennes. Modifiez la classe **Point** de manière à ce que les variables d'instance soient les coordonnées polaires du point, et non plus ses

coordonnées cartésiennes. Faites en sorte que l'interface de la classe ne change pas, afin que les utilisateurs de la classe `Point` (par exemple, la classe `LignePol`) restent valides.

Question 3: Définissez une classe `Individu` composée d'un `nom`, d'une `adresse`, et d'un numéro de `telephone`. Vous écrirez aussi un constructeur, les méthodes de consultation et modification des attributs, et une méthode `toString()`.

Modifiez la classe `Individu` pour pouvoir :

- afficher le nombre d'individus créés,
- gérer la liste des individus créés,
- afficher la liste des individus créés,

Question 4:

A. Ecrivez une classe `Ensemble` pour représenter des sous-ensembles de l'ensemble des entiers compris entre 0 et N-1, où N est une constante donnée, avec les méthodes suivantes :

- `int cardinal()` : nombre d'éléments de l'ensemble,
- `boolean contient(int i)` : test d'appartenance,
- `void ajout(int i)` : ajout d'un élément à l'ensemble,
- `String toString()` : transformation de l'ensemble en une chaîne de caractères de la forme `{5,6,9,12,32,45}`.

Les éléments d'un ensemble seront mémorisés dans une variable d'instance de type tableau d'entiers. Pour essayer la classe `Ensemble`, écrivez un programme déterminant les nombres différents contenus dans une suite de nombres lue sur l'entrée standard.

B. Ajoutez à la classe `Ensemble` la méthode de classe qui crée l'ensemble intersection de deux ensembles a et b (`Ensemble intersection(Ensemble a, Ensemble b)`).

C. Modifiez la classe précédente pour permettre la représentation d'ensembles d'entiers quelconques, c'est-à-dire pas nécessairement compris entre 0 et une constante N connue à l'avance. Faites en sorte que l'interface de cette classe soit identique à celle de la version précédente pour ne rien avoir à modifier dans les programmes qui utilisent la première version de cette classe.

Note : Pour effectuer les lectures d'entiers demandées vous pouvez utiliser la classe suivante (pour lire un entier et le ranger dans une variable x il faut écrire : `x = Lire.entier('Donne x : ');`) :

```
import java.io.*;
public class Lire {
    static public int entier(String prompt) {
        System.out.print(prompt);
        try {
            BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
            return Integer.parseInt(in.readLine());
        }
        catch (Exception e) {
            System.out.println("Erreur de lecture " + e.getMessage());
            System.exit(-1);
        }
        return 0;
    }
}
```

Question 5:

A. Définissez une classe `PileEnt` qui définit le type *pile d'entiers* en utilisant une liste chaînée, avec un constructeur et les méthodes `empiler` et `depiler`. Pour tester votre pile d'entiers, utilisez la classe suivante :

```
public class TestPile{
    public static void main(String[] arg){
        PileEnt p = new PileEnt();
        for (int i=0;i<10;i++) p.empiler(i);
        System.out.println("On vide la pile : \n");
        for (int i=0;i<10;i++)
            System.out.println(p.depiler());
        }
}
```

B. Définissez une nouvelle implémentation de `PileEnt` en utilisant un tableau et un entier représentant l'indice du sommet de la pile. Testez cette nouvelle version avec la classe `TestPile` inchangée.

Feuille 3

Cette série d'exercices traite essentiellement de l'héritage.

Question 1:

A. Reprenez la classe `Individu` définie dans la feuille 2 (ou téléchargez le corrigé). On veut maintenant définir une classe `Etudiant` héritant de la classe `Individu`. Cette classe aura en plus les attributs `formation` et `numero` qui correspondent respectivement à la formation suivie par l'étudiant (une chaîne de caractères) et son numéro d'inscription (un entier). Vous redéfinirez la méthode `afficher`.

Vous testerez cette nouvelle classe en exécutant la classe `TestEtudiant` suivante (mais avant de l'exécuter, suarez vous prédire ce qui sera affiché?!):

```
class TestEtudiant{
    public static void main(String arg[]){
        Etudiant x=new Etudiant("Dupond","Jean","11 rue du bonheur","0491222222","Philosophie",123);
        Etudiant y=new Etudiant("Java","Jaime","13 Avenue de la programmation","11111111","Informatique");
        Etudiant z=new Etudiant("Tragique","Romeo","3 Boulevard Shakespeare","", "Litterature",666);
        Individu t=new Individu("Stallman","Richard","5 rue du libre","0101010");
        x.afficher();
        Etudiant.afficherTous();
        t.afficher();
        t=x;
        t.afficher();
        y=(Etudiant)t;
        y.afficher();
    }
}
```

Question 2: Pour la gestion d'une bibliothèque on nous demande d'écrire une application manipulant des documents de nature diverse : des livres, des dictionnaires, etc. Tous les documents ont un *numéro d'enregistrement* et un *titre*. Les livres ont, en plus, un *auteur* et un *nombre de pages*, les dictionnaires ont une *langue* et un *nombre d'articles*. Ces diverses sortes de chose doivent pouvoir être manipulées de façon homogène en tant que documents.

1. Définissez les classes `Document`, `Livre`, `Dictionnaire`. Donnez à chacune le constructeur qui prend autant d'arguments qu'il y a de variables d'instance à initialiser, et qui se limite à recopier les valeurs des arguments dans les variables correspondantes.
2. Définissez une classe `Bibliothèque` réduite à une méthode `main` permettant de tester les classes précédentes (ainsi que les suivantes).
3. Définissez la classe `ListeDeDocuments` permettant de créer une liste (vide) de documents, d'y ajouter un élément, de parcourir la liste (positionnement au début, avancée, test de fin du parcours).
4. Dans la classe `ListeDeDocuments` définissez une méthode `tousLesAuteurs()` qui affiche la liste des numéros des documents de la liste avec, pour chacun, l'éventuel auteur.

5. Ajoutez une méthode `description()`, définie dans la classe `Document` et redéfinie dans les classes `Livre` et `Dictionnaire` qui renvoie une chaîne de caractères qui est la “fiche de bibliothèque” d’un document.

Ajoutez alors dans la classe `ListeDeDocuments` une méthode `ttesDescr` qui affiche consécutivement les descriptions de tous les documents.

Question 3:

A. Considérez les interfaces

- `Comparable` contenant `boolean plusGrand(Object o)`. L’idée est qu’un objet d’une classe qui implémente cette méthode pourra être comparé à un autre (et donc la méthode retourne `true` si l’instance courante est plus grande, au sens d’une relation d’ordre bien définie, que l’instance passée en paramètre). La méthode `equals` permet de tester l’égalité de deux objets.

```
interface Comparable{
    boolean plusGrand(Object o);
    boolean equals(Object o);
}
```

- `Affichable`

```
interface Affichable{
    String toString();
}
```

- et `ComparableEtAffichable`

```
ComparableEtAffichable extends Comparable,Affichable{}
```

Ecrivez une classe `ListeTrie` pour représenter des ensembles d’objets triés, en utilisant la classe `Vector`. Elle doit posséder un constructeur sans argument et les méthodes `void ajoute(ComparableEtAffichable)` (ajout d’un élément vous utiliserez la méthode `insertElementAt` de la classe `Vector`, consultez la doc!), `void retire(Comparable)` (qui retire un élément, consultez la documentation de la méthode `remove` de la classe `Vector`, vous devriez y trouver une bonne raison pour avoir déclaré la redéfinition de la méthode `equals` dans `Comparable...`), et une méthode qui permet d’afficher tout le contenu de la liste.

B Reprenez la classe `Point` et complétez-la de façon qu’elle implémente l’interface `ComparableEtAffichable` (on considère que la relation d’ordre sur les points est donnée par l’ordre sur leur abscisse (ainsi le point (2,3) est inférieur au point (3,4)).

Testez maintenant le tout avec la classe `Experience` qui crée une liste des résultats de points de mesures réalisés à des temps réguliers (ces mesures seront gardées dans une `ListeTrie` :

```
public class Experience{

    public static void main(String arg[]){
        ListeTrie mesures=new ListeTrie();
        int t;
        for (t=0;t<=10;t=t+2){
            Point p=new Point(t,t*t);// simulation bidon d’une mesure..
            mesures.ajoute(p);
        }
    }
}
```

```
mesures.affiche();  
//retirer une serie de points  
mesures.retire((Comparable)new Point(2,4));  
mesures.retire((Comparable)new Point(8,64));  
System.out.println("Après le retrait de 2 points...");  
mesures.affiche();  
}  
}
```

Feuille 4

Question 1: Ecrivez une classe avec les méthodes `static` suivantes :

- qui retourne le nombre d’occurrences d’une sous-chaîne dans une chaîne,
- qui retourne une nouvelle chaîne, miroir de la chaîne de caractères passée en paramètre,
- qui modifie, en l’inversant (et sans la recopier) la chaîne de caractères passée en paramètre,
- qui reçoit une chaîne passée en paramètre, contenant un nombre décimal, et la transforme en une chaîne contenant une chaîne avec des espaces entre les milliers. Par exemple “12345678” devient “12 345 678”.
- Complétez la méthode précédente, pour qu’elle lève une exception si l’un des caractères de la chaîne n’est pas un chiffre (vous définirez l’exception `notADigit`).

Ecrivez une méthode `main` qui permette de tester les méthodes définies ci-dessus avec une chaîne passée en ligne de commande.

Question 2:

1. Ecrivez un programme qui prend le nom d’un fichier en argument de la ligne de commande et affiche ses caractéristiques (fichier/répertoire, taille, autorisation lecture (o/n), autorisation écriture (o/n)).
2. Ecrivez un programme qui prend le nom d’un répertoire en argument de la ligne de commande et affiche son contenu.

Question 3:

1. Ecrivez un programme qui prend un fichier texte en argument de la ligne de commande et affiche le nombre de mots dans le fichier.
2. Ecrivez un programme qui prend un fichier texte et un mot en argument de la ligne de commande et affiche le nombre d’occurrences du mot dans le fichier.
3. Ecrivez un programme qui prend un fichier texte en argument de la ligne de commande et affiche son contenu en supprimant tous les espaces, tabulations et retours chariots.

Question 4:

Ecrivez un programme qui copie le contenu d’un fichier dans un autre (les deux noms de fichiers sont passés en argument de la ligne de commande).

Question 5: Un filament d’ADN est constitué de 2 brins comportant 4 bases azotées : l’adénine (A), la cytosine (C), la guanine (G) et la thymine (T). Une portion d’ADN peut donc être représentée par un mot constitué des 4 lettres A,C,G et T.

Ecrivez une classe `BrinADN` avec

- un attribut correspondant à la séquence d’un brin (une chaîne de caractères),
- un constructeur prenant une chaîne en paramètre et qui ne construit un nouvel objet `BrinADN` que si la chaîne est conforme (constituée uniquement des lettres ACGT). Ce constructeur lancera une exception du type `SequenceException` (défini ci-dessous).

```

class SequenceException extends Exception {
    static String message=new String("la sequence contient au moins
                                   une lettre non conforme");

    String seq;
    public SequenceException(String s){
        seq=s;
    }
    public String toString(){
        return new String(message+" :\n\t"+seq+"\n");
    }
}

```

Ajoutez à la classe **BrinADN**

- une méthode qui retourne la longueur de la séquence,
- une méthode **toString**,
- une méthode qui permet de vérifier qu'une séquence contient une sous-séquence donnée,
- une méthode qui, étant donnés un caractère **c** et un entier **n**, insère la nouvelle lettre (à condition qu'elle soit conforme) à la position **n** de la séquence. Si **n** est supérieur à la taille de la séquence on pourra considérer que l'insertion se fait à la fin de la séquence,
- une méthode qui permet de déterminer le pourcentage d'adénine de la séquence.

Ecrivez une méthode **main** de façon que le programme prenne une séquence en ligne de commande et affiche sa longueur, et le pourcentage d'adénine qu'elle contient. Voici un exemple d'exécution du programme (en fonction de la commande fournie)

```

> java BrinADN
donner une sequence en ligne de commande !!!
> java BrinADN ATTCGCCCAFTTT
la sequence contient au moins une lettre non conforme !!
> java BrinADN ATTCGCCCATTT
ATTCGCCCATTT de longueur 13 contient 15.384616% d'adenine
> java BrinADN ATATATATAT
ATATATATAT de longueur 10 contient 50.0% d'adenine

```

Ecrivez un constructeur qui prend en paramètre le nom d'un fichier texte dans lequel est stockée une séquence. Comme précédemment, le constructeur lancera une exception si la séquence n'est pas conforme. Modifiez la méthode **main** de façon à lire la séquence dans un fichier dont le nom est fourni en ligne de commande et qui indique la longueur de la séquence et son pourcentage en adénine.

Feuille 5

Question 1: Ecrire une petite application qui ouvre une fenêtre avec 2 boutons et quand l'utilisateur *clique* sur le bouton "Rouge", le fond devient rouge, quand il *clique* sur le bouton "Vert", le fond devient vert (cf. fichier en ligne).

Question 2: Ecrivez une interface qui affiche un nouveau carré de couleur aléatoire chaque fois que l'on clique sur la souris. Vous pourrez envisager plusieurs comportements :

1. où on se contente d'afficher des petits carrés de couleur aléatoire, sans les déplacer, 2. où on affiche des petits carrés de couleur aléatoire, avec la possibilité de les déplacer

Rappel/indication : pour tirer une couleur au hasard, il faut utiliser le générateur pseudo-aléatoire. Pour cela, instancier un objet de classe Random (paquetage java.util), puis "tirer" le prochain entier (qu'il faut ramener entre 0 et 255) :

```
Random tirage=new Random();
....
Color col=new Color(Math.abs(tirage.nextInt())%256,
    Math.abs(tirage.nextInt())%256,
    Math.abs(tirage.nextInt())%256);
```

Question 3: On vous demande de réaliser une application pour compter (des étoiles, des passagers embarqués, des bactéries, etc.). Cela se présente (voir la figure 1) comme un panneau comportant un titre, un nombre entier et un bouton. Chaque fois que l'utilisateur appuie sur le bouton, le nombre augmente de une unité.

A. Pour commencer, réalisez un programme très minimaliste : une classe Compteur avec la méthode main et deux variables statiques (la valeur du nombre et le JLabel chargé de son affichage).

Il vous faudra aussi une classe auxiliaire AuditeurBouton pour représenter l'objet qui détecte et dispatche les pressions sur le bouton ; faites-en une classe interne à Compteur, ce qui lui permettra d'accéder aux variables valeur et affichage (Pour des raisons techniques - mais compréhensibles - cette classe devra elle aussi être qualifiée static).

B. [Légère amélioration du code] Remplacez la classe interne AuditeurBouton par une classe anonyme.

C. [Légère amélioration de l'aspect] Faites en sorte que le bouton « ++ » ait sa largeur préférée (voyez la figure 2), au lieu d'occuper toute la largeur du cadre. Pour cela, intercalez un panneau entre le bouton et le cadre.

D. [Grosse amélioration du code] Faites les choses comme il faut les faire : réorganisez le code précédent afin de définir une classe Compteur, sous-classe de JPanel. Elle est munie d'un constructeur prenant le titre pour argument, et chacune de ses instances représente un panneau supportant un triplet (titre, nombre affiché, bouton).

Pour essayer cette classe, écrivez une méthode main (soit dans la classe Compteur, soit dans une autre classe définie à cet effet) qui crée un cadre et y place un compteur.

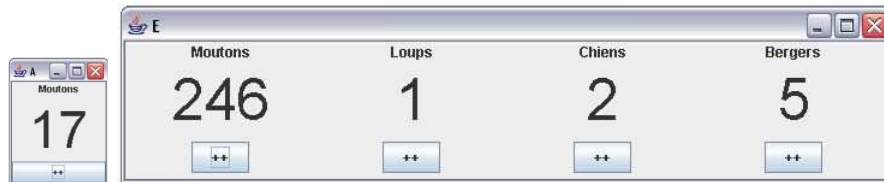


FIG. 2 – Une histoire de compteurs..

E. Pour vous convaincre du bien fondé de la classe précédente, modifiez la méthode `main` précédente afin qu'elle crée un cadre avec, par exemple, quatre compteurs indépendants :

Question 4: Reprenez le dernier exercice de la feuille précédente en développant une interface qui permette à l'utilisateur de rentrer sa séquence dans un champ texte, de *cliquer* sur un bouton "OK" et de voir apparaître le pourcentage d'adénine, de guanine, cytosine, et thymine. Puis,

- définissez une classe `CaractereIllegalException`, dérivée de la classe `Exception`, qui est l'exception levée en cas d'erreur (une lettre non conforme). Dans ce cas, un message d'erreur apparaîtra dans une fenêtre de dialogue (utiliser la méthode `showMessageDialog` de la classe `JOptionPane`).
- rajoutez la possibilité de lire la séquence dans un fichier texte dont l'utilisateur donnera le nom (de même que précédemment, si le fichier n'est pas trouvé, l'exception doit être attrapée et une fenêtre de dialogue ouverte avec un message d'erreur adapté).
- rajoutez la possibilité de déterminer l'inverse complémentaire d'un brin d'ADN (l'adénine est complémentaire à la thymine et la guanine est complémentaire à la cytosine).
- rajoutez la possibilité de chercher un motif spécifié par une expression régulière (voir le paquetage `java.util.regex`).