

Exercice I (Utilisation d'une API tierce)

Les classes `up.mi.jgm.maths.Rationnel` et `up.mi.jgm.maths.Complexe` sont disponibles sur Moodle, dans l'archive Jar nommée `maths.jar`. La documentation de (la partie publique de) ces classes est donnée ci-dessous.

```
public class Rationnel {
    /**
     * Cree un Rationnel de numerateur et denominateur donnees
     *
     * @param num le numerateur
     * @param den le denominateur
     */
    public Rationnel(long num, long den) { ... }

    /**
     * Cree un Rationnel egal a un entier donne
     *
     * @param num le numerateur
     */
    public Rationnel(long num) { ... }

    /**
     * Cree un Rationnel, somme de ce Rationnel et d'un Rationnel donne
     *
     * @param r le rationnel qu'on ajoute a l'objet courant
     * @return un nouveau objet representant la somme de ce rationnel par r
     */
    public Rationnel addition(Rationnel r) { ... }

    /**
     * Cree un Rationnel, produit de ce Rationnel et d'un Rationnel donne
     *
     * @param r le ratinnel par lequel on multiplie l'objet courant
     * @return un nouveau objet representant le produit de ce rationnel par r
     */
    public Rationnel multiplication(Rationnel r) { ... }

    @Override
    public String toString() { ... }
}

public class Complexe {
    /**
     * Construit un nombre complexe a partie de ses deux parties
     *
     * @param partieReelle la partie reelle du nombre
     * @param partieImaginaire la partie imaginaire du nombre
     */
}
```

```

    */
    public Complexe(double partieReelle, double partieImaginaire) { ... }

    /**
     * Permet d'obtenir la partie réelle du nombre
     *
     * @return la partie réelle du nombre
     */
    public double getPartieReelle() { ... }

    /**
     * Permet d'obtenir la partie imaginaire du nombre
     *
     * @return la partie imaginaire du nombre
     */
    public double getPartieImaginaire() { ... }

    @Override
    public String toString() { ... }
}

```

1. Dans une classe utilitaire, définissez une méthode statique qui calcule la puissance n -ème d'un rationnel.
2. Dans la même classe utilitaire, définissez :
 - (a) une méthode statique qui fait la somme de deux nombres complexes.¹
 - (b) une méthode statique qui fait le produit de deux complexes.²

Exercice II (Géométrie et POO)

On souhaite manipuler des objets du plan.

1. (a) Créez une classe Point qui modélise un point du plan, c'est-à-dire une entité composée d'une abscisse et d'une ordonnée.
- (b) Créez une méthode de la classe Point qui prend en entrée un autre Point, et retourne la distance entre les deux points.³
2. (a) Dans un plan, on peut représenter un disque avec deux informations : les coordonnées de son centre d'une part, et son rayon d'autre part. Créez la classe Disque correspondante.
- (b) Un point du plan appartient à un disque si la distance entre ce point et le centre du disque est inférieure ou égale au rayon du disque. Définissez une méthode qui permet de le vérifier.
- (c) Définissez une méthode de la classe Disque qui vérifie si deux disques sont en intersection.
3. (a) On peut représenter un vecteur \overrightarrow{AB} par les deux points A et B qui le composent. Définissez une classe vecteur pour cela.

1. La somme de deux nombres complexes $a + i \times b$ et $c + i \times d$ est $(a + c) + i \times (b + d)$.

2. Le produit de deux nombres complexes $a + i \times b$ et $c + i \times d$ est $(a \times c - b \times d) + i \times (a \times d + b \times c)$.

3. La distance entre deux points (x_a, y_a) et (x_b, y_b) est $\sqrt{(x_b - x_a)^2 + (y_b - y_a)^2}$.

- (b) Deux vecteurs sont égaux si la différence entre les coordonnées de leurs extrémités sont égales. Définissez une méthode `equals()` qui vérifie cela.⁴
- (c) Une translation consiste à déplacer un point dans la direction et la distance représentées par un vecteur. Ajoutez à la classe `Point` une méthode qui prend en entrée un vecteur, et qui retourne un nouveau `Point` correspondant à la translation de l'objet initial par ce vecteur.
- (d) La translation d'un disque par un vecteur consiste en un nouveau disque dont on a déplacé le centre, mais en conservant son rayon. Définissez une méthode dans la classe `Disque` qui effectue la translation du disque par le vecteur donné en paramètre.

Exercice III (Manipulation de listes et généricité)

1. Définissez une méthode qui retourne l'élément maximal d'une liste d'`Integer`.
2. Définissez une méthode qui calcule la moyenne des éléments d'une liste d'`Integer`.
3. Définissez une méthode qui trie une liste d'`Integer`.⁵
4. L'interface `java.lang.Comparable<T>` définit une unique méthode **public int compareTo(T o)** qui sert à comparer deux objets. Si celui qui appelle la méthode est considéré comme plus petit que `o`, alors la méthode retourne un nombre négatif, s'il est plus grand un nombre positif. Enfin, la méthode retourne 0 dans le cas où les deux objets sont considérés comme identiques. C'est ce qui permet, par exemple, de comparer des `String` selon l'ordre lexicographique.
 - (a) Définissez une méthode qui retourne l'élément maximal d'une liste d'objets grâce à l'interface `Comparable`.
 - (b) Définissez une méthode qui trie une liste d'objets grâce à l'interface `Comparable`.

Exercice IV (Promotions d'étudiants)

On souhaite développer un programme qui permet de gérer une promotion d'étudiants. Il doit être possible d'ajouter des étudiants dans la promotion, ainsi que les unités d'enseignement suivies.⁶ Pour chaque unité d'enseignement, chaque étudiant a deux notes : un contrôle continu (CC), et un examen (EX). La note finale d'un étudiant dans une unité d'enseignement est $\max(EX, \frac{CC+EX}{2})$. Écrivez un programme qui propose à l'utilisateur les fonctionnalités suivantes via une interface textuelle (entrée au clavier et affichage sur la console) :

- Ajout d'un étudiant dans la promotion ;
- Ajout d'une note (de contrôle continu ou d'examen) pour un étudiant particulier ;
- Afficher l'ensemble des étudiants de la promotion, avec leur moyenne par unité d'enseignement ;
- Calcul de la moyenne globale de la promotion ;
- Calcul de la moyenne par unité d'enseignement ;
- Recherche du major de promotion (pour simplifier, nous considérons qu'il n'y a qu'un major, en cas d'ex-aequo un seul est affiché) ;
- Recherche du major par unité d'enseignement (même remarque sur les éventuels ex-aequo).

4. Formellement, $\overrightarrow{AB} = \overrightarrow{CD}$ si et seulement si $x_B - x_A = x_D - x_C$ d'une part, et $y_B - y_A = y_D - y_C$ d'autre part.

5. Ne pas utiliser de méthode de tri fournie par l'API standard de Java.

6. Pour simplifier, on suppose que tous les étudiants de la promotion suivent exactement les mêmes unités, et que les unités ont le même coefficient.

Exercice V (Répertoires)

On veut implémenter une classe `RepertoireSimple` qui permet de créer des répertoires téléphoniques simples, permettant :

- l'enregistrement d'une personne identifiée par son prénom, son nom et son numéro de téléphone,
- et la recherche d'un numéro de téléphone en connaissant l'identité de la personne.

La classe `RepertoireSimple` est utilisée dans la classe suivante :

```
public class TestRepertoire {  
    public static void main(String[] args) {  
        RepertoireSimple rep = new RepertoireSimple();  
        rep.addPersonne("John", "Lennon", "0123456789");  
        rep.addPersonne("Paul", "McCartney", "0234567891");  
        rep.addPersonne("George", "Harrison", "0345678912");  
        rep.addPersonne("Ringo", "Starr", "0456789123");  
  
        System.out.println(rep.chercheNumero("John", "Lennon"));  
        System.out.println(rep.chercheNumero("Paul", "McCartney"));  
        System.out.println(rep.chercheNumero("Freddie", "Mercury"));  
    }  
}
```

L'exécution de ce programme donne le résultat suivant :



```
<terminated> TestRepertoire [Java Application] /Library/Java/Java  
0123456789  
0234567891  
L'identite Freddie Mercury est inconnue.
```

1. Dédurre du code précédent les signatures des méthodes de la classe `RepertoireSimple`.
2. Implémenter cette classe.

Exercice VI (HashMap et T9)

Un peu d'histoire (ou de préhistoire) de la téléphonie mobile : le T9 (*Text on 9 keys*) utilisé pour taper des mots sur un clavier de téléphone repose sur l'association de chaque chiffre du clavier à plusieurs lettres (2 correspond à a, b et c, 3 correspond à d, e et f,...). Lorsque l'on tape sur une suite de touche, la série de chiffres peut être associée à un mot correspondant.



Avec le clavier ci-contre, une pression sur les touches 2665687 peut être associée au mot "bonjour" (le 2 est transformé en b, le premier 6 en o, le deuxième 6 en n,...). Cependant, certaines suites de chiffres peuvent correspondre à plusieurs mots : 26663 correspond à "bonne" (2 → b, 6 → o, 6 → n, 6 → n, 3 → e) ou à "comme" (2 → c, 6 → o, 6 → m, 6 → m, 3 → e). Il faut donc mémoriser les différentes possibilités, et les proposer à l'utilisateur lorsque la situation se présente.

On fournit une méthode statique qui renvoie le chiffre T9 associé à un caractère :

```
public static byte getChiffreT9(char c){
    switch (Character.toLowerCase(c)){
        case 'a':
        case 'b':
        case 'c':
            return 2;
        case 'd':
        case 'e':
        case 'f':
            return 3;
        case 'g':
        case 'h':
        case 'i':
            return 4;
        case 'j':
        case 'k':
        case 'l':
            return 5;
        case 'm':
        case 'n':
        case 'o':
            return 6;
        case 'p':
        case 'q':
        case 'r':
        case 's':
            return 7;
        case 't':
        case 'u':
        case 'v':
            return 8;
        case 'w':
        case 'x':
        case 'y':
        case 'z':
            return 9;
        default:
            return 0;
    }
}
```

1. Définissez une méthode statique qui prend en entrée un mot, et qui retourne la chaîne de caractères T9 correspondante (par exemple, le mot "bonjour" est codé en T9 avec "2665687").

On souhaite maintenant créer un dictionnaire de chaînes T9 grâce à une `HashMap<String, ArrayList<String>>`, dont les clés sont des chaînes codées en T9, et les valeurs sont les listes de mots correspondant à ces clés. Par exemple, à la clé "26663" correspond la liste qui contient les mots "bonne" et "comme".

2. Définissez une méthode statique qui prend en entrée une `HashMap<String, ArrayList<String>>` et un `String` et qui place le `String` dans la table de hachage :

```
public static void enregistrer(HashMap<String, ArrayList<String>> dico,
                               String chaine) {
    // ...
}
```

3. Définissez une méthode qui permet d'obtenir la liste des chaînes de caractères correspondant à une chaîne codée en T9 :

```
public static ArrayList recuperer(HashMap<String, ArrayList<String>> dico,
                                   String chaineT9){
    // ...
}
```

Vous pouvez tester les méthodes précédentes avec ce programme :

```
public class TestDicoT9 {
    public static void main(String[] args) {
```

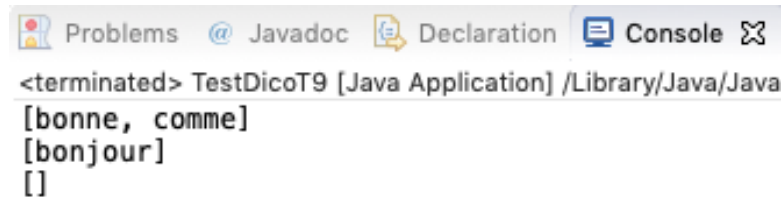
```

HashMap<String , ArrayList<String>> dico = new HashMap<String ,
                                                ArrayList<String>>();

enregistrer(dico , "bonjour");
enregistrer(dico , "bonne");
enregistrer(dico , "comme");

System.out.println(recuperer(dico , "26663"));
System.out.println(recuperer(dico , "2665687"));
System.out.println(recuperer(dico , "123456"));
}
}

```



```

<terminated> TestDicoT9 [Java Application] /Library/Java/Java
[bonne, comme]
[bonjour]
[]

```

Exercice VII (Jeu de cartes)

On a vu dans le cours comment utiliser des Enum pour représenter les couleurs dans un jeu de cartes.

1. Définissez une classe Carte qui permet de représenter une carte à jouer.⁷
2. Les cartes qu'un joueur possède sont appelées sa main. Créez une classe qui permet de représenter la main d'un joueur, d'y ajouter une carte ou d'en retirer une carte.
3. On souhaite que l'un joueur puisse trier sa main en utilisant la méthode de tri générique de l'exercice III. Modifiez la classe Carte pour vous en assurer.⁸

Exercice VIII (Jeu de cartes... amélioré)

On souhaite améliorer la modélisation du jeu de cartes proposée dans l'exercice précédent. Il faut maintenant pouvoir trier les cartes différemment, selon les règles du jeu auquel on joue.

1. Définissez une classe abstraite CarteAbstraite qui permet de représenter les informations de base d'une carte (couleur et valeur).
2. Une classe fille de CarteAbstraite doit correspondre aux cartes associées au système de classement d'un jeu particulier. Par exemple :
 - (a) La classe CarteBasique doit correspondre à un tri simple des cartes, comme vu à l'exercice VII.
 - (b) La classe CarteManille permet de trier les cartes comme à la manille.⁹
3. Assurez vous qu'il est possible pour un joueur de trier les cartes de sa main, dans le cas basique et dans le cas de la manille.

7. On utilisera les cartes d'un jeu classique : coeur, carreau, trèfle et pique pour les couleurs, et les nombres de 1 à 10, ainsi que valet, dame et roi pour les valeurs.

8. On utilisera un ordre « naturel » : si deux cartes sont de la même couleur, on les compare suivant leurs valeurs (l'as est le plus petit, puis 2, puis 3, ..., puis le roi qui a la plus grande valeur), et si elles ont une couleur différente, le coeur est plus petit, puis carreau, puis trèfle, puis pique.

9. Voir l'ordre des cartes : [https://fr.wikipedia.org/wiki/Manille_\(jeu\)](https://fr.wikipedia.org/wiki/Manille_(jeu))

Exercice IX (POO et chimie)

Un atome est composé d'un noyau, qui se décompose en protons et neutrons, autour duquel tournent des électrons. Le nombre de protons est égal au nombre d'électrons (Z), mais le nombre de neutrons (N) peut être différent. Les atomes ont un nom, ainsi qu'une abbréviation officielle. Une molécule peut être vue comme un groupement d'atomes. On peut représenter une molécule sous la forme $X_n Y_m \dots Z_l$ où X, Y, \dots, Z sont les atomes qui la composent, et n, m, \dots, l sont les quantités de chaque atome.

1. Créez une classe qui permet de représenter un atome, et d'afficher les informations qui le caractérisent.
2. Créez une classe qui permet de représenter une molécule, et d'afficher les informations qui la caractérisent.
3. Définissez une méthode qui permet de déterminer le nombre de protons dans une molécule.

Vos classes doivent être utilisables avec le code suivant :

```
public class TestChimie {  
    public static void main(String[] args){  
        Atome hydro = new Atome(0,1,"Hydrogene", "H");  
        Atome oxy = new Atome(8,8,"Oxygene", "O");  
        System.out.println(hydro);  
        System.out.println(oxy);  
        Molecule eau = new Molecule();  
        eau.addAtome(hydro, 2);  
        eau.addAtome(oxy);  
        System.out.println(eau);  
        System.out.println(eau.getNbProtons());  
    }  
}
```