Programmation avancée et application • TD 2

Bases de la POO

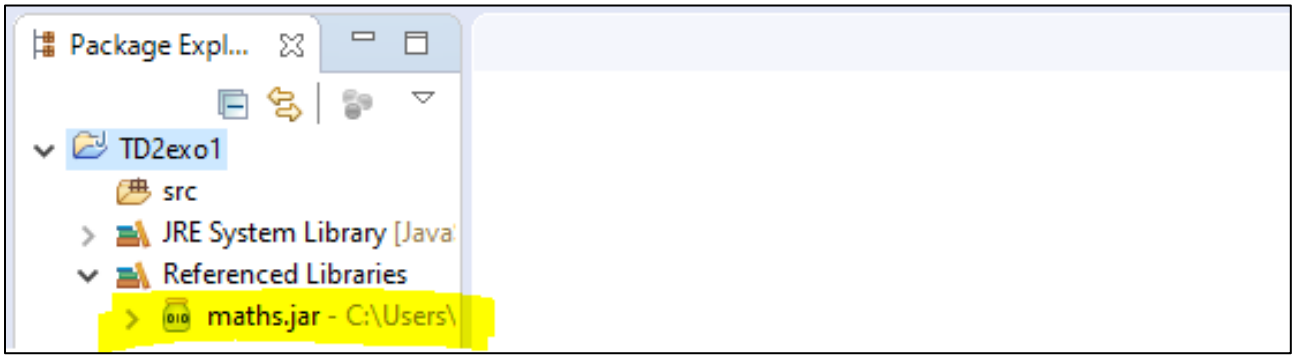
premières classes

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.

To import jar file in your Eclipse IDE, follow the steps given below.

1. Right click on your project
2. Select Build Path
3. Click on Configure Build Path
4. Click on Libraries and select Add External JARs
5. Select the jar file from the required folder
6. Click and Apply and Ok

Source : https://www.edureka.co/community/4028/how-to-import-a-jar-file-in-eclipse



1. Dans une classe utilitaire, définissez une méthode statique qui calcule la puissance n-ème d'un rationnel.

```java
import up.mi.jgm.maths.Rationnel;
// Un nombre rationnel est un nombre qui peut s'exprimer comme le quotient de deux entiers relatifs.
// Un entier relatif est un nombre qui se présente comme un entier naturel auquel on a adjoint un signe positif ou négatif.

public class UtilMath {

	public static Rationnel puissance(int n, Rationnel x) {

		Rationnel result = new Rationnel( 1 );

		for(int i = 0 ; i < n ; i++)
			x.multiplication( x );

		return result;
	}
}
```

2. Dans la même classe utilitaire, définissez :

   (a) une méthode statique qui fait la somme de deux nombres complexes. [1]

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

```java
import up.mi.jgm.maths.Rationnel;
import up.mi.jgm.maths.Complexe;

public class UtilMath {

	public static Complexe sommeComplexes(Complexe c1, Complexe c2) {
		/* Somme de deux nombres complexes a + i*b et c + i*d
		 * (a + c) + i*(b + d)
		 */

		/* Nombre complexe
		 * Un nombre complexe z se présente en général sous forme algébrique
		 * comme une somme a + ib, où a et b sont des nombres réels quelconques
		 * Le réel a est appelé partie réelle de z  , le réel b est sa partie imaginaire
		 */

		double a = c1.getPartieReelle();
		double c = c2.getPartieReelle();

		double b = c2.getPartieImaginaire();
		double d = c2.getPartieImaginaire();

// public Complexe (double p a r t i e R e e l l e  ,double p a r t i e I m a g i n a i r e )
		return( new Complexe( (a + c), (b + d) ) );
	}
}
```

(b) une méthode statique qui fait le produit de deux complexes. [2]

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

```
        public static Complexe produitComplexes(Complexe c1, Complexe c2) {
                /* Produit de deux nombres complexes  a + i*b et c + i*d
                 * (a*c - b*d) + i*(a*d + b*c)
                 */

                /* Nombre complexe
                 * Un nombre complexe z se présente en général sous forme algébrique
                 * comme une somme a + ib, où a et b sont des nombres réels quelconques
                 * Le réel a est appelé partie réelle de z  , le réel b est sa partie imaginaire
                 */

                double a = c1.getPartieReelle();
                double c = c2.getPartieReelle();

                double b = c2.getPartieImaginaire();
                double d = c2.getPartieImaginaire();

// public Complexe (double p a r t i e R e e l l e  ,double p a r t i e I m a g i n a i r e )
                return( new Complexe( (a*c - b*d), (a*d + b*c) ) );
        }
```

```java
import up.mi.jgm.maths.Rationnel;
import up.mi.jgm.maths.Complexe;
import java.util.Scanner;

public class TestUtilMath {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);

		int n;
		long numerateur, denominateur;
		Rationnel x;

		Double a1, b1, a2, b2;
		Complexe c1, c2;

		// Un nombre rationnel est un nombre qui peut s'exprimer comme le quotient de deux entiers relatifs.
		// Un entier relatif est un nombre qui se présente comme un entier naturel auquel on a adjoint un signe positif ou négatif.

		/* public Rationnel (long num,long den )
		 *   num = le numerateur,  den le denominateur
		 */
		System.out.println("TEST fonction puissance : x^n");
		System.out.print("n  = ? ");
		n = sc.nextInt();
		System.out.println("x - Un nombre rationnel - quotient de deux entiers relatifs");
		System.out.print("x - numerateur ?  = ");
		numerateur = sc.nextLong();
		System.out.print("x - denominateur ?  = ");
		denominateur = sc.nextLong();
		x = new Rationnel(numerateur, denominateur); // public Rationnel (long num,long den )

		// public static Rationnel puissance(int n, Rationnel x)
		System.out.println("RESULTAT fonction puissance : " + UtilMath.puissance(n,x) );



		System.out.println("TEST nombres complexes  : z = a + i * b");
		System.out.println("Le réel a = partie réelle de z  , réel b = partie imaginaire");
		System.out.println("**** nombre complexe 1 ****");
		System.out.print("a  = ? ");
		a1 = sc.nextDouble();
		System.out.print("b  = ? ");
		b1 = sc.nextDouble();
		System.out.println("**** nombre complexe 2 ****");
		System.out.print("a  = ? ");
		a2 = sc.nextDouble();
		System.out.print("b  = ? ");
		b2 = sc.nextDouble();

		sc.close(); // Toujours fermer le scanner avec  close() apres la fin de son utilisation

		// public Complexe (double p a r t i e R e e l l e  ,double p a r t i e I m a g i n a i r e )
		c1 = new Complexe(a1, b1);
		c2 = new Complexe(a2, b2);


		// public static Complexe sommeComplexes(Complexe c1, Complexe c2)
		// public static Complexe produitComplexes(Complexe c1, Complexe c2)
		System.out.println("RESULTAT fonction sommeComplexes : " + UtilMath.sommeComplexes(c1,c2) );
		System.out.println("RESULTAT fonction produitComplexes : " + UtilMath.produitComplexes(c1,c2) );

	}
}
```

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.

```java
public class Point {
    private int x; // axe des abscisses
    private int y; // axe des ordonnées

}
```

(b) Créez une méthode de la classe Point qui prend en entrée un autre Point, et retourne la distance entre les deux points. [3]

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

```java
    public double distance(Point p2) {
        /* La distance entre deux points (Xa, Ya) et (Xb, Yb)
         * est : Math.sqrt( (Xb-Xa)^2 + (Yb-Ya)^2 );
         */

        // public static double pow(double a, double b)
        // Returns the value of the first argument raised to the power of the second argument
        double resultX = Math.pow( (p2.x - this.x)  , 2);
        double resultY = Math.pow( (p2.y - this.y)  , 2);

        // public static double sqrt(double a)
        // Returns the correctly rounded positive square root of a double value
        return ( Math.sqrt(resultX + resultY) );
    }
```

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.

```
public class Disque {
	private Point coordonneesDuCentre;
	private double rayon;

	/* Un rayon d'un cercle est un segment de droite quelconque reliant son centre à sa circonférence.
	 * Le rayon est la moitié du diamètre.
	 */

}
```

(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.

```
	public boolean pointAppartientDisque(Point p) {
		// double Point.distance(Point p2)
		double distance = p.distance( this.coordonneesDuCentre );
		boolean result = false;

		if(distance <= rayon)
			result = true;

		return result;
	}
```

(c) Définissez une méthode de la classe Disque qui vérifie si deux disques sont en intersection.

```
	public boolean disquesEnIntersection(Disque d2) {
		return ( d2.pointAppartientDisque( coordonneesDuCentre ) );
	}
```

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.

```java
public class Vecteur {
	private Point p1;
	private Point p2;
}
```

(b) Deux vecteurs sont égaux si la différence entre les coordonnées de leurs extrémités sont égales. Définissez une méthode equals() qui vérifie cela. [1]

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.

```java
public class Point {
	private int x; // axe des abscisses
	private int y; // axe des ordonnées

	public int getX() {
		return x;
	}

	public int getY() {
		return y;
	}

}
```

```java
public class Vecteur {
	public boolean equals(Vecteur v2) {
		/* vecteur AB == vecteur CD
		 * si et seulement si xB-xA=xD-xC dune part
		 * et yB-yA=yD-yC dautre part
		 */
		boolean result = true;

		if ( ( this.p2.getX() - this.p1.getX() ) != ( v2.p2.getX() - v2.p1.getX() ) )
			result = false;

		if ( ( this.p2.getY() - this.p1.getY() ) != ( v2.p2.getY() - v2.p1.getY() ) )
			result = false;

		return result;
	}
}
```

(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.

```java
public class Vecteur {
	Point getP1() {
		return p1;
	}

	Point getP2() {
		return p2;
	}
}
```

```java
public class Point {
	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}

	public Point translation(Vecteur v) {
		/* TRANSLATION ET VECTEURS
		 * https://www.maths-et-tiques.fr/telech/7_Translation_vecteurs.pdf
		 */

		int distanceX =   v.getP2().getX() -  v.getP1().getX();
		int nouveauX = x + distanceX;

		int distanceY =   v.getP2().getY() -  v.getP1().getY();
		int nouveauY = y + distanceY;

		Point nouveauPoint = new Point(nouveauX, nouveauY);
		return nouveauPoint;
	}
}
```

(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.

```java
public class Disque {
	public Disque(Point coordonneesDuCentre, double rayon) {
		this.coordonneesDuCentre = coordonneesDuCentre;
		this.rayon = rayon;
	}

	public Disque translation(Vecteur v) {

		return( new Disque(coordonneesDuCentre.translation(v), rayon)  );
	}
}
```

```java
public class Vecteur {
	public Vecteur(Point p1, Point p2) {
		this.p1 = p1;
		this.p2 = p2;
	}
}
```

```java
import java.util.Scanner;
public class TestExo2 {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int x, y;

		Point p;
		Point pVecteur1 = new Point(0, 3);
		Point pVecteur2 = new Point(0, 7);
		Point apresTranslation;

		Vecteur v;

		System.out.print("x = ? ");
		x = sc.nextInt();
		System.out.print("y = ? ");
		y = sc.nextInt();
		p = new Point(x, y);

		sc.close(); // Toujours fermer le scanner avec close() apres la fin de son utilisation.

		v = new Vecteur( pVecteur1, pVecteur2 );
		System.out.println("Distance vecteur : " + pVecteur2.distance( pVecteur1 ) );

		apresTranslation = p.translation(v);
		System.out.println("Distance apres translation : " + p.distance(apresTranslation) );
		System.out.println("x apres translation : " + apresTranslation.getX() );
		System.out.println("y apres translation : " + apresTranslation.getY() );
	}
}
```

```
x = ? 3
y = ? 0
Distance vecteur : 4.0
Distance apres translation : 4.0
x apres translation : 3
y apres translation : 4
```

Exercice III (Manipulation de listes et généricité)

1. Définissez une méthode qui retourne l'élément maximal d'une liste d'Integer.

```
import java.util.List;

public class UtilListe{

	public static Integer maxList(List<Integer> l) {
		Integer max = null;

		if( !l.isEmpty() ) {
			/* E get(int index)
			 *	Returns the element at the specified position in this list.
			 *	Parameters : index - index of the element to return
			 *	Returns : the element at the specified position in this list
			 */
			max = l.get( 0 );

			for(int i = 0 ; i < l.size() ; i++) {
				if( l.get(i) > max )
					max = l.get(i);
			}
		} // if
		return max;
	}
}
```

2. Définissez une méthode qui calcule la moyenne des éléments d'une liste d'Integer.

```
	public static double moyenneList(List<Integer> l) {
		double moyenne = -1;

		if( !l.isEmpty() ) {
			double sum = 0;

			for(int i = 0 ; i < l.size() ; i++)
				sum += l.get(i);

			moyenne = sum / l.size();
		} // if
		return moyenne;
	}
```

3. Définissez une méthode qui trie une liste d'Integer. [5]

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

Algorithme de tri par sélection

Cours Gael Mahe / L2

Algorithme : Tri par sélection

début

Soit $V : [\![1, n]\!] \to E$ un vecteur non trié. On veut trier V.

```
/* ENTRÉES: Un vecteur V de taille n */
/* SORTIE: Le vecteur V trié */
pour i de 1 à n − 1 faire
    min ← i
    pour j de i + 1 à n faire
        si V(j) < V(min) alors min ← j
    si min ≠ i alors Echange(V, i, min)
    retourner V
```

Parcourir le vecteur : pour tout $i \in [\![1, n-1]\!]$

On cherche l'indice de l'élement minimum de V pour les indices $\{i, \ldots, n\}$

On échange $V(i)$ et $V(min)$

fin

0	1	2	3	4
12	9	3	43	18
12	9	3	43	18
3	9	12	43	18
3	9	12	43	18
3	9	12	43	18
3	9	12	43	18
3	9	12	43	18
3	9	12	18	43

Tri par sélection d'un tableau (1/2)

```
public static void triParSelection(double[] tab) {
  for(int i = 0 ; i < tab.length - 1 ; i++) {
    int indiceMin = rechercheIndicePlusPetit(tab, i);
    if(indiceMin != i) {
      echanger(tab,i, indiceMin);
    }
  }
}
```

Tri par sélection d'un tableau (2/2)

```
private static int rechercheIndicePlusPetit(
                    double[] tab, int indiceMin) {
  for(int j = indiceMin + 1 ; j < tab.length ; j++) {
    if(tab[j] < tab[indiceMin]) {
      indiceMin = j ;
    }
  }
  return indiceMin ;
}

private static void echanger(double[] tab,
                             int i, int j) {
  double tmpVal = tab[i];
  tab[i] = tab[j];
  tab[j] = tmpVal ;
}
```

111

```java
	public static void triParSelection(List<Integer> l) {
		for(int i = 0 ; i < l.size() - 1 ; i++) {
			int indiceMin = rechercheIndicePlusPetit(l, i);
			if(indiceMin != i)  echanger(l, i, indiceMin);
		}

	}

	private static int rechercheIndicePlusPetit(List<Integer> l, int indiceMin) {
		for(int j = indiceMin + 1 ; j < l.size() ; j++) {
			if( l.get(j) < l.get(indiceMin) ) indiceMin = j;
		}
		return indiceMin;
	}

	private static void echanger(List<Integer> l, int i, int j) {
		Integer tmpVal = l.get(i);
		l.set(i, l.get(j) );
		l.set(j, tmpVal);
	}
```

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 lexicograhpique.

(a) Définissez une méthode qui retourne l'élément maximal d'une liste d'objets grâce à l'interface Comparable.

```java
	/* Types bornes : borne superieure T extends A>
	 * On peut indiquer que les elements de la List doivent appartenir a un sous-type de A
	 * On utilise le mot-clé extends
	 * que A soit une classe ou une interface
	 */
	public static <T extends Comparable<T>> T maxListComparable(List<T> l) {
		T max = null;

		if(!l.isEmpty()) {
			max = l.get( 0 );

			/* int java.lang.Comparable.compareTo(T o)
			 * int compareTo(T o)
			 * Compares this object with the specified object for order.
			 * Returns a negative integer, zero, or a positive integer as this object is less than,
equal to, or greater than the specified object.
			 */

			for(T element : l) {
				if( element.compareTo( max ) > 0)
					max = element;
			} // for
		} // if
		return max;
	}
```

(b) Définissez une méthode qui trie une liste d'objets grâce à l'interface Comparable.

```
        public static <T extends Comparable<T>> void triParSelectionComparable(List<T> l) {
                for(int i = 0 ; i < l.size() - 1 ; i++) {
                        int indiceMin = rechercheIndicePlusPetitComparable(l, i);
                        if(indiceMin != i)  echangerComparable(l, i, indiceMin);
                }

        }

        private static <T extends Comparable<T>> int rechercheIndicePlusPetitComparable(List<T> l, int
indiceMin) {
                for(int j = indiceMin + 1 ; j < l.size() ; j++) {
                        if( l.get(j).compareTo( l.get(indiceMin) ) < 0 ) indiceMin = j;
                }
                return indiceMin;
        }

        private static <T extends Comparable<T>> void echangerComparable(List<T> l, int i, int j) {
                T tmpVal = l.get(i);
                l.set(i, l.get(j) );
                l.set(j, tmpVal);
        }
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class TestUtilListe {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                List<Integer> l = new ArrayList<Integer>();

                for(int i = 0 ; i < 5 ; i++) {
                        System.out.print("n" + (i + 1) + "= ? ");
                        l.add( new Integer(sc.nextInt()) );
                }

                System.out.print("La liste : ");
                for(Integer element : l)
                        System.out.print(" " + element.toString() + " ");
                System.out.println();

                System.out.println("maxListe() " + UtilListe.maxList(l) );
                System.out.println("maxListComparable() " + UtilListe.maxListComparable(l) );
                System.out.println("moyenneList() " + UtilListe.moyenneList(l) );

                UtilListe.triParSelection(l);

                System.out.print("La liste apres le tri : ");
                for(Integer element : l)
                        System.out.print(" " + element.toString() + " ");
                System.out.println();

                sc.close();
        }
}
```

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).

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.

```java
import java.util.ArrayList;
import java.util.List;
public class Promotion {
private List<Etudiant> etudiants;

	public Promotion(List<Etudiant> etudiants) {
		this.etudiants = etudiants;
	}

	public Promotion() {
		this.etudiants = new ArrayList<Etudiant>();
	}

	public void ajoutEtudiant(Etudiant etudiant) {
		etudiants.add( etudiant );
	}

	public Etudiant getEtudiant(String nom){
		Etudiant result = null;

		for(Etudiant e : etudiants) {
			if( e.getNom().equals(nom) )
				result = e;
		}
		return result;
	}

	public void afficherPromotion() {
		System.out.println("***** La prommotion *****");
		for(Etudiant e : etudiants) {
			System.out.println( e.toString() );
		}
	}

	public double getMoyennePromotion() {

		double sommeMoyennesEtudiants = 0;
		int nbEtudiants = 0;

		for(Etudiant e : etudiants) {
			sommeMoyennesEtudiants += e.getMoyenneEtudiant();
			nbEtudiants++;
		}
		return (sommeMoyennesEtudiants / (double) nbEtudiants);
	}

	public double getMoyenneUE(String ue) {
		double sommeMoyennesEtudiants = 0;
		int nbEtudiants = 0;
		double moyenneUeEtudiant;

		for(Etudiant e : etudiants) {
			moyenneUeEtudiant = e.getMoyenneUe(ue);

			if(moyenneUeEtudiant != -1){
				sommeMoyennesEtudiants += moyenneUeEtudiant;
				nbEtudiants++;
			}
		}
		return (sommeMoyennesEtudiants / (double) nbEtudiants);
	}

	public Etudiant getMajorPromotion() {
		Etudiant leMajor = null;
		double moyenneMajor = etudiants.get(0).getMoyenneEtudiant();
		double moyenneEtudiant;

		for(Etudiant e : etudiants) {
			moyenneEtudiant = e.getMoyenneEtudiant();
			if(moyenneMajor < moyenneEtudiant){
				leMajor = e;
				moyenneMajor = moyenneEtudiant;
			}

		}
		return (leMajor);
	}

	public Etudiant getMajorUE(String ue) {
		Etudiant leMajor = null;
		double moyenneMajor = etudiants.get(0).getMoyenneUe(ue);
		double moyenneEtudiant;

		for(Etudiant e : etudiants) {
			moyenneEtudiant = e.getMoyenneUe(ue);
			if(moyenneMajor < moyenneEtudiant){
				leMajor = e;
				moyenneMajor = moyenneEtudiant;
			}

		}
		return (leMajor);
	}
}
```

```java
import java.util.ArrayList;
import java.util.List;

public class Etudiant {
	private String nom;
	private String prenom;
	private List<NoteUE> notes;

	public Etudiant(String nom, String prenom, List<NoteUE> notes) {
		this.nom = nom;
		this.prenom = prenom;
		this.notes = notes;
	}

	public Etudiant(String nom, String prenom) {
		this.nom = nom;
		this.prenom = prenom;
		this.notes = new ArrayList<NoteUE>();
	}

	public List<NoteUE> getNotes() {
		return notes;
	}

	public String getNom() {
		return nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public void ajoutNoteCC(String ue, double cc) {
		boolean found = false;

		for(NoteUE element : notes) {
			if( ue.equals(element.getUE()) )
				element.setCC(cc);
				found = true;
		}

		if(!found)// if found == false
			notes.add( new NoteUE(ue, cc, 0.0) );
	}

	public void ajoutNoteExamen(String ue, double examen) {
		boolean found = false;

		for(NoteUE element : notes) {
			if( ue.equals(element.getUE()) )
				element.setExamen(examen);
				found = true;
		}

		if(!found)// if found == false
			notes.add( new NoteUE(ue, 0.0, examen) );
	}

	public double getMoyenneEtudiant() {
		double sommeNotesEtudiant = 0;
		int nbUeEtudiant = 0;

		for (NoteUE ue : notes) {
			sommeNotesEtudiant += ue.getMoyenne();
			nbUeEtudiant++;
		}
		return (sommeNotesEtudiant / (double) nbUeEtudiant);
	}

	public double getMoyenneUe(String ue) {

		for(NoteUE element : notes) {
			if( ue.equals(element.getUE()) )
				return element.getMoyenne();
		}

		return -1;
	}


	public String toString(){
		String result = "NOM : " + nom + "Prenom" + prenom + "\n";
		for(NoteUE element : notes)
			result += "\t UE : " +element.getUE() + "Moyenne : " + element.getMoyenne() + "\n";
		return result;
	}

}
```

```java
public class NoteUE {
	private String ue;
	private double cc;
	private double examen;

	public NoteUE(String ue, double cc, double examen) {
		this.ue = ue;
		this.cc = cc;
		this.examen = examen;
	}

	public NoteUE(String ue) {
		this(ue, 0.0, 0.0);
	}


	public String getUE() {
		return ue;
	}

	public void setCC(double cc) {
		this.cc = cc;
	}

	public void setExamen(double examen) {
		this.examen = examen;
	}

	public double getMoyenne() {
		double tmpMoyenne = ( cc + examen ) / 2;
		return( ( examen >  tmpMoyenne) ? examen : tmpMoyenne  );
	}



}
```

```java
import java.util.Scanner;

public class TestExo4 {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		char choix;
		double note;
		String nom, prenom, ue;
		Promotion promo = new Promotion();
		Etudiant etu;

		do{
			System.out.println("**** MENU ****");
			System.out.println("A  - Ajout d'un étudiant dans la promotion");
			System.out.println("B - Ajout d’une note de contrôle continu pour un étudiant");
			System.out.println("C - Ajout d’une note d’examen pour un étudiant");
			System.out.println("D - Afficher l’ensemble des étudiants de la promotion, avec leur moyenne par unité
d’enseignement");
			System.out.println("E - Calcul de la moyenne globale de la promotion");
			System.out.println("F - Calcul de la moyenne par unité d’enseignement");
			System.out.println("G - Recherche du major de promotion");
			System.out.println("M - Recherche du major par unité d’enseignement");
			System.out.println("Q - Quitter");

			System.out.print("Votre choix ? ");
			choix = sc.nextLine().charAt(0);

			switch( choix ){
			case 'A' : System.out.print("Nom ? ");
					nom = sc.nextLine();

					System.out.print("Prenom ? ");
					prenom = sc.nextLine();

					promo.ajoutEtudiant( new Etudiant(nom, prenom) );

			break;
			case 'B' : System.out.print("Nom ? ");
					nom = sc.nextLine();

					System.out.print("UE ? ");
					ue = sc.nextLine();

					System.out.print("Note CC ? ");
					note = sc.nextDouble();

					etu = promo.getEtudiant(nom);

					if(etu != null) etu.ajoutNoteCC(ue, note);
						sc.nextLine();
			break;
			case 'C' : System.out.print("Nom ? ");
						nom = sc.nextLine();

						System.out.print("UE ? ");
						ue = sc.nextLine();

						System.out.print("Note Exam ? ");
						note = sc.nextDouble();

						etu = promo.getEtudiant(nom);

						if(etu != null) etu.ajoutNoteExamen(ue, note);
						sc.nextLine();
			break;
			case 'D' :  promo.afficherPromotion();
			break;
			case 'E' :  System.out.println( promo.getMoyennePromotion() );
			break;
			case 'F' :  System.out.print("UE ? ");
						ue = sc.nextLine();
						System.out.println( promo.getMoyenneUE(ue) );
			break;
			case 'G' :
						etu = promo.getMajorPromotion();
						if (etu != null)
							System.out.println( etu.getNom() );
			break;
			case 'M' :	System.out.print("UE ? ");
						ue = sc.nextLine();

						etu = promo.getMajorUE(ue);
						if (etu != null)
							System.out.println( etu.getNom() );
			}

		}while(choix != 'Q');


		sc.close();
	}
}
```

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.

```java
public class Personne {
	private String prenom;
	private String nom;
	private String telephone;

	public Personne(String prenom, String nom, String telephone) {
		this.prenom = prenom;
		this.nom = nom;
		this.telephone = telephone;
	}

	public String getNom() {
		return nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public String getTelephone() {
		return telephone;
	}

}
```

```java
import java.util.List;
import java.util.ArrayList;

public class RepertoireSimple {
	private List<Personne> rep;

	public RepertoireSimple() {
		this.rep = new ArrayList<Personne>();
	}

	public void addPersonne(String prenom, String nom, String telephone) {
		rep.add( new Personne(prenom, nom, telephone) );
	}

	public String chercheNumero(String prenom, String nom) {
		for(int i = 0 ; i < rep.size() ; i++) {
			boolean check1 = rep.get(i).getNom().equals(nom);
			boolean check2 = rep.get(i).getPrenom().equals(prenom);
			if(check1 && check2)
				return ( rep.get(i).getTelephone() );
		}
		return "L'identite " + prenom + " " + nom + " est inconnue";
	}
}
```

```java
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" ) );
        }
}
```

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.

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").

```java
public class T9 {
        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;
                }
        }

        public static String getT9(String mot) {
                StringBuffer t9 = new StringBuffer("");
                for(int i = 0 ; i < mot.length() ; i++)
                        t9.append( getChiffreT9( mot.charAt(i) ) );

                return t9.toString();
        }
}
```

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) {
   // ...
}
```

```
import java.util.HashMap;
import java.util.ArrayList;

public class TestDicoT9 {
        public static void enregistrer(HashMap< String, ArrayList<String> > dico, String chaine) {
                String t9 = T9.getT9( chaine );
                if( !dico.containsKey(t9) )
                        dico.put(t9, new ArrayList<String>() );
                dico.get(t9).add( chaine );
        }

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

        }
}
```

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){
   // ...
}
```

```
import java.util.HashMap;
import java.util.ArrayList;

public class TestDicoT9 {

        public static ArrayList<String> recuperer(HashMap< String, ArrayList<String> > dico, String chaineT9) {
                return dico.containsKey( chaineT9 ) ? dico.get( chaineT9 ) : new ArrayList<String>() ;
        }

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

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]

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.

```
public class Carte {
        private CouleurCarte couleur;
        private ValeurCarte  valeur;

        public Carte(CouleurCarte couleur, ValeurCarte valeur){
                this.couleur = couleur;
                this.valeur = valeur;
        }
}
```

```
// Une  énumeration est une classe particuliere qui a un nombre fini et predefini
d'instances
// Lenumeration est juste une liste de constantes, separees par des virgules, qui se termine
par un point virgule.
public enum CouleurCarte {
        COEUR, CARREAU, PIQUE, TREFLE;
}
```

```
// Une  énumeration est une classe particuliere qui a un nombre fini et predefini
d'instances
// Lenumeration est juste une liste de constantes, separees par des virgules, qui se
termine par un point virgule.
public enum ValeurCarte {
        UN, DEUX, TROIS, QUATRE, CINQ, SIX, SEPT, HUIT, NEUF, DIX, VALET, DAME, ROI ;
}
```

2. Les cartes qu'un joueur possède sont appelées sa main. Créez une classe qui permet de représenter la main d'un jouer, d'y ajouter une carte ou d'en retirer une carte.

```
import java.util.List;
import java.util.ArrayList;

public class MainJoueur {
	private List<Carte> laMain;

	public MainJoueur(){
		this.laMain = new ArrayList<Carte>();
	}

	public MainJoueur(ArrayList<Carte> laMain){
		this.laMain = laMain;
	}

	public void ajouterCarte(Carte carte){
		laMain.add( carte );
	}

	public void retirerCarte(Carte carte){
		/* boolean remove(Object o)
		 * Removes the first occurrence of the specified element from this list,if it
is present
		 *  If this list does not containthe element, it is unchanged.
		 *  ore formally, (o==null ? get(i)==null : o.equals(get(i)))(if such an
element exists).
		 *  Returns true if this list contained the specified element (or
equivalently, if this list changedas a result of the call).
		 */
		laMain.remove( carte );


		/* public boolean equals(Object obj)
		 * Indicates whether some other object is "equal to" this one.
		 * The equals method for class Object implements the most discriminating
possible equivalence relation on objects;
		 * that is, for any non-null reference values x and y, this method returns
true if and only if x and y refer to the same object(x == y has the value true).
		 */

	}
}
```

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]

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.

EXO 3

```java
public static <T extends Comparable<T>> void triParSelectionComparable(List<T> l) {
    for(int i = 0 ; i < l.size() - 1 ; i++) {
        int indiceMin = rechercheIndicePlusPetitComparable(l, i);
        if(indiceMin != i)  echangerComparable(l, i, indiceMin);
    }

}

private static <T extends Comparable<T>> int rechercheIndicePlusPetitComparable(List<T> l, int indiceMin) {
    for(int j = indiceMin + 1 ; j < l.size() ; j++) {
        if( l.get(j).compareTo( l.get(indiceMin) ) < 0 ) indiceMin = j;
    }
    return indiceMin;
}

private s
    T tmp
    l.set
    l.set
}
```

```java
/* int java.lang.Comparable.compareTo(T o)
 * int compareTo(T o)
 * Compares this object with the specified object for order.
 * Returns a negative integer, zero, or a positive integer
 * as this object is less than, equal to, or greater than the specified object.
 */
```

```java
public class Carte implements Comparable<Carte>{
        private CouleurCarte couleur;
        private ValeurCarte  valeur;

        public Carte(CouleurCarte couleur, ValeurCarte valeur){
                //…
        }

        @Override
        public int compareTo(Carte carte) {
                /* boolean java.lang.Enum.equals(Object other)
                 * public final boolean equals(Object other)
                 * Returns true if the specified object is equal to thisenum constant.
                 * Overrides: equals in class Object
                 */

                boolean memeCouleur = carte.couleur.equals( this.couleur);
                int resultCompare;

                /* int java.lang.Comparable.compareTo(T o)
                 * int compareTo(T o)
                 * Compares this object with the specified object for order.
                 * Returns a negative integer, zero, or a positive integer
                 * as this object is less than, equal to, or greater than the specified object.
                 */
                if(memeCouleur) // si deux cartes sont de la même couleur, on les compare suivant leurs valeurs
                        resultCompare = this.valeur.ordinal() - carte.valeur.ordinal();
                else
                        resultCompare = this.couleur.ordinal() - carte.couleur.ordinal();

                /* public final int ordinal()
                 * Returns the ordinal of this enumeration constant
                 * (its position in its enum declaration, where the initial constant is assigned an ordinal of zero).
                 */

                return resultCompare;
        }
}
```

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).

```
public abstract class CarteAbstraite {
		private CouleurCarte couleur;
		private ValeurCarte  valeur;

		public CarteAbstraite(CouleurCarte couleur, ValeurCarte valeur){
			this.couleur = couleur;
			this.valeur = valeur;
		}

}
```

```
// Une  énumeration est une classe particuliere qui a un nombre fini et predefini
d'instances
// Lenumeration est juste une liste de constantes, separees par des virgules, qui
se termine par un point virgule.
public enum CouleurCarte {
	COEUR, CARREAU, PIQUE, TREFLE;
}
```

```
// Une  énumeration est une classe particuliere qui a un nombre fini et predefini
d'instances
// Lenumeration est juste une liste de constantes, separees par des virgules, qui se termine
par un point virgule.
public enum ValeurCarte {
	UN, DEUX, TROIS, QUATRE, CINQ, SIX, SEPT, HUIT, NEUF, DIX, VALET, DAME, ROI ;
}
```

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.

```
public abstract class CarteAbstraite {
		private CouleurCarte couleur;
		private ValeurCarte  valeur;

		public CarteAbstraite(CouleurCarte couleur, ValeurCarte valeur){
			this.couleur = couleur;
			this.valeur = valeur;
		}

		public CouleurCarte getCouleur() {
			return couleur;
		}

		public ValeurCarte getValeur() {
			return valeur;
		}


}
```

```
public class CarteBasique extends CarteAbstraite implements Comparable<CarteBasique>{

	public CarteBasique(CouleurCarte couleur, ValeurCarte valeur){
		super(couleur, valeur);
	}

	@Override
	public int compareTo(CarteBasique carte) {
		/* boolean java.lang.Enum.equals(Object other)
		 * public final boolean equals(Object other)
		 * Returns true if the specified object is equal to thisenum constant.
		 * Overrides: equals in class Object
		 */

		boolean memeCouleur = carte.getCouleur().equals( this.getCouleur());
		int resultCompare;

		/* int java.lang.Comparable.compareTo(T o)
		 * int compareTo(T o)
		 * Compares this object with the specified object for order.
		 * Returns a negative integer, zero, or a positive integer
		 * as this object is less than, equal to, or greater than the specified object.
		 */
		if(memeCouleur) // si deux cartes sont de la même couleur, on les compare suivant
leurs valeurs
			resultCompare = this.getValeur().ordinal() - carte.getValeur().ordinal();
		else
			resultCompare = this.getCouleur().ordinal() - carte.getCouleur().ordinal();

		/* public final int ordinal()
		 * Returns the ordinal of this enumeration constant
		 * (its position in its enum declaration, where the initial constant is assigned an
ordinal of zero).
		 */

		return resultCompare;
	}
}
```

(b) La classe CarteManille permet de trier les cartes comme à la manille. [9]

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

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.

abc