Programmation avancée et application • TD 3

POO Classes et heritage

Exercice I (T9 et classes)
On a vu dans le TD précédent comment implémenter des méthodes qui permettent de stocker des chaînes dans un dictionnaire dont les clés sont leur code T9. Certains éléments vus à ce moment là peuvent être réutilisés.

TD 2

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").

TD 2

```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".

TD 2

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) {
   // ...
}
```

TD 2

```
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){
   // ...
}
```

TD 2

```
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) {
           //…
        }
}
```

1. Créez une classe DicoT9 qui possède les méthodes suivantes :
   (a) **public void** enregistrer(String chaine) { ... }
   (b) **public** ArrayList<String> recuperer(String chaineT9){ ... }

```java
public class TestDicoT9 {
        public static void main (String[] args) {
                DicoT9 dico = new DicoT9();

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

        System.out.println( dico.recuperer( "26663" ) ); // = dico.recuperer( "26663" ).toString()
        System.out.println( dico.recuperer( "2665687" ) ); // = dico.recuperer( "2665687" ).toString()
        System.out.println( dico.recuperer( "123456" ) ); // = dico.recuperer( "123456" ).toString()

        }
}
```

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

public class DicoT9 {
        Map< String, ArrayList<String> > dico;

        public DicoT9() {
                dico = new HashMap< String, ArrayList<String> >();
        }

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

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

```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) {
		StringBuilder t9 = new StringBuilder("");
		for(int i = 0 ; i < mot.length() ; i++)
			t9.append( getChiffreT9( mot.charAt(i) ) );

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

Exercice II (Géométrie et POO, le retour)
On souhaite manipuler des objets du plan. On peut réutiliser la classe Point et la classe Vecteur créées au TD précédent.

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

	public double getX() {
		return x;
	}

	public double getY() {
		return y;
	}

	public Point(double x, double y) {
		this.x = x;
		this.y = y;
	}

	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) );
	}

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

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

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

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

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

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

	Point getP1() {
		return p1;
	}

	Point getP2() {
		return p2;
	}

	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;
	}
}
```

1. Créez une classe abstraite Figure qui représente une figure géométrique. Elle doit disposer des méthodes suivantes :
   - **public abstract double** perimetre();
   - **public abstract double** surface();
   - **public abstract** Figure translation(Vecteur v);

```java
public abstract class Figure {

	public abstract double perimetre();
	public abstract double surface();
	public abstract Figure translation(Vecteur v);

	@Override
	public String toString(){
		return ("perimetre = " + perimetre() + " ; surface = " +  surface() );
	}

}
```

2. Créez une classe Disque qui hérite de Figure. [1]

1. Vous pouvez vous inspirer de la classe Disque créée au TD précédent.

TD 2

```java
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.
	 */

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

	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;
	}

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

	public Disque translation(Vecteur v) {

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

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

	/* Calcul du périmètre d'un cercle (ou d'un disque)
	 * Le périmètre d'un cercle est égal au produit de 2 par π (nombre pi) par la longueur du rayon R du cercle.
	 * Périmètre P d'un cercle (ou d'un disque) = 2 x π x R
	 * Source : https://www.calculateur.com/perimetre-d-un-cercle-ou-d-un-disque.html
	 */
	public double perimetre() {
		return ( ( (double) 2 ) * Math.PI * rayon);
	}


	/* Surface et cercle : aire d'un disque
	 * Source : https://www.lememento.fr/surface-et-cercle-aire-dun-disque
	 * La surface délimitée par un cercle de rayon r correspond à l'aire A du disque de rayon r.
	 * L'aire A d'un disque de rayon r est égale à :
	 * A = π r²
	 */
	public double surface() {
		// public static double pow(double a,double b)
		// Returns : the value a^b.
		return ( Math.PI * Math.pow(rayon, 2) );
	}

	/* La translation d'un disque par un vecteur consiste en un nouveau disque
	 * dont on a déplacé le centre, mais en conservant son rayon.
	 * Source : sujet td2 PAA
	 */
	public Disque translation(Vecteur v) {
		return( new Disque(coordonneesDuCentre.translation(v), rayon)  );
	}
}
```

3. Créez une classe abstraite Quadrilatere qui hérite de Figure. Un quadrilatère peut être caractérisé par ses quatre sommets, et son périmètre est facile à calculer : il s'agit de la somme des longueurs de ses quatre côtés. Implémentez cette classe, en déterminant quelles parties sont abstraites, et quelles parties ne le sont pas.

```java
/**
 * En géométrie plane, un quadrilatère est un polygone à quatre côtés.
 * Les trapèzes, parallélogrammes, losanges, rectangles, carrés et cerfs-volants sont des quadrilatères particuliers.
 * Source : https://fr.wikipedia.org/wiki/Quadrilat%C3%A8re
 *
 */
public abstract class Quadrilatere extends Figure{
	// Un quadrilatère peut être caractérisé par ses quatre sommets.
	private Point sommet1;
	private Point sommet2;
	private Point sommet3;
	private Point sommet4;

	public Quadrilatere(Point sommet1, Point sommet2, Point sommet3,  Point sommet4) {
		this.sommet1 = sommet1;
		this.sommet2 = sommet2;
		this.sommet3 = sommet3;
		this.sommet4 = sommet4;
	}

	// périmètre = la somme des longueurs de ses quatre côtés
	public double perimetre() {
		double longueurCote1 = sommet1.distance(sommet2);
		double longueurCote2 = sommet2.distance(sommet3);
		double longueurCote3 = sommet3.distance(sommet4);
		double longueurCote4 = sommet4.distance(sommet1);

		return ( longueurCote1 + longueurCote2 + longueurCote3 + longueurCote4 );
	}

	public abstract double surface();

	public abstract Quadrilatere translation(Vecteur v);
	/* IMPOSSIBLE - "Cannot instantiate the type Quadrilatere" vu que Quadrilatere est une "abstract class"
	 * => donc translation() = abstract methode
	public Quadrilatere translation(Vecteur v){
		return( new Quadrilatere(sommet1.translation(v), sommet2.translation(v) , sommet3.translation(v) , sommet1.translation(v));

	}
	*/

	public Point getSommet1(){
		return sommet1;
	}

	public Point getSommet2(){
		return sommet2;
	}

	public Point getSommet3(){
		return sommet3;
	}

	public Point getSommet4(){
		return sommet4;
	}

}
```

4. Un losange est un quadrilatère dont les quatre côtés ont la même longueur. Créez la classe Losange et implémentez les méthodes nécessaires.

```java
/**
 *      Un losange est un quadrilatère dont les quatre côtés ont la même longueur.
 */
public class Losange extends Quadrilatere{

	public Losange(Point sommet1, Point sommet2, Point sommet3, Point sommet4) {
		super(sommet1, sommet2, sommet3, sommet4);
	}

	/** Calcul de l'aire d'un losange
	 *  L'aire A d'un losange (ou surface d'un losange) est égale
	 *  au produit de 1/2 par la longueur de la grande diagonale D par la longueur de la petite diagonale d,
	 *  soit : Aire A = (D * d) / 2
	 */
	public double surface() {
		double longueurDiagonale1 = getSommet1().distance( getSommet3() );
		double longueurDiagonale2 = getSommet2().distance( getSommet4() );

		return ( (longueurDiagonale1 * longueurDiagonale2) / (double) 2 );
	}

	@Override
	public Losange translation(Vecteur v) {
		return( new Losange(getSommet1().translation(v), getSommet2().translation(v) , getSommet3().translation(v) , getSommet4().translation(v)) );
	}
}
```

```java
public class TestFigures {
	public static void main ( String [] args ) {
		Disque d = new Disque ( new Point(3.5 ,4.0)  ,  4 );
		Losange l = new Losange ( new Point(0 ,0 ) ,
			new Point( 2 , 0 ) , new Point ( 2 ,2 ) , new Point ( 0 ,2 ) ) ;

		System.out.println( d );
		System.out. println( l );

		Vecteur v = new Vecteur(new Point ( 2 , 2 ) , new Point(3 ,3 ) ) ;
		System.out.println( d.translation( v ) ) ;
		System.out.println( l.translation( v ) ) ;

	}
}
```

Exercice III (Répertoires améliorés)
On a créé précédement une classe RepertoireSimple qui permet :

- 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.

TD 2

TD 3

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

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";
	}

	public List<Personne> getRep(){
		return rep;
	}
}
```

```java
public class Personne implements Comparable<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;
	}

	@Override
	public String toString() {
		return prenom + " " + nom + " : " + telephone;
	}

	@Override
	public int compareTo(Personne personne){
		return ( nom.equals(personne.nom) ? prenom.compareTo(personne.prenom) :
nom.compareTo(personne.nom) );
	}
}
```

On souhaite améliorer les répertoires avec de nouvelles fonctionnalités :

- un répertoire est associé à un contact particulier, qui est le propriétaire du répertoire;
- on souhaite pouvoir rechercher l'identité d'une personne en fonction de son numéro de téléphone;
- on souhaite pouvoir afficher l'ensemble des contacts contenus dans le répertoire : d'abord les informations sur le propriétaire du répertoire, puis tous les autres contacts triés par ordre alphabétique (du nom d'abord, du prénom ensuite).

Créez une classe RepertoireAmeliore, qui hérite de RepertoireSimple.

```
public class RepertoireAmeliore extends RepertoireSimple{
	private Personne proprietaire;

	public RepertoireAmeliore(Personne proprietaire) {
		super();
		this.proprietaire = proprietaire;
	}

	public String cherchePersonne(String telephone) {
		for(Personne personne : getRep()) {
			if( personne.getTelephone().equals(telephone) )
				return personne.getPrenom() + " " + personne.getNom();
		}
		return "Le numero " + telephone + " n'appartient a personne.";
	}

	@Override
	public String toString() {
		StringBuilder result = new StringBuilder( "Proprietaire : " +
proprietaire.toString() + "\n");
		UtilListe.triParSelectionComparable( getRep() );
		for(Personne personne :  getRep())
			result.append( personne.toString() + "\n");
		return result.toString();
	}
}
```

```
public class TestRepertoireAmeliore {
	public static void main (String[] args) {
		RepertoireAmeliore rep = new RepertoireAmeliore( new Personne("Jimi",
"Hendrix", "0987654321") );
		rep.addPersonne ("John" , "Lennon", "0123456789");
		rep.addPersonne ("Paul" , "McCartney", "0234567891");
		rep.addPersonne ("George", "Harrison", "0345678912");
		rep.addPersonne ("Ringo",  "Starr", "0456789123");
		rep.addPersonne ("Sean", "Lennon", "0567891234");

		System.out.println( rep.chercheNumero("John", "Lennon" ) );
		System.out.println( rep.chercheNumero("Freddie", "Mercury" ) );
		System.out.println( rep.cherchePersonne("0234567891") );
		System.out.println( rep.cherchePersonne("0234567899") );
		System.out.println( rep );
	}
}
```

TD2 - triParSelectionComparable()

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 $V(min)$

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

COURS Chapitre 1

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);
    }
  }
}
```

110

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

code TD2

```
import java.util.List;
public class UtilListe{

	/* 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-clÍe extends
	 * que A soit une classe ou une interface
	 */

	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);
	}
}
```

Exercice IV (Produits et héritage)
Nous allons modéliser le fonctionnement d'une grande surface.

1. Le taux de TVA en France dépend de la nature du produit vendu. Il y a 4 taux différents : normal (20%), intermédiaire (10%), réduit (5,5%) et particulier (2,1%). Définissez une Enum qui permet de représenter les différents taux de TVA.

```java
// Cours chapitre3 , page 23
public enum TVA {
	NORMAL(0.2), INTERMEDIAIRE(0.1), REDUIT(0.055), PARTICULIER(0.021);

	// normal : 20% => 20/100 = 0.2

	private double tauxTva;

	private TVA(double tauxTva) {
		this.tauxTva = tauxTva;
	}

	public double getTauxTva() {
		return tauxTva;
	}
}
```

2. Un produit est une entité qui possède un prix hors taxes et un taux de TVA. Définissez une classe abstraite pour représenter un produit. Une méthode permet de calculer le prix TTC en fonction du prix HT et du taux de TVA.

```java
public abstract class Produit {
	private double prixHt;
	private TVA tauxTva;
	private double prixTtc;

	public Produit(double prixHt, TVA tauxTva) {
		this.prixHt = prixHt;
		this.tauxTva = tauxTva;
	}

	public void calculerPrixTtc(){
		prixTtc = prixHt + (prixHt * tauxTva.getTauxTva());
	}

	public double getPrixTtc() {
		return prixTtc;
	}
}
```

3. Nous définissons maintenant des classes représentant différents types de produits :
   (a) un DVD possède un titre, un réalisateur, et son taux de TVA est normal ;
   (b) un livre possède un titre, un auteur, un ISBN [2], et son taux de TVA est réduit ;
   (c) un fruit possède un nom, un pays d'origine, et son taux de TVA est réduit ;
   (d) un médicament possède un nom, un laboratoire, le nombre de comprimés dans la boîte, et le taux de TVA est particulier.

```java
public class DVD extends Produit{
        private String titre;
        private String realisateur;

        public DVD(String titre, String realisateur, double prixHt) {
                super(prixHt, TVA.NORMAL);
                this.titre = titre;
                this.realisateur = realisateur;
        }
}
```

```java
public class Livre extends Produit{
        String titre;
        String auteur;
        String ISBN;

        /* International Standard Book Number (ISBN) ou Numéro international normalisé du livre
         * est un numéro internationalement reconnu, créé en 1970,
         * identifiant de manière unique chaque édition de chaque livre publié
         */

        public Livre(String titre, String auteur, String ISBN, double prixHt) {
                super(prixHt, TVA.REDUIT);
                this.titre = titre;
                this.auteur = auteur;
                this.ISBN = ISBN;
        }
}
```

```java
public class Fruit extends Produit{
        private String nom;
        private String paysOrigine;

        public Fruit(String nom, String paysOrigine, double prixHt){
                super(prixHt, TVA.REDUIT);
                this.nom = nom;
                this.paysOrigine = paysOrigine;
        }
}
```

```java
public class Medicament extends Produit{
        private String nom;
        private String laboratoire;
        private int nombreComprimesBoite;

        public Medicament(String nom, String laboratoire, int nombreComprimesBoite, double prixHt) {
                super(prixHt, TVA.PARTICULIER);
                this.nom = nom;
                this.laboratoire = laboratoire;
                this.nombreComprimesBoite = nombreComprimesBoite;
        }
}
```

Exercice V (POO et télévision)
La grille de programmation d'une chaîne de télévision est une suite de programmes de différents types :

- les émissions de divertissement,
- le journal télévisé,
- les reportages,
- les fictions.

Tous les programmes sont caractérisés par leur heure de début et de fin.[3] Les émissions de divertissement ont un présentateur, un nom, et durent toutes deux heures. Les fictions sont définies par leur titre, le nom du réalisateur, et s'il s'agit ou non d'une rediffusion. Le journal télévisé a un présentateur. Enfin, un reportage a un thème parmi trois possibles : histoire, actualité, culture) et un nom.
Définissez les classes qui permettent de modéliser les informations données ci-dessus.

3. Pour simplifier, les heures sont données sous forme de nombres entiers, entre 0 et 23.

```java
public abstract class Programme {
        // Pour simplifier, les heures sont données sous forme de nombres entiers, entre 0 et 23.
        private int heureDebut;
        private int heureFin;

        public Programme(int heureDebut, int heureFin) {
                this.heureDebut = heureDebut;
                this.heureFin = heureFin;
        }
}
```

```java
public class Divertissement extends Programme{
        private String nom;
        private String presentateur;

        public Divertissement(String nom, String presentateur, int heureDebut) {
                super(heureDebut, heureDebut + 2);
                this.nom = nom;
                this.presentateur = presentateur;
        }
}
```

```java
public class Fiction extends Programme{
        private String titre;
        private String realisateur;
        private boolean rediffusion;

        public Fiction(String titre, String realisateur, boolean rediffusion, int
heureDebut, int heureFin) {
                super(heureDebut, heureFin);
                this.titre = titre;
                this.realisateur = realisateur;
                this.rediffusion = rediffusion;
        }
}
```

```java
public class JournalTelevise extends Programme{
	private String presentateur;

	public JournalTelevise(String presentateur, int heureDebut, int heureFin) {
		super(heureDebut, heureFin);
		this.presentateur = presentateur;
	}
}
```

```java
public enum ThemeReportage {
	HISTOIRE, ACTUALITE, CULTURE;
}
```

```java
public class Reportage extends Programme{
	private String nom;
	private ThemeReportage theme;

	public Reportage(String nom, ThemeReportage theme, int heureDebut, int heureFin) {
		super(heureDebut, heureFin);
		this.nom = nom;
		this.theme = theme;
	}
}
```