

Manuel de corrigés Module JAVA

Travaux dirigés

Travaux pratiques

Filière DUT GI
M. Lahmer
m.lahmer@umi-.ac.ma
2002-2023



Partie I Travaux dirigés avec Solution	3
TD 1 Classes et Objets	3
Correction TD 1	4
TD 2 Syntaxe de base en Java	5
Correction TD 2	6
TD 3 Récapitulatif	7
Prototype de correction TD 3	8
TD 4 Application de Gestion de Caisse	9
Correction TD 4	10
TD 5 Héritage et Collections en Java	12
Correction TD 5	13
TD 6 Examen Modele de DS 1	14
Correction de TD DS 1	15
TD 7 Fichiers	17
Correction TD 7	18
TD 8 Interface d'authentification depuis un fichier texte/Correction	24
TD 9 Image Viewer en JavaFx (sans FXML)	26
CorrectionTD 9	27
TD 10 Gestion Produit/Base de données	29
Correction TD 10	30
Partie II Travaux pratiques avec Solution	37
TP 1 Révision et prise en main du langage Java	37
Solution TP 1	38
TP 2 Classes / Objets	39
Solution TP 2	40
TP 3 sur les Classes particulières String, Integer	42
Solution TP3	43
Atelier Gestion Salle Cinéma (classes, Composition, tableaux/collections)	45
Solution	47
Atelier	50
Gestion Comptes bancaires (Heritage et collections)	50
Solution	52
Atelier Java (Héritage, Interfaces)	56
Solution	58
Atelier IO/ Sécurité (Crypto, Compression, Hachage)	63
Solution	65
Atelier Gestion des Contact/(Swing et base de données)	68
Solution	69

Partie I Travaux dirigés avec Solution

TD 1 Classes et Objets

L'objectif de ce TD est de réutiliser le code à travers la composition. En programmant des Objets de la géométrie plane. On s'intéressera en particulier au point, cercle et rectangle.

A- classe point

Ecrire une classe point définie par les coordonnées x et y de types réels. La classe implémentera en plus du constructeur les méthodes suivantes :

- Distance : permet de calculer la distance entre deux points
- Translater : permet de translater un point d'une position à une autre
- Print : permet d'afficher les coordonnées d'un point

B- classe cercle

Un cercle est défini par le centre (un point) et le rayon (un réel). Ecrire une classe cercle qui implémentera en plus du constructeur les méthodes suivantes :

- Surface : retourne la surface du cercle
- Perimetre : retourne le périmètre du cercle
- Translater : idem que point
- Print : permet d'afficher l'état d'un cercle (centre et rayon)

C- classe rectangle

Un rectangle est défini par le point haut gauche et le point bas droit. Ecrire une classe rectangle qui implémentera en plus du constructeur les méthodes suivantes :

- Surface : retourne la surface du rectangle
- Perimetre : retourne le périmètre du rectangle
- Translater : idem que point
- Print : permet d'afficher l'état d'un rectangle

A la fin de chaque question vous allez écrire une classe de Test qui permettra de tester toutes les méthodes à implémenter.

Correction TD 1

```
public class Point {
    private double x;
    private double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }
    public double getX() {
        return this.x;
    }
    public void setX(double x) {
        this.x = x;
    }
    public double getY() {
        return this.y;
    }
    public void setY(double y) {
        this.y = y;
    }
    @Override
    public String toString() {
        return "{" +
            " x:" + getX() + " " +
            " y:" + getY() + " " +
            "}";
    }
    public static void translate(double
dx,double dy){
        this.x=this.x+dx;4
        this.y=this.y+dy;
    }
    //Surcharge
    public Point translate(Point p){
        return new Point(this.x+p.x, this.y+p.y);
    }
    public int compareTo(Point p){
        double n=Math.sqrt(Math.pow(this.x, 2)+Math.pow(this.y, 2));
        double np=Math.sqrt(Math.pow(p.x, 2)+Math.pow(p.y, 2));
        /*if (n==np) return 0;
        if (n>np) return 1;
        return -1;*/
        return (int)(n-np);
    }
    public static int compareTo(Point p,Point q){
        return p.compareTo(q);
    }
}

public class Cercle {
    private Point centre;
    private double r;
    public double surface(){
        return Math.PI*r*r;
    }
    public int compareTo(Cercle c){
        return (int)(this.surface()-c.surface());
    }
    public void translate(double dx,double dy){
        centre.translate(dx,dy);
    }
    public static distance(Cercle c, Cercle d){
        return c.distance(d)
    }
    public static distance(Cercle c){
        return this.distance(c)
    }
}
```

TD 2 Syntaxe de base en Java

N.B : Il est fortement recommandez de créer une classe Java Syntaxe avec la méthode main et tester avant de répondre.

A- Donnez la valeur affichée par l'instructions suivantes :

1. `System.out.println("3+1=" +(3+1));`
2. `System.out.println(" 3+1="+3+1);`
3. `System.out.println("Taille byte en Java : " +Byte.SIZE);`
4. `System.out.println("3 or 5 : " +(3 | 5));`
5. `System.out.println("3 xor 5 : " +(3 ^ 5));`
6. `System.out.println("3 or 5 : " +(3 & 5));`

B- Remplir le tableau ci-dessous

L'expression	C'est quoi l'erreur	L'expression correcte
<code>byte b=300;</code>		
<code>float pi=3.14;</code>		
<code>short i=30; int j=i;</code>		
<code>double f=3.45;float x=f;</code>		
<code>String s=120;</code>		
<code>String s=String.valueOf(13.4);</code>		
<code>int a="123";</code>		
<code>int a= Integer.parseInt("JAVA");</code>		

C- Répondez aux Quiz suivants et reportez fidèlement votre premier score comme réponse à chaque Quiz (Pour des raisons de Qualité):

1. https://www.w3schools.com/java/java_quiz.asp
2. <https://www.proprofs.com/quiz-school/story.php?title=test-java>

Correction TD 2

A- Donnez la valeur affichée par l'instructions suivantes :

1. `System.out.println("3+1=" + (3+1));` // **3+1=4**
2. `System.out.println(" 3+1=" + 3+1);` // **3+1=31**
3. `System.out.println("Taille byte en Java : " + Byte.SIZE);` // **Taille byte en Java : 8**
4. `System.out.println("3 or 5 : " + (3 | 5));` // **3 or 5 : 7**
5. `System.out.println("3 xor 5 : " + (3 ^ 5));` // **3 xor 5 : 6**
6. `System.out.println("3 or 5 : " + (3 & 5));` // **3 or 5 : 1**

L'expression	C'est quoi l'erreur	L'expression correcte
<code>byte b=300 ;</code>	La constante entière est par défaut de type int	<code>byte b=(byte)300 ;</code>
<code>float pi=3.14;</code>	Les constantes réelles sont par défaut de type double	<code>float pi=3.14f;</code>
<code>short i=30; int j=i;</code>	La constante entière est par défaut de type int	<code>short i=(short)30; int j=i;</code>
<code>double f=3.45; float x=f;</code>	On ne peut pas affecter un double à float car le double est codé sur <u>8 octets</u> par contre le float c'est juste sur <u>4 octets</u>	<code>double f=3.45; float x=(float)f;</code>
<code>String s = 120 ;</code>	On ne peut pas convertir de int a string directement	<code>String s=String.valueOf(120);</code>
<code>String s=String.valueOf(13.4);</code>	Il n'y a pas d'erreur car la méthode <code>valueOf()</code> (et puisque on a le surcharge) peut prendre en paramètre tous les types primitives (double, float, int ...)	
<code>int a="123";</code>	On ne peut pas convertir de string à int directement	<code>int a= Integer.parseInt("123");</code>
<code>int a= Integer.parseInt("JAVA") ;</code>	La méthode <code>parseInt()</code> prend en paramètre un string qui contient des nombres non des lettres afin de la retourner en décimal.	<code>int a= Integer.parseInt("222");</code>

TD 3 Récapitulatif

1. Nombre rationnel

On désire représenter les nombres rationnels sous la forme d'une fraction de 2 entiers.

On écrira un constructeur permettant d'instancier des nombre rationnels, de manière à obtenir la forme canonique (c'est à dire sous la forme d'une faction qui ne peut pas être simplifiée). Pour simplifier la fraction, on divisera les 2 entiers par leur PGCD.

On écrira donc une fonction PGCD utilisant l'algorithme d'Euclide :

Calcul du PGCD de deux entiers naturels m et n tels que $m > n$.

Itérer les manipulations suivantes :

Effectuer la division euclidienne de m par n . Soit r le reste.

Remplacer m par n et n par r . On a $n > r$ d'après la définition de la division euclidienne. Le PGCD est le dernier reste non nul. (Si n divise m , le PGCD est n)

On voudra pouvoir récupérer le numérateur et le dénominateur, afficher un rationnel, tester l'égalité de deux rationnels, d'additionner, de multiplier, de diviser des rationnels, de calculer l'inverse...

2. Fractions égyptiennes

Les égyptiens ne connaissaient comme rationnels que les inverses des nombres entiers. En 1201, Fibonacci prouva que tout nombre rationnel pouvait s'écrire sous la forme d'une somme de fractions égyptienne. Remarque : cette décomposition n'est pas unique.

Voici une méthode pour l'obtention de cette somme :

Soit une fraction n/d que l'on cherche à décomposer.

Si n/d est négatif, on traitera son opposé et on changera les signes de la somme.

Si $n=1$, la décomposition est terminée.

Sinon, chercher la fraction égyptienne $1/a$ la plus proche de n/d et recommencer avec $(n/d - 1/a)$

Programmer une méthode statique `DecomposeEnFractionEgyptienne(Rationnel r)` qui ramène sous forme d'une chaîne de caractère la décomposition d'un rationnel en une somme de fractions rationnelles.

Note :

A chaque fois que vous avez à traiter la division penser à capturer la division par zéro en utilisant votre propre Exception.

Prototype de correction TD 3

```

public class Rationnel {
    private int numerateur;
    private int denominateur;

    public void setNumerateur(int
a){
        numerateur=a;
    }
    public int getNumerateur(){
        return numerateur;
    }
    public void
setDenominateur(int a){
        if(a==0)
erreur(DIVISION_PAR_0);
        else
            denominateur=a;
    }
    public int getDenominateur(){
        return denominateur;
    }
    public Rationnel(int n,int d){
        this.setNumerateur(n);
        this.setDenominateur(d);
    }
    public Rationnel(int n){
        this(n,1);
    }
    public Rationnel(){
        this(0,1);
    }

    public static int pgcd(int
a,int b){
        if(a<0) a=-a;
        else
            if(a==0) return b;
        if(b<0) b=-b;
        else
            if(b==0) return a;
        if(a==b)
            return a;
        if(a<b)
            return pgcd(a,b-a);
        return pgcd(a-b,b);
    }

    public Rationnel toCanonique()
{
    return
Rationnel.toCanonique(this);
}

    public Rationnel addition(Rationnel r){
        return Rationnel.addition(this, r);
    }
    public Rationnel soustraction(Rationnel r){
        return Rationnel.soustraction(this, r);
    }
    public Rationnel multiplication(Rationnel r){
        return Rationnel.multiplication(this, r);
    }
    public Rationnel division(Rationnel r){
        return Rationnel.division(this,r);
    }
    public Rationnel puissance(int n){
        return Rationnel.puissance(this,n);
    }
    public int compareTo(Rationnel o) {
        return this.soustraction(o).numerateur;
    }
    @Override
    public String toString(){
        return getNumerateur()
+ "/" + getDenominateur();
    }
    @Override
    public double doubleValue() {
        return (double) getNumerateur()/
getDenominateur();
    }
}

```


TD 4 Application de Gestion de Caisse

On vous propose de réaliser une application gestion de caisse et d'édition de tickets pour le compte d'un salon de thé qui offre deux types de produits :

Boisson : définie par un libellé, une tva fixée à 15% et un prix (café, thé, limonade, jus, ...etc.)

Pâtisserie : définie par un libellé, une tva fixée à 30% et un prix (petit pain, croissant fourré, etc.)

La caisse est calculée selon le total des tickets délivrés. Chaque ticket est associée à une table et il peut comporter les boissons et ou les pâtisseries. Chaque table est identifiée par un numéro.

L'objectif de l'application est de permettre l'édition d'un ticket selon le format ci-dessous :

Table numéro :

Libellé	Prix	Tva	Quantité
----	---	---	---

Total : ---

Elle doit en plus permettre de savoir à n'importe quel moment :

- Le contenu de la caisse qui est la somme des totaux tickets.
 - Le nombre de consommation indépendamment de son type (boisson ou pâtisserie)
- 1) Enumérer l'ensemble de classes possibles avec les propriétés et les constructeurs associés
 - 2) Comment vous allez modéliser la variable nombreConsommation et dans quelle classe vous allez le définir
 - 3) Dans quelle classe vous allez définir les méthodes EditerTicket et getCaisse.
 - 4) Créer une classe de test dans laquelle vous allez modéliser le service de deux tables :
 - a. Table numéro 1 : 1 Café noir, 2 Jus d'orange
 - b. Table numéro 2 : 2 Café crème, 4 petits pains
 - 5) Afficher le ticket associé à chaque table ainsi que le total caisse de la journée

Correction TD 4

```
public abstract class Produit {
    private String libelle;
    private double prix;
    public abstract double getPrixTTC();
    public Produit(String libelle,
double prix) {
        this.libelle = libelle;
        this.prix = prix;
        System.out.println("Produit");
    }
    public String getLibelle() {
        return this.libelle;
    }
    public void setLibelle(String
libelle) {
        this.libelle = libelle;
    }
    public double getPrix() {
        return this.prix;
    }
    public void setPrix(double prix) {
        this.prix = prix;
    }
    public String toString(){
        return libelle+"\t\t"+prix;
    }
}

public class Boisson extends Produit {
    private static final double
tva=0.15;
    public Boisson(String libelle,
double prix) {
        super(libelle, prix);
        System.out.println("Boisson");
    }
    @Override
    public String toString() {
        // TODO Auto-generated method
stub
        return super.toString()
+" \t\t"+tva;
    }
    public double getPrixTTC(){
        return getPrix()*(1+tva);
    }
}

public class Patisserie extends Produit {
    private static final double tva=0.3;
    public Patisserie(String libelle, double
prix) {
        super(libelle, prix);
        //TODO Auto-generated constructor
stub
    }
    @Override
    public String toString() {
        // TODO Auto-generated method stub
        return super.toString()+"\t\t"+tva;
    }
    public double getPrixTTC(){
        return getPrix()*(1+tva);
    }
}

public class Consommation {
    private Produit p;
    private int qte;
    public Consommation(Produit p, int qte)
{
        this.p = p;
        this.qte = qte;
    }
    public Produit getP() {
        return this.p;
    }
    public void setP(Produit p) {
        this.p = p;
    }
    public int getQte() {
        return this.qte;
    }
    public void setQte(int qte) {
        this.qte = qte;
    }
    @Override
    public String toString() {
        // TODO Auto-generated method stub
        return p.toString()+"\t\t"+qte;
    }
}
```

```

import java.util.Vector;
public class Command {
    private Vector <Consommation> vc=new Vector<>();
    private static double caisse;
    public void setVc(Vector<Consommation> vc) {
        this.vc = vc;
    }
    public static double getCaisse() {
        return caisse;
    }
    public int getTable() {
        return this.table;
    }
    public void setTable(int table) {
        this.table = table;
    }
    private int table;
    public Command(int table) {
        this.table=table;
    }
    public Command(Vector<Consommation> vc, int table) {
        this.vc = vc;
        this.table = table;
    }
    public void getTicket(){
        System.out.println("Table : "+table);
        System.out.println("Libelle\t\tPrix\t\tTva\t\tQte");
        double total=0;
        for (Consommation consommation : vc) {
            System.out.println(consommation);
            total=total+consommation.getQte()*consommation.getP().getPrixTTC();
        }
        caisse=caisse+total;
        System.out.println("\t\t\tTotal : "+total);
    }
    public Vector<Consommation> getVc(){
        return vc;
    }
}

```

TD 5 Héritage et Collections en Java

Pile et File sont généralement des structures utilisées pour mémoriser temporairement des transactions qui doivent attendre pour être traitées.

Une **pile** (en anglais **stack**) est une structure de données fondée sur le principe « dernier arrivé, premier sorti » (en anglais LIFO pour *last in, first out*), ce qui veut dire, qu'en général, le dernier élément, ajouté à la pile, sera le premier à en sortir.

Une **file** dite aussi **file d'attente** (en anglais **queue**), est une structure de données basée sur le principe du *premier entré, premier sorti*, (en anglais FIFO (« first in, first out ») ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à en être retirés.

Les primitives communément utilisées pour manipuler ces deux structures sont :

Pop : renvoie un élément et le retire de la structure (le premier dans le cas de la file et le dernier pour la pile)

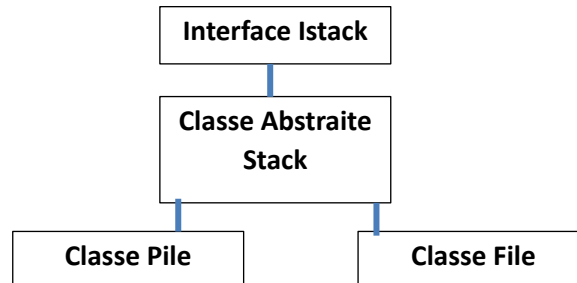
Push : ajoute un élément à la structure

isEmpty : renvoie vrai si la structure est vide

size : renvoie le nombre d'éléments dans la structure

erase : permet de vider la structure

1. En se basant sur le modèle UML suivant illustrant la relation d'héritage, donner les méthodes à mettre dans l'interface **Istack**. Quelles méthodes doivent être abstraites dans la classe Stack



2. Donner une implémentation Java des classes **Pile** et **File** en respectant votre proposition
3. Tester dans une classe **Main**

Correction TD 5

```
public interface Istack<T> {
    public T pop();
    void push(T o);
    boolean isEmpty();
    int size();
    void erase();
    boolean exist(T o);
}

import java.util.Vector;

public abstract class Stack implements Istack<T>{
    private Vector<T> stack=new Vector<>();
    public abstract T pop();
    @Override
    public void push(T o) {
        stack.add(o);
    }
    @Override
    public boolean isEmpty() {
        return stack.isEmpty();
    }
    @Override
    public int size() {
        return stack.size();
    }
    @Override
    public void erase() {
        stack.clear();
    }
    @Override
    public boolean exist(T o) {
        return stack.contains(o);
    }
}

package heritage;
public class Pile extends Stack {

    @Override
    public int pop() {
        int a =
tabElem.get(tabElem.size()-1);
tabElem.remove(tabElem.size()-1);
        return a;
    }
}

package heritage;
public class File extends Stack {

    @Override
    public int pop() {
        int a = tabElem.get(0);
        tabElem.remove(0);
        return a ;
    }
}
```

TD 6 Examen Modele de DS 1

DURÉE: 1H 30

Pour chaque morceau des codes suivants, vous allez dire s'il va s'exécuté correctement et pourquoi ?. Utiliser un tableau de trois colonnes (Question; OUI/NON; Justification)

A) `float f=String.valueOf("3.14");`
B) `char c;short i=30;c=i;`
C) `Boolean b=true;`
D) `String[] ts={"A","B"}; String[] ts1=ts;`
`//On supposera que la classe bus est déjà définie.`
E) `bus b ; System.out.println(b.toString());`
F) `bus [] tb=new bus[10] ; System.out.println(tb[0].toString());`
G) `abstract class A{void m(){} }`
H) `class A{abstract void m();}`
I) `interface I{} class A implements I{}`
`//Dans le main I a= new I();`

PROBLÈME

On vous propose dans ce problème de mettre en place une application java pour le compte d'une bibliothèque qui permettra a ses adhérents à travers le Net de lire un livre. On suppose que tous les livres sont scannés et que le texte des différentes pages est disponible.

A) Écrire une classe page définie par un numéro et un texte et qui implémente les méthodes suivantes :

- Un constructeur,
- `getText(), getNumero()`
- `search` : recherche un mot dans la page, s'il existe elle le retourne si non elle retourne null.

B) Écrire une classe Livre définie par un titre, un auteur et un ensemble de pages et qui implémente les méthodes suivantes :

- un constructeur qui permet de créer un livre avec un ensemble de pages
- **getPage** qui prend comme argument un numéro et retourne le texte de la page associée.
- **nextPage**: retourne la page suivante, si on est à la fin du livre on passe à la première
- **previousPage** : retourne la page précédente, si on est au début on passe à la dernière
- **search**: recherche un mot dans le livre, si celui-ci existe alors elle retourne toutes les pages qui le contient sinon elle retourne null.

C) Un Dictionnaire est un livre dont toute les pages sont triées, écrire la classe Dictionnaire !!!

Correction de TD DS 1

L'expression	Execution	Justification
<code>float f=String.valueOf("3.14");</code>	Non	Un String ne peut pas être un float
<code>float pi=3.14;</code>	Non	Un double vers un float, il faut un cast
<code>short i=30; int j=i;</code>	Oui	Un short est un int
<code>double f=3.45; float x=f;</code>	Non	Un double vers un float, il faut un cast
<code>String s=120;</code>	Non	Un int ne peut être un String
<code>String s=String.valueOf(13.4);</code>	Oui	S contiendra "13.4"
<code>int a="123";</code>	Non	Il faut convertir la chaîne vers int Integer.parseInt()
<code>int a= Integer.parseInt("JAVA");</code>	Non	"JAVA" n'est pas une chaîne numérique

```

class Page {
    private String text;
    private int num;
    public Page(String text,int num){
        this.text=text;
        this.num=num;
    }
    public String getText() {
        return text;
    }
    public void setText(String text) {
        this.text = text;
    }
    public int getNum() {
        return num;
    }
    public void setNum(int num) {
        this.num = num;
    }
    public String search(String word){
        if text.contains(word){
            return text;
        }
        else
            return null;
    }
}

```

```

class Livre {
    private Page[] pages;
    private String author;
    private String title;
    private int current=0;
    public Livre(Page[] pages,String author,String title){
        this.pages=pages;
        this.title=title;
        this.author=author;
    }
}

```

```

    }
    public String getPage(int num){
        for (int i=0;i<pages.length;i++){
            if(i==num) return pages[i].getText();
        }
        return null;
    }
    public String nextPage(){
        if (current==pages.length){
            current=0;
        }
        else {
            current++;
        }
        return pages[current].getText();
    }
    public String previousPage(){
        if (current==0){
            current=pages.length-1;
        }
        else {
            current--;
        }
        return pages[current].getText();
    }
    public Page[] search(String word){
        Page[] temp=new Page[pages.length];
        int j=0;
        for (int i=0;i<pages.length;i++){
            if (pages[i].search(word)!=null){
                temp[j]=pages[i];j++;
            }
        }
        return temp.length>=0:temp:null;
    }
}

```


TD 7 Fichiers

Objectif :

L'objectif est de mettre une classe java cmdLinux qui implémente quelques commandes de base Linux à savoir ls, cp, cat, wc, diff, tail, head, join et find. Toutes les méthodes seront statiques.

Rappel de syntaxe

diff file1.txt file2.txt ::: Affiche les lignes de différence sinon égalité

ls -l rep ::: le contenu du répertoire rep en format ls -l

cp file1 file2 ::: copie file1 dans file2

cat file ::: affiche le contenu de file en ajoutant le numéro de ligne

wc file ::: compte le nombre de mots, de lignes et de caractères de file

grep mot file :: cherche mot dans file s'il existe, elle affiche toutes les lignes contenant mot, sinon n'affiche rien

find rep file ::: affiche de façon récursive l'endroit de file dans rep

head n file ::: affiche les n premières lignes 10 par défaut

tail n file ::: affiche les n dernières lignes par défaut 10

join file1 file 2 ::: concatènera file 2 à file 1

Note

- Il faut traiter les différents cas d'erreurs (fichier inexistant, problèmes de droits)
- Penser à améliorer les différentes commandes

Correction TD 7

```
import java.io.*;
import java.util.Date;
import java.util.Vector;

public class cmdLinux {

    public static void diff(String st1, String st2) {
        File f1 = new File(st1);
        File f2 = new File(st2);
        try {
            if (!f1.exists()) {
                System.out.println(st1 + " Not found");
                System.exit(1);
            } else if (!f2.exists()) {
                System.out.println(st2 + " Not found");
                System.exit(1);
            }

            if (f1.isFile() && f2.isFile()) {
                String ligne1;
                String ligne2;
                System.out.println("\nFichier: " + f1.getName() + " Symbol:
+");
                cat(st1);
                System.out.println("\nFichier: " + f2.getName() + " Symbol:
-");
                cat(st2);
                System.out.println("\nLignes de différences: ");
                BufferedReader br1 = new BufferedReader(new FileReader(st1));
                BufferedReader br2 = new BufferedReader(new FileReader(st2));
                Vector<String> v1 = new Vector<String>();
                Vector<String> v2 = new Vector<String>();
                boolean eg=true;
                while ((ligne1 = br1.readLine()) != null) {
                    v1.add(ligne1);
                }
                while ((ligne2 = br2.readLine()) != null) {
                    v2.add(ligne2);
                }
                for (String e : v1) {
                    if (!v2.contains(e)) {
                        System.out.println("+ " + e);
                        eg=false;
                    }
                }
                for (String e : v2) {
                    if (!v1.contains(e)) {
                        System.out.println("- " + e);
                        eg=false;
                    }
                }
                if(eg) {
                    System.out.println("Aucune. Les fichiers sont égaux.");
                }

                br1.close();
                br2.close();
            } else
                System.out.println("L'un des fichiers est un repertoire");

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public static void ls(String rep) {
        File f1 = new File(rep);
    }
}
```

```

    if (!f1.canRead()) {
        System.out.println(rep + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        if (!f1.exists()) {
            System.out.println(rep + " Not found");
            System.exit(1);
        }
        String w = "-";
        String r = "-";
        String x = "-";
        String t = "-";
        if (f1.isDirectory()) {
            File[] files = f1.listFiles();

            for (File e : files) {
                if (f1.isDirectory()) {
                    t = "d";
                }
                if (e.canExecute()) {
                    x = "x";
                }
                if (e.canRead()) {
                    r = "r";
                }
                if (e.canWrite()) {
                    w = "w";
                }
                if (e.isFile()) {
                    t = "-";
                }
                System.out.println(t + r + w + x + "\t" +
e.getAbsolutePath() + "\t" + new Date(e.lastModified())
                    + "\t" + e.getName());
            }
        } else {
            if (f1.canExecute()) {
                x = "x";
            }
            if (f1.canRead()) {
                r = "r";
            }
            if (f1.canWrite()) {
                w = "w";
            }
            System.out.println(t + r + w + x + "\t" + f1.getAbsolutePath()
+ "\t" + new Date(f1.lastModified())
                + "\t" + f1.getName());
        }
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public static void cat(String st1) {
    File f = new File(st1);
    if (!f.exists()) {
        System.out.println(st1 + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st1 + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        Vector<String> v = new Vector<String>();
        String ligne;

```

```

        int i = 1;
        while ((ligne = br.readLine()) != null) {
            System.out.println(i + " " + ligne);
            i++;
        }
        br.close();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public static void find(String rep, String fich) {
    File f = new File(rep);
    if (!f.exists()) {
        System.out.println(rep + " Not found");
        System.exit(1);
    }
    if (f.isDirectory()) {
        File[] files = f.listFiles();
        for (File e : files) {
            if (e.getName().contains(fich)) {
                System.out.println(e);
            }
            if (e.isDirectory()) {
                find(e.getAbsolutePath(), fich);
            }
        }
    }
}

public static void head(String st, int n) {
    File f = new File(st);
    if (!f.exists()) {
        System.out.println(st + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        String ligne;
        int i = 0;
        while ((ligne = br.readLine()) != null && i < n) {
            System.out.println(ligne);
            i++;
        }
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void head(String st) {
    File f = new File(st);
    if (!f.exists()) {
        System.out.println(st + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        String ligne;
        int i = 0;
        while ((ligne = br.readLine()) != null && i < 10) {
            System.out.println(ligne);
            i++;
        }
        br.close();
    }
}

```

```

    } catch (IOException e) {
        e.printStackTrace();
    }
}
public static void tail(String st) {
    File f = new File(st);
    if (!f.exists()) {
        System.out.println(st + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        Vector<String> v = new Vector<String>();
        String line;
        while ((line = br.readLine()) != null) {
            v.add(line);
        }

        int i;
        for (i = 10; i >= 1; i--) {
            System.out.println(v.get(v.size() - i));
        }
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public static void tail(String st, int n) {
    File f = new File(st);
    if (!f.exists()) {
        System.out.println(st + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        Vector<String> v = new Vector<String>();
        String line;
        while ((line = br.readLine()) != null) {
            v.add(line);
        }

        int i;
        for (i = n; i >= 1; i--) {
            System.out.println(v.get(v.size() - i));
        }
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
public static void join(String st1, String st2) {
    File f1 = new File(st1);
    File f2 = new File(st2);
    if (!f1.exists()) {
        System.out.println(st1 + " Not found");
        System.exit(1);
    } else if (!f2.exists()) {
        System.out.println(st2 + " Not found");
        System.exit(1);
    }
    if (!f1.canRead()) {

```

```

        System.out.println(st1 + " n'a pas le droit de lecture");
        System.exit(1);
    }
    if (!f2.canRead()) {
        System.out.println(st2 + " n'a pas le droit de lecture");
        System.exit(1);
    }
    if (!f1.canWrite()) {
        System.out.println(st1 + " n'a pas le droit d'écriture");
        System.exit(1);
    }
    try {
        BufferedReader br1 = new BufferedReader(new FileReader(f1));
        BufferedReader br2 = new BufferedReader(new FileReader(f2));
        String line1, line2;
        Vector<String> v = new Vector<String>();
        while ((line1 = br1.readLine()) != null) {
            v.add(line1);
        }
        BufferedWriter bw = new BufferedWriter(new FileWriter(f1));
        int i = 0;
        while ((line2 = br2.readLine()) != null) {
            if (i < v.size()) {
                v.set(i, v.get(i) + " " + line2);
                i++;
            } else
                v.add(line2);
        }
        for (String e : v) {
            bw.append(e);
            bw.newLine();
        }
        br1.close();
        br2.close();
        bw.close();
        cmdLinux.cat(st1);
    } catch (IOException e) {
        System.out.println("erreur");
    }
}

public static void wc(String st) {
    File f = new File(st);
    if (!f.exists()) {
        System.out.println(st + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st + " n'a pas le droit de lecture");
        System.exit(1);
    }
    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        String line;
        int nb_chars = 0, nb_words = 0, nb_lines = 0;
        Vector<String> v = new Vector<String>();
        while ((line = br.readLine()) != null) {
            String[] words = line.split("\\s+");
            for (int i = 0; i < words.length; i++) {
                v.add(words[i]);
            }
            nb_lines++;
        }
        nb_words = v.size();
        for (String e : v) {
            nb_chars += e.length();
        }
        br.close();
        cat(st);
        System.out.println("Mots: " + nb_words + " Caractères: " + nb_chars +
            " Lignes: " + nb_lines);
    }
}

```

```

    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public static void grep(String w, String st) {
    File f = new File(st);
    if (!f.exists()) {
        System.out.println(st + " Not found");
        System.exit(1);
    }
    if (!f.canRead()) {
        System.out.println(st + " n'a pas le droit de lecture");
        System.exit(1);
    }

    try {
        BufferedReader br = new BufferedReader(new FileReader(f));
        String line;
        CharSequence word = w;
        while ((line = br.readLine()) != null) {
            if (line.contains(word)) {
                System.out.println(line);
            }
        }
        br.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void cp(String st1, String st2) {
    File f1 = new File(st1);
    File f2 = new File(st2);
    try {
        if (!f1.exists()) {
            f1.createNewFile();
            System.out.println(f1.getName() + " Créé");
        }
        if (!f2.exists()) {
            f2.createNewFile();
            System.out.println(f2.getName() + " Créé");
        }
        if (!f1.canRead()) {
            System.out.println(st1 + " n'a pas le droit de lecture");
            System.exit(1);
        }
        if (!f2.canRead()) {
            System.out.println(st2 + " n'a pas le droit de lecture");
            System.exit(1);
        }
        if (!f2.canWrite()) {
            System.out.println(st2 + " n'a pas le droit d'écriture");
            System.exit(1);
        }

        FileReader br = new FileReader(f1);
        FileWriter bw = new FileWriter(f2);
        int c;
        while ((c = br.read()) != -1) {
            bw.write((char) c);
        }
        System.out.println("Fichier " + f1.getName() + " copié avec
succès.");
        br.close();
        bw.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
}
}

```

TD 8 Interface d'authentification depuis un fichier texte/Correction

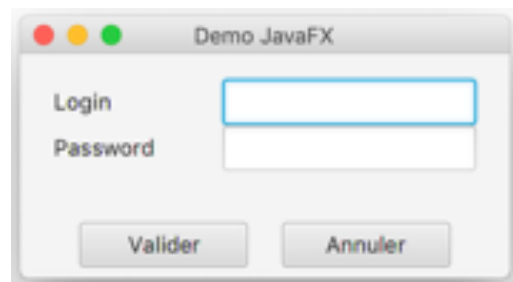
L'objectif est de mettre une interface d'authentification standard basée sur le login et le mot de passe avec deux boutons :

- valider : permet de checker les identifiants depuis un fichier texte contenant les users. Dans le cas où le user existe on affichera un message de bien venu sinon, on affiche une erreur.

Les lignes du fichier contenant les identifiants des utilisateurs est de la forme suivante :

Login;password

- Annuler : permet de vider les champs



```
+++++
package cours.java.graphique;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTextField;

public class Authentification {

    //etape1: Enumerer et declarer les objets
    private JFrame frmAuth;
    private JLabel lbllogin, lblpwd;
    private JTextField txtlogin, txtpwd;
    private JButton btnCon, btnCan;

    public Authentification()
    {
        //Initialisation
        frmAuth=new JFrame("Authentification");
        lbllogin=new JLabel("Login");
        lblpwd=new JLabel("Password");
        txtlogin=new JTextField(10);
        txtpwd=new JTextField(10);
        btnCan=new JButton("Annuler");
        btnCon=new JButton("Connection");
        //Mise en page et ajout
        frmAuth.setLayout(new GridLayout(3, 2));
        frmAuth.add(lbllogin);frmAuth.add(txtlogin);
        frmAuth.add(lblpwd);frmAuth.add(txtpwd);
        frmAuth.add(btnCon);frmAuth.add(btnCan);
    }
}
```



```

btnCan.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        txtlogin.setText("");
        txtpwd.setText("");
    }
});

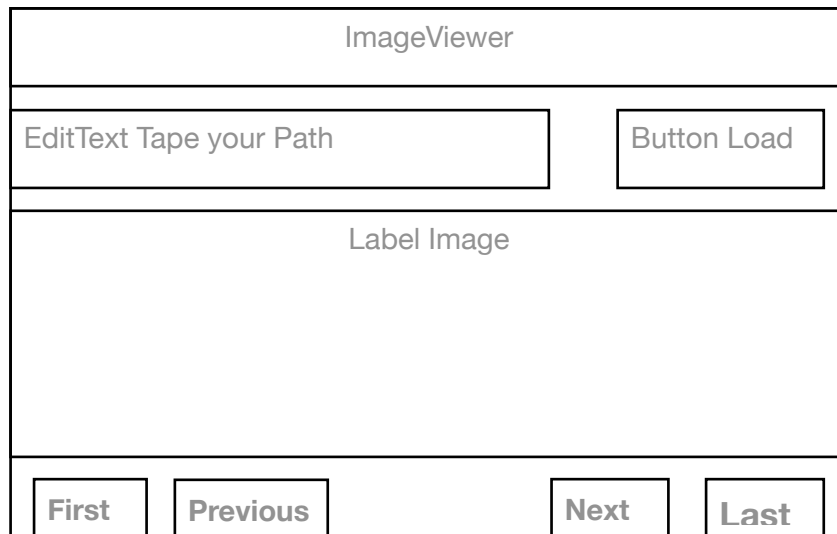
btnCon.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
        File f= new File("/Users/mohammedlahmer/Desktop/Java-2017/coursIL/src/
coursIL/java/graphique/users.txt");
        if(!f.exists())
        {
            JOptionPane.showMessageDialog(null, "Erreur fichier");
            System.exit(0);
        }
        try{
            FileReader fr=new FileReader(f);
            BufferedReader br=new BufferedReader(fr);
            String ligne="";
            while((ligne=br.readLine())!=null)
            {
                String [] T=ligne.split(":");
                if(T[0].equals(txtlogin.getText()) &&
T[1].equals(txtpwd.getText()))
                {
                    JOptionPane.showMessageDialog(null, "Bien venu :"+T[0]);
                    return ;
                }
            }
            br.close();fr.close();
            JOptionPane.showMessageDialog(null, "Login Incorrecte");
        }catch(Exception e1){e1.printStackTrace();}
    }
});
frmAuth.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
//size de la fenetre
frmAuth.pack();
//Position
frmAuth.setLocationRelativeTo(null);
//visibilite
frmAuth.setVisible(true);
}

}

```

TD 9 Image Viewer en JavaFx (sans FXML)

Mettre en place une application graphique pour visionner et naviguer sur des images stockées dans un repertoire que vous allez crée.



Méthodes à realiser :

- Load (String Path) : permet de charger la premiere image dans la zone réservée
- Next() : Charge l'image suivante, une fois on arrive à la dernière en charge la première
- Previous() : Charge l'image précédente dans l'ordre, une fois on arrive à la première en charge la dernière
- Last() : charge la dernière image
- First() : Charge la première image

En cas ou l'utilisateur saisie un chemin incorrecte ou vide on affiche une erreur dans une boite de dialogue.

NB

La taille de l'image stockée est généralement différente de la zone où on va l'afficher alors il faut penser à adapter la taille.

CorrectionTD 9

```
package dut.jfx.imageviewer;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import java.io.IOException;

public class App extends Application {
    private static Scene scene;
    private Button btnnext=new Button(">");
    private Button btnprevious=new Button("<");
    private Button btnfirst=new Button("|<");
    private Button btnlast=new Button(">|");
    private Button btngo=new Button("go");
    private TextField path=new TextField();
    private Label lblindex=new Label("");
    private Label lblimage=new Label("");
    private BorderPane mainPane=new BorderPane();
    private HBox hbBottom=new HBox(btnfirst,btnprevious,lblindex,btnnext,btnlast);
    private HBox hbTop=new HBox();
    private ModelImage modelImage;
    private int index=0;

    @Override
    public void start(Stage stage) throws IOException {

        scene=new Scene(mainPane);
        mainPane.setPadding(new Insets(10));
        hbTop.setAlignment(Pos.CENTER);
        hbTop.setSpacing(10);
        hbBottom.setSpacing(10);
        hbBottom.setAlignment(Pos.CENTER);
        hbTop.getChildren().add(path);hbTop.getChildren().add(btngo);
        mainPane.setTop(hbTop);
        mainPane.setCenter(lblimage);
        mainPane.setBottom(hbBottom);
        lblimage.setPrefSize(400,250);
        btngo.setOnAction(e->{
            String Imagepath=path.getText().toString();
            modelImage=new ModelImage(Imagepath);
            System.out.println(Imagepath);
            //ImageView icon=new ImageView("file:///"+Imagepath);
            lblimage.setGraphic(modelImage.getVimages().get(index));
        });

        btnnext.setOnAction(e->{
            index++;
            if (index==modelImage.getVimages().size())
                index=0;

            lblimage.setGraphic(modelImage.getVimages().get(index));
        });
        stage.setTitle("ImageViewer");
        stage.setScene(scene);
        stage.centerOnScreen();
        stage.show();
    }
    public static void main(String[] args) {
        launch();
    }
}
```

```

    }
}

package dut.jfx.imageviewer;
import java.io.File;
import java.util.Vector;
import javafx.scene.image.ImageView;
public class ModelImage {

    private Vector<ImageView> vimages=new Vector<>();
    public Vector<ImageView> getVimages() {
        return vimages;
    }
    public ModelImage(String path){
        try{
            File d=new File(path);
            System.out.println(path);

            File[] l=d.listFiles();

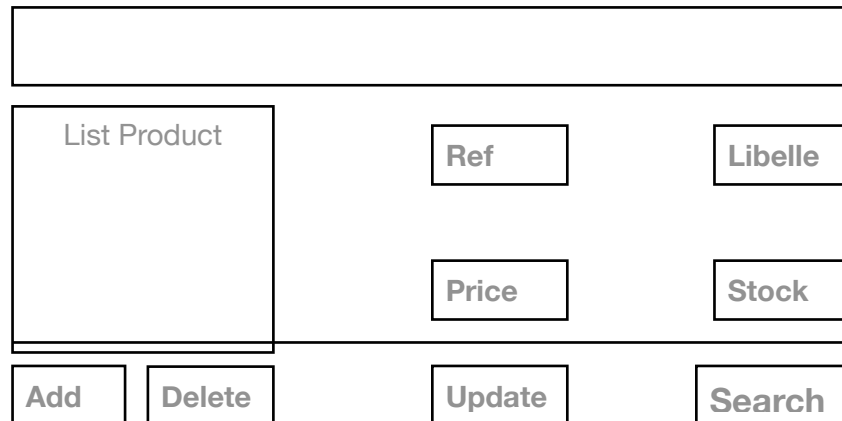
            for (File f:l)
            {
                vimages.add(new ImageView("file:///"+f.getAbsolutePath()));
            }
        }catch(Exception e){
            System.out.println("Problem file");
        }
    }
}

```

TD 10 Gestion Produit/Base de données

Mettre en place une application graphique Java (MVC) avec accès aux bases de données pour la gestion des produit (Implementation des opérations CRUD). Vous allez créer trois packages :

- Package dal (Data Access Layer) : Contient les classes d'accès à la base de données
- Package business : Contient les classes représentant la logique de l'application
- Package presentation : Contient les classes de la vue graphique (voir figure ci-dessous)



Une fois lancée, le libelles de tous les produits de la base de données sont affichés dans la liste. Lorsque on sélectionne un produit, le détail associé est affiché dans la partie detail. Les fonctionnalités à implementer sont :

- Add : une fois cliqué, son nom deviendra save et les autres boutons seront désactivés. Les zones de saisie seront vidées. lorsqu'on remplit les champs, on clique sur save et le bouton redeviendra Add tout en activant les autres
- Delete : permet de supprimer le produit affiché de la liste et de la base de données bien sûr
- Update : permet de modifier le produit en cours à l'exception de la référence (clé de la table)
- Search : une fois cliqué, un inputBox est affiché dans lequel on saisie la référence à chercher

NB

Il faut prendre en considération toutes les erreurs possibles (stocke ou prix negative, produit n'existe pas, produit existe déjà cas d'ajout, ...)

Correction TD 10

package presentation;

```
import java.util.Vector;
import javax.swing.AbstractListModel;
import dal.Produit;
import dal.ProduitDao;
public class ProduitModel extends AbstractListModel<Produit> {
    Vector<Produit> model = new Vector<>();
    ProduitDao dao = new ProduitDao();
    public ProduitModel() {
        model = dao.getAll("products.txt");
    }
    @Override
    public int getSize() {

        return model.size();
    }
    @Override
    public Produit getElementAt(int index) {
        // TODO Auto-generated method stub
        return model.get(index);
    }
    public void add(Produit p) {
        model.add(p);
        System.out.println("added");
        for (Produit d : model)
            System.out.println(d.getLibelle());
        System.out.println("---");
        fireIntervalAdded(this, 0, getSize());
    }
    public void save() {
        dao.save("products.txt", model);
    }
    public boolean update(Produit p) {
        int i=-1;
        for (Produit d : model) {
            if (p.getReference().equals(d.getReference())) {
                i=model.indexOf(d);
            }
        }
        if(i!=-1) {
            model.set(i, p);
            fireContentsChanged(this,0,getSize());
            return true;
        }
        return false;
    }
    public Produit search(String ref) {
        for (Produit d : model) {
            if (d.getReference().equals(ref)) {
                return d;
            }
        }
        return null;
    }

    public boolean delete(Produit d) {
        int i=-1;
        i=model.indexOf(d);
        if(i!=-1) {
            model.remove(i);
            fireIntervalRemoved(this,0,this.getSize());
            return true;
        }
        return false;
    }
}
+++++
```

```

package presentation;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.GridBagLayout;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.border.Border;
import javax.swing.border.EmptyBorder;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import dal.Produit;
import presentation.ProduitModel;
//Benali Yasser – Section 1 – Groupe TP2
public class Gui extends JFrame {

    private ImageIcon searchIcon = new ImageIcon("search.png");
    private ImageIcon delIcon = new ImageIcon("delete.png");
    private ImageIcon addIcon = new ImageIcon("add.png");
    private ImageIcon updateIcon = new ImageIcon("update.png");
    private JButton addButton = new JButton("Add", addIcon);
    private JButton updateButton = new JButton("Update", updateIcon);
    private JButton delButton = new JButton("Delete", delIcon);
    private JButton searchButton = new JButton("Search", searchIcon);
    private JLabel refLab = new JLabel("Reference");
    private JLabel priceLab = new JLabel("Prix");
    private JLabel libeLab = new JLabel("Libelle");
    private JLabel quantLab = new JLabel("Quantity");
    private JTextField refText = new JTextField(10);
    private JTextField priceText = new JTextField(10);
    private JTextField libeText = new JTextField(10);
    private JTextField quantText = new JTextField(10);
    private JMenuBar mBar = new JMenuBar();
    private JMenu mFile = new JMenu("File");
    private JMenuItem mOpen = new JMenuItem("Open");
    private JMenuItem mSave = new JMenuItem("Save");
    private JMenuItem mExit = new JMenuItem("Exit");
    private JList<Produit> products = new JList<Produit>();
    private JScrollPane scroll = new JScrollPane();
    private JPanel pnlCenter = new JPanel();
    private JPanel pnlSouth = new JPanel();
    private ProduitModel produitModel;
    private boolean op = false;
    private ImageIcon iconGestion = new ImageIcon("icon.png");

    public Gui() {
        // frame
        this.setIconImage(iconGestion.getImage());
        this.setLayout(new BorderLayout());
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
        this.setTitle("Gestion Produits");
        // MenuBar
        mFile.add(mOpen);
        mFile.add(mSave);
        mFile.add(mExit);
        mBar.add(mFile);
    }

```

```

this.add(mBar, BorderLayout.NORTH);
// panelCenter
pnlCenter.setLayout(new GridLayout(2, 4, 10, 10));
pnlCenter.add(refLab);
pnlCenter.add(refText);
pnlCenter.add(libelab);
pnlCenter.add(libetext);
pnlCenter.add(priceLab);
pnlCenter.add(pricetext);
pnlCenter.add(quantLab);
pnlCenter.add(quantText);
this.add(pnlCenter, BorderLayout.CENTER);
Border padding = BorderFactory.createEmptyBorder(10, 10, 10, 10);
pnlCenter.setBorder(padding);
// panelSouth
pnlSouth.setLayout(new FlowLayout());
pnlSouth.add(addButton);
pnlSouth.add(updateButton);
pnlSouth.add(delButton);
pnlSouth.add(searchButton);
this.add(pnlSouth, BorderLayout.SOUTH);
scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
scroll.setPreferredSize(new Dimension(80, 80));
scroll.setViewportView(products);
this.add(scroll, BorderLayout.WEST);
this.pack();
this.setLocationRelativeTo(null);
try {
    mExit.addActionListener(e -> {
        System.exit(0);
    });
    mOpen.addActionListener(e -> {
        produitModel = new ProduitModel();
        products.setModel(produitModel);
        op = true;
    });

    mSave.addActionListener(e -> {
        produitModel.save();
        JOptionPane.showMessageDialog(null, "Succs", "Produits Enregistrs",
JOptionPane.INFORMATION_MESSAGE);
    });
    products.addListSelectionListener(new ListSelectionListener() {

        @Override
        public void valueChanged(ListSelectionEvent e) {
            // TODO Auto-generated method stub
            if (e.getValueIsAdjusting()) {
                refText.setText(products.getSelectedValue().getReference());
                libetext.setText(products.getSelectedValue().getLibelle());
                pricetext.setText(products.getSelectedValue().getPrice() + "");
                quantText.setText(products.getSelectedValue().getQuantity() +
"");
            }
        }

    });

    addButton.addActionListener(e -> {
        if (!op) {
            JOptionPane.showConfirmDialog(null, "Vous devez ouvrir un fichier!",
"Erreur",
                JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
        } else {
            Produit p = new Produit();
            p.setReference(refText.getText());
            p.setLibelle(libetext.getText());
            p.setPrice(Double.parseDouble(pricetext.getText()));
            p.setQuantity(Double.parseDouble(quantText.getText()));
            produitModel.add(p);
            refText.setText("");
            libetext.setText("");

```



```

        priceText.setText("");
        quantText.setText("");
    }
});

updateButton.addActionListener(e -> {
    if (!op) {
        JOptionPane.showConfirmDialog(null, "Vous devez ouvrir un fichier!",
"Erreur",
        JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
    } else {
        boolean cont = true;
        Produit p = new Produit();
        try {
            p.setReference(refText.getText());
            p.setLibelle(libText.getText());
            p.setPrice(Double.parseDouble(priceText.getText()));
            p.setQuantity(Double.parseDouble(quantText.getText()));
            if (!produitModel.update(p)) {
                JOptionPane.showMessageDialog(this, "Introuvable", "Produit
introuvable",
                JOptionPane.ERROR_MESSAGE);
                cont = false;
            }
            if (cont) {
                JOptionPane.showMessageDialog(this, "Product updated",
"Succs",
                JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (NumberFormatException e1) {
            JOptionPane.showConfirmDialog(null, "Vous devez entrez des
donnes!", "Erreur",
                JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
        }
    }
});

delButton.addActionListener(e -> {
    if (!op) {
        JOptionPane.showConfirmDialog(null, "Vous devez ouvrir un fichier!",
"Erreur",
        JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
    } else {
        if (produitModel.delete(products.getSelectedValue())) {
            JOptionPane.showMessageDialog(this, "Produit supprim", "Succs",
                JOptionPane.INFORMATION_MESSAGE);
            refText.setText("");
            libText.setText("");
            priceText.setText("");
            quantText.setText("");
        } else {
            JOptionPane.showMessageDialog(this, "Introuvable", "Produit
introuvable",
                JOptionPane.ERROR_MESSAGE);
        }
    }
});

searchButton.addActionListener(e -> {
    if (!op) {
        JOptionPane.showConfirmDialog(null, "Vous devez ouvrir un fichier!",
"Erreur",
        JOptionPane.DEFAULT_OPTION, JOptionPane.ERROR_MESSAGE);
    } else {
        JFrame searchFrame = new JFrame();
        searchFrame.setTitle("Search");
        searchFrame.setLayout(new BorderLayout());
        searchFrame.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        searchFrame.setLocationRelativeTo(null);
        searchFrame.setVisible(true);
        JPanel searchInfo = new JPanel();
        JLabel searchLab = new JLabel("Enter reference: ");
        JTextField searchText = new JTextField(10);
    }
});

```

```

        searchInfo.setBorder(new EmptyBorder(10, 10, 10, 10));
        searchInfo.setLayout(new FlowLayout());
        searchInfo.add(searchLab);
        searchInfo.add(searchText);
        JButton searchButton = new JButton("Search");
        searchFrame.add(searchInfo, BorderLayout.CENTER);
        searchFrame.add(searchButton, BorderLayout.SOUTH);
        searchFrame.pack();
        searchFrame.setLocationRelativeTo(null);
        searchButton.addActionListener(b -> {
            Produit p = produitModel.search(searchText.getText());
            boolean cont = true;

            if (p == null) {
                JOptionPane.showMessageDialog(searchFrame, "Introuvable",
"Produit introuvable",
                JOptionPane.ERROR_MESSAGE);
                cont = false;
            }
            if (cont) {
                JOptionPane.showMessageDialog(searchFrame,
"Reference: " + p.getReference() + "\n" + "Libelle:
" + p.getLibelle() + "\n"
                + "Prix: " + p.getPrice() + "\n" +
"Quantity: " + p.getQuantity(),
                "Rsultat", JOptionPane.INFORMATION_MESSAGE);
            }
        });
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Erreur", "Une erreur s'est produite",
JOptionPane.ERROR_MESSAGE);
        e.printStackTrace();
    }
}

}
+++++
package dal;
public class Produit {

    private String reference;
    private String libelle;
    private double price;
    private double quantity;
    public String getReference() {
        return reference;
    }
    public void setReference(String reference) {
        this.reference = reference;
    }
    public String getLibelle() {
        return libelle;
    }
    public void setLibelle(String libelle) {
        this.libelle = libelle;
    }
    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public double getQuantity() {
        return quantity;
    }
    public void setQuantity(double quantity) {
        this.quantity = quantity;
    }
}

```

```

        public String toString() {
            return libelle;
        }
    }
}

```

package dal;

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.Vector;

public class ProduitDao {
    public Vector<Produit> getAll(String path) {
        Vector<Produit> vp = new Vector<>();
        try {
            FileReader fr = new FileReader(path);
            BufferedReader br = new BufferedReader(fr);
            String line;
            Produit p;
            while ((line = br.readLine()) != null) {
                p = new Produit();
                String[] t = line.split(";");
                p.setReference(t[0]);
                p.setLibelle(t[1]);
                p.setPrice(Double.parseDouble(t[2]));
                p.setQuantity(Double.parseDouble(t[3]));
                vp.add(p);
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        return vp;
    }

    public void save(String path, Vector<Produit> v) {
        try {
            FileWriter fw = new FileWriter(path);
            BufferedWriter bw = new BufferedWriter(fw);
            for (Produit e : v) {
                System.out.println(e);
                bw.write(e.getReference() + ";");
                bw.write(e.getLibelle() + ";");
                bw.write(String.valueOf(e.getPrice()) + ";");
                bw.write(String.valueOf(e.getQuantity()));
                bw.newLine();
            }
            bw.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    public Produit search(String path, String ref) {
        try {
            Vector<Produit> vp = getAll(path);
            for(Produit e:vp) {
                if(e.getReference().equals(ref)) {
                    return e;
                }
            }
        } catch (Exception e) {
            // TODO Auto-generated catch block

```

```
        e.printStackTrace();
    }
    return null;
}
```

Partie II Travaux pratiques avec Solution

TP 1 Révision et prise en main du langage Java

Objectif :

Prise en main de l'environnement de développement vscode et se familiariser avec la syntaxe Java : notion de classe, méthodes statiques.

Calcul de la racine carrée

Il existe plusieurs algorithmes pour calculer la racine carrée d'un nombre positif. En voici un:

1) Sachant que pour calculer la racine carrée de a revient à résoudre l'équation $x^2=a$. On peut transformer celle-ci en:

$$x^2=a \quad ? \quad 2x^2=x^2+a \quad ? \quad x=(x^2+a)/(2x) \quad ? \quad x=(x+a/x)/2$$

L'idée a été de distinguer les x suivant les membres où ils se trouvaient pour donner la formule:

$$x_{i+1} = \frac{1}{2} \left(x_i + \frac{a}{x_i} \right)$$

Écrire un programme qui prend en paramètre un réel x et affiche sa racine carrée.

NOTE: le calcul peut commencer avec n'importe quelle valeur de x (différente de 0).

Paramètres du main

Dans l'en-tête d'une méthode **main** apparaît ce que l'on appelle un paramètre ou un argument nommé `arg` :

public static void main(String[] arg)

Cet argument est de type tableau de String ou encore tableau de chaînes de caractères dont le premier élément est accessible par `arg[0]` et ainsi de suite. Ecrire un programme qui prend en paramètre n et m et :

1. vérifie qu'il y exactement deux paramètres si non affiche une erreur
2. affiche m n fois
3. affiche m entouré de n parenthèses. Exemple 3 4 ? (((4)))

Calcul du zéro d'une fonction

On souhaite calculer la valeur approchée d'une racine d'une équation de la forme : $f(x) = 0$

On utilise pour cela une méthode de dichotomie. La méthode suppose que l'on parte de 2 valeurs x_1 et x_2 encadrant une racine, et telle que $f(x_1)$ et $f(x_2)$ soient de signe contraire. On calcule $c = (x_1 + x_2)/2$ et on remplace x_1 (respectivement x_2) par c suivant que $f(c)$ est du même signe que $f(x_1)$ (respectivement $f(x_2)$). La largeur de l'intervalle considéré est donc divisée par 2 à chaque pas. Le calcul est répété jusqu'à ce que cette largeur soit inférieure à une valeur donnée.

Solution TP 1

```
public static void dichotomie(double a, double b) {
    double epsilon = 1e-6;

    if (Math.abs(b - a) < epsilon) {
        System.out.println("La valeur de la racine est " + (a + b) / 2);
    } else {
        if (function((a + b) / 2) * function(a) < 0) {
            dichotomie((a + b) / 2, a);
        } else if (function((a + b) / 2) * function(a) > 0) {
            dichotomie((a + b) / 2, b);
        } else {
            System.out.println("la valeur est " + (a + b) / 2);
        }
    }
}

}

public static double calculRacineCarree(double x) {
    double a = x;
    for (int i = 0; i < 10; i++) {
        a = (a + x / a) / 2.0;
    }

    return a;
}

public static void function(int n, int m) {
    for (int i = 1; i <= n; i++) {
        System.out.printf("[");
    }
    System.out.printf("%d", m);
    for (int i = 1; i <= n; i++) {
        System.out.printf("]");
    }
}

}

public static void dichotomie(double a, double b) {
    double epsilon = 1e-6;

    if (Math.abs(b - a) < epsilon) {
        System.out.println("La valeur de la racine est " + (a + b) / 2);
    } else {
        if (function((a + b) / 2) * function(a) < 0) {
            dichotomie((a + b) / 2, a);
        } else if (function((a + b) / 2) * function(a) > 0) {
            dichotomie((a + b) / 2, b);
        } else {
            System.out.println("la valeur est " + (a + b) / 2);
        }
    }
}

}
```

TP 2 Classes / Objets

Dans cet exercice, vous serez amenés à créer une classe que vous appellerez *Personne*. Cette classe aura des variables correspondant aux caractéristiques suivantes :

Une variable de classe qui sera déclarée *private*, qui s'appellera *num* et qui stockera le nombre des bonhommes existants.

Une variable d'instance qui stockera une chaîne de caractère qui représente la race, et qui sera d'un type *Race* à créer, et qui est une chaîne de caractères qui ne peut prendre que l'une des valeurs "Africain", "Caucasien", "Asiatique", "Indien Américain", ou "Métis".

Des variables d'instance de type *int* qui stockerons les valeurs de l'âge, du poids, de la taille, et d'un numéro d'identification.

Des variables de type *String* qui stockera le nom, le prénom et la profession. Une fois que les principales caractéristiques de la classe seront définies, vous aurez à définir des comportements de la classe. Cela concernera d'abord les constructeurs suivants :

Un constructeur qui créera une instance de *Bonhomme* qui représentera un bonhomme à sa naissance. Ce constructeur ne prendra aucun argument, et initialisera uniquement les variables suivantes avec les valeurs décrites : *num*++, *age* = 0, *poids* = 3, *taille* = 50, *identification* = *num*.

Un constructeur qui permettra de créer une instance de *Bonhomme* avec toutes les caractéristiques définies, c'est à dire qui prendra toutes les données nécessaires pour initialiser toutes les variables comme argument (sauf *num* qui sera incrémenté et l'identification qui prendra la nouvelle valeur de *num*)

Ensuite, vous aurez à créer des méthodes qui vont accomplir les tâches suivantes :

Une méthode *baptême()* qui prend comme argument le nom et le prénom pour les affecter aux variables correspondantes.

Une méthode *anniversaire()* qui incrémente l'âge.

Une méthode *embauche()* qui prend comme argument la profession, et l'affecte à la variable correspondante.

Une méthode ***ToString*** qui affiche les caractéristiques de l'individu. son implémentation sera la suivante

Complement Tableau :

Une classe *CNSS* composée de plusieurs *Adhérent*. Un *Adhérent* est une *Personne* avec *Profession* et une date d'adhésion

Définir une méthode *add(Adherent adherent)* qui ajoute un adhérent à la *CNSS* et une méthode *list()* qui listera les adhérents triés par date d'adhésion.

Solution TP 2

```
public class Personne {
    private static int num;
    private Race race;
    private int age, poids, taille, identification;
    private String nom, prenom, profession;

    public Personne() {
        num++;
        age = 0;
        poids = 3;
        taille = 50;
        identification = num;
    }

    public Personne(Race race, int age, int poids, int taille, String nom, String
prenom, String profession) {
        num++;
        this.race = race;
        this.age = age;
        this.poids = poids;
        this.taille = taille;
        this.identification = num;
        this.nom = nom;
        this.prenom = prenom;
        this.profession = profession;
    }

    // getters et setters omis

    @Override
    public String toString() {
        return String.format("%1$-3d %2$-10s %3$-10s %4$-10s %5$-3d %6$-3d %7$-3d
%8$-10s",
            identification, (nom == null ? "UNKNOWN" : nom.toUpperCase()), prenom, race,
            age, taille, poids, profession);
    }
}

public enum Race {
    AFRICAINE,
    CAUCASIEN,
    ASIATIQUE,
    INDIEN,
    METIS
}

import java.time.LocalDate;
public class Adherent{
    private Personne bonhomme;
    private LocalDate date_cnss;

    public Adherent(Personne bonhomme, LocalDate date_cnss) {
        this.bonhomme = bonhomme;
        this.date_cnss = date_cnss;
    }

    // la methode compareTo pour comparer 2 Adherent selon id

    public int compareTo(Adherent a) {
        return (this.bonhomme.getIdentification() -
a.getBonhomme().getIdentification());
    }

    public String toString() {
        return bonhomme.toString() + date_cnss.getYear() + "-" + date_cnss.getMonth()
+"-" + date_cnss.getDayOfMonth();
    }
}
```



```

}

public class Cnss {
    private Adherent[] adherants;
    public static final int MAX = 100;
    private int index = 0;
    public Cnss() {
        adherants = new Adherent[MAX];
    }

    // la methode add pour ajouter un adherent au tableau des adherents
    public void add(Adherent new_adherent) {
        if(index == MAX){
            System.out.println("vous avez depasser le max");
            return;
        }
        if(new_adherent.getBonhomme().getProfession() != null) {
            adherants[index] = new_adherent;
            index++;
        }
        else{
            System.out.println("on ne peut pas ajouter cette adherent car profession is
null !!!");
        }
    }

    // la methode lister pour afficher les adherents trie selon l'id d'un ordre croissant
    public void lister() {
        Adherent tmp;
        for(int i=0;i<index-1;i++) {
            int min = i;
            for(int j=i+1;j<index;j++) {
                if(adherants[min].compareTo(adherants[j]) > 0)
                    min = j;
            }
            if(min != i){
                tmp = adherants[min];
                adherants[min] = adherants[i];
                adherants[i] = tmp;
            }
        }
        System.out.println("***** Les adherents trie : *****");
        for(int i=0;i<index;i++){
            System.out.println(adherants[i].toString());
        }
    }
}

```

TP 3 sur les Classes particulières String, Integer

Objectif : Mettre en place une application java pour les calculs d'adresses ipv4

Créer Classe Java Ipv4 définie par une **@ipv4** et un **masque** de types String et les méthodes suivantes. :

Constructeur : doit vérifier si l'adresse et le masque sont valides avant de les affecter aux propriétés de l'objet.

toBytes(String ipv4) : retourne un tableaux de 4 bytes correspondant à l'adresse ou masque donné en argument

getIpClass() : retourne la classe de l'@ipv4 (A,B,C,...)

getIpNet() : retourne l'adresse Ip du réseau auquel appartient l'@ipv4

getIpBroadCast() : retourne l'@ipv4 de diffusion associée

getValidIp() : retourne la plage d'@ipv4 qu'on peut attribuer aux machines du réseau associé

Vous aurez besoins de :

Convertir un byte en binaire	Integer.toString(b)
Nombre de bits à zéro à gauche	Integer.numberOfLeadingZeros(b)
Nombre de bits à zéro à droite	Integer.numberOfTrailingZeros(b)
Convertir un byte en String	String.valueOf(b)

Solution TP3

```
public class Ipv4 {
    private String addressIpv4;
    private String masque;

    public Ipv4(String addressIpv4, String masque) {
        if(isValidIpAddress(addressIpv4) && isValidMasque(masque)) {
            this.addressIpv4 = addressIpv4;
            this.masque = masque;
        }
        else
            throw new IllegalArgumentException("Address Ip or masque non valid");
    }

    //getters et setters omis
    public boolean isValidIpAddress(String addressIpv4) {
        String[] bytes = addressIpv4.split("\\.");
        if(bytes.length != 4) return false;
        for(String octet : bytes)
            if(Integer.parseInt(octet) < 0 || Integer.parseInt(octet) > 255)
                return false;
        return true;
    }

    public boolean isValidMasque(String masque) {
        String [] bytes = masque.split("\\.");
        String s = "";
        for(String b : bytes)
            s += Integer.toBinaryString(Integer.parseInt(b));

        s = s.replaceAll("0+$", "");
        return !s.contains("0");
    }

    public byte[] tobytes(String ipv4) {
        if(!isValidIpAddress(ipv4)) return null;
        byte[] bytes = new byte[4];
        String[] ipOctets = ipv4.split("\\.");
        for(int i=0;i<4;i++)
            bytes[i] = (byte) Integer.parseInt(ipOctets[i]);
        return bytes;
    }

    public char getIpClass() {
        int first = tobytes(addressIpv4)[0] & 255;
        String s = String.format("%08d",
Integer.parseInt(Integer.toBinaryString(first)));
        if(s.charAt(0) == '0')
            return 'A';
        if(s.charAt(1) == '0')
            return 'B';
        if(s.charAt(2) == '0')
            return 'C';
        if(s.charAt(3) == '0')
            return 'D';
        return 'E';
    }

    public String getIpNet() {
        byte[] ipv4 = tobytes(addressIpv4);
        byte[] mask = tobytes(masque);
        String s = "";
        for(int i=0;i<4;i++) {
            int ip = ipv4[i] & 255;
            int msk = mask[i] & 255;
            s += (i < 3) ? (msk & ip) + "." : (msk & ip);
        }
        return s;
    }

    public String getIpbroadcast() {
        byte[] ipNet = tobytes(getIpNet());
        byte[] mask = tobytes(masque);
    }
}
```

```

String s = "";
for(int i=0;i<4;i++) {
    int a = 255 - (mask[i] & 255);
    s += (i < 3) ? (a | (ipNet[i] & 255)) + "." : (a | (ipNet[i] & 255));
}
return s;
}

public String[] getValidIp() {
    byte[] ipNet = tobytes(getIpNet());
    byte[] ipBroadCast = tobytes(getIpbroadcast());
    ipNet[3] = (byte) ((ipNet[3] & 255) + 1);
    ipBroadCast[3] = (byte) ((ipBroadCast[3] & 255) - 1);
    String[] plage = new String[2];
    plage[0] = "";
    plage[1] = "";
    for(int i=0;i<4;i++) {
        plage[0] += (i < 3) ? (ipNet[i] & 255) + "." : (ipNet[i] & 255);
        plage[1] += (i < 3) ? (ipBroadCast[i] & 255) + "." : (ipBroadCast[i] & 255);
    }
    return plage;
}

public String toString() {
    String[] plage = getValidIp();
    return String.format("%1$-15s | %2$-15s | %3$-6s | %4$-15s | %5$-15s | %6$-15s | %7$-15s",
        "Ipv4","Masque","Class","Net Address","Brodcast","first ip","Last ip") + "\n" +
        String.format("%1$-15s | %2$-15s | %3$-6s | %4$-15s | %5$-15s | %6$-15s | %7$-15s",
        addressIpv4, masque, getIpClass(), getIpNet(), getIpbroadcast(), plage[0], plage[1]);
}
}

```

Atelier Gestion Salle Cinéma (classes, Composition, tableaux/collections)

Le but de ce sujet est d'écrire un programme JAVA pour aider à la gestion de la billetterie des différentes salles d'un complexe cinématographique. Les places non numérotées sont vendues selon deux tarifs :

- un tarif "normal" qui est fixé en fonction de la salle et du film qui y est joué,
- un tarif réduit (familles nombreuses, militaires, chômeurs, étudiants) qui correspond à 80% du tarif normal.

1) Après analyse du problème, il est décidé de représenter les salles de cinéma par des objets JAVA instances d'une **classe SalleCinema** définie comme suit :

- une chaîne de caractères qui contient le titre du film joué,
- un entier qui contient le nombre de places de la salle,
- un réel qui contient le prix unitaire d'une place à tarif normal,
- un entier qui contient le nombre de places qui ont été vendues à tarif normal,
- un entier qui contient le nombre de places qui ont été vendues à tarif réduit.

Les valeurs des trois premières caractéristiques (titre du film, nombre de place, prix de la place) sont fixées lors de la création d'un nouvel objet SalleCinema (c'est-à-dire, sont passées en paramètres du constructeur). Quand aux deux autres variables (nombre de places vendues à tarif normal et nombre de places vendues à tarif réduit) elles sont bien sur initialisées à 0.

La classe SalleCinema possède les méthodes suivantes :

nbPlacesDisponibles()

-> calcule et renvoie le nombre de places encore disponibles dans la salle.

vendrePlaces(int nbre, boolean tarifReduit)

-> permet de vendre des billets pour la salle. nbre indique le nombre de places demandées et le booléen tarifReduit indique si une réduction est demandée ou non (si le paramètre tarifReduit vaut true une réduction est demandée, si il vaut false les places sont achetées au tarif normal sans réduction). Si le nombre de places demandé est supérieur au nombre de places disponibles la vente n'est pas effectuée et la méthode affiche un message indiquant que la vente n'est pas possible. Sinon la variable d'instance correspondant au nombre de places vendues à tarif normal ou à tarif réduit (selon la valeur du paramètre tarifReduit) est mise à jour et le prix à payer est affiché.

chiffreAffaires()

-> retourne le chiffre d'affaires produit par la salle pour la séance en cours (total des ventes depuis la création de l'objet salle ou la dernière remise à zero du nombre de places vendues).

tauxRemplissage()

-> retourne le taux (pourcentage) de remplissage de la salle.

toString()

-> retourne une représentation sous forme d'une chaîne de caractères de l'objet SalleCinema. Cette chaîne indique la valeur de chacun des attributs (chacune des variables d'instances) de l'objet (le titre du film, le nombre de places de la salle, le nombre de places vendues à tarif normal, le nombre de places vendues à tarif réduit, le prix de la place). Par exemple, pour une salle de 60 places

jouant le film "Arressala" dont 20 places ont été vendues au tarif normal (de 25 DH) et 14 places ont été vendues au tarif réduit l'affichage de la chaîne retournée par toString pourrait être le suivant :

Film joué : Arressala,

Nombre de places : 60 ,

Prix d'une place : 25 DH,

20 places vendues au tarif normal , 14 places vendues au tarif réduit.

2) Ecrire une classe **billetterie** qui permet d'enregistrer les entrées effectuées dans les différentes salles et de calculer et d'afficher le taux d'occupation et le chiffre d'affaire produit par chaque salle cinéma. Chaque salle est identifiée par un numéro unique (les numéros allant de 1 à n, n étant le nombre total de salles).

En plus du constructeur, vous allez écrire les méthodes suivantes :

vente ()

->Lorsqu'un client se présente, le guichetier tape le numéro de la salle pour lequel le client désire des billets. Le programme affiche alors les différents attributs de la salle sélectionnée (le titre du film, le nombre de places de la salle, le nombre de places vendues...). Le guichetier fournit ensuite au programme le nombre de places que le client désire acheter en indiquant également si le client bénéficie ou non d'une réduction. Si la demande du client peut être satisfaite le programme affiche le prix à payer sinon il affiche un message indiquant que le nombre de places demandé est incorrect. **Etat()**

-> permet d'afficher pour chaque salle son état (la valeur des ses attributs), son taux d'occupation et le chiffre d'affaire produit. Le programme calcule aussi le chiffre d'affaires total et l'affiche.

3) Écrire une classe Main dans laquelle vous allez créer deux salles correspondant aux informations définies dans la table ci-dessous. Deux places à tarif normal puis trois places à tarif réduit doivent être achetées pour la première salle. Pour la deuxième salle trois places à tarif normal puis six places à tarif réduit doivent être ensuite achetées. Finalement afficher l'état des salles.

Titre	Nombre de Places	Prix de la place (tarif normal)
wjaae trrabe	120	30
Arressala	60	25

Indication : pour stocker les salles on pourra utiliser un tableau.

Solution

```
import java.util.Vector;
public class Cinema {
    private Vector<Salle> vSalles;
    public Cinema(Vector<Salle> vSalles) {
        this.vSalles = vSalles;
    }
}

public class Film {
    private String titre;
    public Film(String titre) {
        this.titre = titre;
    }
    public String getTitre() {
        return this.titre;
    }
    public String toString() {
        return titre;
    }
}

public class Salle {
    private Vector<Place> vplaces = new Vector<>();

    private static int nb_salles;
    private int num_salle;
    public Salle(int nb_places) {
        nb_salles++;
        num_salle = nb_salles;
        for(int i=1;i<=nb_places;i++)
            vplaces.add(new Place(i));
    }
    public Vector<Place> getVplaces() {
        return vplaces;
    }
    public int getNum_salle() {
        return num_salle;
    }
    public String toString() {
        return num_salle+"";
    }
}

public class Place {
    private int num_place;
    private boolean state = false;
    private Boolean tarifReduit = null;

    public Place(int num_place) {
        this.num_place = num_place;
    }

    //getters setters omis
}

import java.time.LocalDate;
import java.util.Vector;

public class Presentation {
    private Film film;
    private Salle salle;
    private LocalDate date;
    private double prix;

    public Presentation(Film film, Salle salle, LocalDate date, double prix) {
        this.film = film;
        this.salle = salle;
    }
}
```

```

        this.date = date;
        this.prix = prix;
    }
    public int getNBPlace_reduit() {
        int count = 0;
        for (Place place : salle.getVplaces()) {
            if(place.getTarifReduit() == Boolean.TRUE)
                count++;
        }
        return count;
    }

    public int getNBPlace_normal() {
        int count = 0;
        for (Place place : salle.getVplaces()) {
            if(place.getTarifReduit() == Boolean.FALSE)
                count++;
        }
        return count;
    }

    public int nbPlacesDisponibles() {
        int place_dispo = 0;
        for (Place place : salle.getVplaces()) {
            if(!place.getState())
                place_dispo++;
        }
        return place_dispo;
    }

    public Vector<Place> vendrePlaces(int nbre, boolean tarifReduit) {
        if(nbre > nbPlacesDisponibles()) {
            System.out.println("la vente n'est pas effectuer car le nbre de place
inssufisant");
            return null;
        }

        Vector<Place> vPlace_vendue = new Vector<>();
        for(Place place : salle.getVplaces()) {
            if(nbre == 0) break;
            if(!place.getState()){
                place.setState(true);
                if(tarifReduit)
                    place.setTarifReduit(Boolean.TRUE);
                else
                    place.setTarifReduit(Boolean.FALSE);
                nbre--;
                vPlace_vendue.add(place);
            }
        }
        return vPlace_vendue;
    }

    public double chiffreAffaires() {
        return (getNBPlace_reduit() * prix * 0.8) + (getNBPlace_normal() * prix);
    }

    public double tauxRemplissage() {
        return 100 - (nbPlacesDisponibles() * 100 / (double)salle.getVplaces().size());
    }

    public String toString() {
        String format = "%-10s %-15s %-15s %-15.2f %-15d %-15d %-15d %-15d %-15.2f
%-15.2f";
        return String.format(format, salle, film, date.toString(), prix,
salle.getVplaces().size(),
getNBPlace_normal(), getNBPlace_reduit(), nbPlacesDisponibles(),

```



```

        tauxRemplissage(), chiffreAffaires());
    }
}

import java.util.Vector;

public class Billeterie {
    private Vector<Presentation> vPresentations = new Vector<>();

    public void addPresentation(Presentation presentation) {
        vPresentations.add(presentation);
    }

    public void vente(int num_salle, int nbre_places, boolean tarifReduit) {
        for (Presentation presentation : vPresentations) {
            if (presentation.getSalle().getNum_salle() == num_salle) {
                presentation.vendrePlaces(nbre_places, tarifReduit);
                return;
            }
        }
    }

    public double chiffreAffaireGlobal() {
        double chiffreAffaire = 0;
        for (Presentation presentation : vPresentations) {
            chiffreAffaire += presentation.chiffreAffaires();
        }
        return chiffreAffaire;
    }

    public void Etat() {
        String header = String.format("%-10s %-15s %-15s %-15s %-15s %-15s %-15s %-15s\n",
            "Num Salle", "Titre film", "Date", "Prix norm", "nbre places",
            "places norm", "places réduit", "places dispo", "taux Occupee", "chiffre
affaire");
        System.out.println(header);

        for (Presentation presentation : vPresentations) {
            System.out.println(presentation);
            System.out.println();
        }

        System.out.println("\n\t\t\tChiffre d'affaire total : " +
chiffreAffaireGlobal());
    }
}

```

Atelier

Gestion Comptes bancaires (Heritage et collections)

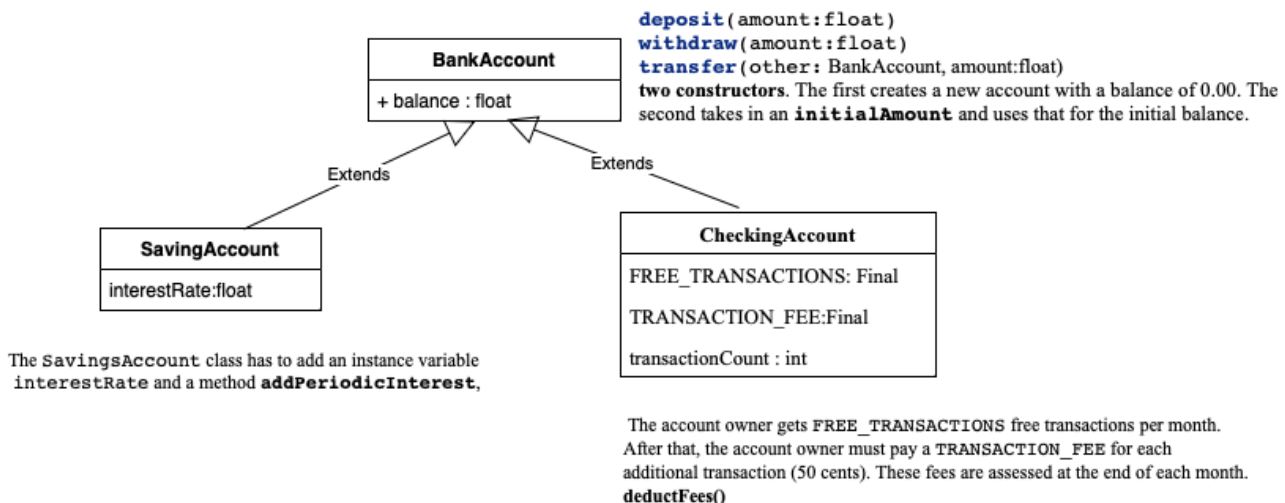
L'objectif est de mettre en place une application de gestion des comptes bancaires. Pour ce faire vous allez implementer en premier lieu les classes du diagramme UML ci-dessous.

BankAccount est définie par la balance (solde) et account_id (numéro de compte auto_increment).

SavingAccount (compte épargne) définie par interestRate (taux intérêt) utilisé par la méthode addPeriodInterest pour calculer l'intérêt qu'il faut ajouter à la balance

CheckingAccount (compte courant) est défini par FREE_TRANSACTIONS (nombre de transactions non imposables). Une fois on le dépasse on applique TRANSACTION_FEE pour déduire les frais du compte.

Les classes dérivées doivent définir leurs propres constructeur. Il est possible d'avoir recours à l'utilisation d'autres variables et méthodes.



La classe Bank centralise la gestion de tous les comptes bancaires à savoir :

search(int num) : retourne le compte dont le numero est num si il existe message d'erreur sinon

create(BankAccount account) : si le compte n'existe pas il l'ajoute sinon erreur, le numero de compte est autoIncrement

list(): lister tous les comptes triés par balance

deposite(BankAccount account,float amount) : déposer le montant amount dans le compte account

withdraw(BankAccount account,float amount): retirer le montant amount du compte account

transfert(BankAccount accountfrom,BankAccount accountto,float amount):

delete (BankAccount account) : supprimer le compte account

receipt(BankAccount account) : affiche le relevé de toutes les opérations du compte account selon le format suivant :

Account : number.	date :	
date / time.	operation.	amount
-----	-----	-----

Pour simplifier l'utilisation de l'application je vous propose de créer une class Menu réutilisable qui affichera le menu textuel suivant :

----- Account Management -----

- 1- List All Account
- 2- Search for Account
- 3- Create Account
- 4- Delete Account
- 5- Deposit Amount
- 6- Withdraw amount
- 7- Transfert
- 8- Receipt
- 9- Exit

Tapez votre choix :

Une classe Menu est définie par un title and list of Options, un constructeur et une méthode choice qui lit le numéro de l'option tapez par l'utilisateur et le retourne.

Finalement une classe App qui contient le main affichant le menu et réalisant le traitement demandé

Solution

```
public class BankAccount {
    private static int nb_accounts;
    private int id_account;
    private float balance;

    private Vector<Historic> historics = new Vector<>();

    public BankAccount() {
        balance = 0;
        nb_accounts++;
        id_account = nb_accounts;
    }

    public BankAccount(float balance) {
        this.balance = balance;
        nb_accounts++;
        id_account = nb_accounts;
    }
}

public class CheckingAccount extends BankAccount{
    public final int FREE_TRANSACTION = 20;
    public final float TRANSACTION_FEE = 0.5f;
    public int transaction_count;
    public void checkTransactions() {
        if(transaction_count >= FREE_TRANSACTION)
            setBalance(getBalance()-TRANSACTION_FEE);
        transaction_count++;
    }
    @Override
    public void deposit(float amount) {
        super.deposit(amount);
        checkTransactions();
    }
    @Override
    public void withdraw(float amount) {
        super.withdraw(amount);
        checkTransactions();
    }
    @Override
    public void transfer(BankAccount other, float amount) {
        super.transfer(other, amount);
        checkTransactions();
    }
    public float deductFees() {
        return (transaction_count-FREE_TRANSACTION)*FREE_TRANSACTION;
    }
}

public class SavingAccount extends BankAccount{
    private static final float interestRate = 0.03f;

    public void addPeriodInterest() {
        deposit(getBalance()*interestRate);
    }
}
```

```

public enum Operation {
    WITHDRAW,
    TRANSFER,
    DEPOSITE
}
import java.time.LocalDate;
import java.util.Collections;
import java.util.Vector;

public class Bank {
    private Vector<BankAccount> vBankAccounts = new Vector<>();

    public BankAccount search(int num) {
        for (BankAccount bankAccount : vBankAccounts) {
            if (bankAccount.getId_Account() == num)
                return bankAccount;
        }
        return null;
    }

    public void create(BankAccount account) {
        if (search(account.getId_Account()) != null) {
            System.out.println("account existe !!");
            return;
        }
        vBankAccounts.add(account);
    }

    public boolean isEmpty() {
        return vBankAccounts.isEmpty();
    }

    public void list() {
        Collections.sort(vBankAccounts, (a1, a2) -> (int) (a1.getBalance() -
a2.getBalance()));
        System.out.println("id_Account \t Balance");
        vBankAccounts.forEach(a -> System.out.println(a));
    }

    public void deposite(BankAccount account, float amount) {
        if (search(account.getId_Account()) == null) {
            System.out.println("account not found !!");
            return;
        }
        search(account.getId_Account()).deposite(amount);
    }

    public void withdraw(BankAccount account, float amount) {
        BankAccount temp = search(account.getId_Account());
        if (temp == null) {
            System.out.println("account not found !!");
            return;
        }
        temp.withdraw(amount);
    }

    public void transfer(BankAccount accountFrom, BankAccount accountTo, float amount) {
        BankAccount tmp1 = search(accountFrom.getId_Account());

```

```

        BankAccount tmp2 = search(accountTo.getId_Account());
        if (tmp1 == null || tmp2 == null) {
            System.out.println("account not found !!");
            return;
        }
        accountFrom.transfer(tmp2, amount);
    }
    public void delete(BankAccount account) {
        BankAccount tmp = search(account.getId_Account());
        if (tmp != null) {
            vBankAccounts.remove(tmp);
            System.out.println("account removed succefully");
        } else
            System.out.println("account not found !!");
    }

    public void receipt(BankAccount account) {
        System.out.println("Account " + account.getId_Account() + "\t Date : " +
LocalDate.now());
        account.getHistorics().forEach(h -> System.out.println(h));
    }
}

import java.util.Scanner;
import java.util.Vector;

public class Menu {
    private String title;
    private Vector<String> options = new Vector<>();

    public Menu(String title, Vector<String> options) {
        this.title = title;
        this.options = options;
    }

    public Vector<String> getOptions() {
        return this.options;
    }

    public static int choice(){
        System.out.print("Taper votre choix : ");
        Scanner scanner = new Scanner(System.in);
        int choix = scanner.nextInt();
        return choix;
    }

    public void list() {
        System.out.println("-----" + title +
"-----");
        options.forEach(o->System.out.println(o));
    }
}

import java.time.LocalDate;

public class Historic {
    private LocalDate date;
    private Operation operation;
    private float amount;

```

```
public Historic(Operation operation, float amount) {  
    this.date = LocalDate.now();  
    this.operation = operation;  
    this.amount = amount;  
}  
public String toString() {  
    return date.toString() + "\t" + operation + "\t" + amount;  
}  
}
```

Atelier Java (Héritage, Interfaces)

Dans cet atelier, vous serez amenés à créer un programme qui gérera les carburants de machines comprenant des véhicules et des appareils. Le programme concernera les véhicules suivants :

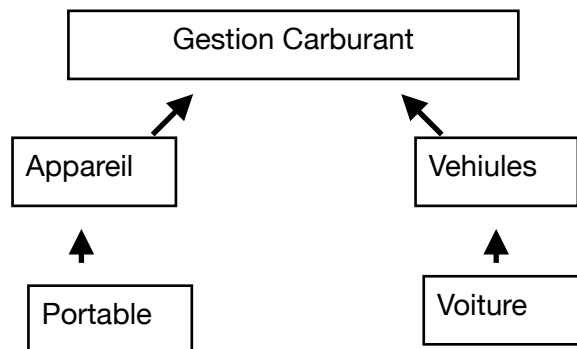
- Voitures (utilisent le carburant "essence")

ainsi que les machines suivantes :

- Téléphones portables (utilise des batteries rechargeables)

Vous aurez à gérer le niveau de carburant disponible, et le rajout de carburant. Ces comportements dépendent du type de véhicule ou de machine, puisque ce ne sont guère les mêmes processus.

Vous allez donc définir plusieurs classes et interfaces selon le schéma suivant



L'interface gestion carburants comprendra les méthodes suivantes :

- **Jauge()** : Renvoie la valeur actuelle du niveau du carburant.
- **Consommer()** : Décrémente le niveau du carburant par la consommation.
- **Remplir()** : Incrémente le niveau du carburant de la valeur spécifiée si le réservoir peut encore contenir cette valeur, sinon fait le plein et indique la valeur mise dans le réservoir
- **Plein()** : Affecte la valeur du maximum des réserves au niveau du carburant.

La classe Véhicules aura les caractéristiques suivantes :

- **marque** : Variable qui stocke le nom de la marque du véhicule
- **typeCarburant** : Stocke le type de carburant utilisé
- **réservoir** : Stocke la capacité en litres du réservoir
- **niveau** : Stocke le niveau actuel des réserves
- **prix** : Stocke le prix du carburant.
- **consommation** : Stocke la consommation des véhicules tous les 100km.

La classe Voiture ajoutera les caractéristiques suivantes :

- **passagers** : Nombre de passagers
- **typeMoteur** : Cylindrée du moteur
- **numéroMoteur** : Numéro du moteur
- **châssis** : Numéro du châssis
- **couleur** : Couleur du véhicule

La classe Appareils définira les caractéristiques suivantes :

- **Marque** : Stocke la marque de l'appareil
- **typeCarburant** : Stocke le type de carburant ("piles", "piles rechargeables" ou "batteries")
- **niveau** : Stocke le niveau actuel des réserves
- **rechargeable** : Vrai si l'appareil utilise une batterie ou des piles rechargeables

La classe Portable ajoutera les caractéristiques suivantes :

- **type** : Stocke le type du portable ("GSM", "AMPS" ou "NTM")

- wap : Vrai si le portable peut utiliser les services WAP.
- consommation : Consommation du portable en une heure en pourcentage de la capacité totale.

Chacune des classes de bas niveau (Voiture, Portable) définira deux méthodes :

- AfficherAttributs() : Elle permettra d'afficher les attributs de l'objet en question
- DéfinirAttributs() : Elle affichera un menu qui renvoie vers une commande permettant de spécifier, à travers le clavier, les valeurs des variables de l'objet en question.

Une classe Test sera créée, et gèrera l'exécution. Cette exécution qui comprendra la création d'une instance de chacune des classes de bas niveau, et un menu permettant de déclencher les diverses opérations possibles, y compris la définition des attributs de chacune des machines, le déclenchement du remplissage des réserves, de la consommation, du jaugeage, ... etc. On vous suggère d'intégrer en plus deux opérations supplémentaires dans le menu :

- Save() : Elle permettra d'enregistrer les objets créés dans un fichier dont le chemin sera saisi par clavier
- Load() : devra charger les données à partir d'un emplacement donné

Solution

```
public interface GestionCarburant{
    public double jauge();
    public void consommer();
    public double remplir(double value);
    public void plein();
}

public abstract class Vehicules implements GestionCarburant, IAttributs, Serializable{
    protected String marque;
    protected TypeCarburant typeCarburant;
    protected double reservoir;
    protected double niveau;
    protected double prix;
    protected double consommation;
    public Vehicules(String marque, TypeCarburant typeCarburant, double reservoir,
double niveau, double prix,double consommation) {
        this.marque = marque;
        this.typeCarburant = TypeCarburant.Essence;
        this.reservoir = reservoir;
        this.niveau = niveau;
        this.prix = prix;
        this.consommmation = consommation;
    }
    public Vehicules() {};
    @Override
    public double jauge() {
        return this.niveau;
    }
    @Override
    public void consommer() {
        if(niveau >= consommation)
            niveau -= consommation;
        else
            niveau = 0;
    }
    @Override
    public double remplir(double value) {
        double niveau_empty = reservoir - niveau;
        if(niveau_empty < value){
            plein();
            return niveau_empty;
        }
        niveau += value;
        return value;
    }
    @Override
    public void plein() {
        niveau = reservoir;
    }
}

public abstract class Appareils implements GestionCarburant,IAttributs,Serializable{
    protected String marque;
    protected TypeCarburant typeCarburant;
    protected double niveau;
    protected boolean rechargable;

    public Appareils(String marque, TypeCarburant typeCarburant, double niveau) {
        this.marque = marque;
    }
}
```

```

        this.typeCarburant = typeCarburant;
        this.niveau = niveau;
        if(typeCarburant == TypeCarburant.Batteries || typeCarburant ==
TypeCarburant.Piles_Rechargables)
            this.rechargable = true;
    }
    public Appareils(){};
    public void setNiveau(double niveau) {
        this.niveau = niveau;
    }
    @Override
    public double jauge() {
        return niveau;
    }
    @Override
    public abstract void consommer();
    @Override
    public double remplir(double value) {
        double niveau_empty = 100 - niveau;
        if (niveau_empty < value) {
            plein();
            return niveau_empty;
        }
        niveau += value;
        return value;
    }
    @Override
    public void plein() {
        niveau = 100;
    }
}
public enum TypeCarburant {
    Piles,
    Piles_Rechargables,
    Batteries,
    Essence
}
public enum TypePortable {
    GSM,
    AMPS,
    NTM
}
import java.io.Serializable;
import java.util.Scanner;

public class Portable extends Appareils implements Serializable{
    private TypePortable type;
    private boolean wap;
    private double consommation;

    public Portable(String marque, TypeCarburant typeCarburant, double niveau,
TypePortable type, double consommation) {
        super(marque, typeCarburant, niveau);
        this.type = type;
        this.consommation = consommation;
        if (type == TypePortable.GSM)
            wap = true;
    }

    public Portable() {};

```

```

@Override
public void consommer() {
    if (jauge() < consommation)
        setNiveau(0);
    else
        setNiveau(jauge() - consommation);
}

@Override
public void afficherAttributs() {
    String headerString = String.format("%-10s %-10s %-10s %-13s %-13s %-10s %-15s",
        "marque", "typeCarb", "niveau", "rechargeable", "typePortable", "IsWap",
"consommation");
    String attString = String.format("%-10s %-10s %-10s %-13s %-13s %-10s %-15s",
        marque, typeCarburant, niveau + "%", rechargeable, type, wap,
consommation + "%");
    System.out.println(headerString);
    System.out.println(attString);
}

@Override
public void definirAttributs() {
    Scanner scanner = new Scanner(System.in);
    System.out.print("Entrez la marque: ");
    this.marque = scanner.nextLine();
    System.out.print("Entrez le type de carburant(B:Batteries, P:Piles,
PR:Piles_Rechargeables): ");
    String type = scanner.nextLine();
    if (type.equalsIgnoreCase("B")) {
        this.typeCarburant = TypeCarburant.Batteries;
    } else if (type.equalsIgnoreCase("P")) {
        this.typeCarburant = TypeCarburant.Piles;
    } else if (type.equalsIgnoreCase("PR")) {
        this.typeCarburant = TypeCarburant.Piles_Rechargeables;
    } else {
        System.out.println("Type de portable non reconnu it will take batteries as
default type !!");
        this.typeCarburant = TypeCarburant.Batteries;
    }

    System.out.print("Entrez le niveau: ");
    this.niveau = scanner.nextInt();
    scanner.nextLine();

    System.out.println("Choisissez le type de portable (GSM/AMPS/NTM): ");
    String portableTypeInput = scanner.nextLine();
    if (portableTypeInput.equalsIgnoreCase("GSM")) {
        this.type = TypePortable.GSM;
        wap = true;
    } else if (portableTypeInput.equalsIgnoreCase("AMPS")) {
        this.type = TypePortable.AMPS;
    } else if (portableTypeInput.equalsIgnoreCase("NTM")) {
        this.type = TypePortable.NTM;
    } else {
        System.out.println("Type de portable non reconnu.");
    }

    System.out.print("Entrez la consommation: ");
    this.consommmation = scanner.nextDouble();
}

```

```

    }
}

public class Voiture extends Vehicules implements Serializable{
    private int passagers;
    private String typeMoteur;
    private int numMoteur;
    private int chassis;
    private String color;

    public Voiture(String marque, TypeCarburant typeCarburant, double reservoir, double
niveau, double prix,
        double consommation, int passagers, String typeMoteur, int numMoteur, int
chassis, String color) {
        super(marque, typeCarburant, reservoir, niveau, prix, consommation);
        this.passagers = passagers;
        this.typeMoteur = typeMoteur;
        this.numMoteur = numMoteur;
        this.chassis = chassis;
        this.color = color;
    }
    public Voiture(){};
    @Override
    public void afficherAttributs() {
        String headerString = String.format("%-10s %-10s %-10s %-10s %-10s %-15s %-10s
%-10s %-10s %-10s %-10s",
            "marque", "typeCarb", "reservoir", "niveau", "prix", "consommation",
"passagers", "typeMoto", "numMoto", "chassis", "color");
        String attString = String.format("%-10s %-10s %-10s %-10s %-10s %-15s %-10d
%-10s %-10d %-10d %-10s",
            marque, typeCarburant, reservoir + " L", niveau+" L", prix +" DH",
consommation+" L/100km", passagers, typeMoteur, numMoteur, chassis, color);
        System.out.println(headerString);
        System.out.println(attString);
    }

    @Override
    public void definirAttributs() {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Entrez la marque: ");
        this.marque = scanner.nextLine();
        this.typeCarburant = TypeCarburant.Essence;
        System.out.print("Entrez la capacité du réservoir: ");
        this.reservoir = scanner.nextInt();
        System.out.print("Entrez le niveau actuel: ");
        this.niveau = scanner.nextDouble();
        System.out.print("Entrez le prix: ");
        this.prix = scanner.nextDouble();
        System.out.print("Entrez la consommation: ");
        this.consommation = scanner.nextDouble();
        System.out.print("Entrez le nombre de passagers: ");
        this.passagers = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Entrez le type de moteur: ");
        this.typeMoteur = scanner.nextLine();
        System.out.print("Entrez le numéro de moteur: ");
        this.numMoteur = scanner.nextInt();
        scanner.nextLine();
        System.out.print("Entrez le numéro de châssis: ");
        this.chassis = scanner.nextInt();
    }
}

```

```

        scanner.nextLine();
        System.out.print("Entrez la couleur: ");
        this.color = scanner.nextLine();
    }
}

public interface IAttributs {
    public void afficherAttributs();
    public void definirAttributs();
}

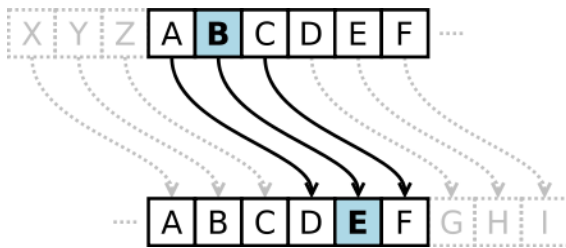
public static void save(Vehicules v, Appareils a, String path) {
    try (ObjectOutputStream outputStream = new ObjectOutputStream(new
FileOutputStream(path))) {
        outputStream.writeObject(v);
        outputStream.writeObject(a);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static void load(String path) {
    try {
        FileInputStream fileInS = new FileInputStream(path);
        ObjectInputStream objectInS = new ObjectInputStream(fileInS);
        Vehicules voiture = (Voiture) objectInS.readObject();
        Appareils portable = (Portable) objectInS.readObject();
        toyota = voiture;
        iphone = portable;
        objectInS.close();
        fileInS.close();
        System.out.println("Objets chargés avec succès !");
    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}

```

Algorithme de César

Le chiffrement par substitution est le premier algorithme qui a été mise en place pour rendre un message confidentiel. Le chiffre de César fonctionne par décalage des lettres de l'alphabet par un nombre (clé) bien défini. Par exemple dans l'image ci-dessus, il y a une distance de 3 caractères, donc B devient E dans le texte codé.



Le chiffrement peut aussi être représenté en utilisant les congruences sur les entiers. En commençant par transformer chaque lettre en un nombre ($A = 0, B = 1, \dots, Z = 25$), pour encoder une lettre avec une clé k il suffit d'appliquer la formule : $E(x) = (x + k) \bmod 26$

Le déchiffrement consiste à utiliser la clé opposée (à la place de $+$) : $D(x) = (x - k) \bmod 26$.

- 1) Écrire une fonction `Encrypte(char c, int k)` qui retourne le caractère crypté associé à c en utilisant la clé k
- 2) écrire de la même façon `Decrypte(char c, int k)` qui retourne le caractère décrypté associé à c en utilisant la clé k
- 3) Écrire une fonction `EncrypteMessage(String fichierClaire, String fichierCrypte, int k)` qui permet de chiffrer le fichier en clair `fichierClaire` dans `fichierCrypte`.
- 4) Écrire de la même façon `DecrypteMessage(String fichierCrypte, String fichierClaire, int k)`
- 5) Écrire un programme Java qui va vous permettre de tester toutes les fonctions demandées

Algorithme de Blaise de Vigenère

Il existe un autre chiffrement plus robuste que ce lui de César et il est du à Blaise de Vigenère dont le principe est comme suit :

La clé est cette fois ci un mot de taille 26 caractères numéroté ($A=0, B=1, \dots$). Chaque caractère de la clé définit le décalage à faire par exemple :

Mot en clair	C	H	I	F	F	R	E	D	E	V	I	G	E	N	E	R	E
Clé	B	A	C	H	E	L	I	E	R	B	A	C	H	E	L	I	E
Décalage	1	0	2	7	4	11	8	4	17	1	0	2	7	4	11	8	4
Mot Chiffré	D	H	K	M	J	C	M	H	V	W	I	I	L	R	P	Z	I

- 6) Réécrire les fonctions de 1) à 5) de la partie A sachant que la clé est ESTMGIGL

Compression Run-Length Encoding

Mettre en œuvre un programme Java qui compresse un fichier texte en utilisant un algorithme de compression simple (par exemple, le codage en longueurs d'exécution) et écrit le contenu compressé dans un nouveau fichier.

(RLE) est un algorithme de compression simple qui représente les caractères identiques consécutifs dans une chaîne par un seul caractère préfixe du nombre d'occurrences. Il est particulièrement efficace pour compresser des données avec de longues séquences de caractères répétés.

Voici comment fonctionne l'algorithme de codage en longueurs d'exécution :

Encodage :

Chaîne d'entrée : "AAAABBBCCDAA"

Chaîne de sortie : "4A3B2C1D2A"

Hashage à base de XOR

- Écrire un programme Java qui prend en argument un fichier de type quelconque et génère le hashcode associé en utilisant l'algorithme simplifié suivant :
- 1- décomposer le fichier en matrices de byte de taille 16 colonnes si le fichier est de taille qui n'est pas multiple de 16 compléter par des zéro
- 2- calculer le xor de chaque colonne dans un tableau hash de byte de taille 16
- 3- répéter 1 et 2 en sommant dans hash jusqu'à la fin de fichier Tester votre code sur des fichiers de différente taille

	bit 1	bit 2	• • •	bit n
block 1	b_{11}	b_{21}		b_{n1}
block 2	b_{12}	b_{22}		b_{n2}
	•	•	•	•
	•	•	•	•
	•	•	•	•
block m	b_{1m}	b_{2m}		b_{nm}
hash code	C_1	C_2		C_n

Figure 3.3 Simple Hash Function Using Bitwise XOR

Solution

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Securite {
    public static char encryptChar(char c, int k) {
        if(Character.isAlphabetic(c))
            return (char) (((Character.toUpperCase(c) - 'A' + k) % 26) + 'A');
        else
            return c;
    }
    public static char decryptChar(char c, int k) {
        if(Character.isAlphabetic(c))
            return (char) (((Character.toUpperCase(c) - 'A' - k + 26) % 26) + 'A');
        else
            return c;
    }
    public static void encryptFile(String fileSource, String fileDest, int k) {
        File fs = new File(fileSource);
        try (FileReader fr = new FileReader(fs); FileWriter fw = new
FileWriter(fileDest)) {
            int c;
            while ((c = fr.read()) != -1)
                fw.write(Securite.encryptChar((char) c, k));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void decryptFile(String fileSource, String fileDest, int k) {
        File fs = new File(fileSource);
        try (FileReader fr = new FileReader(fs); FileWriter fw = new
FileWriter(fileDest)) {
            int c;
            while ((c = fr.read()) != -1)
                fw.write((int) Securite.decryptChar((char) c, k));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public static void encryptFile(String fileSource, String fileDest, String key) {
        try (FileReader fr = new FileReader(fileSource);
            FileWriter fw = new FileWriter(fileDest)) {

            char[] data = new char[key.length()];
            int[] keys = new int[key.length()];
            int tmp;

            for (int i = 0; i < key.length(); i++)
                keys[i] = Character.toUpperCase(key.charAt(i)) - 'A';

            while ((tmp = fr.read(data)) != -1) {
                for (int i = 0; i < tmp; i++) {
                    fw.write(encryptChar(data[i], keys[i]));
                }
            }

        } catch (IOException e) {
```

```

        e.printStackTrace();
    }
}

public static void decryptFile(String fileSource, String fileDest, String key) {
    try (FileReader fr = new FileReader(fileSource);
        FileWriter fw = new FileWriter(fileDest)) {

        char[] data = new char[key.length()];
        int[] keys = new int[key.length()];
        int tmp;

        for (int i = 0; i < key.length(); i++)
            keys[i] = Character.toUpperCase(key.charAt(i)) - 'A';

        while ((tmp = fr.read(data)) != -1) {
            for (int i = 0; i < tmp; i++) {
                fw.write(decryptChar(data[i], keys[i]));
            }
        }

    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static String compress(String fileSource, String fileDest) {
    File fs = new File(fileSource);
    try (FileReader fr = new FileReader(fs);
        FileWriter fw = new FileWriter(fileDest)) {
        int tmp = fr.read();
        int count = 1;
        int c;
        int score = 0;
        while ((c = fr.read()) != -1) {
            if (c != tmp) {
                fw.write(String.valueOf(count) + (char) tmp);
                count = 0;
                score += 2;
            }
            tmp = c;
            count++;
        }
        fw.write(String.valueOf(count) + (char) tmp);
        score += 2;
        return String.valueOf(score + "/" + fs.length());
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

```

```

public static void decompress(String fileSource, String fileDest) {
    File fs = new File(fileSource);
    try (FileReader fr = new FileReader(fs);
        FileWriter fw = new FileWriter(fileDest)) {

        for (int i = 0; i < fs.length(); i+=2) {
            char num = (char) fr.read();

```

```

        char c = (char)fr.read();
        for(int j = 0; j < Integer.parseInt(String.valueOf(num)); j++) {
            fw.write(c);
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
}

public static String hashage(String fileSource) {
    File fs = new File(fileSource);
    try (FileInputStream fr = new FileInputStream(fs)) {
        byte[] hashCode = new byte[16];
        byte[] data = new byte[16];
        int tmp;
        while ((tmp = fr.read(data)) != -1) {
            if(tmp != 16)
                for(int i=tmp; i<16; i++)
                    data[i] = 0;
            for(int i=0; i<16; i++) {
                hashCode[i] = (byte)(data[i] ^ hashCode[i]);
            }
        }
        StringBuffer hash = new StringBuffer();
        for (byte b : hashCode)
            hash.append(bytesToHex(b));
        return hash.toString();
    } catch (IOException e) {
        e.printStackTrace();
    }
    return null;
}

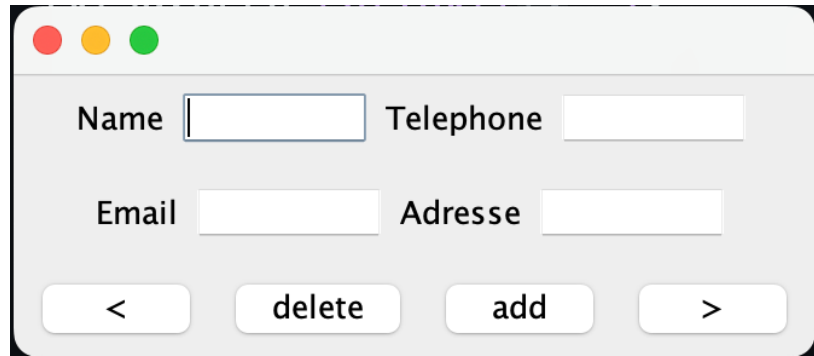
public static String bytesToHex(byte b) {
    return Integer.toHexString(0xff & b);
}
}

```

Atelier Gestion des Contact/(Swing et base de données)

Mettre en place une application graphique Java (MVC) avec accès aux bases de données pour la gestion des contacts Vous allez créer deux packages :

- Package dal (Data Access Layer) : Contient les classes d'accès à la base de données
- Package presentation : Contient les classes de la vue graphique (voir figure ci-dessous)



Un contact est défini par un nom, un téléphone, un email et une adresse. On supposera que le email est la clé de la table que vous allez créer dans mysql

Delete : permet de supprimer le contact en cours

Add : une fois cliquer le bouton deviendra save et on videra les champs après remplissage des une fois on clique sur save le bouton deviendra add

Les boutons > et < permettent respectivement de passer au suivant et au précédent

Solution

Package `estm.java.dal`

Classe `BdConnection`

```
import java.sql.Connection;
import java.sql.DriverManager;

public class Bdconnection {

    private static Connection cnx=null;
    public static Connection getConnection() {
        if(cnx==null) {
            try {
                Class.forName("com.mysql.cj.jdbc.Driver");
                cnx=DriverManager.getConnection("jdbc:mysql://localhost:3306/
Contact", "root", "");
            } catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        return cnx;
    }
}
```

```
public class Contact {
    private int id ;
    private String nom;
    private String phone;
    private String email;
    //getters et setters omis
}
```

Public Classe `ContactDao`

```
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;

public class ContactDao {

    private Connection cnx;
    private Statement stm;
    private ResultSet rs;

    public ContactDao() {

        cnx=Bdconnection.getConnection();
        try {
            stm=cnx.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_UPDATABLE);
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

```

public Vector<Contact> getAll() {

    Vector<Contact> vp = new Vector<>();
    try {
        rs=stm.executeQuery("Select * from T_contact");
        Contact c;
        while(rs.next()) {
            c=new Contact();
            c.setId(rs.getInt(1));
            c.setNom(rs.getString(2));
            c.setPhone(rs.getString(3));
            c.setEmail(rs.getString(4));
            vp.add(c);
        }
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return vp;
}

public void add(Contact p) {
    int id = p.getId();
    String nom =p.getNom();
    String phone=p.getPhone();
    String email=p.getEmail();

    try {
        stm.execute("insert into T_contact
values("+id+", '"+nom+"', '"+phone+"', '"+email+"');");
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public void delete(int id) {

    try {
        stm.execute("delete from T_contact where id = "+id+";");
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

}

```

Package estm.java.views

```

public class Contact_bd extends JFrame{
    JLabel lblId = new JLabel("id");
    JLabel lblnom = new JLabel("Nom");
    JLabel lblphone = new JLabel("phone");
    JLabel lblemail = new JLabel("Email");
    JButton btnPrecedent = new JButton("<");
    JButton btnNext = new JButton(">");
    JButton btnAdd = new JButton("Add");
    JButton btnDellet = new JButton("Delet");

    JTextField txtId = new JTextField(10);
}

```

```

JTextField txtnom = new JTextField(10);
JTextField txtphone = new JTextField(10);
JTextField txtEmail = new JTextField(10);
JPanel pnNouth = new JPanel();
JPanel pnsouth = new JPanel();
JPanel pnhaut = new JPanel();
Vector<Contact> vc = new Vector<>();
Contact ct= new Contact();
ContactDao dao = new ContactDao();
private int curent=0;

public Contact_bd() {
    this.setLayout(new BorderLayout());
    pnNouth.setLayout(new GridLayout(2,4));
    pnNouth.add(lblId);
    pnNouth.add(txtId);
    pnNouth.add(lblnom);
    pnNouth.add(txtnom);
    pnNouth.add(lblphone);
    pnNouth.add(txtphone);
    pnNouth.add(lblemail);
    pnNouth.add(txtEmail);
    this.add(pnNouth,BorderLayout.NORTH);
    pnsouth.setLayout(new FlowLayout());
    pnsouth.add(btnPrecedent);
    pnsouth.add(btnDellet);
    pnsouth.add(btnAdd);
    pnsouth.add(btnNext);
    this.add(pnsouth,BorderLayout.SOUTH);
    //Properties
        this.setTitle("Gestion contact");
        this.pack();
        this.setLocationRelativeTo(null);
        this.setVisible(true);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

    //pour remplir les field text
        vc=dao.getAll();
        txtId.setText(vc.get(curent).getId()+"");
        txtnom.setText(vc.get(curent).getNom()+"");
        txtphone.setText(vc.get(curent).getPhone()+"");
        txtEmail.setText(vc.get(curent).getEmail()+"");
    txtId.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e) {
            vc=dao.getAll();
            int a=0;
            for(Contact c :vc)
                if(Integer.parseInt(txtId.getText()) == c.getId()) {

                    txtId.setText(c.getId()+"");
                    txtnom.setText(c.getNom());
                    txtphone.setText(c.getPhone());
                    txtEmail.setText(c.getEmail());
                    a=1;
                    break;
                }
            if(a!=1)

```

```

        JOptionPane.showMessageDialog(null, "cette contact
n'exist pas");

    }
});
//Add
btnAdd.addActionListener(e->{
    if(btnAdd.getText().compareTo("Add")==0) {
        int id=Integer.parseInt(txtId.getText());
        String nom=txtnom.getText();
        String p =txtphone.getText();
        String em =txtEmail.getText();
        ct.setId(id);
        ct.setNom(nom);
        ct.setPhone(p);
        ct.setEmail(em);
        txtId.setText("");
        txtnom.setText("");
        txtphone.setText("");
        txtEmail.setText("");
        btnNext.disable();
        btnPrecedent.disable();
        btnDelllet.disable();
        btnAdd.setText("valider");}
    if(btnAdd.getText().compareTo("valider")==0) {

        btnNext.enable();
        btnPrecedent.enable();
        btnDelllet.enable();
        String [] options={"oui","non"};
        int n =JOptionPane.showOptionDialog(this,"voulez vous vraiment
Enregistre le contact?","Quitter",
JOptionPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE, null,options,options[0]);
        if(n==1) { btnAdd.setText("Add");

    }

        if(n==0) {btnAdd.setText("Add");

        dao.add(ct);
    }
}

});
//Next
btnNext.addActionListener(e->{

    curent++;
    if(vc.size()<=curent) curent=0;

    txtId.setText(vc.get(curent).getId()+"");
    txtnom.setText(vc.get(curent).getNom()+"");
    txtphone.setText(vc.get(curent).getPhone()+"");
    txtEmail.setText(vc.get(curent).getEmail()+"");

});
//precedent
btnPrecedent.addActionListener(e->{
    if(0>=curent) curent=vc.size()-1;
    else

```



```

        curent--;

        txtId.setText(vc.get(curent).getId()+"");
        txtnom.setText(vc.get(curent).getNom()+"");
        txtphone.setText(vc.get(curent).getPhone()+"");
        txtEmail.setText(vc.get(curent).getEmail()+"");

    });
//delete
    btnDelllet.addActionListener(e->{
        int a=0;
        for(Contact c :vc)
            if(Integer.parseInt(txtId.getText()) == c.getId())
                a=1;
        if(a==1){
            String [] options={"oui","non"};
            int n =JOptionPane.showOptionDialog(this,"voulez vous vraiment supprimer
cette contact?","Quitter", JOptionPane.YES_NO_OPTION,JOptionPane.QUESTION_MESSAGE,
null,options,options[0]);
            if(n==1) {}
            if(n==0) {
                txtId.getText();
                dao.delete(Integer.parseInt(txtId.getText()));
            }
        }
        else {
            JOptionPane.showMessageDialog(null, "cette contact
n'existe"+Integer.parseInt(txtId.getText())+" pas");

        }

    });

}}

```