Programmation avancée et application • TD 4

Exceptions, Entrées-Sorties

Exercice I (Exceptions et UtilMath)

1. Reprenez la classe UtilMath vue dans le TD1 (Exercice I).

TD 1

Exercice I (UtilMath)

Définir la classe UtilMath possédant les méthodes suivantes :

- **public static int** somme3(**int** a, **int** b, **int** c) qui fait la somme de trois entiers ;
- **public static long** fact(**int** n) qui calcule la factorielle $n!$ d'un entier positif n ;
- **public static long** comb(**int** n, **int** p) qui calcule la combinaison $\binom{n}{p}$ de deux entiers positifs p et n ;[1]
- **public static long** puissance(**int** n, **int** m) qui calcule la puissance m-ème de n : n^m, avec $m \geqslant 0$.

Rappel : $\binom{n}{p} = \frac{n!}{p! \times (n-p)!}$, avec $p \leqslant n$.

TD 1

```java
public class UtilMath {

	// fait la somme de trois entiers
	public static int somme3(int a, int b, int c)
	{
		return (a + b + c);
	}

	// calcule la factorielle n! d'un entier positif n
	public static long fact(int n)
	{
		long result = 1;
		for(int i = 1 ; i <= n ; i++)
			result *= i;

		return result;
	}

	// calcule la combinaison de deux entiers positifs p et n
	public static long comb(int n, int p)
	{
		return ( ( fact(n) ) / ( fact(p) * fact(n - p) ) );
	}

	// calcule la puissance m-ème de n: n^m, avec m >= 0
	public static long puissance(int n, int m)
	{
		long result = 1;


		for(int i = 1 ; i <= m ; i++)
			result *= n;

		//Si m == 0 , alors result = n^0 = 1
		return result;
	}

}
```

(a) Si le paramètre de la méthode fact(**int** n) est négatif, levez une IllegalArgumentException avec un message d'erreur informatif.

(b) De la même façon, gérez les erreurs possibles dans les méthodes comb(**int** n, **int** p) (si $p > n$) et puissance(**int** n, **int** m) (si $m < 0$).

```
public class UtilMath {

	// fait la somme de trois entiers
	public static int somme3(int a, int b, int c)
	{
		return (a + b + c);
	}

	// calcule la factorielle n! d'un entier positif n
	public static long fact(int n) throws IllegalArgumentException
	{
		// Le mot-cle throws dans la signature indique que la methode est susceptible
de lever une exception

		if(n < 0)
			throw new IllegalArgumentException("Le paramètre n est négatif.");
		// Le mot-cle throw suivi d'un objet de type Exception permet de lever
l'exception.

		/* java.lang.IllegalArgumentException.IllegalArgumentException(String s)
		 * Constructs an IllegalArgumentException with thespecified detail message.
		 * Parameters:s the detail message.
		 */

		long result = 1;
		for(int i = 1 ; i <= n ; i++)
			result *= i;

		return result;
	}

	// calcule la combinaison de deux entiers positifs p et n
	public static long comb(int n, int p) throws IllegalArgumentException
	{
		if(p > n)
			throw new IllegalArgumentException("Le paramètre p est plus grand que
le paramètre n.");

		return ( ( fact(n) ) / ( fact(p) * fact(n - p) ) );
	}

	// calcule la puissance m-ème de n: n^m, avec m >= 0
	public static long puissance(int n, int m) throws IllegalArgumentException
	{
		if(m < 0)
			throw new IllegalArgumentException("Le paramètre m est négatif.");

		long result = 1;
		for(int i = 1 ; i <= m ; i++)
			result *= n;

		//Si m == 0 , alors result = n^0 = 1
		return result;
	}

}
```

2. Les modifications apportées aux méthodes de la classe UtilMath doivent être impactées sur le programme qui les utilise (TD1, Exercice VI). Faites les modifications nécessaires pour que l'utilisateur soit informé lors d'une erreur, et que le programme ne s'arrête pas brutalement.

3. Faites les modifications nécessaires pour que l'utilisateur soit informé de l'erreur s'il tape autre chose que des nombres entiers, et que le programme ne s'arrête pas brutalement.

TD 1

Exercice VI (Interface textuelle pour UtilMath)

1. Écrire un programme qui demande à l'utilisateur s'il souhaite calculer la somme de trois entiers, la factorielle d'un entier, la combinaison de deux entiers, ou la puissance m-ème d'un entier. En fonction de sa réponse, le programme demande ensuite à l'utilisateur le (ou les) nombre(s) nécessaire(s) pour le calcul, et affiche le résultat.

2. Écrire une variante du programme dans laquelle le programme revient au menu après avoir affiché le résultat du calcul. Dans ce cas, il faut prévoir une option "Quitter" en plus du choix des opérations.

TD 1

```java
import java.util.Scanner;
public class TestUtilMath {
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {
        int menu;

        do{
            System.out.println("******* MENU ******");
            System.out.println("1 - interface UtilMath");
            System.out.println("2 - quitter");
            System.out.print("Votre choix ? ");
            menu = sc.nextInt();

            switch(menu){
                case 1 : interfaceMath();
            }

        }while(menu != 2);
        sc.close();
    }
```

```
    public static void interfaceMath() {
            int choix, a, b, c;

            do{
                  System.out.println("***** UtilMath - interface *****");
                  System.out.println("Menu");
                  System.out.println("1 - somme de trois entiers");
                  System.out.println("2 - factorielle d'un entier");
                  System.out.println("3 - combinaison de deux entiers");
                  System.out.println("4 - la puissance m-ème d'un entier");
                  System.out.println("5 - quitter");
                  System.out.print("Votre choix ? ");
                  choix = sc.nextInt();

                  switch(choix) {
                        case 1 : System.out.print("numero 1 = ? ");
                                 a = sc.nextInt();
                                  System.out.print("numero 2 = ? ");
                                  b = sc.nextInt();
                                  System.out.print("numero 3 = ? ");
                                  c = sc.nextInt();
            System.out.println( "Resultat = " + UtilMath.somme3(a, b, c) );
                                  break;

                        case 2 : System.out.print("numero = ? ");
                                  a = sc.nextInt();
                          System.out.println( "Resultat = " + UtilMath.fact(a) );
                                  break;

                        case 3 : System.out.print("n = ? ");
                                 a = sc.nextInt();
                                 System.out.print("p = ? ");
                                 b = sc.nextInt();
                                System.out.println( "Resultat = " + UtilMath.comb(a, b) );
                                 break;

                        case 4 : System.out.print("n = ? ");
                                  a = sc.nextInt();
                                  System.out.print("m = ? ");
                                  b = sc.nextInt();
                  System.out.println( "Resultat = " + UtilMath.puissance(a, b) );

                  }
            }while( choix != 5 );
       } // interfaceMath()
} // Class
```

```java
import java.util.Scanner;
import java.util.InputMismatchException;
public class TestUtilMath {
	static Scanner sc = new Scanner(System.in);

	public static void main(String[] args) {
		int menu;

		do{
			System.out.println("******* MENU ******");
			System.out.println("1 - interface UtilMath");
			System.out.println("2 - quitter");
			System.out.print("Votre choix ? ");
			menu = getInt();

			switch(menu){
				case 1 : interfaceMath();
			}

		}while(menu != 2);
		sc.close();
	}

	public static int getInt() {
		boolean check = false;
		int entier = 0;

		/* java.util.InputMismatchException
		 * Thrown by a Scanner to indicate that the token retrieved does not match the
pattern for the expected type,
		 * or that the token is out of range for the expected type. See
Also:java.util.Scanner
		 */

		do {
			try {
				entier = sc.nextInt();
				check = true;
			} catch (InputMismatchException e) {
				System.out.print("Erreur ! Essayez de taper à nouveau un numéro : ");
				sc.nextLine();
			}
		}while( check == false );

		return entier;
	}
```

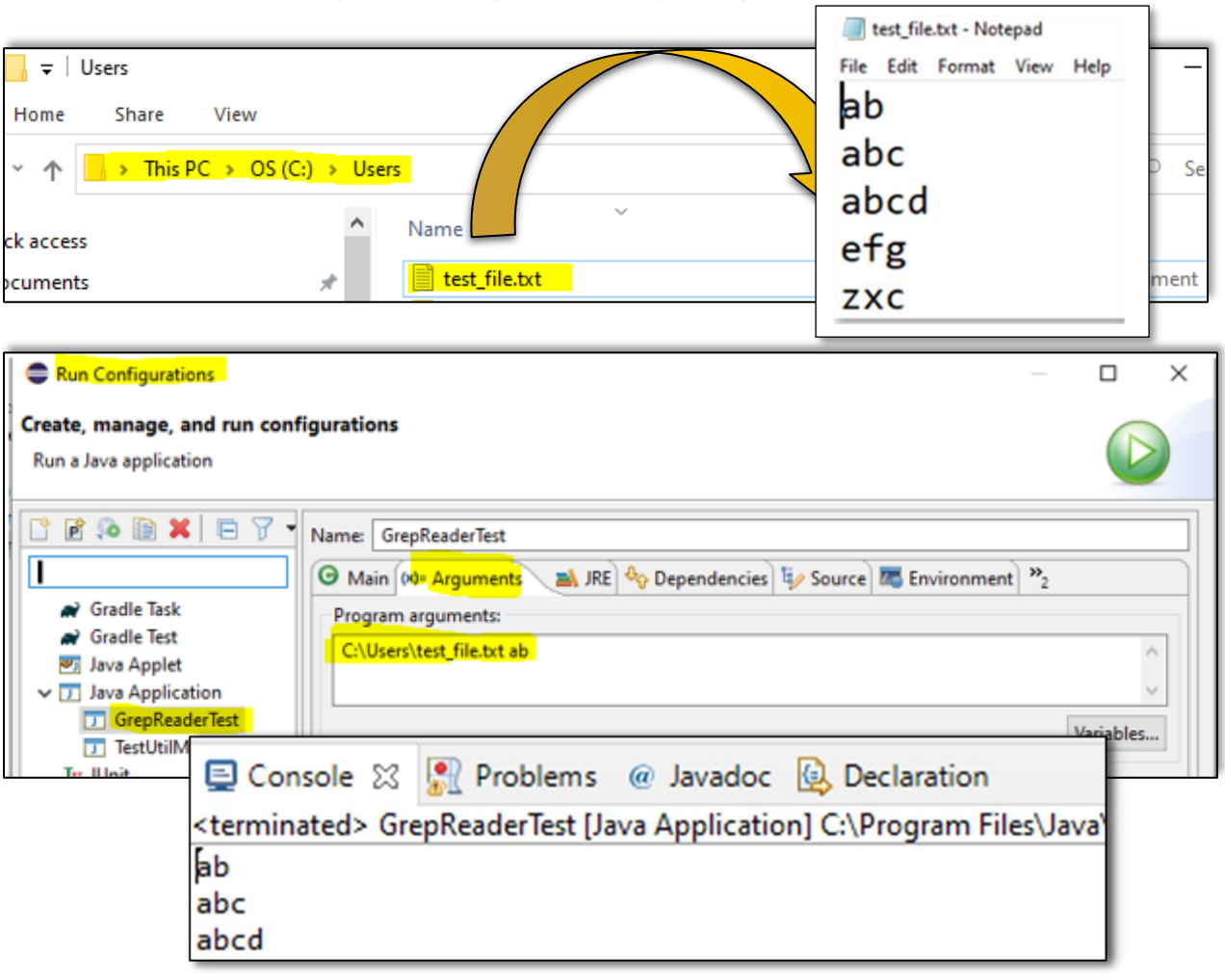
```
	public static void interfaceMath() {
		int choix, a, b, c;

		do{
			System.out.println("***** UtilMath - interface *****");
			System.out.println("Menu");
			System.out.println("1 - somme de trois entiers");
			System.out.println("2 - factorielle d'un entier");
			System.out.println("3 - combinaison de deux entiers");
			System.out.println("4 - la puissance m-ème d'un entier");
			System.out.println("5 - quitter");
			System.out.print("Votre choix ? ");
			choix = getInt();

			switch(choix) {
				case 1 : System.out.print("numero 1 = ? ");
					a = getInt();
					System.out.print("numero 2 = ? ");
					b = getInt();
					System.out.print("numero 3 = ? ");
					c = getInt();
					System.out.println( "Resultat = " + UtilMath.somme3(a, b, c) );

					break;

				case 2 : System.out.print("numero = ? ");
					a = getInt();
					try {
						System.out.println( "Resultat = " + UtilMath.fact(a) );
					} catch(IllegalArgumentException e) {
						System.out.println( e.getMessage() );
					}
					break;

				case 3 : System.out.print("n = ? ");
					a = getInt();
					System.out.print("p = ? ");
					b = getInt();
					try {
				System.out.println( "Resultat = " + UtilMath.comb(a, b) );

					} catch(IllegalArgumentException e) {
						System.out.println( e.getMessage() );
					}
					break;

				case 4 : System.out.print("n = ? ");
					a = getInt();
					System.out.print("m = ? ");
					b = getInt();
					try {
					System.out.println( "Resultat = " + UtilMath.puissance(a, b) );

					} catch(IllegalArgumentException e) {
						System.out.println( e.getMessage() );
					}
			}
		}while( choix != 5 );
	} // interfaceMath()
} // Class
```

Exercice II (Quelques commandes UNIX)
On souhaite développer des programmes qui correspondent à certaines commandes UNIX :

- `grep` sert à filtrer un texte en n'affichant que les lignes qui contiennent un certain motif;
- `wc` compte le nombre de lignes, de mots et de caractères dans un texte;
- `cp` permet de copier un fichier.

1. Écrivez un programme qui prend en entrée le nom d'un fichier texte et une chaîne de caractères, [1] et qui affiche sur la console toutes les lignes du fichier qui contiennent la chaîne de caractères donnée en entrée.
   Indice : vous pouvez créer une classe GrepReader qui hérite de BufferReader, et qui redéfinit la méthode readLine() de manière adéquate.

1. Pour qu'un programme lancé dans Eclipse utilise les paramètres de la ligne de commande, il faut les indiquer dans le menu Run → Run Configurations → Arguments → Program arguments.

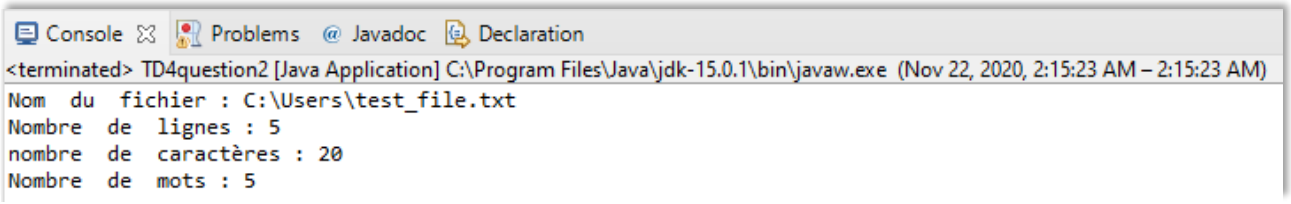


```java
import java.io.BufferedReader;
import java.io.Reader;
import java.io.IOException;
/* "Indice : vous pouvez créer une classe GrepReader qui hérite de BufferReader,
 * et qui redéfinit la méthode readLine() de manière adéquate." (Sujet TD4)
 */
public class GrepReader extends BufferedReader{

	/* Le principal intérêt de la classe BufferredReader est la méthode readLine()
	 * qui va lire une ligne de texte.
	 * (Cours POO L2 S4 / Page 201)
	 */

	/* BufferedReader est une classe fille de Reader qui propose une lecture
	 * efficace de caracteres et de lignes dans un flux d'entree
	 *
	 * readLine()  est une methode de BufferedReader
	 * qui lit une ligne a partir du flux d'entree et la retourne sous forme de  String
	 *
	 * readLine ()  retourne null lorsque la fin du fichier est atteinte.
	 * (Cours PAA Chapitre 3 / pages 56-57)
	 */

	 /* Reader
	  * Classe abstraite, mere de la classe dediee a la lecture de flux de caracteres.
	  * (Cours PAA Chapitre 3)
	  */

	private String grep;

	public GrepReader(Reader in, String grep) {
		super(in);
		/* super(in) - Call to BufferedReader(Reader in)
		 * Creates a buffering character-input stream that uses a
		 * default-sized input buffer.
		 *
		 * Parameters: in A Reader
		 * (JavaDoc)
		 */
		this.grep = grep;
	}

	/* "Indice : vous pouvez créer une classe GrepReader qui hérite de BufferReader,
	 * et qui redéfinit la méthode readLine() de manière adéquate." (Sujet TD4)
	 */

	/* BufferedReader.readLine() throws IOException
	 * Returns: A String containing the contents of the line, not including
	 * any line-termination characters, or null if the end of the stream has
	 * been reached without reading any characters.
	 * (JavaDoc)
	 */

    @Override
    public String readLine() throws IOException {
        String newLigne = null;
        do {
		newLigne = super.readLine(); // Call to BufferedReader.readLine()
        }while( (newLigne != null) &&  (newLigne.contains(grep) == false) );
        /* Si la ligne n'inclut pas le contenu de la chaîne grep,
         * nous l'ignorons et passons à la ligne suivante.
         */

        return (newLigne);
    }

}
```

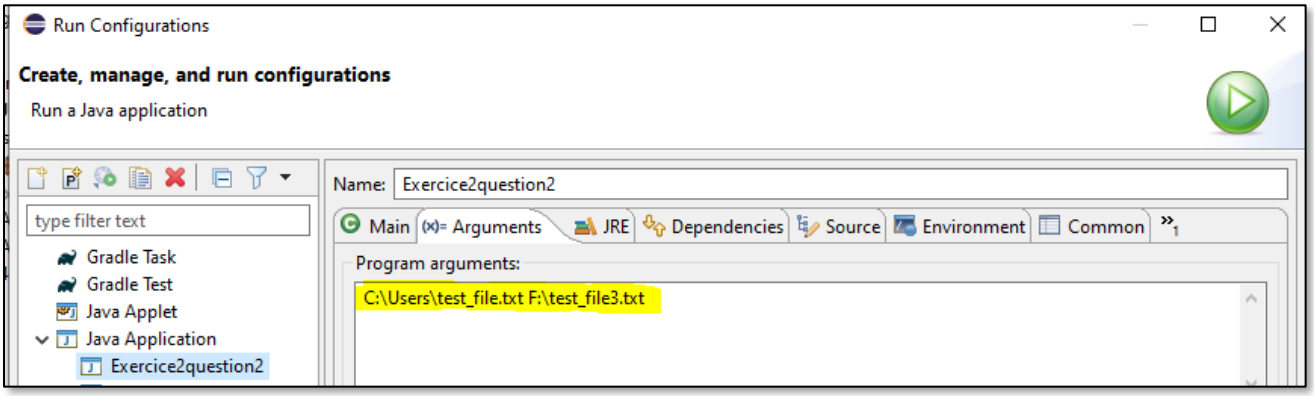
```java
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
public class GrepReaderTest {
	public static void main(String[] args) {
		/* String[] args est un tableau de String representant les parametres
		 * du programme sur la ligne de commande.
		 * (Cours PAA Chapitre 1 / Page 21)
		 */

		if(args.length < 2) {
			System.out.println("Utilisation sur la ligne de commande : java GrepReaderTest
nom_fichier_texte une_chaine");
			System.exit(1);
			/* System.exit(int status)
			 * Terminates the currently running Java Virtual Machine.
			 * The argument serves as a status code; by convention, a non zero status code
indicates abnormal termination.
			 * (JavaDoc)
			 */
		}

		String fileName = args[0];
		String grep = args[1];

		/* Exemple de procedure standard pour lire les lignes d'un fichier
		 * (Cours PAA Chapitre 3 / Page 57)
		 */

		FileReader fReader = null;
		try {
			// FileReader(String fileName) throws FileNotFoundException
			fReader = new FileReader(fileName);
		} catch (FileNotFoundException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}


		try {
			GrepReader gReader = new GrepReader(fReader, grep);

			String newLigne = null;
			// GrepReader.readLine() throws IOException
			while( (newLigne = gReader.readLine()) != null )
				System.out.println( newLigne );
		} catch(IOException e) {
				System.out.println( e.getMessage() );
		}

	} // main
} // class
```

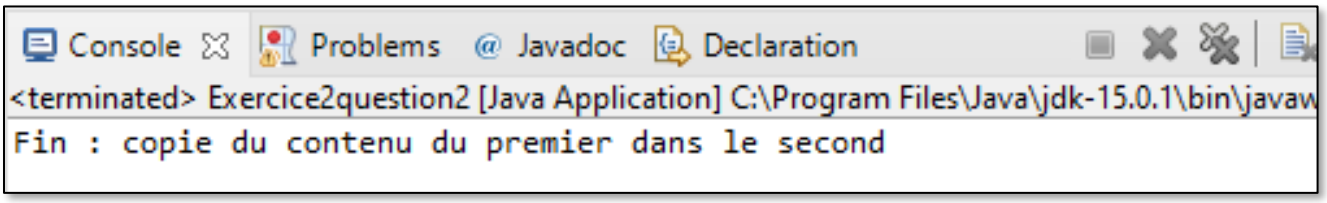
2. Écrivez un programme qui prend en entrée le nom d'un fichier texte, et qui affiche sur la console le nombre de lignes, de mots, et de caractères dans ce fichier, suivi du nom du fichier.
   On prend en compte les caractères '\n', '\t' et '␣' pour les séparateurs de mots.

```java
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.IOException;
public class TD4question2 {
	/* Écrivez un programme qui prend en entrée le nom d'un fichier texte,
	 * et qui affiche sur la console le nombre de lignes, de mots,
	 * et de caractères dans ce fichier, suivi du nom du fichier.
	 */

	public static void main(String[] args) {
		if(args.length < 1) {
			System.out.println("Utilisation sur la ligne de commande : java TD4question2
chemin_fichier");
			System.exit(1);
		}

		String filePath = args[0];

		/* Exemple de procedure standard pour lire les lignes d'un fichier
		 * (Cours PAA Chapitre 3 / Page 57)
		 */
		FileReader fReader = null;
		try {
			// FileReader(String fileName) throws FileNotFoundException
			fReader = new FileReader(filePath);
		} catch (FileNotFoundException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}

		BufferedReader bReader = new BufferedReader(fReader);
		StringBuilder str = new StringBuilder("");
		int numberOfLignes = 0;
		try {
			String newLine;
			while( (newLine = bReader.readLine()) != null ) {
				numberOfLignes++;
				str.append( newLine );
				str.append("\n");
			}
		} catch (IOException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}

		System.out.println("Nom du fichier : " + filePath);
		System.out.println("Nombre de lignes : " + numberOfLignes);
		System.out.println("nombre de caractères : " + str.toString().length());

		// On prend en compte les caractères '\n','\t'et ' ' pour les séparateurs de mots.
		System.out.println("Nombre de mots : " + str.toString().split("[ \n\t]").length);

		/* String[] split(String regex)
		 * Splits this string around matches of the given regular expression.
		 * Returns: the array of strings computed by splitting this string
		 * around matches of the given regular expression.
		 */

	} // main
} // class
```



3. Écrivez un programme qui prend en entrée deux noms de fichiers[2] et copie le contenu du premier dans le second. Si le second fichier existe déjà, son contenu initial est écrasé, dans le cas contraire le fichier est créé.





```java
import java.io.FileInputStream;
import java.io.FileOutputStream;

import java.io.FileNotFoundException;
import java.io.IOException;

import java.io.File;
public class Exercice2question2 {
	public static void main(String[] args) {
		if(args.length < 2) {
			System.out.println("Utilisation sur la ligne de commande : java Exercice2question2 chemin_premier_fichier chemin_second_fichier");
			System.exit(1);
		}

		/* Lire et ecrire des donnees brutes dans un fichier
		 * FileInputStream  et FileOutputStream : lecture/ écriture de donnees brutes
		 * La lecture et l'ecriture se font avec les methodes de bas niveau heritees
		 * des classes InputStream et OutputStream
		 * Utile si on doit manipuler directement des donnees non textuelles
		 * (image, son, fichier ex écutable,...)
		 */

		/* InputStream : la base de la lecture de donnees
		 * Classe abstraite, mere de toutes les classes qui correspondent
		 * a des flux d'entree d'octets
		 * La methode abstrait read()  doit etre definie par les classes filles
		 * concretes ; elle retourne un int qui correspond au prochain octet
		 * a lire dans le flux d'entree
		 */

		/* OutputStream : la base de l'ecriture de donnees
		 * Contre partie d'InputStream : classe abstraite, mere
		 * de toutes les classes qui correspondent a des flux de sortie d'octets
		 * La methode abstrait  write (int b) doit etre definie par les classes
		 * filles concretes ; elle ecrit un octet donnee sous forme d'int dans le flux de sortie
		 * Sortie de bas niveau : on recupere directement les octets sans tenir compte du genre de donnees manipule
		 */

		FileInputStream file1 = null;
		FileOutputStream file2 = null;
		try {
			// FileInputStream(String name) throws FileNotFoundException
			file1 = new FileInputStream( args[0] );

			// FileOutputStream(String name) throws FileNotFoundException
			file2 = new FileOutputStream( args[1] );
		} catch(FileNotFoundException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}

		/*  int FileInputStream.read() throws IOException
		 * Reads a byte of data from this input stream.
		 * This method blocks if no input is yet available.
		 * Returns : the next byte of data, or -1 if the
		 * end of the file is reached.
		 * Throws : IOException - if an I/O error occurs.
		 *
		 */

		try {
			int LectureProchainOctet;
			while( (LectureProchainOctet = file1.read()) != -1 )
				file2.write( LectureProchainOctet );

			/* void FileOutputStream.write(int b) throws IOException
			 * Writes the specified byte to this file output stream.
			 * Parameters : b the byte to be written.
			 * Throws : IOException - if an I/O error occurs.
			 */

			System.out.println("Fin : copie du contenu du premier dans le second");
		} catch (IOException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}

	} // main
} // class
```

Exercice III (Sauvegarde de répertoire)
On réutilise la classe RepertoireAmeliore utilisée précédemment.

TD 3

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

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

On souhaite améliorer les répertoires avec de nouvelles fonctionnalités :

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

Créez une classe RepertoireAmeliore, qui hérite de RepertoireSimple, et dont l'utilisation est illustrée dans le code suivant :

```
public class TestRepertoireAmeliore {
```

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

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

```java
public class Personne implements Comparable<Personne>{
	private String prenom;
	private String nom;
	private String telephone;

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

	public String getNom() {
		return nom;
	}

	public String getPrenom() {
		return prenom;
	}

	public String getTelephone() {
		return telephone;
	}

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

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

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

public class RepertoireSimple {
	private List<Personne> rep;

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

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

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

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

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

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

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

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

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

	/* Types bornes : borne superieure T extends A>
	 * On peut indiquer que les elements de la List doivent appartenir a un sous-type de A
	 * On utilise le mot-cle extends
	 * que A soit une classe ou une interface
	 */

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

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

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

On réutilise la classe RepertoireAmeliore utilisée précédemment. On souhaite pouvoir enregisitrer un répertoire dans un fichier texte, ou au contraire charger un répertoire à partir d'un fichier texte. Le fichier est composé d'autant de lignes que de personnes dans le répertoire :

- `proprietaire(XXX,YYY,ZZZ)` pour la description du propriétaire du répertoire, avec `XXX` le prénom, `YYY` le nom et `ZZZ` le numéro de téléphone ;
- `contact(XXX,YYY,ZZZ)` pour la description des contacts, avec `XXX` le prénom, `YYY` le nom et `ZZZ` le numéro de téléphone.

1. Écrivez une classe ParserRepertoire qui permet de lire un fichier texte décrivant un répertoire, et de l'utiliser pour construire une instance de RepertoireAmeliore.
2. Écrivez une classe SauvegardeRepertoire qui permet d'enregistrer un RepertoireAmeliore dans un fichier texte.

```java
import java.io.FileWriter;
import java.io.IOException;
import java.io.BufferedWriter;
import java.io.PrintWriter;

/**
 * classe SauvegardeRepertoire qui permet d'enregistrer
 * un RepertoireAmeliore dans un fichier texte.
 */
public class SauvegardeRepertoire {
        public static void saveRepertoireToFile(String fileName, RepertoireAmeliore repertoire) {
                /* FileReader FileWriter sont des implémentations de Reader et Writer
                 * pour la gestion de fichiers de textes.
                 * Les différents constructeurs prennent toujours :
                 * soit un objet de type File, soit une chaîne de
                 * caractères représentant le chemin du fichier.
                 * (Cours POO L2)
                 */
                FileWriter fWriter = null;
                try {
                        // FileWriter(String fileName) throws IOException
                        fWriter = new FileWriter( fileName );
                } catch(IOException e) {
                        System.out.println( e.getMessage() );
                        System.exit(1);
                }

                // PrintWriter(Writer out)
                BufferedWriter bWriter = new BufferedWriter( fWriter );
                /*  BufferedWriter est une classe fille de Writer
                 * qui propose une écritures efficace de caracteres
                 * dans un flux de sortie. Un BufferedWriter peut
                 * uniquement écrire dans un autre Writer, et rend
                 * les écritures plus efficaces
                 */

                // PrintWriter(Writer out)
                PrintWriter pWriter = new PrintWriter( bWriter );
                /* La classe PrintWriter fournit des methodes d'ecriture
                 * ( print (...) / println (...) )
                 * Sans le BufferedWriter , l'appel a pw.println ´
                 * necessiterait, pour chaque caractere a écrire,
                 * sa conversion en octet et son écriture sur le fichier.
                 */
                pWriter.print("proprietaire(");
                pWriter.print(repertoire.getProprietaire().getPrenom() + ",");
                pWriter.print(repertoire.getProprietaire().getNom() + ",");
                pWriter.println(repertoire.getProprietaire().getTelephone() + ")");

                for(Personne personne : repertoire.getRep()) {
                        pWriter.print("contact(");
                        pWriter.print(personne.getPrenom() + ",");
                        pWriter.print(personne.getNom() + ",");
                        pWriter.println(personne.getTelephone() + ")");
                }


        } // saveRepertoireToFile
} // class
```

```java
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.BufferedReader;
import java.io.IOException;

import java.util.List;
import java.util.ArrayList;
/**
 * une classe ParserRepertoire qui permet de lire un fichier texte décrivant un répertoire,
 * et de l'utiliser pour construire une instance de RepertoireAmeliore.
 */
public class ParserRepertoire {
	public static RepertoireAmeliore createRepertoireAmelioreFromFile(String fileName) {
		FileReader fReader = null;
		try {
			// FileReader est une implémentation de Reader
			// pour la gestion de fichiers de textes (Cours POO / L2)
			fReader = new FileReader( fileName );
			/* FileReader(String fileName) throws FileNotFoundException
			 * Creates a new FileReader, given the name of the file to read,
			 * using the platform's default charset.
			 * Parameters : fileName the name of the file to read
			 * Throws : FileNotFoundException - if the named file does not exist,
			 * is a directory rather than a regular file,
			 * or for some other reason cannot be opened for reading.
			 * (JavaDoc)
			 */
		} catch (FileNotFoundException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}

		BufferedReader bReader = new BufferedReader( fReader );
		/* BufferedReader(Reader in)
		 * Creates a buffering character-input stream that
		 * uses a default-sized input buffer.
		 * Parameters : in A Reader
		 * (JavaDoc)
		 */

		/* Le principal intérêt de la classe BufferredReader
		 * est la méthode readLine() qui va lire une ligne de texte.
		 * (Cours POO L2 / Page 201)
		 */

		List<Personne> contacts = new ArrayList<Personne>();
		Personne proprietaireDuRepertoire = null;

		try {
			String nextLine;
			// readLine() throws IOException
			while( (nextLine = bReader.readLine()) != null ) {
				// String.startstWith()
				// Tests if this string starts with the specified prefix.
				if( nextLine.startsWith("contact") )
				{
					contacts.add(  createPersonneFromLineOfFile(nextLine) );
				}
				else
				{
					if( nextLine.startsWith("proprietaire") )
					{
						proprietaireDuRepertoire = createPersonneFromLineOfFile(nextLine);
					}
					else
					{
						System.out.println("Error !");
						System.exit(1);
					} // else
				} // else
			} // while
		} catch(IOException e) {
			System.out.println( e.getMessage() );
			System.exit(1);
		}

		RepertoireAmeliore repertoire = new RepertoireAmeliore( proprietaireDuRepertoire );
		for(Personne personne : contacts)
			repertoire.addPersonne(personne.getPrenom(), personne.getPrenom(), personne.getTelephone());
		return repertoire;
	} // createRepertoireAmelioreFromFile

	public static Personne createPersonneFromLineOfFile(String lineFromFile) {
		Personne personne;
		String[] str1 = lineFromFile.split("\\(");
		String[] str2 = str1[1].split("\\)");
		String[] str3 = str2[0].split(",");

		personne = new Personne(str3[0], str3[1], str3[2]);
		return personne;
	} // createPersonneFromLineOfFile

} // class
```

Ajouter une fonction *get* dans RepertoireAmeliore

```
	public Personne getProprietaire() {
		return proprietaire;
	}
```

a