EXEMPLES DU COURS

Programmation Java

2éme année Ingenieur

Remarques :

- Ce Document reprend uniquement les exemples donnés en cours par Mme Nouira. Il ne constitue en aucun cas un support de cours complet.

Ce document doit être Imprimer pour que les étudiants puissent suivre chaque exemple.

Les Principes de la programmation OO

Exemple1 :

```
package enginevehicle1;
class Engine {
  // attributs
    String company;
    int horsePower;
     // constructeur
   Engine (String company,int horsePower)
   { this.horsePower = horsePower;
   this.company=company;}
   // on utilise this en cas de confusion entre le paramètre et l'attribut

    // méthode vitesse du moteur
   double speedEngine(double timeSec, int weightPounds)
   { return horsePower*timeSec*47997.64/weightPounds;}
   // méthode d'affichage
   void afficher(){
      System.out.println("Ce moteur est "+ company +" dont le nombre de chevaux est " + horsePower );
   }
}

package enginevehicle1;
public class EngineVehicle1 {
    public static void main(String[] args) {
        double timeSec = 10.0;
        int horsePower = 246;
        Engine engine = new Engine("Marque",horsePower);
        engine.afficher();
    }

}
```

Exemple2 :

```
public class Engine {
 // attributs: conseillé qu'il soient privés
    private String company;
    private  int horsePower;

 // constructeur
      Engine (String company,int horsePower)
   { this.horsePower = horsePower;
   this.company=company;}
      // méthode vitesse du moteur
   public double speedEngine(double timeSec, int weightPounds)
   { return horsePower*timeSec*47997.64/weightPounds;}
   // méthode d'affichage
   public void afficher(){
      System.out.println("Ce moteur est "+ company +" dont le nombre de chevaux est " + horsePower );
   }
   // avec des attributs privés il vaut mieux utliser des getters-setters
   public void setCompany(String company)
   {  this.company=company; }
   public String getCompany()
   {
      return (company); }
```

```
  public void setHorsePower(int horsePower)
  {  this.horsePower=horsePower; }
  public int getHorsePower()
  {
     return (horsePower); }
}

public class EngineVehicle2 {

   public static void main(String[] args) {
        int horsePower = 246;
      Engine engine = new Engine("Marque",horsePower);
      engine.afficher();
      engine.setHorsePower(300);
      engine.afficher();
      System.out.println(engine.getCompany());
   }

}
```

Exemple 3 :

```
public class Engine {
   // attributs: conseillé qu'il soient privés
   private String company;
   private  int horsePower;

 // constructeur
     Engine (String company,int horsePower)
  { this.horsePower = horsePower;
  this.company=company;}
    // méthode vitesse du moteur
  public double speedEngine(double timeSec, int weightPounds)
  { return horsePower*timeSec*47997.64/weightPounds;}
  // méthode d'affichage
  public void afficher(){
     System.out.println("Ce moteur est "+ company +" dont le nombre de chevaux est " + horsePower );
  }
  // avec des attributs privés il vaut mieux utliser des getters-setters
  public void setCompany(String company)
  {  this.company=company; }
  public String getCompany()
  {
     return (company); }

  public void setHorsePower(int horsePower)
  {  this.horsePower=horsePower; }
  public int getHorsePower()
  {
     return (horsePower); }
}

public class Vehicle {
  // attributs
   private Engine engine;
   private int weightPounds;
 // constrcuteur
   public Vehicle(Engine engine,int weightPounds)
   {   this.engine=engine;
      this.weightPounds=weightPounds; }
```

```
    public double speedEngineVehicle(double timeSec)
    {
       return engine.speedEngine(timeSec, weightPounds);
    }
     // getters-setters
     public void setWeightPounds(int weightPounds)
    {  this.weightPounds=weightPounds; }
    public int getWeightPounds()
    {
       return weightPounds; }
    public void setEngine(Engine engine)
    {
       this.engine = engine;
    }
    public Engine getEngine ()
    {
       return engine;
    }
}

public class Car extends Vehicle {
private int passengersCount;
public Car(Engine engine,int weightPounds, int passengersCount ){
super(engine,weightPounds);
this.passengersCount = passengersCount;
}
// getters-setters
public int getPassengersCount() {
   return passengersCount;
}
public void setPassengersCount(int passengerCount) {
   this.passengersCount = passengersCount;
}

}


public class EngineVehicle3 {
   public static void main(String[] args) {
double timeSec = 10.0;
int horsePower = 246;
int vehicleWeight = 4000;
Engine engine = new Engine("Marque",horsePower);
Vehicle vehicle = new Vehicle(engine,vehicleWeight);
Car car = new Car(engine,vehicleWeight,4);
// aussi permise:
//Vehicle car = new Car(engine,vehicleWeight,4);
/*grace au polymorphisme, la reference à un objet de classe Car peut etre assignée
à la référence de la classe mère Vehicle.L'objet a deux types: Car et vehicle*/
System.out.println("Vehicle speed (" + timeSec + " sec)=" + vehicle.speedEngineVehicle(timeSec) + "
mph");
System.out.println("Car speed (" + timeSec + " sec)=" + car.speedEngineVehicle(timeSec) + " mph");
   }

}
```

un exemple complet : Encapsulation -Héritage - Polymorphisme

//******la classe Ville

```
public class Ville {
 private String nom;
 private int nbHabitants;

/********* 3 constructeurs ***********************************************
* 1 .....
* 2 .....
* 3 .....*/
public Ville() {
 nom="VILLE INCONNUE";
}
public Ville(String nomVille) {
 nom=nomVille.toUpperCase();
}
public Ville(String nomVille, int nombre) {
 this(nomVille);      //.....
 if (nombre<=0) erreur("nbHabitants doit être positif !");
 else nbHabitants=nombre;
}

        //******** qq méthodes d'accès aux variables privées *********************
public String getNom() {
 ..............................
}
public void setNom(String nomVille) {
 if (nom.equals("VILLE INCONNUE") ) nom=nomVille.toUpperCase();
 else erreur(" il est interdit de changer le nom de la ville !");
}
public boolean nbHabConnu() {
 return nbHabitants >0;
}
public int getNbHabitants() {
 if (!nbHabConnu()) erreur("nombre d'habitants inconnu ");
 return nbHabitants;
}
public void setNbHabitants(int nombre) {
 if (nombre<=0) erreur("nbHabitants doit être positif !");
 else nbHabitants=nombre;
}

//*********** méthode pour décrire un objet Ville ************************/
public String Affiche() {
 String ch="Ville( "+nom+", ";
 if (nbHabConnu()) ch+=nbHabitants+" ) ";
 else  ch+=" nombre d'habitants inconnu )";
 return ch;
}
/****** méthode privée de classe pour signaler les erreurs ***************/
private static void erreur(String message) {
System.out.println("--> erreur classe Ville : "+message);
System.exit(1);
}
}
```

//*******fin classe Ville

/******Classe d'essai EssaiVille1 de la classe Ville

```
* création de 3 objets Ville, utiliser les 3 constructeurs,dont le constructeur par défaut. */
```

```
public class EssaiVille1 {
   public static void main(String args[]) {

   Ville t =new Ville();
   System.out.println(t.Affiche());

   Ville v =new Ville("Tunis");
   System.out.println(v.Affiche());

/** essai de changement direct de nom pour la ville v
 ……
 Ville w =new Ville("Monastir");
 System.out.println(w.Affiche());
 w.setNbHabitants(100000);
 System.out.println(w.Affiche());

// Bien sûr il est préférable d'utiliser le constructeur "complet"
 Ville e = new Ville("Sousse", 60000);
 System.out.println(e.Affiche());

// et si on fait une erreur sur le nombre d'habitants ?
 Ville f = new Ville("Sousse", -60000);
 System.out.println(f.Affiche());
 f.setNbHabitants(50000);
 }
}
```

Exécution obtenue

……
……
……
……

//********** la classe Capitale **********/

```
public class Capitale extends Ville {
 private String pays;

// définition de 3 constructeurs
/** ce constructeur est-il correct ?   ……*/
public Capitale(String nomPays) {
  pays = nomPays.toUpperCase();
}
/** Que fait le deuxième constructeur ?      ……*/
public Capitale(String nomVille, String nomPays) {
  this(nomPays);
  setNom(nomVille);
}
/** Que fait Ce troisième constructeur ?     …… */
public Capitale(String nomVille, String nomPays, int nombre) {
 super(nomVille, nombre);
 pays = nomPays.toUpperCase();
}

//********** méthodes d'accès à la variable privée pays **********/
public String getPays() {
     ……;
}
public void setPays(String nomPays) {
 ……;
}
```

```
//**Affiche() redéfinir pour intégrer le pays en utilisant la methode Affiche()*************/

public String Affiche() {
 return  …… +" capitale de "+ pays;
}
```

//******** fin classe Capitale

/******** Classe d'essai EssaiCapitale1 de la classe Capitale**********/

```
/* 1er essai d'utilisation de la classe Capitale --> sans redéfinition de la méthode Affiche()
, les capitales ici se présentent comme de simples villes.
* --> création de 3 capitales, pour utiliser les 3 constructeurs,dont le constructeur par
défaut.*/

public class EssaiCapitale1 {
   public static void main(String args[]) {

/** Quel est le constructeur de Capitale appelé          ……*/
   Capitale c1 = new Capitale("Tunisie");
   System.out.println(c1.Affiche());

   Capitale c2 = new Capitale("madrid","espagne");
   System.out.print(c2.Affiche());
   System.out.println(" capitale de "+ c2.getPays());

   Capitale c3 = new Capitale("rome","italie",1000000);
   System.out.print(c3.Affiche());
   System.out.println(" capitale de "+ c3.getPays());
}
}
```

Exécution obtenue

……
……
……
……

/*** Classe d'essai EssaiCapitale2 de la classe Capitale ***/

```
/** 2ème essai d'utilisation de la classe Capitale après la redéfinition dans cette classe  de
la méthode Affiche(). cette redéfinition fait d'ailleurs appel à la méthode Affiche()de la
sur-classe Ville. */

public class EssaiCapitale2 {

public static void main(String args[]) {
 Capitale c1 = new Capitale("Tunisie");
 System.out.println(c1.Affiche());

 Capitale c2 = new Capitale("madrid","espagne");
 System.out.println(c2.Affiche());

 Capitale c3 = new Capitale("rome","italie",1000000);
 System.out.println(c3.Affiche());
}
```

Exécution obtenue

……
……

/******** Classe d'essai EssaiCapitale3 **********

```
/** 3ème essai : illustration du polymorphisme la méthode Affiche(), a été réécrite dans la
classe Ville,  et redéfinie dans Capitale pour permettre au code de s'adapter, au moment de
l'exécution, à l'objet auquel la méthode est adressée.*/
```

```
// méthode polymorphe pour décrire un objet Ville

public String Affiche() {
 String ch=getClass().getName() ;
 ch+= "( "+nom+", ";
 if (nbHabConnu()) ch+=nbHabitants+" ) ";
 else  ch+=" nombre d'habitants inconnu )";
 return ch;
}

public class EssaiCapitale3 {
   public static void main(String args[]) {

   Ville v ;
     v = new Ville("rome",1000000);
   System.out.println(v.Affiche());

   v = new Capitale("rome","italie",1000000);
   System.out.println(c.Affiche());
}
}
```

Exécution obtenue

.....
.....
.....

I - Package et encapsulation

Exemple 1 : Classe imbriquée

```
public class Voiture {
        class Roue {
                private String modele ;
                Roue(String modele) { this.modele = modele ; }
        }
        private Roue[] roues = new Roue[4] ;
        private int puissance =10 ;


        public Voiture(String modele_roue, int puissance) {
                this.puissance = puissance ;
                for ( int i=0 ; i<roues.length ; i++ ) roues[i] = new Roue(modele_roue) ;
        }
}
```

II – Interface

Exemple 2

```
interface Printable {          /* exprime le fait de pouvoir être imprimé */
        void print() ;
}

interface InputStream { /* exprime le fait de pouvoir être une source de caractères */
        public int read() ;
}
interface OutputStream { /* exprime le fait de pouvoir accepter des caractères */
        public void write(int) ;
}

interface DataStream extends InputStream{
        public double readDouble() ;
        public void writeDouble(double) ;
}


class MonStream implements DataStream, Printable {
        void print() { // Définition de la méthode …}
        public int read() { // Définition de la méthode …}
        public double readDouble(){// Définition de la méthode …}
        public void writeDouble(double) { // Définition de la méthode …}
}
```

Exemple 3 : L'operateur intanceof

Une interface définie un nouveau type : il est possible de définir des références du type d'une interface, c'est à dire permettant de référencer des objets dont tout ce que l'on sait est qu'ils implémentent cette interface.

```
interface Persistent { void save() ; }

class Client extends Personne implements Persistent {
        private int numero ;
        …
        public void save() {
                /* Enregistrement sur fichier ou Base de donnée */
        }
}

class PersistentManager {
        private final static int MAX=100 ;
        private static Persistent[] liste_des_objets = new Persistent[MAX] ;
        static  int nb = 0 ;
        …
        public static void addPersistent(Persistent objet) {
                if (nb<MAX) liste_des_objets[nb++] = objet ;
        }

        public static void saveAll () {
                for (int i=0 ; i<liste_des_objets.length ; i++)
                        if ( liste_des_objets[i] != null ) liste_des_objets[i].save() ;
        }
}


        …
        Client client = new Client("Toto",4529) ;
        PersistentManager.addPersistent(client) ;
        …
        PersistentManager.saveAll() ;
        …
```

L'opérateur **instanceof** peut être utilisé pour savoir si un objet implémente une interface donnée :

```
Point point = new Point() ;
…
if ( point instanceof Printable ) ((Printable)point).print() ;
…
```

L'exemple suivant met en œuvre le polymorphisme au travers de l'utilisation d'une interface. Cette interface définie un service (Persistent) qui est implémenté par un certain nombre de classes de l'application (la classe Client dans l'exemple).
On stocke une collection d'objet implémentant le service Persistent dans un tableau de type Persistent. Chacun de ces objets dispose donc de la méthode save(), avec sa propre implémentation. La classe PersistentManager peut alors manipuler tous ces objets avec l'interface Persistent, sans rien savoir de leur nature réelle.

Exemple Package :

- Reprendre l'exemple Ville-Capital et définir un package Ville qui contient uniquement la classe ville, quelle modification faut –il apporter au programme.

III- Les Exceptions

Exemple 4 : Propagation des exceptions , le mot clé throws

```
class PasDeSolution extends Exception {}

class Equation { /* Equation du second degré ax2+bx+c */
        private double a, b, c ;
        public Equation(double a, double b, double c) { this.a = a ; this.b = b ; this.c = c ; }

        public double resultat() throws PasDeSolution { // Cette méthode propage une exception
                double discriminent  = b*b-4*a*c ;
                if (discriminent < 0) throw new PasDeSolution() ;
                return ( b + Math.sqrt(discriminent) ) / (2 * a) ;
        }
}
…
void calcul() throws PasDeSolution {
        Equation eq = new Equation(1,0,1) ;
        Eq.resultat() ;
}
// Cette méthode doit déclarer la propagation de l'exception PasDeSolution que Eq.Solution() peut
// déclencher, car elle ne la traite pas localement
```

Exemple 5 : Capture des exceptions : catch, try et finally

```
void calcul() {
        try {
                Equation eq = new Equation(1,0,1) ;
                double resultat = Eq.resultat() ;
                System.out.println("Resultat = " + resultat) ;
        }
        catch ( PasDeSolution e ) {
                System.out.println("Pas de solutions") ;
        }
}
```

Exercice TD : Soit une classe **EntierNaturel** permettant de manipuler des entiers naturels (positifs ou nuls). Cette classe disposera :

- D'un constructeur à un argument de type **int**
- D'une méthode **getNbr** fournissant sous forme d'un **int**, la valeur encapsulée dans un objet de type EntierNaturel

1) Réaliser la classe EntierNaturel en identifiant l'emplacement de génération d'une exception que vous pouvez appeler ExceptNeg
2) Implémenter la classe ExceptNeg de type Exception
3) Ecrire un programme d'utilisation qui traite ExceptNeg en affichant un message d'erreur, et sortir dans tous les cas.
4) Implémenter dans la classe EntierNaturel des méthodes statiques de somme, de différence et de produit de deux naturels ; elles génèreront respectivement des exceptions ExceptSom et ExceptDiff lorsque le résultat ne sera pas représentable. Noter bien que la limite des valeurs des naturels sera fixée à la plus grande valeur de type int (Integer.Max_Value).
5) Implémenter les différentes classes de type Exception.
6) Ecrire deux exemples d'utilisation de la classe EntierNaturel :

- L'un se contentant d'intercepter sans discernement les exceptions, on peut identifier une superclasse ExceptNaturel
- L'autre qui explicite la nature de l'exception en affichant les informations disponibles.

Les deux exemples pourront figurer dans deux blocs try d'un même programme

Correction

1)

```
public class EntierNaturel
{
  private int Nbr ;
  // constructeur
  public EntierNaturel (int Nbr) throws
  ExceptNeg
  {
  // il existe une exception lorsque le nombre
  donné est négatif
  // on peut générer une exception qu'on peut
  appeler ExceptNeg
  if (Nbr < 0) throw new ExceptNeg(Nbr) ;
  this.Nbr = Nbr ;
  }
  public int getNbr () { return Nbr ;}
}
```

2)

```
public class ExceptNeg extends Exception
{
        private int value;
        // constructeur
        public ExceptNeg (int value)
        { this.value = value ;}
        public int getValue { return value;}
}
```

3)

```
public class Testentiernaturel
{
 public static void main (String args[])
 {
 // on peut tester en changeant les valeurs tels que
 100, -50…
 int n;
 System.out.println("donner un entier naturel : ") ;
 n = Clavier.lireInt() ;
try
{
 EntierNaturel Nbr1 = new EntierNaturel (n);
 System.out.println("le nombre naturel est valide et
 =" + Nbr1.getNbr()) ;
 // la question à penser : pourquoi on a utilisé
 getNbr et non pas Nbr1.Nbr ?
}
catch ( ExceptNeg e)
{
 Sytem.out.println ("**********erreur de
 construction du nombre********** ") ;}
finally
  {      System.exit(-1) ;}
}
}
```

4)

```
public class EntierNaturel
{ …
  public  static EntierNaturel somme (EntierNaturel Nbr1,
  EntierNaturel Nbr2) throws ExceptSom,

  ExceptNeg
  {
  int op1 = Nbr1.Nbr; int op2= Nbr2.Nbr;
  int S = op1 + op2 ;
  if (S > Interger.Max_Value) throw new ExceptSom
  (op1,op2);
  EntierNaturel res = new EntierNaturel (S);
  return res;
}
 public  static EntierNaturel diff (EntierNaturel Nbr1,
 EntierNaturel Nbr2) throws ExceptDiff, ExceptNeg
 {
    int op1 = Nbr1.Nbr; int op2= Nbr2.Nbr;
    int D = op1 - op2 ;
    if (D<0) throw new ExceptDiff (op1,op2);
    // autre façon en utilisant classe anonyme
    return new EntierNaturel(D);
 }
```

5)

```
public class ExceptSom extends Exception
{
        public int value1, value2;
        // constructeur
        public ExceptSom (int value1, int value2)
        { this.value1 = value1 ;
        this.value1 = value1 ;}
}

public class ExceptDiff extends Exception
{
```

6) On peut définir une classe : class ExceptNaturel extends Exception {}

Les autres seront des sous-classes :

```
class ExceptNeg extends ExceptNaturel {…….}
class Except_op extends ExceptNaturel{…….}
class ExceptSom extends Except_op {……..}
class ExceptDiff extends Except_op {……}
```

```
public class Test2entiernaturel
{
```

```
        public int value1, value2;
        // constructeur
        public ExceptDiff (int value1, int value2)
        { this.value1 = value1 ;
        this.value1 = value1 ;}
}

//une autre façon est de créer une classe Except_op et des sous classes  ExceptSom et ExceptDiff, et appeler super(value1, value2)
```

```
public static void main (String args[])
{
 // on peut tester en changeant les valeurs tels que 100, -50…
 int n1, n2;
 System.out.println("donner un premier entier naturel : ") ;
 n1 = Clavier.lireInt() ;
 System.out.println("donner un second entier naturel : ") ;
 n2 = Clavier.lireInt() ;
 try{
     EntierNaturel Nbr1 = new EntierNaturel (n1);
     EntierNaturel Nbr2 = new EntierNaturel (n2);
     EntierNaturel Som =  EntierNaturel.somme (n1,n2) ;
     EntierNaturel Di =  EntierNaturel.diff (n1,n2) ;
 }
// exception sans différentiation
 catch ( ExceptNaturel e)
 { Sytem.out.println ("****erreur entier naturel******* ") ;}
// test avec différentiation des exceptions
 try {
       EntierNaturel Nbr1 = new EntierNaturel (n1);
       EntierNaturel Nbr2 = new EntierNaturel (n2);
       EntierNaturel Som =  EntierNaturel.somme (n1,n2) ;
       EntierNaturel Di =  EntierNaturel.diff (n1,n2) ;
  }
 catch ( ExceptNeg e)
 { Sytem.out.println ("erreur construction entier
                          naturel "+e.getValue()) ;  }
 catch ( ExceptSom e)
{ Sytem.out.println ("erreur somme des entiers
                       naturels "+e.value1 +" "+e.value2 ) ;  }
catch ( ExceptDiff e)
{ Sytem.out.println ("erreur difference des entiers
                         naturels "+e.value1 +" "+e.value2 ) ; }
finally
        {System.exit(-1) ;}
  }
}
```

TRAVAIL A faire

Dans le code source suivant se trouve une méthode solveEquation() qui résout une équation quadratique :

```
public static double[] solveEquation(double A, double B, double C)
{    if (A==0 && B==0) {      ; // Error: no solution    }
       else { double discriminant = B * B - 4 * A * C;
      if (discriminant < 0 ; // Error: discriminant < 0
      double sol1 = (-B + Math.sqrt(discriminant)) / (2 * A);
      double sol2 = (-B - Math.sqrt(discriminant)) / (2 * A);
      return new double[] {sol1, sol2};    }
    return null;  // To avoid compiler error (remove after code completion)
}
```

Modifier la méthode solveEquation() de telle manière que les erreurs qu'elle peut produire soient signalées par des exceptions (par ex. du type IllegalArgumentException) qui seront traitées dans la méthode main().

Dans la méthode main() répéter la lecture des paramètres (coefficients) et l'affichage de la solution (ou d'un message d'erreur) jusqu'à ce que l'utilisateur souhaite arrêter l'application.

IV - Les classes de Bases

Exemple 6 : Classes Object

```
class Personne  { // dérive de Object par défaut
        private String nom ;
        public Personne(String nom) { this.nom = nom ; }
        public String toString() {
                return "Classe : " + getClass().getName() + " Objet : " + nom ;
        }
        boolean equals(Personne p) { return p.nom.equals(nom) ; }
}

Personne p1 = new Personne("Jean Dupond") ;
Personne p2 = new Personne("Jean Dupond") ;

if ( p1 == p2 ) … // Faux, les références sont différentes
if (p1.equals(p2)) … // Vrai, comparaison sémantique

System.out.println( p1 ) ;
// Affiche : Classe : Personne Objet : Jean Dupond
```

Exemple 7 : Classes Wrapper

```
// Exemple d'accès aux valeurs min/max d'un type
double f ;
int i ;
…
if ( f > Integer.MIN_VALUE and f < Integer.MAX_VALUE) i = (int) f ;
…
```

```
// Exemple de conversion chaîne => entier (Variante 1)
int stoi(String s ) {
      try { return Integer.parseInt(s) ; }
      catch (Exception e) { return 0 ; }
}
```

```
// Exemple de conversion chaîne => entier (Variante 2)
int stoi(String s ) {
      return (new Integer(s)).intValue() ;
}
```

```
// Exemple de conversion chaîne => entier (Variante 3)
int stoi(String s ) {
      return Integer.valueOf(s).intValue() ;
}
```

```
// Exemple de conversion entier => chaîne (Variante 1)
String itos(int i ) {
      return (new Integer(i)).toString() ;
}
```

```
// Exemple de conversion entier => chaîne (Variante 2)
String itos(int i ) {
      return "" + i ;
}
```

```
// Exemple de conversion entier => chaîne (Variante 3)
String itos(int i ) {
      return String.valueOf(i)  ;
}
```

Exemple 8 : Classes Tokenizer

```
void AfficheParMots(String texte) {
            StringTokenizer st = new StringTokenizer(texte, ",:") ;

            while ( st.hasMoreTokens() ) {
                  String mot = st.nextToken() ;
                  System.out.println(mot) ;
            }
}
…      AfficheParMots("Lundi,Mardi:Mercredi;Jeudi");
// Resultat Affiche :
Lundi
Mardi
Mercredi;Jeudi
```

Exemple 9 : Classe Vector

```
Vector vec = new Vector() ;
for (int i=0 ; i<10 ; i++) {
      Integer element = new Integer(i) ;
      vec.addElement(element) ;  // Ajout en fin de Vecteur
}                                    // => 0 1 2 3 4 5 6 7 8 9
…
Integer i = new Integer(15) ;
vec.insertElementAt(i,5) ;                  // Insertion à la position indiquée
// => 0 1 2 3 4 15 5 6 7 8 9
…
vec.removeElementAt(0) ;                    // Suppression de l'élément indiqué
// => 1 2 3 4 15 5 6 7 8 9
…
Integer j = (Integer)vec.elementAt(6) ;   // j contient une référence sur l'objet Integer contenant 5
…
vec.removeAllElements() ;                   // Suppression de tous les éléments
// =>
```

VIII- Les Collections

Interface collection:

```
public interface Collection {
// Basic Operations
     int size();
      boolean isEmpty();
      boolean contains(Object element);
      boolean add(Object element); // Optional
      boolean remove(Object element); // Optional
      Iterator iterator();
      int hashCode();
              boolean equals(Object element);
              // Bulk Operations
              boolean containsAll(Collection c);
              boolean addAll(Collection c); // Optional
              boolean removeAll(Collection c); // Optional
              boolean retainAll(Collection c); // Optional
              void clear(); // Optional
              // Array Operations
              Object[] toArray();
Object[] toArray(Object a[]);}
```

Interface List

```
public interface List extends Collection {
        // Positional Access
        Object get(int index);
        Object set(int index, Object element); // Optional
        void add(int index, Object element); // Optional
        Object remove(int index); // Optional
        boolean addAll(int index, Collection c); // Optional
        // Search
        int indexOf(Object o);
        int lastIndexOf(Object o);
        // Iteration
        ListIterator listIterator();
        ListIterator listIterator(int index);
        // Range-view
        List subList(int fromIndex, int toIndex);
}
```

Interface Map

```
public interface Map {
            // Basic Operations
            Object put(Object key, Object value);
            Object get(Object key);
            Object remove(Object key);
            boolean containsKey(Object key);
            boolean containsValue(Object value);
            int size();
            boolean isEmpty();
            // Bulk Operations
            void putAll(Map t);
            void clear();
            // Collection Views
            public Set keySet();
            public Collection values();
            public Set entrySet();
            // Interface for entrySet elements
            public interface Entry {
            Object getKey();
            Object getValue();
            Object setValue(Object value);
}} values retourne les valeurs sous la forme d'une Collection.
```

Exemple Map

```
import java.util.*;
public class MapExample {
public static void main(String args[]) {
       Map map = new HashMap();
       Integer ONE = new Integer(1);
       for (int i=0, n=args.length; i<n; i++) {
     String key = args[i];
          Integer frequency =
                    (Integer)map.get(key);
       if (frequency == null) {
    frequency = ONE;
       }
       else {
    int value = frequency.intValue();
         frequency = new Integer(value + 1);
       }
     map.put(key, frequency);
       }
        System.out.println(map);
        Map sortedMap = new TreeMap(map);
        System.out.println(sortedMap);}
}
```

Les collections

Exemple 1 : Set

```
import java.util.HashSet;
import java.util.Set;
import java.util.TreeSet;

public class Ensembles {

    public static void main(String[] args) {
        //creation du set
Set<String> se =  new HashSet<String>() ;
// ajout d'element
System.out.println("J'ajoute un : " + se.add("un")) ;
System.out.println("J'ajoute deux : " +
se.add("deux")) ;
 // ajout d'un doublon : échec
System.out.println("J'ajoute encore un : " +
se.add("un")) ;
 // affichage de la taille du set
System.out.println("Taille du set : " + se.size()) ;
Set se1 = new HashSet(); // Une table de Hachage
se1.add("Bernadine");
se1.add("Elizabeth");
se1.add("Gene");
se1.add("Elizabeth");
se1.add("Clara");
System.out.println(se1);
Set setTrie = new TreeSet(se1); // Un Set trié
System.out.println(setTrie);
// avec forEach
setTrie.forEach(System.out::println);

    }

}
```

```
J'ajoute un : true
J'ajoute deux : true
J'ajoute encore un : false
Taille du set : 2
[Bernadine, Elizabeth, Gene, Clara]
[Bernadine, Clara, Elizabeth, Gene]
Bernadine
Clara
Elizabeth
Gene
```

Exemple 2 : List

```
import java.util.*;
public class ListExample {
public static void main(String args[]) {
  List list = new ArrayList();
  list.add("Bernadine");
  list.add("Elizabeth");
  list.add("Gene");
  list.add("Elizabeth");
  list.add("Clara");
  System.out.println(list);
  System.out.println("2: " + list.get(2));
  System.out.println("0: " + list.get(0));
  LinkedList queue = new LinkedList();
  queue.addFirst("Bernadine");
   queue.addFirst("Elizabeth");
  queue.addFirst("Gene");
  queue.addFirst("Elizabeth");
  queue.addFirst("Clara");
  System.out.println(queue);
  queue.removeLast();
  queue.removeLast();
  System.out.println(queue);}
}
```

```
Bernadine, Elizabeth, Gene, Elizabeth, Clara]
2: Gene
0: Bernadine
[Clara, Elizabeth, Gene, Elizabeth, Bernadine]
[Clara, Elizabeth, Gene]
```

```
import java.util.*;

public class ListeChainée {
public static void afficher(LinkedList li){
       ListIterator <String> iter = li.listIterator();
       while(iter.hasNext())
System.out.print(iter.next()+"  ");
           // avec forEach
       li.forEach(System.out::print);
         System.out.println();
}
public static void main(String[] args) {
   LinkedList <String> lin = new LinkedList();
   System.out.print("liste en A:  ");
   afficher(lin); // car statique dans la classe courante
   lin.add("a");lin.add("b");// ajout en fin de liste
   System.out.print("liste en B:  ");
   afficher(lin);
   ListIterator <String> it = lin.listIterator();
   it.next(); // on se place sur le premier élément
   it.add("c"); it.add("b"); // on ajoute deux éléments
   System.out.print("liste en C:  ");
   afficher(lin);
   it = lin.listIterator();
   it.next(); // on progresse d'un élément
   it.add("b"); it.add("d"); // on ajoute deux éléments
   System.out.print("liste en D:  ");
   afficher(lin);
   it = lin.listIterator(lin.size()); // on se place en fin de
liste
   while (it.hasPrevious()) { // on recherche le dernier
b
     String ch = it.previous();
     if (ch.equals("b")) it.remove();// on le supprime
     break;    }
       System.out.print("liste en E:  ");
       afficher(lin);
       it = lin.listIterator();
   it.next(); it.next(); // on se place sur le deuxième
élément
   it.set("x"); // on le remplace par "x"
   System.out.print("liste en F:  "); afficher(lin);
      }
}

liste en A:
liste en B:  a  b  ab
liste en C:  a  c  b  b  acbb
liste en D:  a  b  d  c  b  b  abdcbb
liste en E:  a  b  d  c  b  abdcb
liste en F:  a  x  d  c  b  axdcb
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.ListIterator;

public class VecteurDynamique {

public static void main(String[] args) {
Collection <String> col =  new ArrayList<String>() ;
// ajout des elements a cette collection
col.add("un") ; col.add("deux") ; col.add("trois") ;
// test d'appartenance de "deux"
boolean b1 = col.contains("deux") ;
System.out.println(b1) ;  // affiche true
 // test d'appartenance de "DEUX"
 boolean b2 = col.contains("DEUX") ;
System.out.println(b2) ;  // affiche false
// parcourir les elements de la collection avec  un iterateur
Iterator<String> it = col.iterator() ;
 while (it.hasNext()) {
String element = it.next() ;  // retourne un objet de type String
System.out.println(element) ; }
// balayer les elements de la collection avec un for each
for (String element : col) {
System.out.println(element) ; }
// balayer les éléments de la collection avec forEach
//forEach (consumer<? super String>)void
col.forEach(System.out::println);
// conversion d'une collection en tableau
String [] tab = col.toArray(new String[] {}) ;
// affichage du contenu du tableau
for (String element : tab) {
System.out.println(element) ;}
// exemple avec Liste
List <String> li =  new ArrayList <String>() ;
 // ajout d'éléments à cette liste
li.add("un"); li.add("deux") ; li.add("trois") ;
 // ajout d'un element à un index
li.add(1,  "avant deux") ;
 // positionnement d'un element donné
li.set(3,  "TROIS") ;
// creation d'un listIterator sur cette liste
ListIterator <String> it1 = li.listIterator() ;
 while(it1.hasNext()) {
    // on ajoute un élément supplémentaire après chaque element
de la liste
   String elt = it1.next() ;   it1.add(elt +  " et demi") ;}

// vérification du résultat
 for (String s : li) {   System.out.println(s) ;}
 // balayage avec forEach
 li.forEach(System.out::print);
    }
}
```

```
true
false
un
deux
```

```
trois
un
deux
trois
un
deux
trois
un
deux
trois
un
un et demi
avant deux
avant deux et demi
deux
deux et demi
TROIS
TROIS et demi
unun et demiavant deuxavant deux et demideuxdeux et demiTROISTROIS et demi
```

```
import java.util.Collection;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;


public class TablesAssociative {

   public static void main(String[] args) {
   Map <String,String> m = new HashMap();
   m.put("c", "10"); m.put("f", "20");m.put("k",
"30");
   m.put("x", "40");m.put("p", "50");m.put("g",
"60");
   System.out.println("Map intial: "+m);
   // retrouver la valeur associée à la clé "f"
   String ch = m.get("f");
   System.out.println("valeur associée à f est:
"+ch);
   // ensemble des valeurs (Collection et non Set)
   Collection <String> valeurs = m.values();
   System.out.println("liste des valeurs initiales:
"+ valeurs);
   valeurs.remove("30"); // on supprime la valeur
30 par la vue associée
   System.out.println("liste des valeurs après
suppression: "+ valeurs);
   // ensemble des clés (ici Set)
   Set <String> cles = m.keySet();
   System.out.println("liste des clés initiales:
"+cles);
   cles.remove("p"); // on supprime la clé p par la
vue associée
   System.out.println("liste des clés après
suppression: "+cles);
```

```
    System.out.println("Map après les deux
suppressions: "+m);
    // modification de la valeur associée à la clé "x"
    String old = m.put("x","25");
    if (old!=null) System.out.println("valeur
ancienne pour la clé x = " + old);
    System.out.println("Map après modification:
"+m);
    System.out.println("liste des valeurs après
modification: "+ valeurs);
    // On parcout les entrées (Map, Entry) du map
jusqu'à trouver la valeur 20
    // on supprime l'élement correspondant (en
supposant qu'il existe)
    Set <Map.Entry<String,String>> entrees =
m.entrySet();
    Iterator <Map.Entry<String,String>> iter =
entrees.iterator();
    Map.Entry<String,String> entree;
    String valeur;
    while (iter.hasNext()){
       entree = iter.next();
       valeur = entree.getValue();
       if (valeur.equals("20")) {
          System.out.println("valeur 20 trouvée en
clé: " + entree.getKey());
          iter.remove(); // suppression de la vue
associéé
          break;      }    }
 System.out.println("Map après suppression
élément suivant 20: "+m);
 // on supprime l'élement de clé f
 m.remove("f");
System.out.println("liste des clés après
suppression de f: "+cles);
System.out.println("liste des valeurs après
suppression de f: "+valeurs);      }
}
```