UFR Sciences et Technologies
Master 1 E3A

EC923 : Génie Informatique Programmation Objet et IHM

Malik Mallem - Bâtiment Pelvoux

UFR S&T, ,40 Rue du Pelvoux F-91020 Evry Cedex Tél :+33-1-69 47 75 15
Fax :+33-1-69 47 75 99

Enseignement : II74 GEII & GSI & RVSI- POO & IHM - Programmation orienté objet (JAVA)
Objectif : Ce module se situe dans la continuité des modules GI62 et GI63 et permet de perfectionner les connaissances de la modélisation et de la programmation orientées objet. Les notions de base de création d'interface homme-machine graphique seront également abordés.
Le langage JAVA est pris comme exemple.
Avertissement : Ce cours suppose la connaissance et la maîtrise du langage C++

CONTENU DU COURS

HISTORIQUE

1990-1992, projet Oak

- Joy, Gosling (Sun)
- Langage de programmation pour systèmes embarqués:TV interactive,PDA,…
- But: avoir une plateforme commune.
- Projet abandonné en 1992

1996, Java(Sun Microsystems.Joy et Gosling)

- Projet Oak appliqué à WWW.
- Java, combiné avec un browser spécial Hotjava, apporte le concept d'applet.
- Avec le développement du web, Java est né.

LIENS WEB

- http://java.sun.com/docs/books/jls/
- http://java.sun.com/reference/docs/index.html

HISTORIQUE (Suite)

Java combine les caractéristiques des langages :
- Smalltalk(Xerox: Palo Alto (Californie) :Langage d'objets communiquant par messages) (1980),
- ADA (Défense américaine : Langage mutitaches) (1979),
- C++ (Bell Lab : Langage pseudo objet) (1992).

- En Smalltalk :

  1 programme = objets +messages

- En JAVA :

  1 programme = objets + interfaces

1.INRODUCTION

1.1 Objectif de Java

public class Pgme{
  public static void main(...)...}

Compilation

javac Pgme.java

Code intermédiaire (Bvte code)

« Pgme.class »
local ou distant

Interprétation

Sun

java Pgme.class

ou par l'intermédiare d'un navigateur web

HP

PC

1.2 Machine Virtuelle Java

- Rôle :

Interprète le byte code en langage machine de la cible: - interprétation

- supervise l'exécution des classes,
- gère les classes et les méthodes communes (constr. Init, classe Object, …),
- gère la pile d'exécution et la mémoire(lance le ramasse miettes – garant d'une meilleure stabilité des applications),
- interprète le code intermédiaire.

- Constitution :
  - Types de données
  - Les registres
  - Jeu d'instructions

1.3 Caractéristiques de Java

- **« Simple »**
  syntaxe du langage "C"
- **« sûr »**
  pas de pointeurs, vérification de l'intégrité du code à l 'exécution et des accès réseau et/ou fichiers,cryptage des données véhiculées sur le réseau
- **Orienté Objet**
  Complètement orienté objet(fichier,type primitif, chaine de caractères, exception, thread,…) , pas de variables ni de fonctions globales.
- **Robuste**
  ramasse miettes, fortement typé, gestion des exceptions,…
- **Indépendant d 'une architecture**
  Portabilité assurée par la présence d 'un interpréteur de bytecode sur chaque machine cible.
- **Environnement riche**
  Classes pour l 'accès Internet
  classes standard complètes
  fonctions graphiques évoluées
- **Applications interactives et réseaux**

(alphanumérique ou graphique)

« Stand alone » : executable en local
« applet » : executable en local /réseau dans un navigateur Web

Programme Minimal

- **Exemple 1.1 : pgme minimal ss java faisant appel à une méthode de sortie(printl)de l'objet (out) de la classe System :**

```
public class PgmeMinimal                                    // (1)
{ public static void main (String[] args)                   // (2)
  { System.out.println ("Universite d'Evry, UFR Sciences et Technologies, cours de java") ; // (3)
  }}
```

(1) : PgmeMinimal : classe théorique limitée à la définition de la méthode main ; public utilisé par la MV(Machine Virtuelle) de java pour accéder à la classe.
(2) : static : main n'est pas associée à un objet (instance de la classe)
String[] args : tableau d'objets de type String équivalent à argv du C/C++
public utilisé par la MV(Machine Virtuelle) de java pour accéder à la méthode main.
(3) : System : la classe
out : champ représentant la fenêtre d'affichage,
println : méthode d'affichage avec saut de ligne non attaché à un objet.

- **Mise en œuvre**

Compilation :

[mallem@gsc18]$ /usr/java/j2sdk1.4.1_03/bin/javac PgmeMinimal.java

Le Programme doit avoir le même nom que la classe théorique

La compilation produit un code intermédiaire("bytecode"), nommé PgmeMinimal.class

Execution :

[mallem@gsc18]$ /usr/java/j2sdk1.4.1_03/bin/java PgmeMinimal

Universite d'Evry, UFR Sciences et Technologies, cours de java

Lancement de la MV de java (Interprétation/execution)

Remarque :

Une application Java peut être exécutée sur une plate-forme quelconque à condition que celle-ci dispose d'une *Java Virtual Machine*(JVM) présente dans :

- un environnement de développement, par exemple le *Java Development Kit* (JDK),
- un environnement d'exécution comme le *Java Runtime Environment* (JRE) - (voir les produits de Sun),
- un navigateur Web capable **java** d'exécuter des « applet » Java.

- **Exemple 1.2 : Calcul trigonometrique - Utilisation des classes Clavier(Utilisateur) et System (java)**

```
// La classe Calcul utilise la classe Clavier
public class Calcul
{ public static void main (String[] args)
  { int i ;
    double angle,angledeg;
    double sinus,cosinus ;
      { System.out.print ("Donnez un angle en degres : ") ;
        angledeg  = Clavier.lireDouble ()  ; angle=angledeg*3.14/180.;
        System.out.print ("Cosinus (0) ou Sinus (1)? ") ;
        choix  = Clavier.lireInt ()  ;
        if (choix==0)
          { cosinus = Math.cos(angle) ;  System.out.println (angledeg + " a pour cosinus : "  + cosinus) ;}
        else {
        sinus = Math.sin(angle) ; System.out.println (angledeg + " a pour sinus : "  + sinus) ;}
     }}}
/* résultat d'exécution
[mallem@gsc18 02]$ /usr/java/j2sdk1.4.1_03/bin/java Calcul
Donnez un angle en degrés : 90
Cosinus(0) et Sinus(1)? 0
90.0 a pour cosinus : 0
Donnez un angle en degrés : 0
Cosinus(0) et Sinus(1)? 1
0 a pour sinus : 0*/
```

2.SPECIFICITES DE JAVA

2.1 Introduction

- pas de structures et d'unions,
- pas de define
- pas de pointeurs
- pas d'héritage multiple bien que Java permet à une classe d'implémenter plus d'une interface.
- pas de surcharge d'opérateurs
- Les opérateurs du langage Java sont quasi identiques à ceux du langage C.

Java peut être considéré comme un C++ allégé de sa complexité formelle.

2.2 Mots clés

abstract	boolean	break	**byte**	case
catch	char	class	const	continue
default	do	double	else	**extends**
final	**finally**	float	for	goto
if	**implements**	**import**	**instanceof**	int
interface	long	**native**	new	**package**
private	protected	public	return	short
static	**super**	switch	**synchronised**	this
throw	**throws**	**transient**	**try**	void
volatile	while			

En soulignés les nouveautés par rapport au C++

2.3 Types de base

- **entier**
  signés seulement
  type **byte** (8 bits), **short** (16 bits), **int** (32 bits), **long** (64 bits)
- **flottant**
  standard IEEE 754
  type **float**(32 bits), **double** (64bits)
- **booléen**
  type **boolean** (true,false)
- **caractère**
  unicode,
  type **char** (16 bits)

Conversion implicite :

Byte → short → int → long → float → double
Char → short → int → long → float → double

2.4 Opérateurs

Priorité	Opérateur(s)						Associativité
Max	()	[ ]	.	**++(post)**	**--(post)**		→
	+ - ~(unaires)		!	cast new	**++(pre)**	**--(pre)**	←
	*	/	%				→
	+	-					→
	>>	<< (arith)	**>>>**	**<<<** (logique)			→
	>	>=	<	<=	**instanceof**		→
	==	!=					→
	&						→
	^						→
	|						→
	&&						→
	||						→
	?:						→
Min	=	*op*=					←

En gras les nouveautés par rapport au C++

A noter l'absence des opérateurs → sizeof : par rapport au C/C++

2.5 Tableaux

- **Déclaration**

```
int[] tableau_entiers;
double tableau_reels[];
long[][] tableau_2D;  ou long[]tableau_2D [] ;
```

- **Création**

```
tableau_entiers = new int[12];
tableau_reels = new double[23];
tableau_2D = new long[5][4]; // peut avoir des lignes de taille différentes
tableau_entiers.length() ; //donne la taille du tableau
```

- **Utilisation**

```
public static int [] suite(int n)    // Méthode permettant de créer et
{ int [] tab = new int [n] ;          //initialiser  un tableau d'entiers et le
for (int i=0; i<n; i++)               // retourner
tab[i] = 2 * i;
return tab ; }
```

Exemple d'un tableau à 2 dimensions :

```
int[][] t = new int[4][];
for (int i=0; i<t.length(); i++) { //instantiation des lignes
t[i] = new int[4-i];}
for (int i=0; i<t.length(); i++) {  //initialisation des colonnes
for (int j=0; j<t[i].length(); j++) {
t[i][j] = i+j;}}
```

t[0]	0	1	2	3
t[1]	1	2	3	
t[2]	2	3		
t[3]	3			
	[0]	[1]	[2]	[3]

- **Remarques**
- Un tableau multidimensionnel est un "tableau de tableaux".
- La notion de tableau est généralisée.

La dimension d'un tableau ne peut figurer dans la déclaration(int t[4]→ erreur)

3.CLASSES ET OBJETS

3.1 Définition d'une classe :

class Point

{

// définition de Point

...

}

- **Remarques**
- Le code de la classe Point doit être écrit dans un fichier Point.java.
- La déclaration et la définition d'une classe sont combinées.
- Une classe défini un **type**.

3.2 Instantiation

```
Point p1; // déclaration
p1 = new Point(); // instantiation
Point p2 = new Point(); // déclaration avec initialisation
Point p3 = null;
```

- **Remarques**
- Une variable est déclarée avec un type.
- Une variable contient une **référence** sur un objet.
- Un objet est créé en instantiant une classe.
- new est un opérateur d'instantiation.
- null est une **valeur** qui dénote l'absence d'objet.
- null n'a pas de type.
- Contrairement à Smalltalk, null n'est pas un objet.

// **Exemple 3.2 :** l'instanciation d'un objet ne permet pas de le créer, il faut avoir recours a une allocation dynamique (new).

```
class Obj
{ public Obj() //constructeur
 { System.out.print ("++ creation objet Obj
; ") ;
    nb ++ ;
    System.out.println ("il y en a " + nb) ; }
  public static long nbObj ()
  { return nb ;}
  private static long nb=0 ;
}
public class TstObj2
{ public static void main (String args[])
  { Obj a ; //declaration
    System.out.println ("(1)  : nbre  objets =
" + Obj.nbObj() ) ;
    a = new Obj() ; //instantiation
    System.out.println ("(2)  : nbre objets =
" + Obj.nbObj() ) ;
    Obj b ;
    System.out.println ("(3)  : nbre objets =
" + Obj.nbObj() ) ;
    b = new Obj() ;
    Obj c = new Obj() ;
    System.out.println ("(4)  : nbre objets =
" + Obj.nbObj() ) ;
  }
}
```

```
/*  Résultats d'execution
(1)  : nbre objets = 0
++ creation objet Obj ; il y en a 1
(2)  : nbre objets = 1
(3)  : nbre objets = 1
++ creation objet Obj ; il y en a 2
++ creation objet Obj ; il y en a 3
(4)  : nbre objets = 3
```

3.3 Notion de constructeur

Si une classe ne dispose pas de constructeur, Java fait appel à un constructeur par défaut.

Remarques :

. pas de valeur de retour (comme en C++),

. possibilité de surdéfinir un constructeur,

. appel direct d'un constructeur depuis une autre méthode (super),

3.4 Exemples

// Exemple 3.4a : appel du constructeur par défaut

```
class Point
{
 public void Affiche()
  {System.out.println ("constructeur par defaut : " + x + " " + y) ;}
  private int x, y ;
}
public class _ConstDefaut
{ public static void main (String args[])
    {
    Point a = new Point() ;
    a.Affiche();
  }
}
/* Resultats
[mallem@gsc18 objets]$ /usr/java/j2sdk1.4.1_03/bin/java _ConstDefaut
constructeur par defaut : 0 0
[mallem@gsc18 objets]$*/
```

// Exemple 3.4b surdefinition de constructeurs, un constructeur ss arguments appelant un const. a 2 arguments

```
class Point
{ public Point(int abs, int ord) //constructeur a 2 parametres
  { x = abs ;y = ord ;
    System.out.println ("constructeur deux arguments : " + x + " " + y) ;}
  public Point() //constructeur sans parametres
  { this (0,0) ;  // appel Point (0,0) ; doit etre la premiere instruction
    System.out.println ("constructeur sans argument") ;}
  private int x, y ; //donnees membres
}
public class _Consthis
{ public static void main (String args[])
 { Point a = new Point (1, 2) ;  Point b = new Point() ;}
}
/* Resultats
[mallem@gsc18 objets]$ /usr/java/j2sdk1.4.1_03/bin/java _Consthis
constructeur deux arguments : 1 2
constructeur deux arguments : 0 0
constructeur sans argument
//Java permet de réaliser des appels entre constructeurs à l'intérieur d'une classe
```

// Exemple 3.4c : surdéfinition de constructeurs (suite)

```
class Point
{ public Point(int abs, int ord)
  { x = abs ;  y = ord ;
    System.out.println ("constructeur deux arguments : " + x + " " + y) ;}
 public void Affiche()
  {
    System.out.println ("coordonnees du point : " + x + " " + y) ;}
  private int x, y ;
}public class _ConstDefaut2
{ public static void main (String args[])
    { Point a = new Point (1, 2) ;      Point b = new Point() ;    b.Affiche();}
}/* Resultats
[mallem@gsc18 objets]$ /usr/java/j2sdk1.4.1_03/bin/javac _ConstDefaut2.java
_ConstDefaut2.java:21: cannot resolve symbol
symbol  : constructor Point ()
location: class Point
    Point b = new Point() ;
                  ^
1 error // le constructeur par défaut n'est pas appelé dès l'instant ou au moins un
constructeur est défini
```

//**Exemple 3.4d :** Surdefinition de constructeurs (suite)

```
class Point
{ public Point ()                    // constructeur 1 (sans argument)
  { x = 0 ;  y = 0 ;
  }
  public Point (int abs)             // constructeur 2 (un argument)
  { x = y = abs ;
  }
  public Point (int abs, int ord ) // constructeur 3 (deux arguments)
  { x = abs ; y = ord ;
  }
  public void affiche ()
  { System.out.println ("Coordonnees : " + x + " " + y) ;
  }
  private int x, y ;
}
```

```
public class _Surdef2
{ public static void main (String args[])
  { Point a = new Point () ;      // appelle constructeur 1
    a.affiche() ;
    Point b = new Point (5) ;     // appelle constructeur 2
    b.affiche() ;
    Point c = new Point (3, 9) ;  // appelle constructeur 3
    c.affiche() ;
  }
}

/*
[mallem@gsc18 objets]$ /usr/java/j2sdk1.4.1_03/bin/java _Surdef2
Coordonnees : 0 0
Coordonnees : 5 5
Coordonnees : 3 9
```

// Exemple 3.4e : appel du constructeur par recopie

```
class Point
{public Point(int abs, int ord)
  { x = abs ;   y = ord ;
    System.out.println ("constructeur deux arguments : " + x + " " + y) ;}
public Point(Point p) //constructeur par recopie
  { x = p.x ;    y = p.y ;
   System.out.println ("constructeur par recopie : " + x + " " + y) ;}
 public void Affiche()
  { System.out.println ("coordonnees du point : " + x + " " + y) ; }
  private int x, y ;
}
public class _ConstRecopie
{ public static void main (String args[])
    {     Point a = new Point (1, 2) ; Point b = new Point(a) ; b.Affiche(); }
}/* Resultats
[mallem@gsc18 objets]$ /usr/java/j2sdk1.4.1_03/bin/java _ConstRecopie
constructeur deux arguments : 1 2
constructeur par recopie : 1 2
coordonnees du point : 1 2
*/
```

3.5 Champs d'un objet

- Initialisation des types primitifs :

Boolean : false,
Char, short, int, long, float, double : null ou 0

Tout champ doit être initialisé !
Un champ donnée peut être un objet d'une classe différente

- **Champ statique :**

Plusieurs objets d'une même classe peuvent se partager un champ statique

Exemple [cf. exemple 3.2]

```
class B
{static int n ;
float y ;
}
```

B b1 = new B(), b2 = new B() ;
n est un champ de classe, car il commun à la classe et à tout objet de la classe :
Les écritures suivantes sont équivalentes :
b1.n , b2.n et B.n

3 .6 Méthode de classe

C'est une méthode, dont la définition est précédée de static, et qui existe en un seul exemplaire. Cette méthode ne peut agir que sur des champs de classe.

Cette notion est appliquée aux méthodes des librairies standard telles que Math (exemple : Math.sqrt()).

Surdéfinition de méthodes

Il s'agit du même mécanisme qu'en C++.

Exemple : constructeur par recopie

```
public Point (Point p)
{x=p.x ; y=p.y ;}
Point a = new Point (1,2) ; //constructeur usuel
Point b = new Point (a) ;    //constructeur par recopie
}
```

3.8 Passage de paramètres entre méthodes

Par valeur. Les objets étant connus par leur référence (adresse). La méthode appelée travaille directement sur le même objet que la méthode appelante. Java ne dispose pas de pointeurs (absence des opérateurs * et &,->). Dans ce cas, si une méthode appelée doit modifier un paramètre pour la méthode appelante, ce dernier doit obligatoirement être un objet.
Un constructeur peut faire appel à un autre en utilisant « this » qui doit être la 1ere instruction de ce constructeur.

Exemple 1 : Passage d'un objet en paramètre sans modification du paramètre
a.coincide(b) ; a et b objets de la classe Point.

Exemple 2 : Passage d'un objet en paramètre avec modification du paramètre
a.permute(b) ; a et b objets de la classe point
a= a.permute(b) ;

3.9 Classe interne

C'est une classe définie à l'intérieur d'une autre classe. Dans ce cas, la création d'un objet de la classe externe implique celle de l'objet de la classe interne. Les objets externe et interne accèdent alors à leur champs respectifs même privés. L'inconvénient est la limitation de la ré utilisation des classes.

- **Exemple : Une classe(Segment) encapsulant une autre(Point)**

```
class Segment  // {   //moyen de contourner l'héritage entre classes
class Point
  { public void affiche()
    { System.out.println ("point de coordonnees : " + x + " " + y) ;}
    public Point (int x, int y){ this.x = x ; this.y = y ;}
    private int x, y ;}  //Point
  public Segment (int xd, int yd, int xf, int yf, byte c)
  { orig =   new Point (xd, yd) ; extrem = new Point (xf, yf) ; this.c = c ;}
  public void affiche()
  { System.out.println ("segment de coordonnees : " + this.orig.x + " " +  this.orig.y + " " +
this.extrem.x +  " " + this.extrem.y + " " + this.c) ;
  private Point orig, extrem ; private byte c ;} // Segment
public class _TstSeg
{ public static void main (String args[])
{ Segment s = new Segment (1, 3, 5, 9, (byte)2) ; s.affiche() ;} } //_TstSeg
[mallem@gsc18 ]$ /usr/java/j2sdk1.4.1_03/bin/java _TstSeg  segment de coordonnees : 1 3 5 9 2
```

3.10 Classe enveloppe

Correspond à une classe associé à un type primitif(boolean, char, short, int, long, float, double).

Toute classe enveloppe(Boolean,Character,Short,Integer,Long,Float,Double) dispose d'un constructeur ayant un type primitif en argument et de méthodes,dont :

- xxxValue (valeur de la variable),
- toString() (conversion valeur-ascii),
- parsexxx() (conversion ascii – valeur)

(cf. paquetage : java.lang)

Exemple :

```
int n=12 ;
Integer nobj = new Integer(n) ;

int m = nobj.intValue() ;
```

3.11 Classe String

Classe standard permettant d'instancier des objets non modifiable.
Exemple :

- Création

```
String s = new String() ;  // chaîne vide – appel du constr. Sans argument
String s1 = new String(« chaîne des caracteres » ) ;  //  appel du constr. a 1  argument
String s2 = new String(s1) ;  // reference sur s2 copie de s1–appel du constr.par recopie
String s3=s1+s2 ;                    //concaténation
```

- Opérations

```
s=clavier.lireString() ;       //lecture
System.out.Println(s) ;       //affichage
int n=s.lenght ();                 //longueur de la chaîne s
s.charAt(i) ;                        //accès au caractère i
n=s1.indexOf('d') ;    //recherche de l'index de 'd' dans s1
s2=s1.replace('d','c') ;      // remplace 'd' par 'c'
s1.compareTo(s2) ;    //comparaison de s1 et s2
```

Tableau de chaîne de caractères

String args[] //paramètre de la méthode de classe « main »
int argc=args.length ; //permet d'obtenir le nombre de paramètres de la ligne de commande
for(int i=0 ; i<argc ; i++)System.out.Println(args[i]) ; //affiche les paramètres de la ligne de commande

(Illustrer par le programme args.java)

La classe StringBuffer permet de manipuler des chaînes de caractères modifiables.

3.12 Classe abstraite

C'est une classe qui ne permet pas d'instancier des objets. Elle sert de classe de base pour une dérivation. Elle est destinée à contenir la déclaration des fonctionnalités de la descendance.

Ex : Abstract class A

A a = new A(…) ; cause une erreur à la compilation

Il existe également la notion de méthode abstraite qui est déclarée (prototype) mais non définie.

public abstract void f(…)

Règles :

. Une classe est abstraite si elle contient au moins une méthode abstraite,

. Une classe abstraite est forcément publique, car destinée à participer à une dérivation,

. Une classe dérivée d'une classe abstraite n'est pas obligée de re définir toutes les méthodes de la classe abstraite. Si elle ne définit pas toutes les méthodes de la classe abstraite, elle devient abstraite.

4. INTERFACES

4.1 Définition

Comme les classes abstraites, les interfaces permettent de déclarer des champs (données et méthodes) sans les définir. Leur définition sera faite dans les classes dérivées. La notion de classe abstraite est + générale que celle d'interface car elle permet de définir des méthodes.

Exemple :

```
public interface I                          class A implements I
{                                           {
      void f(int n) ;                           //A doit définir les méthodes de I
      void g() ;                            }
}
```

Une classe peut implémenter plusieurs interfaces, il faudrait alors définir toutes les méthodes de ces interfaces.

class A implements I1,I2,…,In (→ héritage multiple)

4.2 Remarques sur les interfaces

- Une interface ne possède pas d'implémentation.
- Une interface ne peut pas être instantiée, mais défini un **type** (de la même manière qu'une classe abstraite).
- Une interface peut aussi définir des constantes (`static` et `final`).

```
interface Rotateable
{
static final double PI = 3.1415926535;
public void rotate();
}
```

- Une interface peut étendre une ou plusieurs autres interfaces.

```
interface Transformable extends Scalable, Rotateable {}
```

- Une interface hérite de toutes les méthodes et constantes de sa super-interface.
- Une interface dénote ce qu'un objet peut faire. Par **convention**, le nom d'une interface se termine généralement par `-able`.

```
//4.2 Interfaces
interface Affichable
{ void affiche() ;}
class Entier implements Affichable
{ public Entier (int n)
  { valeur = n ;}
  public void affiche()
  { System.out.println ("entier de valeur "
+ valeur) ;}
  private int valeur ;
}
class Flottant implements Affichable
{ public Flottant (float x)
  { valeur = x ; }
  public void affiche()
  { System.out.println ("flottant de valeur "
+ valeur) ;}
  private float valeur ;
}
public class _Interface
{ public static void main (String[] args)
  { Affichable [] tab ;
    tab = new Affichable [3] ;
    tab [0] = new Entier(12) ;
    tab [1] = new Flottant (1.23f) ; ;
    tab [2] = new Entier (34) ;
    int i ;
    for (i=0 ; i<3 ; i++)
      tab[i].affiche() ;
  }
}
/*
[mallem@gsc18 objets]$
/usr/java/j2sdk1.4.1_03/bin/java _Interface
entier de valeur 12
flottant de valeur 1.23
entier de valeur 34
[mallem@gsc18 objets]$
*/
```

5. PAQUETAGE

5.1 Définition

C'est un regroupement logique, sous un identificateur, d'un ensemble de classes qui se partageront des données et méthodes.

5.2 Règles :

- Toutes les classes contenues dans un même fichier source appartiennent au même paquetage par défaut.
- Le nom du paquetage correspond à celui du répertoire où il se trouve.
- Un paquetage est unique pour une implémentation donnée.
- L'identification d'un paquetage se fait avec le mot clé « package »(Ex. package user)

La désignation d'un paquetage se fait de plusieurs façons :

- Nom_paquetage.Nom_classe //dans une instruction
- import Nom_paquetage.* //inclusion
- import Nom_paquetage. Nom_classe //inclusion

Paquetages standards :

java.lang (contient les classes System, Math, Float, Integer ,…)

java.awt ; java.swing (paquetages graphiques)

6. HERITAGE ET POLYMORPHISME

6.1 Introduction:

- Terminologie :
  - «mère», «ancêtre», «parente», «super-classe»
  - «héritière», «fille», «dérivée», «sous-classe»
- Héritage simple : (Java, Smalltalk)
  - Toutes les classes descendent d'une superclasse « Object »
  - Une et une seule classe mère
  - Java : Une classe hérite d'une autre mais peut implémenter plusieurs interfaces.
- Héritage multiple : (C++)
  - Une classe peut ne pas avoir d'ancêtre
  - Plusieurs classes mères
  - Difficultés pour l'héritage «répété» de membres :
    - Membres de même nom
    - Même classe héritée plusieurs fois

6.2 : Héritage simple en Java

Syntaxe :

class Derivee extends Base

L'objet de la classe Derivee peut accéder aux membres publiques de sa classe de base, mais pas aux membres privées (Comme en C++).

6.3 Droits d'accès aux classes et interfaces

Modificateur	Signification
Public	Accès pour tous
/	Accès depuis les classes du même paquetage

// 6.2.1. Heritage simple avec appel explicite du constr. de la classe de base

```
class Point
{ public Point (int x, int y)
  { this.x = x ; this.y = y ;}
  public void deplace (int dx, int dy)
  { x += dx ; y += dy ;}
  public void affiche ()
  { System.out.println ("Je suis en " + x + " " +
y) ;}
  protected int x, y ;
}
class Pointcol extends Point
{ public Pointcol (int x, int y, byte couleur)
  { super (x, y) ;// obligatoirement comme
première instruction (remplace : du C++)
    this.couleur = couleur ;}
  public void affiche ()
  { super.affiche() ;
    System.out.println ("  et ma couleur est : " +
couleur) ;}
  private byte couleur ;
}
public class _HerSimple
{ public static void main (String args[])
  { Point p = new Point (1, 2) ;
    p.affiche() ;  // appelle affiche de Point
    Pointcol pc = new Pointcol (3, 4, (byte)2) ;
    p = pc ;       // p de type Point, reference un
objet de type Pointcol
    p.affiche() ;  // on appelle affiche de Pointcol
    p = new Point (5, 6) ;  // p reference a nouveau
un objet de type Point
    p.affiche() ;  // on appelle affiche de Point
  }
}
/*[mallem@gsc18 objets]$[mallem@gsc18
objets]$ /usr/java/j2sdk1.4.1_03/bin/java
_HerSimple
Je suis en 1 2
Je suis en 3 4
  et ma couleur est : 2
Je suis en 5 6
[mallem@gsc18 objets]$
*/
```

//

6.3 Droits d'accès pour les membres et classes internes

Modificateur	Signification
Public	Accessible dès l'instant où sa classe l'est
	Accessible depuis les classes du même paquetage
protected	Accessible depuis les classes dérivées ou celles du même paquetage
private	Accès restreint à la classe où est faite la déclaration du membre ou de la classe interne

Rem. :

1)Il n'ya pas d'exception faite, comme en C++, pour les fonctions amies

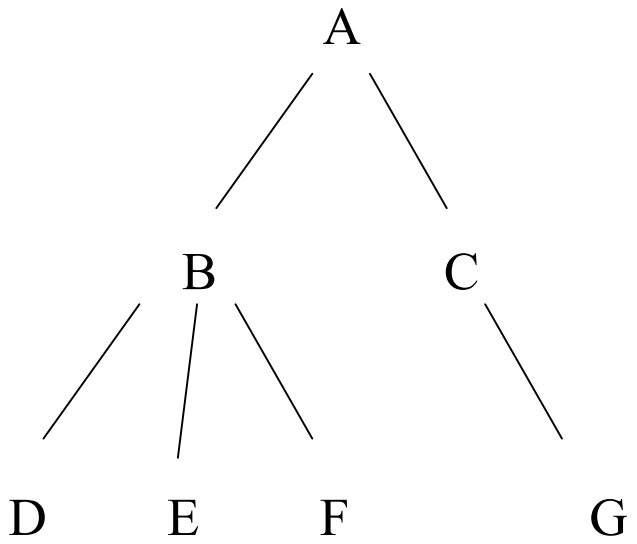
2)Il n'ya pas, comme en C++, de type d'héritage « private/protected ».

6.4 Dérivations successives

En java, toute classe dérivée dispose d'une seule classe mère.

Exemple :

class D extends B extends A
class E extends B extends A
class F extends B extends A
class G extends C extends A



6.5 Finalisation

- Mot clé « final » : permet de figer l'implémentation d'une classe, d'une méthode, ou d'une variable
- De classe
  - Il ne pourra pas y avoir d'héritières à cette classe
  - exemple : public final class AFinalClass { … }
- De méthode
  - Une telle méthode ne pourra pas être redéfinie dans les classes héritières (protection du code)
  - exemple : public final static void main (String args[])
- De variable
  - Une telle variable est une constante (on l'avait déjà vu !)
  - exemple : final float PI = 3.1415927

6.6 Polymorphisme

Basé sur les notions de redéfinition et de surdéfinition de méthodes

6.6.1 Redéfinition

Surcharge d'une méthode en apportant une extension à celle-ci. La redéfinition impose la même signature (paramètres) et la même valeur de retour à la fonction redéfinie que celles de la méthode surchargée.

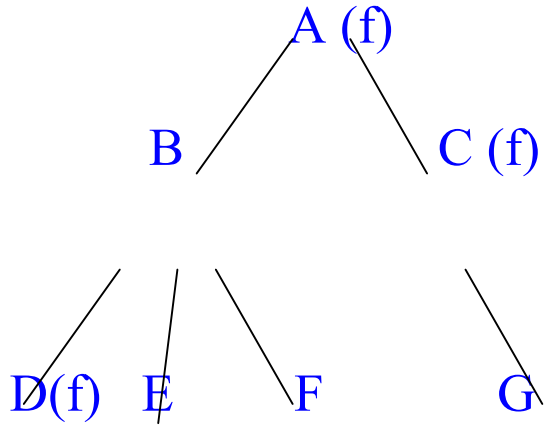
6.6.2 Surdéfinition

Surcharge d'une méthode en apportant une modification à celle-ci. La surdéfinition n'impose pas la même signature (paramètres) et la même valeur de retour à la fonction redéfinie que celles de la méthode surchargée.

Le polymorphisme permet de manipuler des objets sans en connaître parfaitement le type.

Exemple : f est redéfinie dans A, C et D.

dans la classe de l'objet l'ayant invoquée. Si échec, elle est recherchée dans les classes ascendantes.



//6.6.3a polymorphisme1 : f() redéfinie dans 2 classes (mère et fille)

```
class A
{ void f()
  { System.out.println ("appel f de A") ;
  }
}
class B extends A
{ void f()        //f redéfinie
  { System.out.println ("appel f de B") ;
  }
  public void test()
  { A a = new A();
    a.f() ; (1)
    super.f() ; (2)
    this.f() ; (3)
  }
}
public class _Super
{ public static void main (String args[])
  { B b = new B() ;
    b.test() ;
  }
}
/*
[mallem@gsc18 objets]$
/usr/java/j2sdk1.4.1_03/bin/java _Super
appel f de A  (1) //car a est un objet de A
appel f de A   (2)//appel de f() de A
appel f de B   (3)//appel f() de B
*/
```

//6.6.3b - polymorphisme :2 fonctions redéfinies dans 2 classes apparentées A,B et utilisée dans une classe fille C.

```
class A
{ void f()
  { System.out.println ("appel f de A") ;}
}
class B extends A
{ void f()   //f redéfnie
  { System.out.println ("appel f de B") ;
  }
  public void test()
  { A a = new A();
    a.f() ;  (1)
    super.f() ; (2)
    this.f() ; (3)
  }
}
class C extends B //partie rajoutée(rouge)
{public void test()  // test() redéfinie
  { C c = new C();
    c.f() ; } (4)
}
public class _Super2
{ public static void main (String args[])
  { B b = new B() ;
    b.test() ;
    C c = new C();
    c.test();   // appel test() de C
  }
}
/*[mallem@gsc18 objets]$
/usr/java/j2sdk1.4.1_03/bin/java _Super2
appel f de A  (1) //car a est un objet de A
appel f de A   (2)//appel de f() de A
appel f de B   (3)//appel f() de B
appel f de B  (4)//appel f() de B car C ne
contient pas de f()
```

//6.6.3c – polymorphisme 3 : f() et test() surdéfinies et utilisée dans B et C. A B et C sont apparentées

```
class A
{ void f(int i)
    { System.out.println ("appel f de A
avec un argument i = "+ i ) ; }
}
class B extends A
{ void f()
  { System.out.println ("appel f de B") ;}
  public void test()
  { A a = new A();
    a.f(1) ;        (1)
    super.f(2) ;    (2)
    this.f() ; }    (3)
}
class C extends B
{
  public void test(int i) //test() surdéfinie
  { C c = new C();
    c.f(i) ; }      (4)
}
public class _Super3
{ public static void main (String args[])
  { B b = new B() ;
    b.test() ;
    C c = new C();
    c.test(3); // appel test() de C
  }
}
/*
[mallem@gsc18 objets]$
/usr/java/j2sdk1.4.1_03/bin/java _Super3
appel f de A avec un argument i = 1    (1)
appel f de A avec un argument i = 2    (2)
appel f de B                           (3)
appel f de A avec un argument i = 3    (4)
```

//6.6.3d polymorphisme 4 : f() surdéfinie dans B et utilisée dans B et C. A B et C sont apparentées

// Un objet de la classe mere peut référencer un objet de la classe fille

```
class A
{ void f(int i)
    {
System.out.println ("appel f de A avec un
argument i = "+ i ) ;}
}
class B extends A
{ void f()
  { System.out.println ("appel f de B") ;}
  public void test()
  { A a = new A();
    a.f(1) ;          (1) (4)
    super.f(2) ;      (2) (5)
    this.f() ;        (3) (6)
  }}
class C extends B
{public void test(int i)
  { C c = new C();
    c.f(i) ;
  }
}
public class _Super4
{ public static void main (String args[])
  { B b = new B() ;
    b.test() ;
    B c = new C();
    c.test();}// appel test() de B car pas ds C
}
```

```
/*mallem@gsc18 objets]$
/usr/java/j2sdk1.4.1_03/bin/java _Super4
appel f de A avec un argument i = 1   (1)
appel f de A avec un argument i = 2   (2)
appel f de B                          (3)
appel f de A avec un argument i = 1   (4)
appel f de A avec un argument i = 2   (5)
appel f de B                          (6)
```

//6.6.3e polymorphisme 5 : f() surdefinie dans B et redefinie dans C. A B et C sont apparentees. Un objet de la classe grand mere ne peut pas referencer un objet de la classe fille

```
class A
{ void f(int i)
    {
System.out.println ("appel f de A avec
un argument i = "+ i ) ;}
}
class B extends A
{ void f()
  { System.out.println ("appel f de B") ;}
  public void test()
  { A a = new A();   a.f(1) ;
    super.f(2) ;
    this.f() ;}}
class C extends B
{ public void test(int i)
  { C c = new C();
    c.f(i) ;}
}
public class _Super5
{ public static void main (String args[])
  { B b = new B() ;
    b.test() ;
    A c = new C();
    c.test();}
}
```

```
/*[mallem@gsc18 objets]$
/usr/java/j2sdk1.4.1_03/bin/java _Super5
_Super5.java:32: cannot resolve symbol
symbol  : method test ()
location: class A
    c.test();
     ^
1 error
```

6.7 Super classe Object

Classe mère

• toutes les autres classes Java descendent d'`Object`
• implictement une classe descend d'`Object`
• **Méthodes de la Classe `Object`**
• Ces méthodes servent à décrire un objet et devraient être surchargées par les descendants de la classe:
`toString()` rend une version textuelle de l'objet appelé implicitement si nécessaire.
`equals()` compare avec un autre objet rend `true` si égal.

Exemples :

```
Object o1 = new Point(1,2) ;
Object o2 = new Point(1,2) ;
  o1.equal(o2) ;  // retourne false
```

o1.toString() ; //retourne une chaîne de caractères contenant le nom de la classe et l'@ de l'objet en hexadécimal.

7. EXCEPTION

7.1 Introduction

- **C'est une anomalie**

• Une exception représente une erreur qui survient à l'exécution. Elle peut avoir +sieurs sources :

- générée par le système (dépassement d'index de tableau), implicites et gérées par la MV. L'instant de leur occurrence n'est pas connu à l'avance.
- générée explicitement par le programmeur et contrôlé par ce dernier.

- **Considérée comme un objet**

• Une exception est représentée dans le système par un objet.
• Les exceptions sont des instances de la classe `Exception`.

- **Traitée par le programme utilisateur**

• Une exception peut être installée, interceptée et manipulée dans un programme.

7.2 Utilisation

- Création

Pour définir une nouvelle catégorie d'exception, il suffit de créer une sous-classe d'`Exception`. Une exception peut avoir ses propres variables d'instance et donc transporter une information supplémentaire.

- Soulèvement d'une exception :

- clause « throw » :
- les méthodes susceptibles de lever une exception doivent le signaler : clause « throws » suivant l'entête de la méthode.

• Capture d'une exception : bloc « try … catch »

try { *bloc d'instructions pouvant soulever une ou plusieurs exceptions* }
catch *type d'exception* { *bloc de traitement de l'exception* }
catch *autre type d'exception* {*bloc de traitement de l'exception* }

• Règles d'utilisation

- Une clause throw implique la présence d'une clause catch : une exception doit être traitée,
- La clause finally permet de spécifier un bloc de code qui doit être exécuté avant le programme de l'exception (rangement/sauvegarde,…).

7.3a Exemple : Exception utilisateur générée explicitement. La classe l'ayant générée ne la traite pas.

```
//lancement de l'exception Exception1  car p à
pour abscisse -1, le bloc catch de main est
executé car f n'en a pas.  La suite de f n'est
ensuite pas executée. Il ya retour dans main et
poursuite en séquence de main(). Si f doit être
poursuivie en séquence, il faut y prévoir un bloc
try - catch.
class Point
{ public Point(int x, int y) throws Exception1
  { if ( (x<0) || (y<0))throw new Exception1() ; (2)
    this.x = x ; this.y = y ;
  }
  public void f()  throws Exception1
    {Point p = new Point (-1, 0) ;                (1)
 System.out.println ("les nouvelles coordonnees
de p sont : "+ p.x + "  " + p.y) ;
 System.out.println ("apres exception ans f") ;
   }
  public int x, y ;
} // class Point
class Exception1 extends Exception
{ Exception1 ()
    { this.abs=0; this.ord=0;}
   public int abs, ord;
} //class Exception1
public class Parcour1
{ public static void main (String args[])
  { try
    { Point a = new Point (1, 2) ;
      a.f() ;
   System.out.println ("apres appel de  f  dans
main") ;
    }
    catch (Exception1 e) (3)
    { System.out.println ("dans catch
(Exception1) de main") ;
    }
 System.out.println("apres try dans  main") ; (4)
  }
}
/*
[mallem@gsc18 7exception]$
/usr/java/j2sdk1.4.1_03/bin/java Parcour1
dans catch (Exception1) de main    (3)
apres try dans  main                (4)
```

7.3b Exemple : lancement de l'exception Exception1 dans f car p à pour abscisse -1, le bloc catch de f est exécuté. La suite de f est ensuite exécutée. Il y a retour dans main et poursuite en séquence de main(). f traite l'exception.

```
class Point
{ public Point(int x, int y) throws Exception1
  { if ( (x<0) || (y<0))throw new Exception1();(2)
    this.x = x ; this.y = y ;}
  public void f()
  { try{ Point p = new Point (-1, 0) ; (1)
 System.out.println ("les nouvelles coordonnees
de p sont : "+ p.x + "  " + p.y) ; }
    catch  (Exception1 e) (3)
    { System.out.println ("dans catch
(Exception1) de f , le pb est corrige, les
coordonnees sont : "+ e.abs + "  " +e.ord) ;
    }
 System.out.println ("apres bloc try de f"); (4)}
  private int x, y ;
} //class Point
class Exception1 extends Exception
{ Exception1 ()
    { this.abs=0; this.ord=0;}
    public int abs, ord;
} //class Exception1
public class Parcour2
{ public static void main (String args[])
  { try{ Point a = new Point (1, 2) ;
      a.f() ;System.out.println ("apres appel de  f
dans  main") ; (5)
    }
    catch (Exception1 e)
    { System.out.println ("dans catch
(Exception1) de main , le pb est corrige, les
coordonnees sont : "+ e.abs + "  " +e.ord) ;}
System.out.println("apres bloc try demain");(6)
  }
}
```

```
[mallem@gsc18 7exception]$
/usr/java/j2sdk1.4.1_03/bin/java Parcour2
dans catch (Exception1) de f , le pb est corrige,
les coordonnees sont : 0  0       (3)
apres bloc try de f                (4)
apres appel de  f  dans  main      (5)
apres bloc try de main             (6)
```

7.3c Exemple : Récupération d'erreur

```
//lancement de l'exception Exception1 car b a
une abscisse < 0. L'exception remet a zero les
coord. de b, ce qui permet au programme de se
poursuivre. On suppose que si les coord sont <0,
le calcul ne peut se poursuivre.
class Point
{ public Point(int x, int y) throws Exception1
  { if ( (x<0) || (y<0))throw new Exception1();(2)
    this.x = x ; this.y = y ; }
public Point()
  {  this.x = 0 ; this.y = 0 ; }
  public int x, y ;
} //class Point
class Exception1 extends Exception
{ Exception1 ()
    { this.abs=0; this.ord=0;}
    public int abs, ord;
} //class Exception1
public class Parcour11
{ public static void main (String args[])
  {
     Point a = new Point () ;   // objet de calcul
   try
      { // Point b = new Point (1, 2) ;
Point b = new Point (-1, 2) ; //objet de saisie(1)
a=b;  // si cood >0 , initialiser objet de calcul
    }
    catch (Exception1 e) (3)
    { System.out.println ("dans catch
(Exception1) de main, les coordonnees sont : "+
e.abs + "  " +e.ord) ;
    a.x=e.abs; a.y=e.ord;      // si coord < 0
initialiser objet de calcul à 0
    }
    System.out.println ("apres try  dans  main les
coordonnees de a sont : "+ a.x + "  " +a.y) ; (4)
  }
}
/*[mallem@gsc18 7exception]$
/usr/java/j2sdk1.4.1_03/bin/java Parcour1
apres try  dans  main les coordonnees de a sont :
1  2    // pour b=(1,2)
//2° execution
dans catch (Exception1) de main, les
coordonnees sont : 0  0  //pour b=(-1,2)        (3)
apres try  dans  main les coordonnees de a sont :
0  0                                              (4)
```

7.3d Exemple : Relance d'exception

```
//lancement de l'exception Exception1 dans f car
p à pour abscisse -1, le bloc catch de f est
exécuté, ensuite il ya lancement de l'exception
Exception2. Comme il n'ya pas de bloc catch
correspondant dans f, celui de main est alors
exécuté. Ensuite  il y a sortie du bloc try.
class Point
{ public Point(int x, int y) throws Exception1
  { if ( (x<0) || (y<0))throw new Exception1();(2)
    this.x = x ; this.y = y ;}
  public void f() throws Exception2
  { try
    { Point p = new Point (-1, 0) ; }  (1)
    catch  (Exception1 e)
    { System.out.println ("dans catch
(Exception1) de f") ; (3)
throw new Exception2() ;        // on lance une
nouvelle exception}
 System.out.println ("apres bloc try de f") ; }
  private int x, y ;
}
class Exception1 extends Exception
{ }
class Exception2 extends Exception
{}
public class Relance
{ public static void main (String args[])
  { try
    { Point a = new Point (1, 2) ;
      a.f() ;
    }
    catch (Exception1 e)
    { System.out.println ("dans catch
(Exception1) de main") ;
    }
    catch (Exception2 e)
    { System.out.println ("dans catch
(Exception2) de main") ; (4)
    }
System.out.println("apres bloc try demain");(5)
 }}
```

```
$ /usr/java/j2sdk1.4.1_03/bin/java Relance
dans catch (Exception1) de f              (3)
dans catch (Exception2) de main           (4)
apres bloc try de main                    (5)
```

7.3eExemple : Inteception de la division/zéro

```
class DivideByZeroException
extends Exception
{public DivideByZeroException () {}
 public DivideByZeroException (String msg)
 {super(msg); }}
class Fraction {
 protected double numerator;
 protected double denominator;
 Fraction()
 {numerator = 0.0; denominator = 0.0; }
 Fraction(double numerator, double
denominator)
 {this.numerator = numerator;
  this.denominator = denominator;}
 double divide() throws DivideByZeroException
 { if (denominator == 0.0)
    throw new DivideByZeroException("It
happened, I can't believe it!"); (3)
  else
    return numerator / denominator; }}
class testDivision
{public static double tryToDivide(Fraction frac)
 {double result = Double.MAX_VALUE;
  try {result =  frac.divide(); }
  catch (DivideByZeroException dze)
  {System.out.println("Sorry: " +
dze.toString()); }   (2)
  return result;
 }
 public static void main (String args[])
 {Fraction frac1 = new Fraction();
  Fraction frac2 = new Fraction(1.0, 2.0);

  // essaye de calculer les fractions
  System.out.println("Dividing frac1"); (1)
  tryToDivide(frac1);
  System.out.println("Dividing frac2");  (4)
  tryToDivide(frac2);
}
```

```
[MALLEM@ens-unix ii34_linux]$
/usr/java/j2sdk1.4.2_02/bin/java testDivision
Dividing frac1          (1)
Sorry: DivideByZeroException: It happened, I
can't believe it!          (2) et (3)
Dividing frac2          (4)
```

Exceptions - classe Throwable

(Classe mère de Exception)

• Constructeurs

• public Throwable()

• public Throwable(String) // Constructeur

permettant de créer une nouvelle exception et d'y associer un message.

• Méthodes

• String getMessage()

Retourne le message associé à l'exception.

• String toString()

Donne une représentation textuelle de l'exception pouvant être affichée.

• void printStackTrace()

Affiche le contenu de la pile au moment où l'exception a été levée.

• Throwable fillInStackTrace()

Permet de remettre à jour la pile de l'exception. Surtout utile lorsqu'une exception est créée et jetée depuis des endroits différents.

Exceptions - clause finally

```
try {...
} catch (Exception e) {
...
} finally {
... }
```

• Remarques

• La clause finally permet de spécifier un bloc de code qui doit être exécuté dans tous les cas.

• Si une exception n'est pas interceptée ou alors est relevée dans un des blocs "catch", cette exception reste levée après l'exécution du bloc "finally".

• La clause finally permet notamment de fermer ou de libérer des ressources utilisées dans le bloc try (fermer un fichier, libérer une connexion,…)

```
// Méthode appelant « main »
public static void _system_(String [] args)
{try {
// initialise le système.
main(args);
} catch (Throwable e) {
e.printStackTrace();
} finally {// libère les ressources du système ;}}
```

8. THREADS

8.1 Introduction

- Un thread est un objet de la classe Thread. Il s'agit d'un pseudo processus géré par la Machine virtuelle.

- La classe Thread est prédéfinie (package java.lang)

- Catégories de thread : - démons (MV),
  - utilisateur

- Création d 'une nouvelle instance
  - Thread unThread = **new** Thread()
- Exécution du processus
  - unThread.start();
    - Cet appel engendre l'exécution de la méthode unThread.run(),

    (rend eligible *unThread* dont le corps est défini par la méthode *run())*
    - new Thread().start() : Création + passage à l'état éligible.

8.1a Exemple :

```
class ThreadExtends extends Thread {
   public void run(){
     while(true){
          System.out.println("dans ThreadExtends.run");
        }
   }
}
public class Thread1 {
  public static void main(String args[]) {
        ThreadExtends t0 = new ThreadExtends();
        t0.start();
        while(true){
          System.out.println("dans Thread1.main");
        }
  }
}
```

8.1b La classe Thread

- *import* java.lang.Thread *// par défaut pour java.lang*

Les constructeurs publics

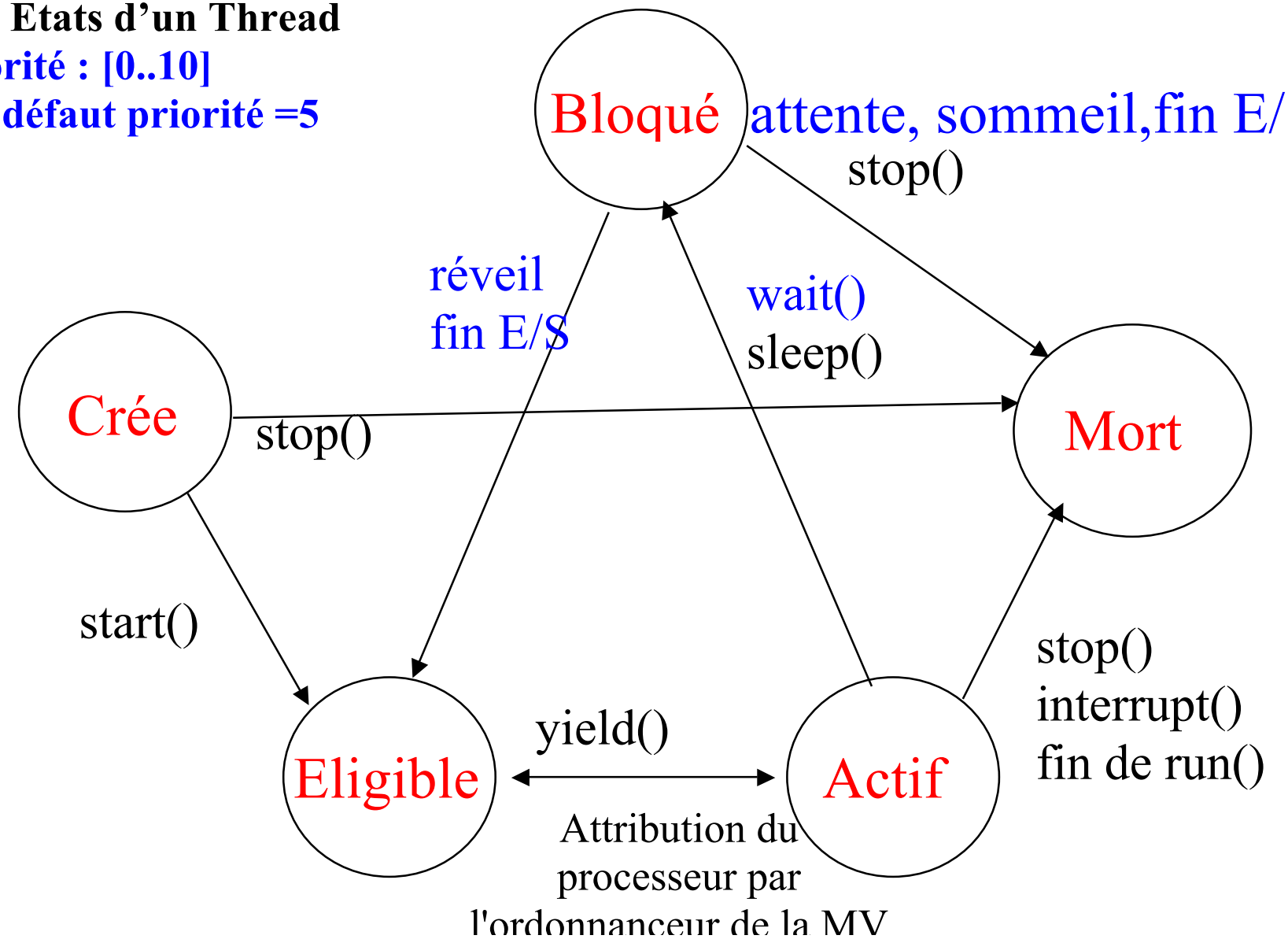
- *Thread();*
- *Thread(Runnable target);*
- ...

Les méthodes publiques

- *void start();*
- *void run();*
- *void stop();*
- *static void sleep(long ms);*
- *void interrupt() ;*
- ...

8.1c Etats d'un Thread

priorité : [0..10]
par défaut priorité =5



8.2a Exemple

```
//Exécution alternée de 3 threads
public class threads
{ public static void main (String args[])
  { Affiche t1 = new Affiche ("t1 ", 20) ;
    Affiche t2 = new Affiche ("t2 ", 10) ;
    Affiche t3 = new Affiche ("t3 ", 30) ;
    t1.start() ;  t2.start() ; t3.start() ;
    Clavier.lireString();
    t1.interrupt() ;     // interruption premier
thread
    System.out.println ("\n*** Arret premier
thread ***") ;
    Clavier.lireString();
    t2.interrupt() ;     // interruption deuxième
thread
    t3.interrupt() ;     // interruption troisième
thread
    System.out.println ("\n*** Arret deux
derniers threads ***") ;
  }}
class Affiche extends Thread
{ public Affiche (String texte, long attente)
  { this.texte = texte ;
    this.attente = attente ;
  }
  public void run ()
  { try {  while (true)
       { if (interrupted()) return ;
         System.out.print (texte) ;
         sleep (attente) ; }}
    catch (InterruptedException e) {}
  }
  private String texte ;
  private long attente ;
}
```

```
$ /usr/java/j2sdk1.4.1_03/bin/java threads
t1 t2 t3 t2 t1 t2 t3 t1 t2 t2 t1 t3 t2 t1 t2 t3 t2 t1 t2
t3 t1 t2 t2 t1 t3 t2 t1 t2 t3 t2 t1 t2 t3 t1 t2 t2 t1 t3
t2 t2 t1 t3 t2 t1 t2 t3 t2 t1 t2

*** Arret premier thread ***
t3 t2 t2 t3 t2 t2 t3 t2 t2 t3 t2 t2 t3 t2 t2 t3 t2 t2 t3
t2 t2 t3 t2 t2 t3 t2 t2 t3 t2 t2 t3 t2 t2 t3 t2 t2 t3 t2
*** Arret deux derniers threads ***
```

8.2b Exemple // Synchronisation de 2 threads permettant l'affichage d'un nombre et de son carré.

```
public class _Sync2threads
{ public static void main (String args[])
  { Nombres nb = new Nombres() ;
 System.out.println ("nb dans main "+
nb.toString()) ;
    Thread cal = new ThrCal (nb) ;
    Thread aff  = new ThrAff (nb) ;
    System.out.println ("tapez entree  pour
sortir") ;
    cal.start() ; aff.start() ;
    Clavier.lireString() ;
    cal.interrupt() ; aff.interrupt() ;
  }
}
// La classe Nombres contient 2 methodes
synchroniees. Chaque methode s'execute de
façon atomique.
class Nombres
{ public synchronized void calcul()
  { n++ ;
    carre = n*n ;
  }
  public synchronized void affiche ()
  { System.out.println (n + " a pour carre " +
carre ) ;
  }
  private int n=0, carre ;
}

// l'objet nb passe au thread est accessible dans
la methode run
class ThrCal extends Thread
{ public ThrCal (Nombres nb)
  { this.nb = nb ;
   System.out.println ("nb dans Thrcal "+
nb.toString()) ;
  }
  public void run ()
  { try
    { while (!interrupted())
      { nb.calcul () ;
        sleep (1000) ;
      }
    }
```

```
catch (InterruptedException e) {}
  }
    private Nombres nb;  // il s'agit d'une
reference à nb
}
```

```
class ThrAff extends Thread
{ public ThrAff (Nombres nb)
  { this.nb = nb ;
 System.out.println ("nb dans ThrAff "+
nb.toString()) ;
  }
   public void run ()
  { try
    { while (!interrupted())
      {
        nb.affiche() ;
        sleep (1000) ;
      }
    }
```

```
    catch (InterruptedException e) {}
  }
    private Nombres nb ;   // il s'agit d'une
reference à nb
}
```

```
/*
1 a pour carre 1
2 a pour carre 4
4 a pour carre 16
5 a pour carre 25
7 a pour carre 49
8 a pour carre 64
10 a pour carre 100
// Les threads « cal » et « aff » travaillent sur la
référence à nb. Donc, la modification faite par
« cal » est connue par « aff ».

« cal » est le producteur
« aff » est le consommateur
```

8.3 Mécanismes de synchronisation

8.3.1 Synchronised

Ce mécanisme permet de simuler les opérations P et V d'un sémaphore. P est appliquée en début de la méthode « synchronisée » et V à la fin de celle-ci.

8.3.2 wait() – notify()

Ce sont 2 méthodes permettant de synchroniser des threads sur l'accès à une ressource critique. Ci –dessous les modif. De l'exemple précédent :

```
// La classe Nombres contient 2 methodes synchroniees. Chaque methode s'execute de façon atomique.
class Nombres
{ public synchronized void calcul()
  { if( !pret) {n++ ;   carre = n*n ; pret=true ; notify() ; }
  } else wait() ;

  public synchronized void affiche ()
  { if( pret) { System.out.println (n + " a pour carre " + carre ) ;
     pret=false ; notify() ; }
  } else wait() ;
  private int n=0, carre ;
  private boolean pret=false ;
} // L'utilisation de notify() et wait() évite au thread « cal » de faire « sleep() » après le calcul.
```

9. Programmation graphique

9.1 Introduction

Version Java	Année	Librairies
1.0	1996	Awt,applet
1.1	1998	«
1.2	1999	« +swing
1.3	2000	
1.4	2002	

- **JFC (Java Foundation Classes de JAVA 2 (1999)) : ensemble de classes qui enrichi de manière significative la gestion de l'interface graphique actuelle AWT. Il se décompose en différents types de fonctionnalités :**
  Swing : ces nouvelles classes définissent un nouvel ensemble de composants reprenant les composants classiques (bouton, liste,...) enrichi de composants plus complexes (arbre, tableau,...). Par ailleurs elle défini le *Model-View-Controller* (MVC) qui permet une meilleure gestion de l'apparence des composants, des évènements et de la représentation des données.
  Une interface utilisateur obéissant à l'architecture MVC est composé de trois type d'objets qui communiquent entre-eux:
  - **un *modèle* qui une représentation logique de l'interface utilisateur**
  - **une *vue* qui est la représentation "visuelle" du modèle**
  - **un *contrôleur* qui gère l'interaction avec l'utilisateur.**
- **Java 2D : ces nouvelles classes du package java.awt permettent d'effectuer des opérations plus riches de dessins : définitions et compositions de formes, transformations, antialiasing, gestion plus riche des polices de caractères...**
- **Java 3D (compatible Java 2) : cette bibliothèque permet de programmer de réaliser des scènes virtuelles en 3D, et permet de gérer les éclairages, les textures, les animations d'objets 3D.**

9.1a - Comment créer une fenêtre ?

Exemple :

```
import javax.swing.* ;
public class fenetre0
{ public static void main (String args[])
  { JFrame fenetre = new JFrame() ;   // Création de l'objet de taille nulle
    fenetre.setSize (300, 50) ;     // Dimensionnement de la fenêtre
    fenetre.setTitle ("Ma Fenetre") ; // Titre de la fenêtre
    fenetre.setVisible (true) ;           //Affichage de la fenêtre
  }
}
```

La fenêtre(Affichée en haut à gauche) est représentée par un thread(qui subsiste après la fin de main()). Pour la fermer, cliquer sur

9.2 Évènements

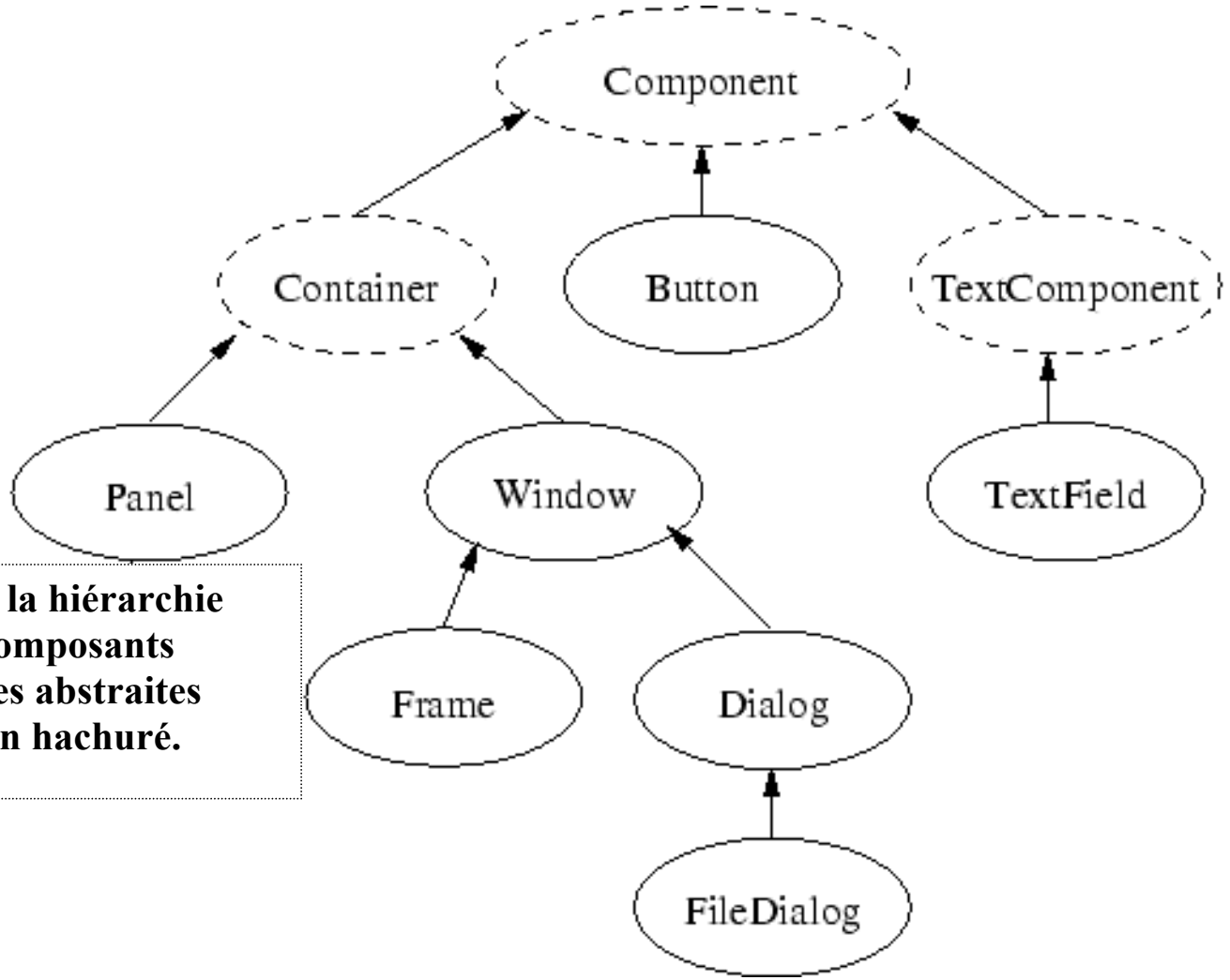
9.2.1 Introduction

Un évènement est un message qu'un objet envoie à un autre objet lui signalant la survenue de quelque chose de ``remarquable''. Il y a donc un émetteur et un récepteur. Dans une application graphique, il y a généralement une quantité importante d'objets et d'évènements possibles. Aussi, pour être récepteur d'un évènement, un objet devra signaler son intérêt pour cet évènement. Un même évènement peut intéresser plusieurs objets et inversement un même objet peut s'intéresser à plusieurs évènements.

LA JVM réagit aux évènements qui surgissent dans l'environnement du système d'exploitation (clic souris, frappe d'une touche ..) et les traduit en créant des objets de la classe EVENT (dans java.awt.Event).
Pour intercepter les évènements il faut :

- importer **java.awt.Event.*;**
- poser des **écouteurs d'événements** sur les composants qui doivent interagir avec l'utilisateur ,
- redéfinir des méthodes *gestionnaires d'événements*, **automatiquement appelées**.

9.2.2. Les classes de composants



Vue partielle de la hiérarchie d'héritage des composants AWT. Les classes abstraites sont entourées en hachuré.

9.2.3 Tableau des interfaces à implémenter en fonction des événements à gérer

Evénement généré par	Interface d'écoute à implémenter	Signatures des méthodes abstraites (à redéfinir obligatoirement)
Button, List, MenuItem, TexField	`ActionListener`	`public void actionPerformed(ActionEvent e)`
Scrollbar	`AdjustmentListener`	`public void adjustmentValueChanged(AdjustmentEvent e)`

Dialog, Frame	`ComponentListener`	`public void componentHidden(ComponentEvent e)`
		`public void componentMoved(ComponentEvent e)`
		`public void componentResized(ComponentEvent e)`
		`public void componentShown(ComponentEvent e)`

Container	`ContainerListener`	`public void componentAdded(ContainerEvent e)` `public void componentRemoved(ContainerEvent e)`
Component	`FocusListener`	`public void focusGained(FocusEvent e)` `public void focusLost(FocusEvent e)`
Checkbox,CheckboxMenuItem, Choice	`ItemListener`	`public void itemStateChanged(ItemEvent e)`

Component	`KeyListener`	`public void keyPressed(KeyEvent e)`
		`public void keyReleased(KeyEvent e)`
		`public void keyTyped(KeyEvent e)`

Canvas, Dialog, Frame, Panel, Window	`MouseListener`	`public void mouseClicked(MouseEvent e)`
		`public void mouseEntered(MouseEvent e)`
		`public void mouseExited(MouseEvent e)`
		`public void mousePressed(MouseEvent e)`
		`public void mouseReleased(MouseEvent e)`

Canvas, Dialog, Frame, Panel, Window	`MouseMotionListener`	`public void mouseDragged(MouseEvent e)` `public void mouseMoved(MouseEvent e)`
TextField	`TextListener`	`public void textValueChanged(TextEvent e)`

Dialog, Frame	`WindowListener`	`public void windowActivated(WindowEvent e)`
		`public void windowClosed(WindowEvent e)`
		`public void windowClosing(WindowEvent e)`
		`public void windowDeactivated(WindowEvent e)`
		`public void windowDeiconified(WindowEvent e)`
		`public void windowIconified(WindowEvent e)`
		`public void windowOpened(WindowEvent e)`

9.2.4. Boutons

. Création

- créer une fenêtre(constituée d'une racine, d'un contenu et d'une vitre),
- créer le ou les boutons, les associer au contenu de la fenêtre, - le bouton est visible par défaut,
- choisir le gestionnaire de formes, permettant de redimensionner et de placer les composants dans la fenêtre

. Sélection (avec un périphérique d'entrée quelconque)

- créer un écouteur sur le bouton qui implémente l'interface ActionListener,
- associer l'écouteur à la source de l'évènement.

Quand l'événement est déclenché, la méthode ActionPerformed de l'interface ActionListener est exécutée.

Si plusieurs boutons, la méthode getSource(), permet de les distinguer.

- 9.2.4. Exemples

9.2.4a - Comment désigner dans une fenêtre avec la souris?

```
import javax.swing.* ;  // Pour JFrame
import java.awt.event.* ;  // pour MouseEvent
et MouseListener
class MaFenetre extends JFrame implements
MouseListener
{ public MaFenetre ()    // constructeur
  { setTitle ("Ma fenetre + Désignation") ;
    setBounds (10, 20, 300, 200) ;
    addMouseListener (this) ;
// la fenetre sera son propre écouteur
// d'évènements souris
  }
//Redéfinition des méthodes événements
  public void mouseClicked(MouseEvent ev)
  { System.out.println ("clic dans fenetre") ;
  }
public void mousePressed (MouseEvent ev)
{}
public void mouseReleased(MouseEvent ev)
{}
public void mouseEntered (MouseEvent ev)
{}
  public void mouseExited  (MouseEvent ev)
{}
}
public class Designe
{ public static void main (String args[])
  { MaFenetre fenetre = new MaFenetre() ;
    fenetre.setVisible(true) ;
  }}
```

$ java Designe

clic dans fenetre

Explications : Identication des touches

9.2.4b - Comment saisir des info. clavier dans une fenêtre?

Exemple :

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;

class MaFenetre extends JFrame implements
KeyListener
{ public MaFenetre ()
  { setTitle ("Exemple lecture clavier") ;
    setSize (300, 180) ;
    addKeyListener (this) ;
  }
  public void keyPressed (KeyEvent e)
  { int code = e.getKeyCode() ;
    System.out.println ("Touche "+code+ " pressee :
" + e.getKeyText (code)) ;
  }
  public void keyReleased (KeyEvent e)
  { int code = e.getKeyCode() ;
    System.out.println ("Touche"+code+ " relachee :
" + e.getKeyText (code)) ;
  }
  public void keyTyped (KeyEvent e)
  { char c = e.getKeyChar() ;
    System.out.println ("Caractere frappe : " +
c + " de code " + (int)c) ;
  }
}
public class Clavier0
{ public static void main (String args[])
  { MaFenetre fen = new MaFenetre() ;
    fen.setVisible(true) ;
  }
}
```

9.2.4b - Comment créer des boutons dans une fenêtre + écouteur ?

Exemple

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;
class Fenetre extends JFrame implements
ActionListener
{ public Fenetre ()
  { setTitle ("Fenêtre + Boutons") ;
    setSize (300, 200) ;
    Bouton1 = new JButton ("Bouton 1") ;
    Bouton2 = new JButton ("Bouton 2") ;
    Container contenu = getContentPane() ;
    contenu.setLayout(new FlowLayout()) ;
    contenu.add(Bouton1) ;
    contenu.add(Bouton2) ;
    Bouton1.addActionListener(this);
    Bouton2.addActionListener(this);
  }
public void actionPerformed (ActionEvent
ev)
  { if (ev.getSource() == Bouton1)
    System.out.println ("action sur bouton 1") ;
if (ev.getSource() == Bouton2)
System.out.println("action sur bouton 2");
  }
  private JButton Bouton1, Bouton2 ;
}
public class Boutons
{ public static void main (String args[])
  { Fenetre fenetre = new Fenetre() ;
    fenetre.setVisible(true) ;
  }
}
```

$ java Boutons

action sur bouton 1

Explications sur l'exemple 9.2.4b précédent

Quand un "clic" a lieu sur un bouton (c-à-dire presser et relacher un bouton de la souris), une instance de la classe **`ActionEvent`** est envoyée.(La souris génére d'autres événements, voir la classe **`MouseEvent`**).
Le bouton cliqué émet de toute façon un événement de type **`ActionEvent`**.
Mais pour le détecter dans son code, le programmeur doit poser un écouteur spécifique sur ce bouton. Cela s'effectue par la méthode **`addActionListener()`** qui est définie dans la classe **`java.awt.Button`**
Plus précisément, le bouton **bouton1** est **la source** d'un objet-événement de la classe `ActionEvent`
Le paramètre **`this`** signifie que c'est la classe courante, ici **`Fenetre`**, qui est destinataire de l'événement et qui va se charger d'y répondre grâce à la méthode **`actionPerformed`**.
Pour pouvoir répondre à l'événement clic émis, la classe à laquelle est déléguée la charge de répondre, doit implémenter obligatoirement l' interface **`ActionListener`** .
Alors, Java exécute automatiquement toutes les méthodes de toutes les interfaces implémentées .
Dans l'interface **`ActionListener`**, il n'y a qu'une seule méthode **`actionPerformed (ActionEvent ev)`** exécutée *automatiquement*
Le programmeur doit "profiter" de l'appel automatique de cette méthode pour y associer le code qui décrit la réaction souhaitée au clic, ou plus généralement le traitement souhaité en réaction à l'événement **ev**.
Ici, le code se borne à vérifier que c'est bien le bouton **bouton1** qui a été cliqué, et le signale en affichant le message " **action sur bouton 1**" sur la console.
La méthode de base pour reconnaître le composant émetteur est **`getSource()`** qui adressé à l'objet-événement **ev** renvoit l'objet de la classe mère **Object**.

9.2.4c. Les contrôles usuels

. Case à cocher : permet d'effectuer un choix de type oui/non

- Création : caseAcocher = new JcheckBox(« case ») ;
- associer à un contenu : getContentpane().add(caseAcocher) ;
- associer un écouteur de la catégorie Item et rédéfinir sa méthode ItemStateChanged(),
- consulter l'état de la case à cocher : caseAcocher.isSelected() renvoie true/false
- positionner case à cocher : caseAcocher.setSelected(), positionne à true la case et génère un événement Item.

. Bouton radio : permet d'effectuer un choix de type oui/non

- Création : BoutonRadio = new JRadioButton(« boutonradio ») ;
- La suite est identique à celle de la case à cocher

. Champ de texte : insertion d'une texte
- Création : Texte = new JTextField(« texte … ») ;
- La suite est identique à celle de la case à cocher

9.3. Dessin

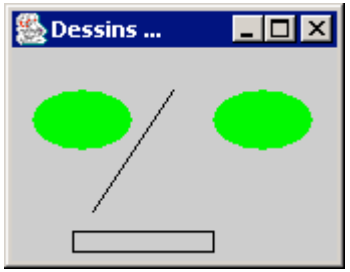
. Création

- Peut se faire sur n'importe quel composant,
- Le rafraîchissement du dessin est assuré paintComponent(), appelé automatiquement par la JVM,
- Le composant fenêtre utilise paint() qui ne réalise pas le rafraîchissement, c'est la raison pour laquelle il faut utiliser un panneau pour le dessin,
- Le panneau n'a ni titre, ni bordure, il est confondu à la fenêtre.
- Dans la classe panneau, il faut redéfinir paintComponent(), en commençant par faire appel à la méthode paintComponent() de la classe mère Jcomponent qui réalise le rafraîchissement.

9.3 Dessin

9.3.2 Exemple

```
import javax.swing.* ;
import java.awt.* ;
class Fenetre extends JFrame
{ Fenetre ()
  { setTitle ("Dessins ...") ;
    setSize (300, 150) ;
    p = new Panneau() ;
    getContentPane().add(p) ;
  }
  private JPanel p ;
}
class Panneau extends JPanel
{ public void
paintComponent(Graphics g)
  { super.paintComponent(g) ;
    g.drawLine  (40, 80, 80, 20) ;
g.drawRect (30,  90,  70, 10) ;
    g.setColor(Color.GREEN);
    g.fillOval (100,  20, 50, 30) ;
g.fillOval (10,  20, 50, 30) ;
  }
}
public class dessins
{ public static void main (String
args[])
  { Fenetre f = new Fenetre() ;
    f.setVisible(true) ;
  }
```



9.4. Dynamique des composants :

En java, la gestion des composants est dynamique :
. Création : composant.validate() force le gestionnaire de formes à placer le composant,
. Suppression : composant.remove supprime le composant de son conteneur
. Désactivation : composant.setEnabled(false) rend le composant non désignable
. Activation : composant.setEnabled(true) rend le composant désignable

9.4. Menus

. Création Barre de menu : BarreMenu = new JmenuBar() ;
setJmenuBar(BarreMenu) ;

. Création d'un sous menu : Menu = new Jmenu(« Titre ») ;

. Création d'un raccourci clavier : Menu.setMnemonic('T') ; //l'accès au ss menu « Titre » peut être obtenu par l'appui sur la touche 'T'.

. Ajout d'un sous menu à la barre de menu : BarreMenu.add(Menu) ;

. Création action dans un ss menu :

```
choix = new JcheckMenuItem(« action1 ») ; //case à cocher
couleur = new ActionCouleur[1] ; //Attribut graphique
attribut_geo = new JmenuItem(« rayon ») ; // Attribut géométrique
```

. Installation d'un écouteur sur action :

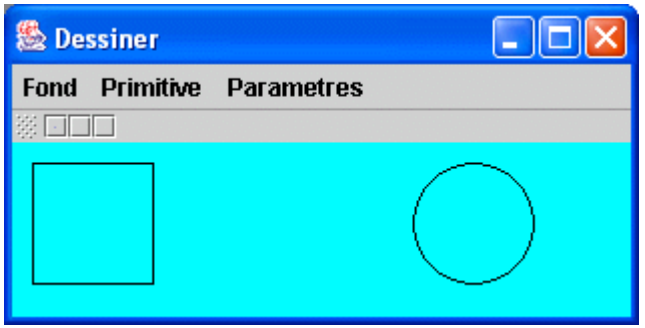
```
{choix,couleur, attribut_geo}.addActionListener(this) ;
```

. Création d'un menu surgissant : Surg = new JPopupMenu() ;

. Création d'une bulle d'aide : composant.setToolTipText(String)

Exemple : /* creation barre des menus */

```
barreMenus = new JMenuBar() ;
setJMenuBar(barreMenus) ;

   /* creation menu formes et ses options rectangle et ovale */
formes = new JMenu ("Primitive") ; formes.setMnemonic('P') ;
barreMenus.add(formes) ;
rectangle = new JCheckBoxMenuItem ("Rectangle") ;
formes.add(rectangle) ;
rectangle.addActionListener (this) ;
```




```
ovale = new JCheckBoxMenuItem ("Ovale") ;
formes.add(ovale) ;
ovale.addActionListener (this) ;

/* creation menu Parametres et ses options Hauteur et Largeur */
 Parametres = new JMenu ("Parametres") ;
barreMenus.add(Parametres) ;
largeur = new JMenuItem ("Largeur") ;
 Parametres.add(largeur) ;
largeur.addActionListener (this) ;
hauteur = new JMenuItem ("Hauteur") ;
Parametres.add(hauteur) ; hauteur.addActionListener (this) ;
```

```
//Modification de la largeur du rectangle - dans actionPerformed()
if (source == largeur)
    { String ch = JOptionPane.showInputDialog (this, "Largeur") ;
      pan.setLargeur (Integer.parseInt(ch)) ;
    }
```

9.4. Menus (exemples - suite)

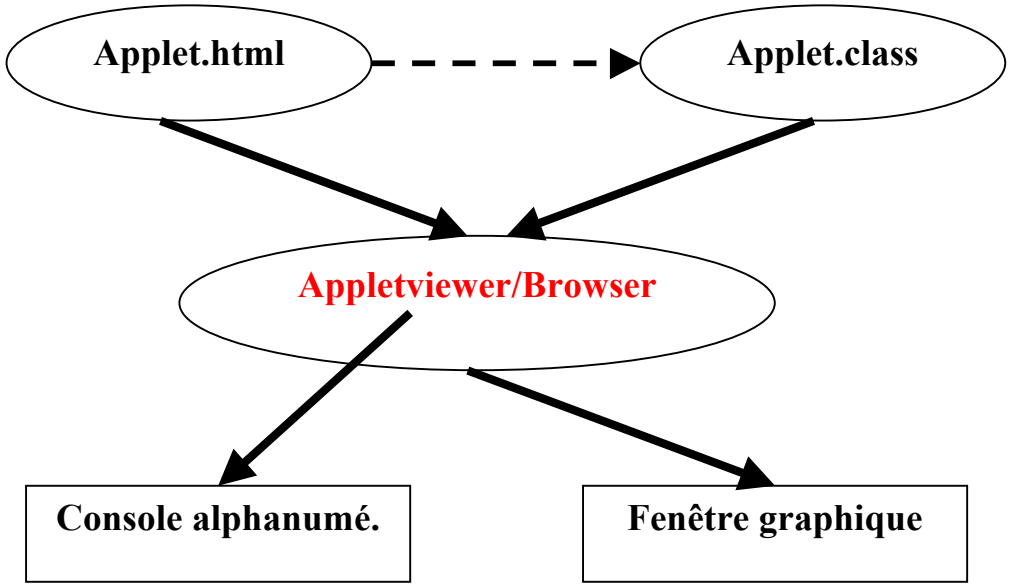
```
//Preparation des ss menus couleur appeles actions commune a plusieurs menus
  static public final String[] nomCouleurs =  //noms des ss menus de Fonds
                          {"jaune",    "vert",      "cyan"} ;
  static public final Color[] couleurs =   //les couleurs reelles correspondantes
                          {Color.yellow,   Color.green, Color.cyan} ;
  static public final String[] nomIcones  =   //les fichiers correspondants aux couleurs
                          {"jaune.gif", "vert.gif",  "cyan.gif"} ;

int nbCouleurs = nomCouleurs.length ;  // Nombre de couleurs envisagées
       /* creation des actions */
    actions = new ActionCouleur [nbCouleurs] ;  //actions est un meta objet{jaune/vert/cyan}
    for (int i=0 ; i<nbCouleurs ; i++)  //Initialisation du meta objet "actions"
    { actions[i] = new ActionCouleur (nomCouleurs[i], couleurs[i],
                                      nomIcones[i], pan) ; //pan : destination des ss menus de
"Fond"
    }

/* creation menu surgissant Couleur et ses options */
    couleurSurg = new JPopupMenu () ;
    for (int i=0 ; i<nomCouleurs.length ; i++)
      couleurSurg.add(actions[i]) ;  //ajout au menu surgissant des ss menus {jaune/vert/cyan}


/* affichage menu surgissant sur clic dans fenêtre */
    addMouseListener (new MouseAdapter ()
        { public void mouseReleased (MouseEvent e)
          { if (e.isPopupTrigger())
              couleurSurg.show (e.getComponent(), e.getX(), e.getY()) ;
          }
        }) ;
```

```
/*              creation barre d'outils couleurs                  */
      /* (avec suppression textes associes et ajout de bulles d'aide */
    barreCouleurs = new JToolBar () ;
    for (int i=0 ; i<nomCouleurs.length ; i++)
    { JButton boutonCourant = barreCouleurs.add(actions[i]) ;
      boutonCourant.setText(null) ;
      boutonCourant.setToolTipText
         ((String)actions[i].getValue(Action.SHORT_DESCRIPTION)) ; //bulle d'aide
    }
    contenu.add(barreCouleurs, "North") ;
  }
```

10.Applet

10.1 Introduction

10.1a Def. :Une applet est un programme qui est exécuté suite à l'analyse d'un fichier HTML contenant le nom et les paramètres de l'applet.



10.1b Méthodes de contrôle d'une applet

Une applet est controlée au moyen des quatre méthodes suivantes :

public void init()appelée après le chargement de l'applet.
public void start()appelée chaque fois que l'on entre dans le document la contenant.
public void stop()appelée chaque fois que l'on sort dans le document la contenant.
public void destroy()appelée pour détruire les ressources allouées par l'applet.

La méthode stop() ne fait rien par défaut, l'applet pourra donc continuer à s'exécuter en arrière plan.
Entre un appel à start() et l'appel suivant à stop() la méthode isAlive retourne true.

10.1c Fichier HTML de l'applet

Une applet est incluse dans un document HTML grâce à la balise <APPLET>,

exemple:

```
<APPLET
     CODE = "Applet0.class"
     WIDTH    = 250
     HEIGHT   = 100
  >
</APPLET>
```

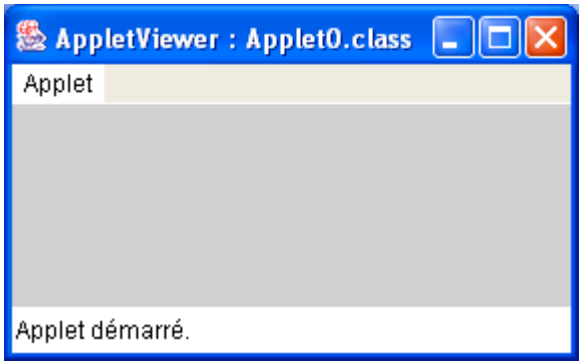
La méthode suivante permet de récupérer les paramètres passés dans la page HTML :

public String getParameter(String name)

10.1d Exemple : Applet

```
// Applet0.html
<HTML>
 <BODY>
  <APPLET
    CODE = "Applet0.class"
    WIDTH    = 250
    HEIGHT   = 100
  >
  </APPLET>
 </BODY>
</HTML>
```

```
// Applet0.java
import javax.swing.* ;
public class Applet0 extends JApplet
{ public void init ()
  { ;
    System.out.println ("Applet0") ;

  }
}
```

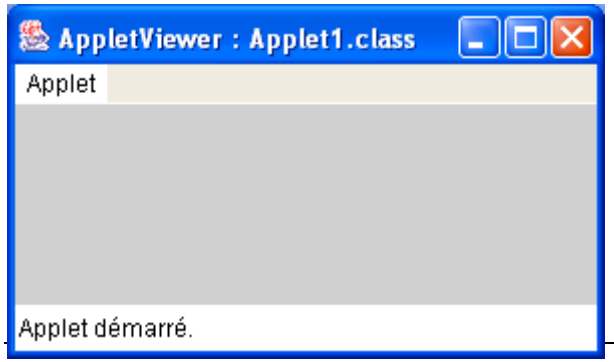


Affichage sur la console :

```
Applet0
```

10.1e Exemple : Applet avec paramètres

```
// Applet1.html
<HTML>
 <BODY>
  <APPLET
    CODE = "Applet1.class"
    WIDTH    = 250
    HEIGHT   = 100
  >
   <PARAM NAME = "fichier", VALUE = "f.dat">
   <PARAM NAME = "proprio", VALUE = "250">
   >
  </APPLET>
 </BODY>
</HTML>
```




```
// Applet1.java
import javax.swing.* ;
public class Applet1 extends JApplet
{ public void init ()
  { String fichier = getParameter ("fichier") ;
    String proprio = getParameter ("proprio") ;
    int uid ;
    System.out.println ("Fichier = " + fichier) ;
    System.out.println ("Proprietaire = " + proprio ) ;
    uid = Integer.parseInt(proprio) ;// conversion ASII-
INT
  }
}
```

Affichage sur la console :

```
Fichier = f.dat
Proprietaire = 250
```

10.3 Restrictions d'une applet

un certain nombre de manipulations sont interdites à l'applet :

1. **accès au système de fichier local**
2. **lancement de tâche au moyen de exec()**
3. **chargement de librairies**
4. **accès au System.getProperty() donnant des informations sur l'utilisateur, la machine locale.**
5. **modification des propriétés système**
6. **accès à un autre groupe de thread.**
7. **changement de ClassLoader, SocketImplFactory, SecurityManager, ContentHandlerFactory, URLStreamHandlerFactory**
8. **ouvrir une connexion réseau vers une autre machine que celle dont elle provient**
9. **accepter des connexions.**

Il existe des gestionnaires de sécurité qui prennent en compte la notion « d'applet certifiée »

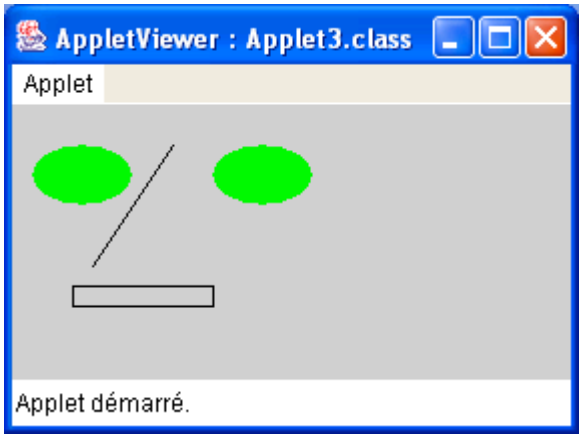
10.4 Transformation d'une application en applet

10.4a Actions

1) Supprimer « main() »,
2) Transformer l'objet JFrame (crée en général par main()) en un objet dérivé de JApplet. Eviter de doter cette classe de constructeur et confier ce travail à la méthode init(),
3) Supprimer les fonctions de dimensionnement (setSize(), setBounds()) et de commentaires (setTitle()) de fenêtre,
4) Editer le fichier HTML de l'applet,

10.4b Exemple – Transformation de l'ex. de dessin §9.3.2 en applet

```
// Applet3.html
<HTML>
 <BODY>
  <APPLET
    CODE = "Applet3.class"
    WIDTH   = 250
    HEIGHT   = 100
  >
  </APPLET>
 </BODY>
</HTML>
// Applet3.java
import javax.swing.* ;
import java.awt.* ;
/*
class Fenetre extends JFrame
{ Fenetre ()
  { setTitle ("Dessins ...") ;
    setSize (300, 150) ;*/
public class Applet3 extends JApplet
{public void init ()
  {
    p = new Panneau() ;
    getContentPane().add(p) ;
  }
  private JPanel p ;
}
class Panneau extends JPanel
{ public void paintComponent(Graphics g)
  { super.paintComponent(g) ;
g.drawLine(40,80,80,20);  g.drawRect (30,  90,  70, 10) ;
   g.setColor(Color.GREEN);
   g.fillOval (100,  20, 50, 30) ; g.fillOval (10,  20, 50, 30);
  }}/*
public class dessins
{ public static void main (String args[])
  { Fenetre f = new Fenetre() ;
    f.setVisible(true) ;
  }
}*/
```



11. LES FLUX (STREAMS)

11.1. Def. : Ce sont des canaux de transmission d'une suite d'octets à partir d'une source ou vers une destination (fichier, périphérique, emplacement mémoire, connexion à un site distant). Les flux sont unidirectionnels. On distingue les flux à accès séquentiel et les flux à accès direct.

11.2. Catégories

On en distingue :
- **flux binaires (les images, vidéos, sons, …)**
- **flux texte (fichiers issus de traiteur de textes).**

11.3 Classes utilisées

paquetage *java.io* contient une cinquantaine de classes.

On distingue quatre classes principales dont toutes les autres héritent des représentations et comportements propre à chaque famille.
- **java.io.InputStream (flot binaire en lecture),**
- **java.io.OutputStream (flot binaire en écriture),**
- **java.io.Reader (flot texte en lecture),**
- **java.io.Writer (flot texte en écriture).**

11.4 Flux binaire en sortie à accès séquentiel
Exemple :Fichier binaire d'entiers lûs au clavier

```
import java.io.* ;
public class bin_seq_out
{ public static void main (String args[])
throws IOException
  {
   int n ; //information saisie
  DataOutputStream f= new
DataOutputStream
    (new FileOutputStream ("f.dat")) ;
do { System.out.print ("Saisir un entier :
") ;
        n = Clavier.lireInt() ;
        if (n != 0)
         { f. writeInt (n)  ;
         }
      }
   while (n != 0) ;
  f.close () ;
    System.out.println("f.dat est complet !
") ;
  }
}
```

```
// Résultats :
Saisir un entier : 1
Saisir un entier : 2
Saisir un entier : 45
Saisir un entier : 5
Saisir un entier : 0
f.dat est complet !
```

OutputStream(Classe abstraite) ← FileOutputStream (Classe manipulant les flux en sortie)
OutputStream(Classe abstraite) ← DataOutputStream (Classe manipulant les flux en sortie avec formatage)

11.5 Flux binaire en entrée à accès séquentiel

Exemple :

Fichier binaire d'entiers lûs dans un fichier

```
import java.io.* ;
public class bin_seq_in
{ public static void main (String args[]) throws
IOException
  { String f = "f.dat" ;
    int n = 0 ;

DataInputStream entree = new
DataInputStream
                    ( new FileInputStream (f)) ;
 System.out.println ("valeurs lues dans le fichier
" + f + " :") ;
boolean eof = false ;  // sera mis a true par
exception EOFile
    while (!eof)
    { try
      {  n = entree.readInt ()  ;
      }
      catch (EOFException e)
      { eof = true ;
      }
      if (!eof) System.out.println (n) ;
    } //while
    entree.close () ;
    System.out.println ("fin du fichier  :  " + f );
  }
}
```

```
// Résultats d'exécution
valeurs lues dans le fichier f.dat :
1
2
45
5
fin du fichier  :  f.dat
```

11.6 Flux binaire en entrée à accès direct

Exemple :

Accès direct dans fichier binaire d'entiers

```
import java.io.* ;
public class bin_dir_in
{ public static void main (String args[]) throws
IOException
  {
    String f;
    int n, i ;
    RandomAccessFile e ;
    System.out.print ("Nom du fichier : ") ;
    f = Clavier.lireString() ;
    e = new RandomAccessFile (f, "  ") ;

    do
     { System.out.print ("Numéro de l'entier
recherché : ") ;
       i = Clavier.lireInt() ;
       if (i == 0) break ;
       e.seek (4*(i-1)) ;
       n = e.readInt() ;
       System.out.println ("  valeur = " + n) ;
     }
    while (i != 0) ;

    e.close () ;
    System.out.println ("fin de la recherche dans
" +f);
  }
}
// Résultats d'exécution
Nom du fichier : f.dat
Numéro de l'entier recherché : 3
  valeur = 45
Numéro de l'entier recherché : 0
fin de la recherche dans f.dat
```

11.7 Flux texte en sortie
Exemple : Création d'un fichier texte

```
import java.io.* ;
public class txt_seq_out
{ public static void main (String args[]) throws
IOException
  {
   String f ;
   int  n ;
  System.out.print ("Nom du fichier a creer : ") ;
    f = Clavier.lireString() ;
 PrintWriter s = new PrintWriter (new
FileWriter (f)) ;

    do
     { System.out.print ("Saisir un entier : ") ;
       n = Clavier.lireInt() ;
       if (n != 0)
        { s.println (n + " a pour carre " + n*n)  ;
        }
     }
    while (n != 0) ;
    s.close () ;
System.out.println ("fin creation fichier " + f);
  }
}
```

```
// Résultats d'exécution
Nom du fichier a creer : t.txt
Saisir un entier : 1
Saisir un entier : 2
Saisir un entier : 3
Saisir un entier : 45
Saisir un entier : 0
fin creation fichier t.txt

//Contenu du fichier t.txt
1 a pour carre 1
2 a pour carre 4
3 a pour carre 9
45 a pour carre 2025
```

11.8 Flux texte en entrée

Exemple : Création d'un fichier texte

```
import java.io.* ;
public class txt_seq_in
{ public static void main (String args[]) throws
IOException
  {
    String f ;
    String ligne ;
    int n ;
    System.out.print ("Donnez le nom du fichier a
lister : ") ;
    f = Clavier.lireString() ;
    BufferedReader entree = new BufferedReader
(new FileReader (f)) ;

    do
      { ligne = entree.readLine() ;
        if (ligne != null) System.out.println (ligne) ;
      }
    while (ligne != null) ;
    entree.close () ;
    System.out.println ("fin fichier " +f);
  }
```

```
}

// Résultats d'exécution
Donnez le nom du fichier a lister : t.txt
1 a pour carre 1

2 a pour carre 4

3 a pour carre 9

45 a pour carre 2025

fin fichier t.txt
```

12 LES COLLECTIONS

L'interface *Collection* représente la racine de la hiérarchie des collections Java.
Une collection représente un groupe d'objets, dénommé éléments pour les listes (*List*) ou les ensembles (*Set*).

L'interface *Collection* possèdent trois sous-interfaces *List*, *Set* et *SortedSet*.
Les implémentations de l'interface *Collection* sont *AbstractCollection*, *AbstractList*, *AbstractSet*, *ArrayList*, *HashSet*, *LinkedHashSet*, *LinkedList*, *TreeSet* et *Vector*.

12.3 concepts des collections

Une collection peut contenir des éléments de type différents. A ces éléments peuvent être appliqués plusieurs opérations :

- **positionnement (méthode set()),**
- **interrogation (méthode get()),**
- **parcours (méthodes next() , previous), procurées par l'interface iterator. Le parcours est dédié à un objet iterator**
- **comparaison (méthode compareTo()),**
- **ajout (add(element)),**
- **suppression (remove(element)),**
- **…**

Des méthodes applicables à des collections :

- **Fusion (collection1.addAll(collection2)),**
- **Suppression (collection1.removeAll(collection2)), - supprime de collection1 tous les éléments trouvés dans collection2,**
- **Suppression (collection1.retainAll(collection2)), - supprime de collection1 tous les éléments non trouvés dans collection2,**
- **…**

12.4 Exemple – Liste chaînée

```
import java.util.* ;
public class Liste1
{ public static void main (String args[])
  { LinkedList l = new LinkedList() ;  //def. de la liste
chainée

   System.out.print ("Liste en A : ") ; affiche (l) ;
   l.add ("a") ; l.add ("b") ;  // ajouts en fin de liste
   System.out.print ("Liste en B : ") ; affiche (l) ;

   ListIterator it = l.listIterator () ;
   it.next() ; // on se place sur le premier élement
   it.add ("c") ; it.add ("b") ;  // et on ajoute deux
elements
   System.out.print ("Liste en C : ") ; affiche (l) ;

   it = l.listIterator() ;
   it.next() ; // on progresse d'un element
   it.add ("b") ; it.add ("d") ;     // et on ajoute deux
elements
   System.out.print ("Liste en D : ") ; affiche (l) ;

   it = l.listIterator (l.size()) ;   // on se place en fin de
liste
   while (it.hasPrevious())     // on recherche le dernier b
   { String ch = (String) it.previous() ;
     if (ch.equals ("b"))
     { it.remove() ;         // et on le supprime
       break ;
     }
   }
   System.out.print ("Liste en E : ") ; affiche (l) ;

  }
  public static void affiche (LinkedList l)
  { ListIterator iter = l.listIterator ();
    while (iter.hasNext()) System.out.print (iter.next() +
" ") ;
    System.out.println () ;
  }
}

//Résultats
Liste en A :

Liste en B : a b

Liste en C : a c b b

Liste en D : a b d c b b

Liste en E : a b d c b
```

12.4 Exemple – Vecteurs dynamiques

```
import java.util.* ;
public class Array1
{ public static void main (String args[])
  { ArrayList v = new ArrayList () ;
    System.out.println ("En A : taille de v = " +
v.size() ) ;
     /* on ajoute 10 objets de type Integer */
    for (int i=0 ; i<10 ; i++) v.add (new Integer(i)) ;
    System.out.println ("En B : taille de v = " +
v.size() ) ;
     /* affichage du contenu, par acces direct (get) a
chaque element */
    System.out.println ("En B : contenu de v = ") ;
    for (int i = 0 ; i<v.size() ; i++)
      System.out.print (v.get(i) + " ") ;
    System.out.println () ;
     /* suppression des elements de position donnee
*/
    v.remove (3) ;
    v.remove (5) ;
    v.remove (5) ;
    System.out.println ("En C : contenu de v = " + v)
;
     /* ajout d'elements a une position donnee */
    v.add (2, new Integer (100)) ;
    v.add (2, new Integer (200)) ;
    System.out.println ("En D : contenu de v = " + v)
;
     /* modification d'elements de position donnee
*/
    v.set (2, new Integer (1000)) ;  // modification
element de rang 2
    v.set (5, new Integer (2000)) ;  // modification
element de rang 5
    System.out.println ("En D : contenu de v = " + v)
;}
}
/*
En A : taille de v = 0
En B : taille de v = 10
En B : contenu de v =0 1 2 3 4 5 6 7 8 9
En C : contenu de v = [0, 1, 2, 4, 5, 8, 9]
En D : contenu de v = [0, 1, 200, 100, 2, 4, 5, 8, 9]
En D : contenu de v = [0, 1, 1000, 100, 2, 2000, 5,
8, 9]

*/
```

13. Enoncés des TDs et des TPs

TD1 : Notions de classe et de méthodes

Exercice 1 :

Écrivez une classe en Java appelée **`Calcul`** effectuant la somme et la moyenne des éléments du tableau **t** défini ainsi :
int[] t = {12, 24, 12, -4, 6, 99, 22} ;
dans la fonction **main** qui se charge d'appeler les deux méthodes statiques **somme** et **moyenne** et d'afficher les résultats.
Ces deux méthodes ont pour prototypes :
public static int somme(int[] tab)
public static int moyenne(int[] tab)

Exercice 2 :

Écrire une classe en Java appelée **Puissance** effectuant la puissance des éléments du tableau **t** défini ainsi dans la méthode **main** :
double [] t = {12, -5.3, 0, 32.4} ;
Cette classe calcule et affiche la valeur de x^n (lire x puissance n) où **x** est un réel (élément du tableau t) et **n** un entier introduit comme argument de la ligne de commande. Pour une exécution du programme, **n** est unique et sera appliqué à tous les éléments du tableau **t**. Ci-deesous sont indiqués des exemples d'execution du programme Puissance.class.

```
/* EXEMPLES DE RESULTATS
//execution pour n=2
 $ java Puissance.class 2
 12.0 puissance 2 = 144.0
 -5.3 puissance 2 = 28.09
 0.0 puissance 2 = 0.0
 32.4 puissance 2 = 1049.76

//execution pour n=-2
$ java Puissance.class -2

 12.0 puissance -2 = 0.006944444444444444
 -5.3 puissance -2 = 0.0355998576005696
 0.0 puissance -2 = Pas défini
 32.4 puissance -2 = 9.525986892242037E-4

//execution pour n=0
$ java Puissance.class 0
  12.0 puissance 0 = 1
 -5.3 puissance 0 = 1
 0.0 puissance 0 = Pas défini
 32.4 puissance 0 = 1
*/
```

Note : Il ne s'agit pas d'utiliser une classe pré existante dans les packages java mais à écrire votre propre classe.

Exercice 3 :

On appelle palindrome un mot pouvant se lire indifféremment dans les deux sens : par exemple *Laval*, *elle*, *ici*,… Plus formellement, un mot de longueur *l* est un palindrome si pour tout indice $i<l$, le caractère à la position *i* est égal au caractère à la position *l-i*-1.
Il s'agit d'écrire une classe en Java appelée **Palindrome** constituée de :

1. Une méthode statique appelée **palindrome** prenant en argument un tableau de caractères **s** et renvoyant un booléen indiquant si le tableau est un palindrome. Le prototype de cette dernière est :
   public static boolean palindrome(String s)

2. La méthode statique **`main`** permettant de tester cette fonction. La chaîne de caractères à tester sera récupérée par **`main`** sur la ligne de commande.

Exercice 4 :

Ecrire une classe JAVA, nommé **Mention**, permettant d'afficher la mention obtenue en fonction de la note moyenne saisie au clavier.

Cette classe doit contenir :
Q.1)une méthode de saisie de la moyenne de prototype :
public static double saisieDouble()
Cette méthode permet de lire une chaîne terminée par ENTER saisie au clavier(**System.in**)
et la convertir en double.

Q.2) une méthode permettant de traduire la moyenne en une mention selon :

Note moyenne	Mention
note >= 18,	Excellent
18 > note >= 16,	TresBien
16 > note >= 14,	Bien
14 > note >= 12,	AssezBien
10 >=note < 12,	Passable
note < 10.	Ajourné

Le prototype de cette méthode est :
static String traduction(double moyenne)

Des exemples de résultats sont fournis ci-dessous :

```
$ java Mention.class
 Entrez votre moyenne et tapez ENTER : 16.56
 Votre mention est : TresBien

$ java Mention.class
 Entrez votre moyenne et tapez ENTER : 11.89
 Votre mention est : Passable

$ java Mention.class
 Entrez votre moyenne et tapez ENTER : 9.5
 Votre mention est : Ajourné
```

TD2 : Notions de tableaux

Exercice 4 :

Quel est le résultat affiché par le programme ci-après ?
Quel est son rôle ?

```
public class TabArg1

{ public static void main (String args[])

  { final int N = 4 ;

    int t1[] = new int [N] ;

    int t2[] = new int [N] ;

    for (int i=0 ; i<N ; i++) t1[i] = i+1 ;

    for (int i=0 ; i<N ; i++) t2[i] = 2*i+1 ;

        // affichage des valeurs de t1 et de t2

    System.out.print ("t1 = ") ; affiche (t1) ;

    System.out.print ("t2 = ") ; affiche (t2) ;

    t1 = t2 ;

    t1[0] = 10 ; t2[1] = 20 ; t1[2] = 30 ; t2[3] = 40 ;

        // affichage des valeurs de t1 et de t2

    System.out.print ("t1 = ") ; affiche (t1) ;

    System.out.print ("t2 = ") ; affiche (t2) ;

  }

  static void affiche (int [] t)

  { for (int i=0 ; i<t.length ; i++)

      System.out.print (t[i] + " ") ;

    System.out.println () ;

  }
```

Exercice 5 :

Quel est le résultat affiché par le programme ci-après ?
Quel est son rôle ?

```
public class Tab2Ind1

{ public static void main (String args[])

  { int [] [] t = new int [3][] ;

    for (int i=0 ; i<3 ; i++)

      { t[i] = new int [i+1] ;

        for (int j=0 ; j<t[i].length ; j++)

          t[i][j] = i+j ;

      }

    for (int i=0 ; i<3 ; i++)

      { System.out.print ("tableau numero " + i + " = ") ;

        for (int j=0 ; j<t[i].length ; j++)

          System.out.print (t[i][j] + " ") ;

        System.out.println () ;

      }

  }

}
```

TD3 : Notions de classes et d'héritage

Exercice 1 :

Il s'agit d'analyser le programme Java suivant , et de répondre aux différentes questions.

```
abstract class Homme{
protected int age;
protected boolean masculin;

Homme(){
age = 20;
masculin = false;}

Homme(boolean masc){
age = 20;
if (masc){
masculin = true;}
else {
masculin = false;}}

abstract public void direBonjour();
public int getAge(){
return age;}

public boolean getMasc(){
return masculin;}
}  // fin classe Homme

class Francais extends Homme{
Francais(){
super();}
Francais(boolean masc){
super(masc);}

public void direBonjour(){
System.out.println("Bonjour");}
}  //fin classe Francais

class Anglais extends Homme{
Anglais(){
super();}

Anglais(boolean masc){
super(masc);}

public void direBonjour(){
System.out.println("Good morning ");}
}  //fin classe Anglais

class Hollandais extends Homme{
Hollandais(){
super();}

Hollandais(boolean masc){
super(masc);}

public void direBonjour(){
System.out.println("Goeie dag ");}
} //fin classe Hollandais

class Bonjour{
public static void main(String[] argv){
Homme personnes[] = new Homme[3];
personnes[0] = new Anglais();
personnes[1] = new Hollandais(false);
personnes[2] = new Francais(true);
for (int i = 0; i < 3; i++){
personnes[i].direBonjour();
System.out.println("J'ai                "          +
personnes[i].getAge() + " ans.");
System.out.print("Je suis ");
if (personnes[i].getMasc()){
System.out.println("un homme.");
}
else{
System.out.println("une femme.");
}
}
```

Q. 1- Que signifie la présence de "abstract" dans la définition de la classe "Homme" ?

Q. 2- Que signifie la présence de "abstract" dans la définition de la méthode "direBonjour" de la classe "Homme" ?

Q. 3-Quel est l'affichage produit par l'exécution de ce programme ?

Q. 4-Quel intérêt présente une classe abstraite ?

Exercice 2 : Indiquer le cas échéant le numéro de la ou des lignes provoquant une erreur de compilation. Préciser l'origine de l'erreur et justifier votre réponse.

```
1.  class A {
2.  }
3.  class B  extends A {
4.  }
5.  public class Main {
6.  public static void main(String args[]) {
7.  A a=new A();
8.  B b=new B();
9.  A a1=b;
10. a1=(B)b;
11. b=(B)a;
12. }
13. }
```

Exercice 3 : quel est le résultat affiché par l'exécution de ce programme. Justifier votre réponse.

```
1.  class A {
2.  public boolean equals(A obj) {
```

```
3.  System.out.println("A equals");
4.  return true;
5.  }
6.  }
7.  class B extends A {
8.  public boolean equals(B obj) {
9.  System.out.println("B equals");
10. return true;
11. }
12. }
13. class Main2 {
14. public static void main(String args[]) {
15. B b=new B();
16. A a=b;
17. Object o=a;
18. System.out.println(b.equals(b));
19. System.out.println(b.equals(a));
20. System.out.println(a.equals(b));
21. System.out.println(o.equals(b));
22. }
23. }
```

Exercice 5:

Il s'agit d'analyser le programme Java suivant , et de répondre aux différentes questions.

```
1.  public class Threads extends Thread {

2.  private static int nbThreads = 0; // nb de threads
    crees
3.  private int numero; // numéro du thread
4.  Threads () {
5.  numero = nbThreads;
6.  nbThreads = nbThreads + 1;
7.  System.out.println("Threads : Thread numero " +
    numero + " cree." );
8.  }

9.  public void run() {
10. System.out.println("Run : Thread numero " +
    numero + " demarre." );
11.     try {
12. Thread.sleep(1);
13. } catch (InterruptedException e) {
14. return;
15. }
16. System.out.println("Run : Thread numero " +
    numero + " termine." );
17. }

18. public static void main(String args[]) {
19. System.out.println("Main : Programme demarre..");
20. for (int i=0; i < 3; i++) {
21. Thread unThread = new Threads();
22. unThread.start();
23. }
24. System.out.println("Main : Programme principal
    termine.");
25. }
26. }
```

1 Combien de threads ce programme permet il d'engendrer ?
2 quel est le rôle de la ligne 12 ?
3 Quel est l'affichage produit par l'exécution de ce programme ?
4 Quel est le nouveau affichage produit par l'exécution de ce programme, si les lignes 11-15 sont supprimées?

Exercice 4:

Q1 :Quel est le rôle du programme suivant et qu'affiche t il suite à son exécution ? Justifier votre réponse.

```
1.  class Somme extends Thread {
2.  Somme(String s) {
3.  super(s);
4.  System.out.println(s);
5.  }
6.  public void run() {
7.  int j=0;
8.  for (int i=0;i<9;i++) {
9.  j=j+i;
10. }
11. System.out.println(j);
12. }
13. }
14. class Main3
15. {
16. public static void main(String args[]) {
17. for (int i=0;i<4;i++) {
18. new
    Somme("Somme"+Integer.toString(i)).start();
19. }
20. }
21. }
```

Q2 : L'ajout de l'instruction « sleep(j) » après la ligne 11 engendre une erreur. Laquelle ?

Proposez une modification du programme pour permettre l'insertion de cette instruction dans le programme.

TD4 Java – programmation graphique

Exo 1 : Champs de texte

Soit l'interface graphique suivante :
et ci dessous le code Java correspondant incomplet .

```
import javax.swing.* ; class CopyTexte extends
Jframe{
private JtextField rightTextField ; private Jbutton
copyButton ; private JtextField lefTextField ;
public CopyTexte() { lefTextField = new JtextField() ;
copyButton = new Jbutton() ; rightTextField = new
JtextField() ; lefTextField.setEditable(false) ; //Ajout
rightTextField.setEditable(false) ; //Ajout
getContentPane().setLayout(new FlowLayout()) ;
setTitle(« Recopie Texte ») ;
getContentPane().add(lefTextField) ;
copyButton.setText(« RECOPIE ») ;
getContentPane().add(copyButton) ;
getContentPane().add(rightTextField) ; pack() ;
}
} public class A {
}
public static void main(String args[]) { new
CopyTexte().setVisible(true) ;
}
```

Sachant que l'interface d'écoute des événements de type `ActionEvent` déclenchés par un clic sur un bouton
est `ActionListerner` et est définie comme suit :

```
public interface ActionListener {public void
actionPerformed(ActionEvent e) ;
}
```

Donner ci dessous les ajouts que vous apporteriez au code précédent pour que lorsque l'utilisateur clique sur le bouton « RECOPIE » le contenu du champ texte de gauche soit recopié dans le champ texte de droite (méthodes `String getText()` pour récupérer et `setText(String s)` pour affecter une valeur à un `JtextField`).

Exo 2 : Fenêtre de dialogue

Il s'agit de compléter le programme Java suivant , et de répondre aux différentes questions.

```
import javax.swing.* ;
import java.awt.* ;
import java.awt.event.* ;
class FenDialog extends JFrame implements
ActionListener
{ public FenDialog ()
  { setTitle ("Essai boite de dialogue") ;
    setSize (400, 200) ;
    lanceDial = new JButton ("Lancement dialogue")
; // Partie A - à compléter

……
…… ……
……
……
……
……
……
……
….
; // partie B - à compléter

……
…… ……
……
……
……
……
……
……
….
  private JButton lanceDial ;
}
public class Dialog1
{ public static void main (String args[])
  { FenDialog fen = new FenDialog() ;
    fen.setVisible(true) ;
  }
}
```

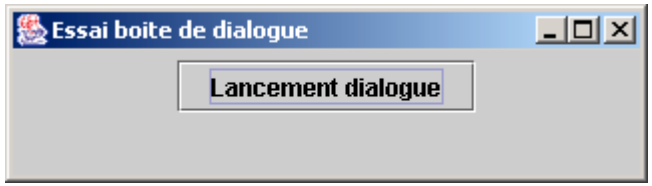
Ce programme consiste à créer une fenêtre principale intitulée "Essai boite de dialogue" contenant un composant "Bouton" intitulé "Lancement dialogue".
L'action sur ce bouton permet d'afficher une fenêtre secondaire de la classe JDialog dont un aperçu de la documentation est fournie ci-après :

"public JDialog(Frame owner,String title)
Creates a non-modal dialog with the specified title and with the specified owner frame. If owner is null, a shared, hidden frame will be set as the owner of the dialog."

Comme JFrame, la classe JDialog hérite de la Component qui contient les méthodes setSize et setVisible.

5.1 : Compléter la partie A indiquée qui doit permettre de créer le bouton de la fenêtre principale et de préparer l'action sur ce dernier.

5.2 : Compléter la partie B indiquée qui doit permettre de prendre en compte l'action sur le bouton et d'afficher la fenêtre secondaire.



II74 : Programmation Orientée Objet et Interface Homme-Machine

TP1: Exceptions, Threads et Héritage

Malik Mallem, Jean-Yves Didier, Samir Otmane

Connaisances requises

- Connaissance générale du langage Java,
- Notions de programmation orientée objet,
- Notions de *thread* et d' *exception*.

But du TP

Ce TP se propose de :
- vous faire mettre en oeuvre les mécanismes liés aux exceptions Java,
- vous familiariser avec les threads et la programmation concurrentielle,
- vous faire appliquer les notions d'héritage des classes.

Situation Initiale

Le producteur et le consommateur : *Un producteur produit des objets qui sont consommés par un consommateur . A une lointaine époque, le consommateur pouvait demander directement l'objet au producteur. Toutefois, la legislation actuelle, pour des raisons d hygiène, impose que le producteur ne soit plus en contact direct avec le consommateur. L'échange d objets se fait donc par un compartiment muni de deux trappes. Une trappe est destiné au producteur, l'autre au consommateur. Il ne peut y avoir qu'une seule trappe ouverte en même temps. Le producteur, son objet terminé au bout d'un temps plus ou moins long, ouvre la trappe et met l'objet dans le compartiment. Le consommateur, quant à lui, ouvre la trappe dès qu'il peut pour voir si l'objet de sa convoitise se trouve dans le compartiment.*

Nous nous proposons de simuler par un programme Java le comportement d'un producteur et d'un consommateur qui s'échangent des objets via un compartiment.

Le producteur ainsi que le consommateur peuvent être assimilés à des threads Java au comportement bien déterminé.

1 Une première solution "naïve"

Quatre classes ont été programmées pour simuler le comportement décrit plus haut :
- `Producer.java` : la classe qui décrit le comportement du producteur,
- `Consumer.java` : la classe qui décrit le comportement du consommateur,
- `CubbyHole.java` : la classe qui décrit le comportement du compartiment,
- `ProducerConsumerTest.java` : le point de départ de notre application.

Le producteur et le consommateur se communiquent les objets en employant les méthodes `put(...)` et `get()` du compartiment comme montré sur la figure ci-dessous :

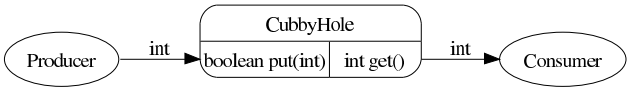


FIG. 1 – Communication produteur/consommateur via le compartiment

Ces classes sont accessibles sur la machine `ens-unix` à l'adresse `~mallem/ii74/tp1/baseTP/`. Pour la suite du TP, vous aurez également besoinde vous référer à la documentation de l'API Java que l'on peut trouver à l'adresse : `http ://java.sun.com/j2se/1.5.0/docs/api/index.html`

Travail demandé

1. Recopier les classes citée plus haut sur votre compte.
2. Compiler l'exemple. Quel est l'erreur donnée par le compilateur ? Quelle est la classe concernée et à quelle ligne ? Quelle est l'instruction qui pose problème ? Se reporter à la documentation en ligne de Java pour découvrir le pourquoi de cette erreur.
3. Modifier le programme pour traiter l'exception levée par l'instruction que vous avez repérée à la question précédente.
4. Compiler et exécuter. Lorsque l'on regarde attentivement le résultat de l'exécution, l'application a-t'elle le comportement désiré ? Sinon, pourquoi ?

2 Héritages de classe et Exceptions

Le but est d'employer le mécanisme des exceptions pour repérer le comportement anormal du programme et y mettre fin. Pour y arriver, le compartiment devra lever une exception quand le consommateur reçoit deux fois le même objet.

Travail demandé

1. Créer une nouvelle classe `MagicCubbyHoleException` qui héritera de la classe `Exception` de l'API Java standard. La nouvelle classe appellera un constructeur de sa superclasse afin de contenir un message de votre choix indiquant le fonctionnement abhérant du programme lorsque celui-ci survient.
2. Modifier la classe `CubbyHole` pour lever une exception `MagicCubbyHoleException` quand le consommateur prend deuxfois le même objet.

3. Modifier la classe `Consumer` pour traiter l'exception`MagicCubbyHoleException` levée par les objets de type `CubbyHole`. Si cette exception est attrappée, alors le programme doit se terminer immédiatement (Se référer à la documentation de la classe `System` de l'API Java).
4. Compiler et exécuter. Arrive-t'on à passer 10 objets ?

3 Threads et synchronisation de threads

Le mot-clé `synchronized`

Ce mot-clé est un modificateur de méthode qui permet de vérouiller l'objet ('lock') quand la méthode est appelée. Pendant ce temps, tout autre appel à une méthode de l'objet (la même ou une autre) faite par un autre objet sera impossible. Par contre, une méthode `synchronized` peut appeller une autre méthode `synchronized` du même objet. C'est ce que l'on appelle les 'verrous réentrants' (reentrant lock).

Travail Demandé

1. A l'aide de la documentation Java, expliquer :
   (a) dans la classe `Producer`, pourquoi est implémantée une méthode `run()`,
   (b) dans la classe `Producer`, à quoi sert l'appel `setPriority(Thread.MIN_PRIORITY)`,
   (c) dans la classe `CubbyHole`, ce que fait l appel `sleep((int)(Math.random()*100)`,
   (d) dans la classe `ProducerConsumerTest`, ce que fait l'appel `p1.start()`.
2. Modifier la classe `CubbyHole` pour empêcher le producteur de mettre un objet quand le consommateur est en train d'en retirer un et inversement.
3. Compiler et exécuter. L'exception `MagicCubbyHoleException` est-elle encore levée ?

4 Optimisation des threads

Le 'polling'

Le polling est une portion de code qui consiste en une boucle dans laquelle la seule opération effectuée est un test. Le succés du test décide de la sortie de la boucle ou non. L'inconvénient d'une telle méthode est qu'elle 'mange' des cycles processeur tout en ne faisant rien par ailleurs. Une solution élégante est donc d'endormir le processus en attendant que les conditions du test assurent son succés, puis de le réveiller.

Les méthodes `wait()`, `notify()`, `notifyAll()` de la classe `Object`

Chaque objet en Java possède la faculté d'endormir le thread qui appelle ses méthodes et de le réveiller le cas échéant.

- La méthode `wait()` endort le processus pour un temps déterminé ou non. Une exception de type `InterruptedException` est levée lorsque le thread est réveillé.
- La méthode `notify()` d'un objet réveille un et un seul thread qui a été endormi par l'appel de la méthode `wait()` du même objet.
- La méthode `notifyAll()` réveille tous les threads qui ont été endormis par l'appel de la méthode `wait()` du même objet.

Travail demandé

1. Consulter la documentation Java sur les méthodes citées ci-dessus,
2. Repérer, dans les classes `Producer` et `Consumer`, les portions de code associées à du 'polling',
3. Modifier la classe `CubbyHole` de manière à utiliser `wait()` et `notify()` aux endroits appropriés,
4. Compiler et exécuter. Vérifier que tout fonctionne comme avant.
5. Quelle conséquences ces modifications entraînent-elles ?

A Classes de base du TP

Classe `Producer`

```
public class Producer extends Thread {
     private CubbyHole cubbyhole;

     public Producer(CubbyHole c) {
          cubbyhole = c;
     }

     public void run() {
          setPriority(Thread.MIN_PRIORITY);
          for (int i = 0; i < 10; i++) {
               int times = 0;
               while (!cubbyhole.put(i)) { times++; }
               System.out.println("Producteur\ta mis\t"
                                  + i + "\t" + times);
          }
     }
}
```

Classe `Consumer`

```
public class Consumer extends Thread {
     private CubbyHole cubbyhole;

     public Consumer(CubbyHole c) {
          cubbyhole = c;
     }

     public void run() {
          setPriority(Thread.MAX_PRIORITY);
          int value = 0;
          for (int i = 0; i < 10; i++) {
               int times = -1;
               do {
                    value = cubbyhole.get();
                    times ++;
               } while (value == -1);
               System.out.println("Consommateur\tprend\t"
                                  + value + "\t" + times);
          }
     }
}
```


Classe `CubbyHole`

```
public class CubbyHole {
     private int contents;
     private boolean available = false;

     public int get() {
          if (available == false)
               return -1;
          available = false;
          randomSleep();
          return contents;
     }

     public boolean put(int value) {
          if (available == true)
               return false;
          available = true;
          randomSleep();
          contents = value;
          return true;
     }

     public static void randomSleep() {
          Thread.sleep((int)(Math.random() * 10));
     }

}
```

Classe `ProducerConsumerTest`

```
public class ProducerConsumerTest {
  public static void main(String[] args) {
    CubbyHole c = new CubbyHole();
    Producer p1 = new Producer(c);
    Consumer c1 = new Consumer(c);

    System.out.println("Thread actif\tAction\tObjet\tCycles d'attente");
    System.out.println(
         "--------------------------------------------------");

    p1.start();
    c1.start();
  }
}
```

TP2 java – programmation graphique

Programmation graphique

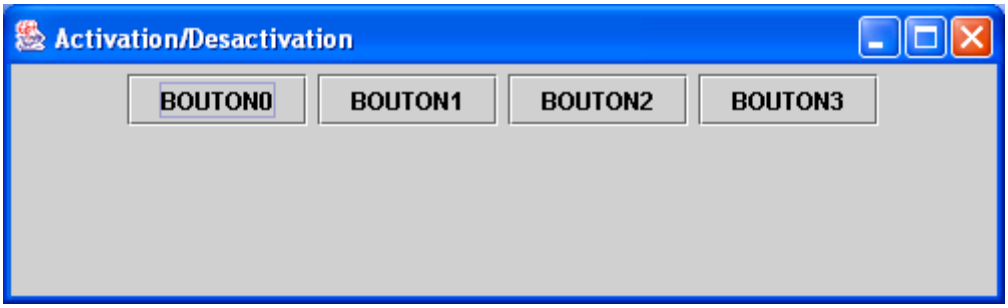
Sujet :

Le sujet se trouve dans le repertoire :
ens-unix $ /export/home/mmallem/ii74/tp2

Question 1 :

Le programme " boutons.java " donné comme base, et étudié en cours, permet de créer deux boutons, en utilisant la librairie swing de java 2 et de mettre en place un écouteur sur ces derniers.

Il s'agit d'étudier, de comprendre et de commenter ce programme d'une part et de le modifier d'autre part. La modification consiste à le généraliser afin de créer plusieurs boutons(quatre comme exemple), de les identifier, de les désactiver suite à chaque clic sur chacun d'eux.
Vous pouvez constater le resultat attendu en executant le fichier " bouttons2.class ".



Question 2 :

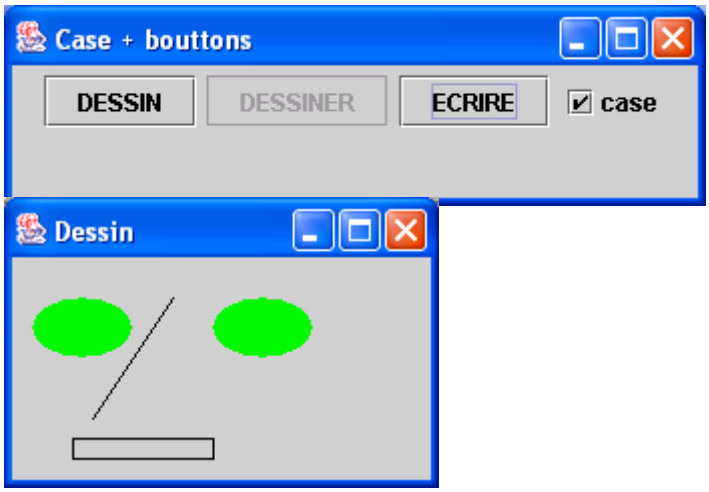
Le programme " cases1.java " donné comme base permet de créer une case à cocher et d'installer l'écouteur correspondant et un bouton donnant l'état de la case à cocher.

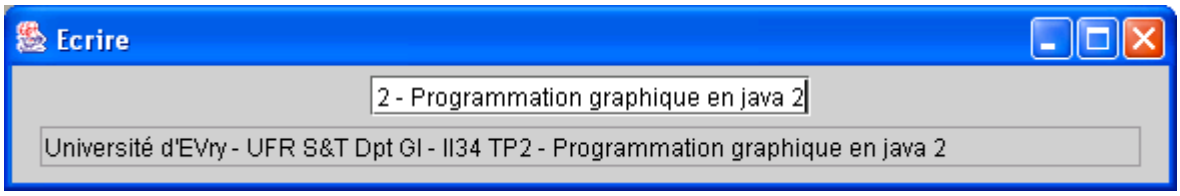
Il s'agit d'étudier, de comprendre et de commenter ce programme d'une part et de le modifier d'autre part. La modification consiste à le combiner avec votre programme, solution de la question 1. En effet la case à cocher, à l'état " true ", servira à rendre actifs tous les boutons quelque soit leur état.
Vous pouvez constater le resultat attendu en executant le fichier " case_bouttons.class ".

Question 3 :

Le programme " dessins.java " donné comme base, et étudié en cours, permet de créer des dessins 2D en java 2.

Il s'agit d'étudier, de comprendre et de commenter ce programme d'une part et de le modifier d'autre part. La modification consiste à le combiner avec votre programme, solution de la question 2. En effet les bouttons serviront à représenter des ordres d'un dessin 2D pre existant et de saisie de texte.
Vous pouvez constater le resultat attendu en executant le fichier " case_bouttons_dessins.class ".
N.B. : Etudier la classe JTextField utilisant les interfaces ActionListener et FocusListener

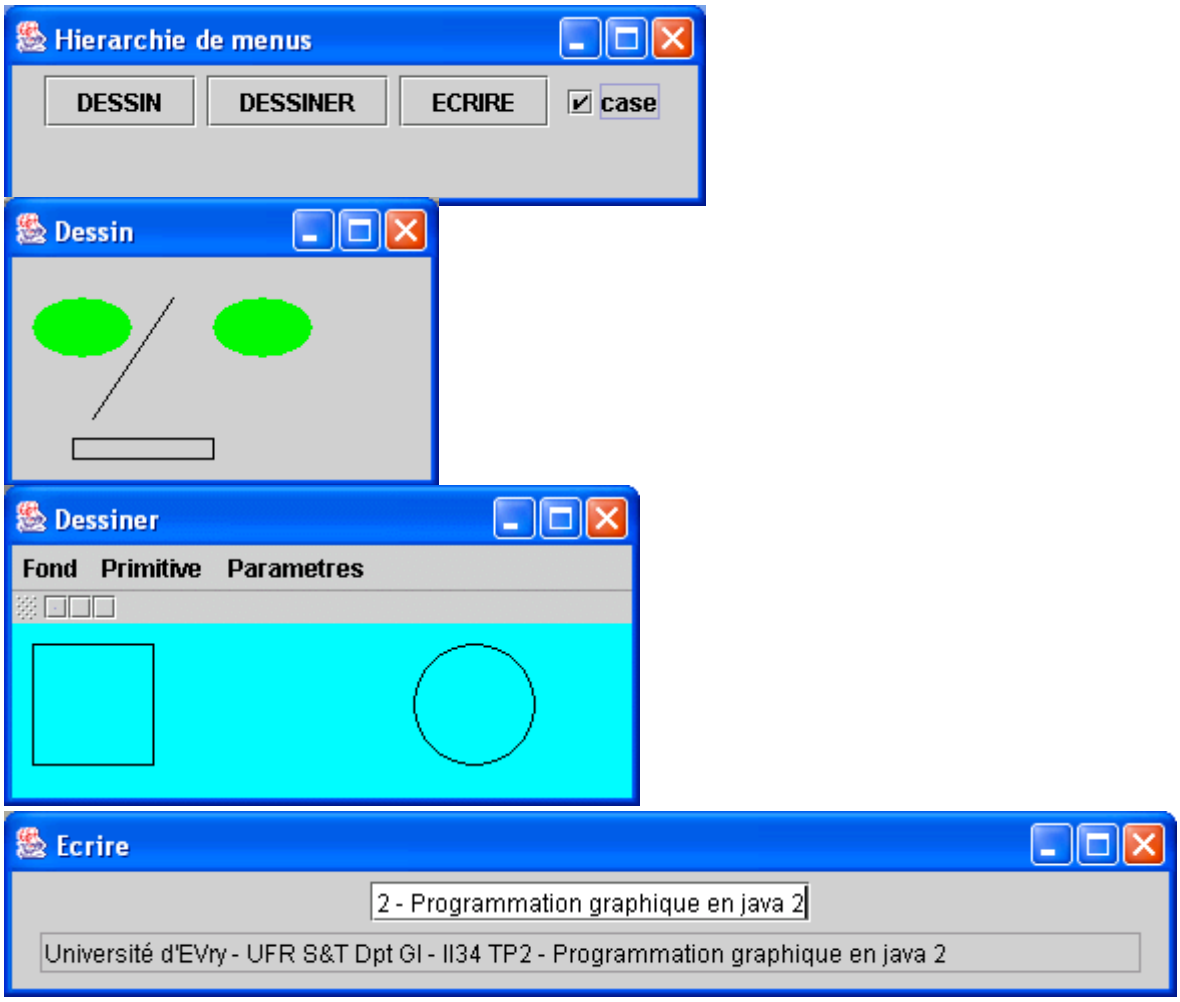




Question 4 :

Il s'agit de modifier votre programme, solution de la question 3. En effet les bouttons serviront à représenter des ordres d'un dessin 2D pre existant, de saisie de texte et de dessin de primitives 2D parametrables. Vous pouvez constater le resultat attendu en executant le fichier " case_bouttons_dessiner.class ".

N.B. : Etudier notamment les classes :
JMenuBar,JMenu,JMenuItem,JCheckBoxMenuItem,JPopupMenu,couleurSurg,JToolBar et ActionCouleur.



TP3 java : Programmation graphique et Applet

Sujet :

Le sujet se trouve dans le repertoire :
ens-unix $ /export/home/mmallem/ii74/tp3

Question 1 :

Il s'agit de modifier l'application Boutons2.java réalisée dans le TP précédent (q1tp2) en une applet JAVA (Boutons_Applet.java) en utilisant la librairie *JApplet* de swing ainsi que les méthodes de bases *init(), start(), stop() et destroy()* gérant la vie d'une applet.

1.1 Construisez un fichier **index1.html** nécessaire pour le chargement de l'applet Boutons2_Applet (cette classe étant fournie pour constater le résultat attendu).
1.2 Exécutez « appletviewer index1.html » et utilisez également votre Navigateur WEB pour charger le fichier index1.html. Commentez le résultat.
1.3 Transformez l'application Bouton2.java en une applet Boutons_Applet.java puis la compiler. Exécutez « appletviewer index1.html » après avoir apporté les modifications nécessaires.
1.4 Surchargez les méthodes *init(), start(), stop() et destroy()* en affichant des messages prouvant leur implications dans le cycle de vie de l'applet. Vous pouvez le vérifier en utilisant les différentes possibilités offertes par le menu de votre appletviewer une fois lancé (Arrêter, Recharger, Fermer etc.)

Notes : Le fichier Boutons2_Applet.class n'est pas exécuté par l'interpréteur `java.exe` (lequel appelle toujours la méthode principale `main()`). C'est l'interpréteur Java **intégré dans le navigateur** (ou dans un visualisateur d'applet tel que **appletviewer**) qui prend en charge l'exécution en chargeant le fichier repéré dans la balise <APPLET …> et qui doit se trouver obligatoirement dans l'attribut **`CODE`** (éventuellement en le cherchant à l'URL indiquée par **`CODEBASE`**) dans le fichier **index1.html**.

Question 2 :

Il s'agit de transformez l'application case_boutons.java (réalisée dans la question 2 tu TP2)
en une applet JAVA et d'utiliser le passage de paramètres entre le fichier HTML et le programme java.

2.1 Construisez un fichier index2.html nécessaire pour le chargement de l'applet case_bouton_Applet (cette classe étant fournie pour constater le résultat attendu).
Ajoutez les balises permettant de définire 3 paramètres INIT, START et STOP ayant comme valeur II34, POO et IHM respectivement.
2.2 Exécutez **appletviewer index2.html** et commentez le résultat.
2.3 L'obtention des paramètres souhaités se fera par une méthode que vous ajouterez qui sous la forme (public String Obtenir_Parametre(String s)). En entrée, vous lui donnerez le nom du paramètre et elle vous retourne la valeur de ce paramètre.

Question 3 :

Poursuite du TP2 et Transformation de l'application « case_bouttons_dessins» en applet JAVA

Question 4 :

Poursuite du TP2 et Transformation de l'application « case_bouttons_dessiner» en applet JAVA

Annexe

```
/* Explications sur les méthodes de la classe JApplet
    Une applet possède un comportement standard, un "cycle de vie" par défaut.
```

Ceci grâce à 4 méthodes prédéfinies init(), start(), stop() et destroy().
Ces méthodes standard, automatiquement appliquées à un objet qui hérite de
la classe Applet ou JApplet permettent de contrôler son exécution.
Ces mèthodes peuvent être adaptées (surchargées) par le programmeur.

****Initialisation de l'applet : public void init() { ..}
Le navigateur lance cette méthode au chargement de l'applet (état construit).
Cette méthode a pour rôle de récupérer des informations telles que images ou sons,
ainsi que les valeurs des paramètres passés dans la balise <applet ...>.
Ainsi, la méthode init() joue souvent le rôle de constructeur. L'applet se trouve
ensuite dans l'état initialisée.

****Démarrage : public void start() {...}
Après avoir été initialisée, l'applet est démarrée, grâce à la méthode start(), et
se trouve alors dans l'état actif Alors que l'initialisation n'a lieu qu'une fois
dans la vie d'une applet, il peut y avoir des appels répétés à la méthode start().
En effet, une applet est redémarrée après avoir été stoppée par la méthode stop().

****Arrêt : public void stop() {...}
Cette méthode est appelée lorsque la page HTML n'est elle-même plus active; l'applet
se trouve alors dans l'état inactif.Par exemple, parce que l'utilisateur a quitté la page
qui contient l'applet ou l'a redimensionnée..A noter que la redéfinition de stop() est
nécessaire pour permettre de suspendre l'exécution des threads.

****Destruction : public void destroy() {...}
L'applet est détruite lorsque l'on sort du navigateur, ou avant que l'applet ne soit
rechargée. L'applet passe d'abord par l'état inactif par stop(), avant d'être détruite.
*/