DEUST Systèmes d'information et réseaux, gestion et réalisation
2ème année

Langage à objets : JAVA

2021-2022

1. Les applications Java

Compilation / exécution

Considérons une application en Java enregistrée dans un fichier nommé Test.java.

Compilation : …….
La compilation génère un ou plusieurs fichiers d'extension …….
Exécution : ……

Attributs et méthodes

Exemple 1

Considérons l'application suivante. Elle doit être enregistrée dans un fichier nommé ……

Compilation : ……
Fichiers générés : ……
Execution : …….

```
class ObjetTableau{
    int[] t;

   ObjetTableau(int a){
          t=new int[3];
         for (int i=0;i<3;i++)
             t[i]=a+i;
      }

   void affiche(){
      System.out.println("Affichage d'un objet");
       for (int i=0;i<3;i++)
            System.out.print(t[i]+" ");
       System.out.println();
      }
   }

public class ProgObjet1{
   public static void main(String args[]){

          ObjetTableau T1=new ObjetTableau(0);
          T1.affiche();

          ObjetTableau T2=new ObjetTableau(10);
          T2.affiche();
   }
}
```

Que va t-on obtenir à l'écran lors de l'exécution de cette application ?

T1 et T2 sont des **instances** de la classe……..

t : attribut ……..…….

affiche() : méthode ……..……

ObjetTableau() : ……..…….de classe. C'est le ……..……...

ProgObjet1 : **classe principale** (ou classe maître). La classe principale contient le *main* et donne son nom au fichier.

Exemple 2

Considérons l'application suivante. Elle doit être enregistrée dans un fichier nommé ……

Compilation : ……
Fichiers générés : ……
Execution : …….

```
class ObjetTableau{
   static int nb=0;
   int[] t;

   ObjetTableau(int a){
          t=new int[3];
         for (int i=0;i<3;i++)
             t[i]=a+i;
         nb++;
     }

   void affiche(){
     System.out.println("Affichage d'un objet");
     for (int i=0;i<3;i++)
         System.out.print(t[i]+" ");
     System.out.println();
     }

   static void nbObjets(){
      System.out.println("Nb d'objets :"+nb);
      }
   }

public class ProgObjet2{
   public static void main(String args[]){
          ObjetTableau.nbObjets();

          ObjetTableau T1=new ObjetTableau(0);
          T1.affiche();

          ObjetTableau.nbObjets();

          ObjetTableau T2=new ObjetTableau(10);
          T2.affiche();

          ObjetTableau.nbObjets();
   }
}
```

Que va t-on obtenir à l'écran lors de l'exécution de cette application ?

nb : attribut …….(signalé par *static*).

nbObjets : méthode…….(signalée par *static*).

La **classe principale** est ……..……

TD1 : commandes MS-DOS

Indiquer pour chaque action la commande MS-DOS correspondante.

1. afficher le contenu du dossier courant : ……..……
2. aller dans le répertoire TEST : ……..……
3. revenir au dossier « parent » : ……..……
4. créer le dossier EXEMPLE (make directory) : ……..……
5. effacer le fichier Ex.java : ……..……
6. effacer le dossier EXEMPLE (remove directory) : ……..……
7. renommer le fichier Projet.java en Appli.java : ……..……
8. copier le fichier Ex1.java en Ex2.java : ……..……

2. Les paquetages Java

Qu'est-ce qu'un paquetage ?

Les paquetages (packages) sont des librairies de **classes,** de classes d'**exceptions** et d'**interfaces** regroupées selon leur fonction, qui sont fournies en même temps que le compilateur javac et l'interpréteur java.

La documentation concernant chaque paquetage est accessible : **http://docs.oracle.com/javase/8/docs/api**

java.lang

Le paquetage **java.lang** est le **seul** paquetage dont l'emploi ne doit jamais être déclaré.
Il comprend notamment la classe **Object**, la classe **System**, les **classes enveloppantes**, la classe **Math** et la classe **String** (classe des chaînes de caractères).

La classe **Object** est à l'origine de toutes les classes. Elle contient une méthode d'instance appelée **toString()** dont hérite donc tout objet. Cette méthode permet de convertir (au mieux) l'objet en une chaîne de caractères affichable.

La classe **System** contient des variables et des méthodes du système. Quand on écrit `System.out.println(...)`, on fait appel à la **variable de classe** appelée **out** de la classe **System**.

Les classes enveloppantes : les variables des types de données primitifs (int, char, double, etc) sont les seuls éléments qui ne sont **pas** des objets dans Java. Mais on peut, en cas de besoin, les transformer en objets à l'aide de classes enveloppantes suivantes :

type	nature	classe enveloppante
boolean	true ou false	Boolean
char	caractère 16 bits	Character
byte	entier 8 bits signé (petit entier)	Byte
short	entier 16 bits signé (entier court)	Short
int	entier 32 bits signé (entier)	Integer
long	entier 64 bits signé (entier long)	Long
float	virgule flottante 32 bits (réel simple)	Float
double	virgule flottante 64 bits (réel double)	Double

Exemples d'utilisation :

```
int n = 5 ;                              // n n'est pas un objet
Integer obj1 = new Integer(n) ;          // obj1 est un objet

double f = 1/3 ;                         // f n'est pas un objet
Double obj2 = new Double(f) ;            // obj2 est un objet
```

La classe **Math** contient toutes les **méthodes de classe** nécessaires pour effectuer tous les calculs mathématiques.
Exemples :

```
double j = -3.145 ;
double k = Math.abs(j) ;                 // valeur absolue
double l = Math.pow(j,k) ;               // j exposant k
double m = Math.sqrt(l) ;                // racine carrée de l
```

La classe **String** contient de très nombreuses **méthodes d'instance** pour le traitement des chaînes de caractères ainsi qu'une **méthode de classe** pour les conversions : **valueOf.**

java.util

Le paquetage java.util contient des classes utilitaires, notamment :
Calendar, **GregorianCalendar** : pour les dates
Random : pour les nombres aléatoires

Pour utiliser une partie d'un paquetage autre que java.lang, on utilise le mot-clé **import** :

- `import java.util.GregorianCalendar ;`
  on se donne le droit d'utiliser la classe GregorianCalendar (qui permet de créer des objets de type date (jour-mois-année-heure-minute-seconde…)
- `import java.util.Random ;`
  on se donne le droit d'utiliser la classe Random
- `import java.util.Vector ;`
  on se donne le droit d'utiliser la classe Vector
- `import java.util.* ;`
  on se donne le droit d'utiliser toutes les classes du paquetage

La notion d'héritage

On déclare qu'une classe dérive d'une autre classe par :

```
class classe-fille extends classe-mère {
        …
        }
```

Si une classe B dérive d'une classe A, elle hérite des champs et des méthodes de A.

Un objet de la classe fille est donc un objet de la classe mère (éventuellement complété par des champs et des méthodes supplémentaires), mais la réciproque est fausse.

Toutes les classes héritent de la classe **Object**. Si on précise pas de clause d'héritage dans la définition d'une classe, par défaut, elle hérite de Object.

Les méthodes héritées d'une classe mère A peuvent être redéfinies dans la classe fille B. Cette possibilité de redéfinir les méthodes est appelée le **polymorphisme** : un objet de type A et un objet de type B sont capables de répondre au même message, mais en faisant appel à des méthodes de contenus différents.

Protection des attributs et méthodes

Les attributs et méthodes peuvent être déclarés **private**, **public**, **protected** ou ne pas être précédés de déclaration de protection.

Exemple :

```
class Truc {
        private int a ;
        protected int b ;
        public int c ;
        int d ;
        }
```

Les attributs et méthodes déclarés **private** ne sont accessibles qu'à l'intérieur de leur propre classe
→ a ne sera donc accessible qu'aux méthodes de la classe Truc.

Les attributs et méthodes déclarés **protected** ne sont accessibles qu'aux sous-classes (qu'elles soient dans le même paquetage ou pas) et aux classes du même paquetage
→ b sera donc accessible aux sous-classes de Truc et aux classes du même paquetage que Truc.

Les attributs et méthodes déclarés **public** sont toujours accessibles
→ c sera donc toujours accessible

Les attributs et méthodes **sans aucune déclaration de protection** sont accessibles par toutes les classes du même paquetage. C'est le mode de protection par défaut
→ d sera donc accessible aux classes du même paquetage que Truc.

Les interfaces

Une interface est une sorte de classe spéciale, dont toutes les méthodes sont déclarées public et abstract (sans code). En fait, une interface **propose des services**, mais sans donner le code permettant d'exécuter ces services.

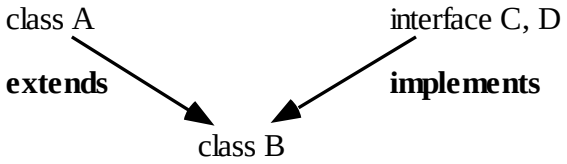
Une classe qui souhaite utiliser une interface le signale par le mot **implements** :

```
class B implements C {…} ;
```

On dit alors qu'elle **implémente** l'interface. Dans la classe B, on doit alors donner le code de **toutes** les méthodes de l'interface.

Une classe B peut hériter d'une classe A et implémenter une ou plusieurs interfaces.

```
class B extends A implements C, D {…} ;
```



Les méthodes des interfaces C et D doivent alors **obligatoirement** être définies dans la classe B.

Les classes d'exceptions

Lorsqu'une méthode est exécutée, il peut se produire des erreurs. On dit alors qu'une **exception** est **levée**. Une exception est un objet, instance d'une des nombreuses sous-classes de la classe **Exception**.

Chaque méthode qui risque de provoquer une ou des erreur(s) le signale par le mot **throws** suivi de la nature des erreurs susceptibles de se produire.

Exemple **:** dans la classe FilterInputStream, une méthode de lecture d'un fichier est déclarée sous la forme :

public int read() **throws IOException**

Cela signale que cette lecture peut provoquer une erreur d'entrée-sortie (Input/Output Exception).

Lorsqu'on a besoin d'utiliser une méthode risquant de provoquer des erreurs, on englobe l'appel à cette méthode dans un bloc **try** (pour capturer l'erreur) et on précise ce qu'on fera en cas d'erreur dans une clause **catch** (pour traiter l'erreur).

Exemple :

```
try       {
          fichier.read() ;
          }
catch (IOException) {
          System.out.println(" erreur de lecture ") ;
          return ;
          }
```

Les différentes classes d'erreurs possibles sont spécifiées dans chaque paquetage.

TD2 : la classe GregorianCalendar

Utiliser la documentation pour compléter le fichier UtilDate.java du dossier UTILDATE. Cette application permet d'afficher à l'écran « Nous sommes le ../../…. et il est ..h..min ».

3. Le paquetage graphique java.awt

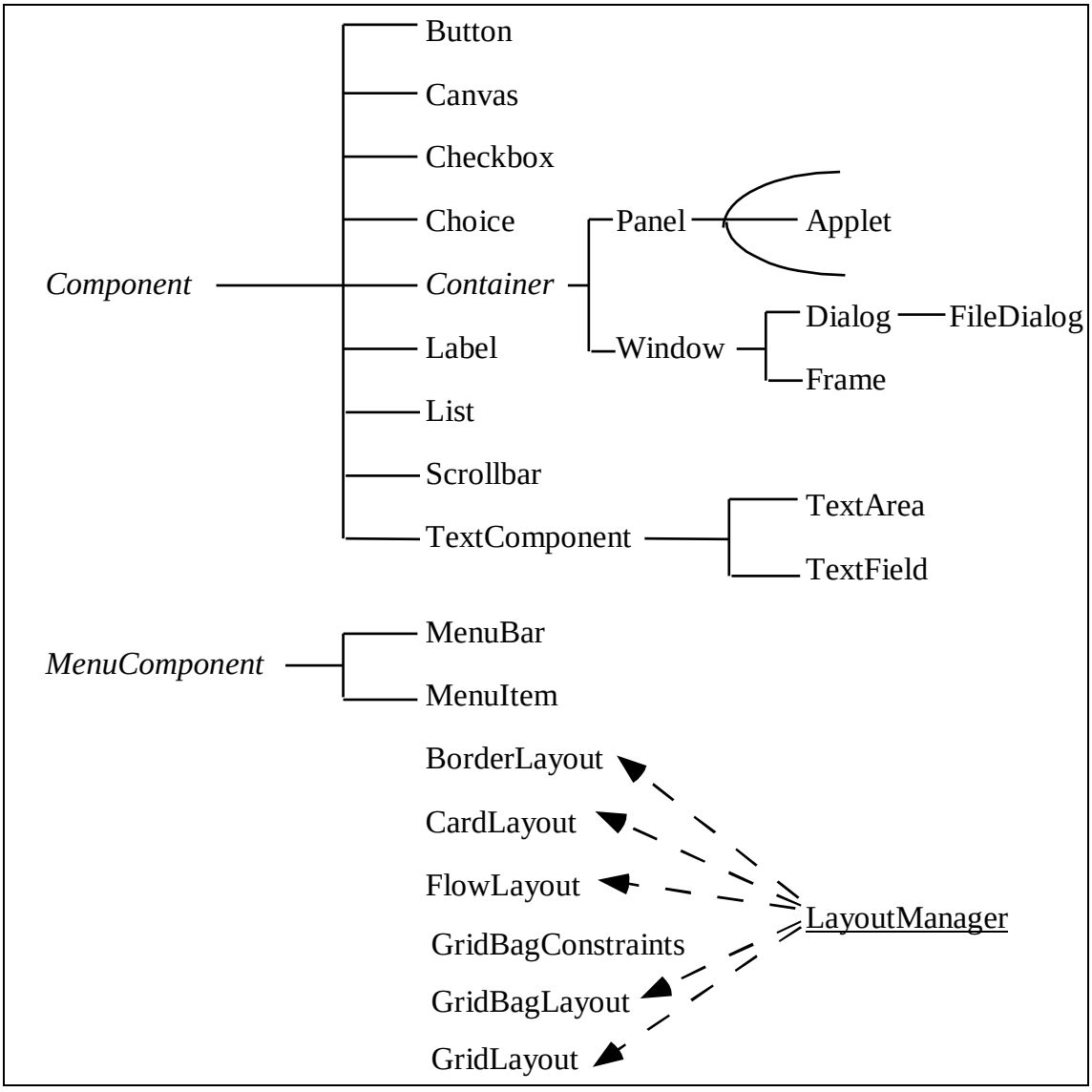
Le paquetage **java.awt** fournit toutes les classes permettant de bâtir des interfaces graphiques.

La classe **Component** est à l'origine de la plupart des classes graphiques.

En italique, les classes abstraites (qui n'autorisent pas la création d'instances). La classe Component est ainsi une classe abstraite ; son rôle est de proposer des méthodes qui seront héritées par tous les objets de ses sous-classes.

Dans la parabole, ce qui est extérieur au paquetage java.awt, mais dérive d'une classe de java.awt.

En flèches pointillées, les implémentations d'interfaces.



Principe de construction d'une interface graphique

Les composants (fenêtres, boutons, cases à cocher, etc) sont insérés dans un objet dérivant de la classe **Container**, qui groupe les composants et les dispose pour l'affichage.

Container contient **Panel** et **Window** ; les objets de type Window sont des fenêtres simples tandis que les objets de type **Frame** servent à définir des fenêtres avec scrollbar, titre, etc. Les fenêtres de type **Dialog** définissent les fenêtres de dialogue avec l'utilisateur.

Chaque container possède un **LayoutManager** (gestionnaire de placement), qui définit comment sont placés les objets qu'il contient. **LayoutManager** est en fait une interface qu'implémentent plusieurs classes, dont :

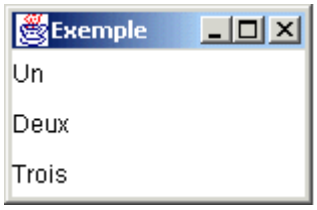
- **BorderLayout**, qui répartit les objets au Nord, Sud, Est, Ouest et Centre (c'est l'option par défaut pour un Frame)
- **FlowLayout** qui les répartit de gauche à droite puis passe à la ligne suivante (c'est l'option par défaut pour un Panel)
- **GridLayout** qui les répartit dans un tableau dont toutes les cases ont la même taille.

Exemples

Appli1.java

```
import java.awt.* ;
class Appli1 {
        public static void main(String arg[]) {
                Frame w =  new Frame("Exemple") ;
                w.add("North", new Label("Un")) ;
                w.add("Center", new Label("Deux")) ;
                w.add("South", new Label("Trois")) ;
                w.pack() ;
                w.setVisible(true);
                }
        }
```

BorderLayout par défaut :



Appli2.java

```
import java.awt.* ;
class Appli2 {
        public static void main(String arg[]) {
                Frame w =  new Frame("Exemple") ;
                w.setLayout(new FlowLayout()) ;
                w.add(new Label("Un")) ;
                w.add(new Label("Deux")) ;
                w.add( new Label("Trois")) ;
                w.pack() ;
                w.setVisible(true);
                }
        }
```

FlowLayout choisi :

Appli3.java

```
import java.awt.* ;
class Appli3 {
        public static void main(String arg[]) {
                Frame w =  new Frame("Exemple") ;
                w.setLayout(new GridLayout(2,2)) ;
                w.add(new Label("Un")) ;
                w.add(new Label("Deux")) ;
                w.add( new Label("Trois")) ;
                w.add( new Label("Quatre")) ;
                w.pack() ;
                w.setVisible(true);
                }
        }
```

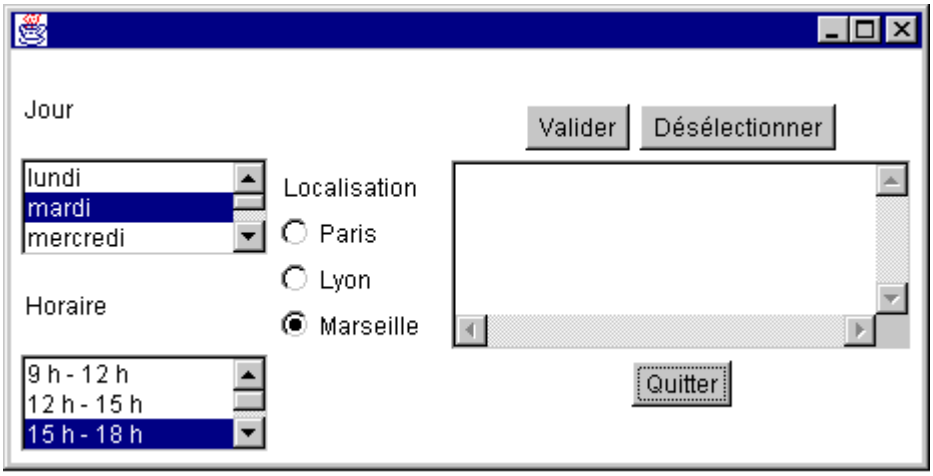
GridLayout choisi :

Remarque :
Pour afficher un objet **w** de la classe **Frame** on écrit : w.setVisible(true);
A la place de la 2e instruction, certains programmes utilisent : w.show() ;
Cette instruction, dépréciée, va générer un avertissement à la compilation.

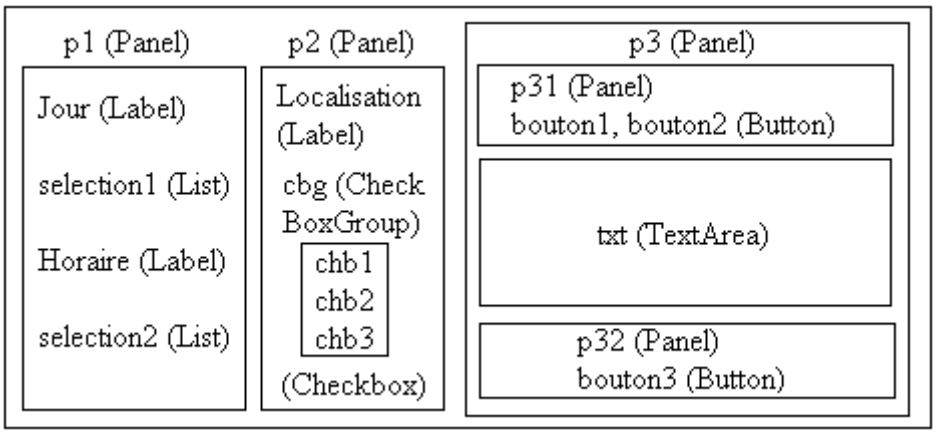
Exemple de construction d'interface

Un objet de la classe **Panel** est un **Container** puisque la classe Panel dérive de la classe Container.
Il peut donc lui-même comporter des composants, dont d'autres Panel.
Chaque Panel gère alors ses propres composants avec le LayoutManager qui lui est propre.

On va construire l'interface suivante :



correspondant au schéma :



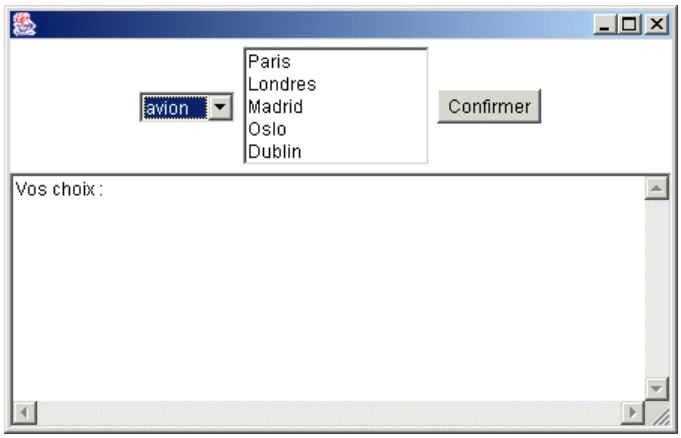

```
import java.awt.* ;
class Fenetre extends Frame {
        protected Panel p1, p2, p3, p31, p32 ;
        protected TextArea txt ;
        protected Button bouton1, bouton2, bouton3 ;
        protected CheckboxGroup cbg ;
        protected Checkbox chb1, chb2, chb3 ;
        protected List selection1, selection2 ;

        Fenetre() {
                setLayout(new FlowLayout()) ;

                // construction du Panel p1
                p1 = new Panel() ; p1.setLayout(new GridLayout(4,1)) ;
                p1.add(new Label("Jour")) ;
                selection1 = new List(3) ;
                selection1.addItem("lundi") ; selection1.addItem("mardi") ;
                selection1.addItem("mercredi") ; selection1.addItem("jeudi") ;
                selection1.addItem("vendredi") ; selection1.addItem("samedi") ;
```

```
                selection1.addItem("dimanche") ;
                p1.add(selection1) ;
                p1.add(new Label("Horaire")) ;
                selection2 = new List(3) ;
                selection2.addItem("9 h - 12 h") ; selection2.addItem("12 h - 15 h") ;
                selection2.addItem("15 h - 18 h") ; selection2.addItem("18 h - 21 h") ;
                p1.add(selection2) ;
                add(p1) ;

                // construction du Panel p2
                p2 = new Panel() ;
                p2.setLayout(new GridLayout(4,1)) ;
                p2.add(new Label("Localisation")) ;
                cbg = new CheckboxGroup();
                chb1 = new Checkbox("Paris", cbg, true);
                p2.add(chb1) ;
                chb2 = new Checkbox("Lyon", cbg, false);
                p2.add(chb2) ;
                chb3 = new Checkbox("Marseille", cbg, false);
                p2.add(chb3) ;
                add(p2) ;

                // construction du Panel p3
                p3 = new Panel() ; p3.setLayout(new BorderLayout()) ;
                p31 = new Panel();
                bouton1 = new Button("Valider") ; p31.add(bouton1) ;
                bouton2 = new Button("Désélectionner") ; p31.add(bouton2) ;
                p3.add("North", p31) ;
                txt = new TextArea(5,30) ;
                p3.add("Center",txt) ;
                p32 = new Panel() ;
                bouton3 = new Button("Quitter") ; p32.add(bouton3) ;
                p3.add("South", p32) ;
                add(p3) ;
                }                                       // fin du constructeur
        }                                               // fin de la classe Fenetre

public class UtilFenetre {                              // classe utilisateur
        public static void main(String args[]) {
                Fenetre f = new Fenetre() ;
                f.pack() ;
                f.setVisible(true);
                }
        }
```

Les deux classes constituent le fichier **UtilFenetre.java** construisant l'interface souhaitée (voir dossier UTILFENETRE-INITIALE).

TD 3

On souhaite obtenir l'interface suivante, comportant : un bouton de choix (de la classe **Choice**), une liste (de la classe **List**), un bouton (de la classe **Button**) et un champ de texte (de la classe **TextArea**).



1) Ouvrir avec un éditeur le fichier **UtilAwt.java** qui se trouve dans le dossier **utilawt** :

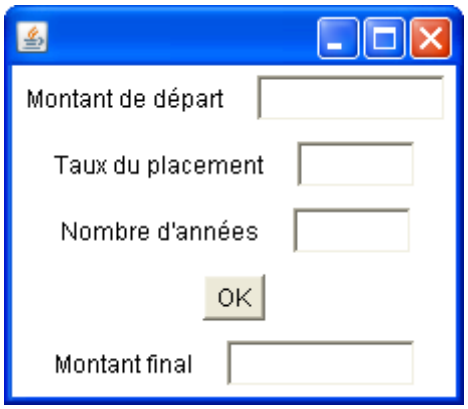
```
import java.awt.* ;
class Fenetre extends Frame {
        protected Panel p ;
        protected Choice c ;
        protected List l ;
        protected Button suite ;
        protected TextArea texte ;
        Fenetre() {                         // le constructeur
                p = new Panel() ;
                c = new Choice() ;
                ...                         // ajout des items dans l'objet Choice
                p.add(c) ;
                l = new List(5) ;
                ...                         // ajout des items dans l'objet List
                p.add(l) ;
                ...                         // création du bouton suite
                p.add(suite) ;
                add("North", p) ;
                texte = new TextArea();
                ...                         // remplissage du champ de texte
                add("South", texte) ;
                }
        }
public class UtilAwt {
        public static void main(String args[]) {
                Fenetre f = new Fenetre() ;
                f.pack() ;
                f.setVisible(true);
        }
}
```

2) Compléter le constructeur Fenetre() afin d'aboutir au résultat escompté.

3) Compiler et exécuter cette application. Pour quitter l'application (l'événement de fermeture de la fenêtre n'étant pas pris en compte pour le moment), revenir dans la fenêtre d'invite de commandes et taper **CRTL C**.

TD4

On veut créer l'interface suivante :



Cette interface comporte :

- un Label pour le texte "Montant de départ"
- un TextField appelé dep pour la saisie du montant de départ
- un Label pour le texte "Taux du placement"
- un TextField appelé tx pour la saisie du taux
- un Label pour le texte "Nombre d'années"
- un TextField appelé nb pour la saisie du nombre d'années
- un Button b pour valider
- un Label pour le texte "Montant final"
- un TextField appelé res pour l'affichage du montant final

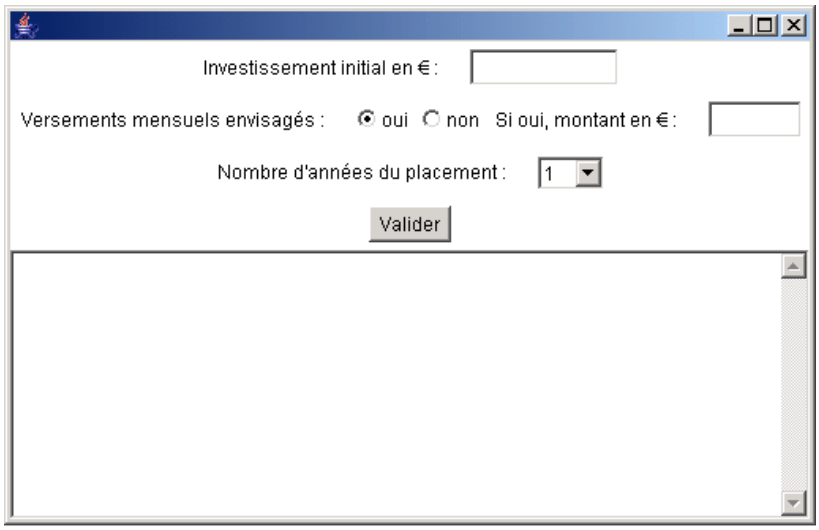
qui sont rangés dans des Panel pour améliorer la présentation.

Ouvrir le fichier **Placement.java** du dossier **Placement** et remplir les Panel p1, p2, p3, p4, p5 afin d'arriver au résultat souhaité :

```
import java.awt.* ;
class MonPlacement extends Frame {
        protected Panel p, p1,p2,p3,p4,p5 ;
        protected Button b ;
        protected TextField dep, tx, nb, res ;
        MonPlacement () {
                p = new Panel() ;
                p.setLayout(new GridLayout(5, 1)) ;
                p1=new Panel();
                ...
                p.add(p1);

                p2=new Panel();
                ...
                p.add(p2) ;

                p3=new Panel();
                ...
                p.add(p3);

                p4=new Panel();
                ...
                p.add(p4);

                p5=new Panel();
                ...
                p.add(p5) ;
                add("Center", p) ;
                }
        }
public class Placement {
        public static void main(String[] args) {
                MonPlacement f = new MonPlacement() ;
                f.pack() ;
                f.setVisible(true);
                }
        }
```

TD5

Créer un **dossier** appelé **MAFENETRE** puis créer le fichier **MaFenetre.java** générant l'interface suivante :



Cette interface comporte :

- un Label pour le texte "Investissement initial en €"
- un TextField (nommé m1) pour la saisie du montant initial
- un Label pour le texte "Versements mensuels envisagés"
- un CheckboxGroup pour regrouper les deux Checkbox "oui" et "non" (nommé cbg)
- les Checkbox "oui" et "non" (nommés chb1 et chb2)
- un Label pour le texte "Si oui, montant en €"
- un TextField (nommé m2) pour la saisie du montant du versement mensuel
- un Label pour le texte "Nombre d'années du placement"
- un Choice (nommé c) pour la durée du placement (proposer de 1 à 10 ans)
- un Button (nommé ok) pour valider
- un TextArea (nommé texte) pour le résultat

ces objets étant rangés dans des Panel pour obtenir une présentation correcte de l'interface.

Compiler et exécuter ensuite le programme.

4. La gestion des événements

Principe de la gestion des événements

L'interface graphique délègue la gestion des événements à des classes extérieures. Les traitements consécutifs aux événements sont ainsi totalement séparés de l'interface elle-même.

Modèle MVC (METHODE - VUE - CONTROLEUR)
Interface = VUE (instance d'une classe)
Prise de décision = CONTROLEUR (instance d'une autre classe)
Action = METHODE (instance d'une troisième classe)

Mise en œuvre en Java

Une interface graphique contient plusieurs objets. On va attacher un **listener** (surveillant) aux objets graphiques que l'on souhaite particulièrement surveiller (Button, List, Window, Canvas …)

Listener étant une famille d'**interfaces**, le listener que l'on va créer sera nécessairement un objet d'une classe implémentant une de ces interfaces.

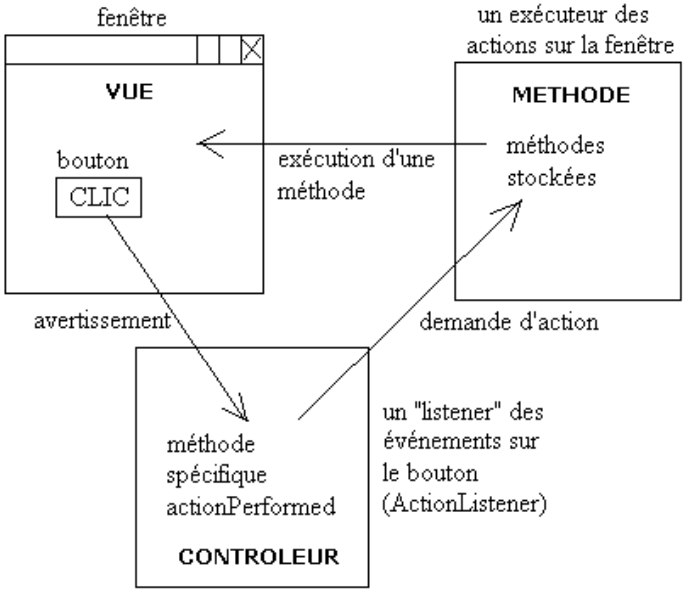
On appellera cette classe la classe **Adaptateur**, car elle sera chargée d'adapter le comportement de l'interface aux actions effectuées par l'utilisateur.

L'objet de la classe Adaptateur, étant devenu un **listener**, sera alors automatiquement averti des événements (**event**) qui se produisent sur les objets surveillés.
Il décidera alors des actions à entreprendre en fonction de la demande de l'utilisateur, et donnera l'ordre à un **exécuteur** d'effectuer ces actions.

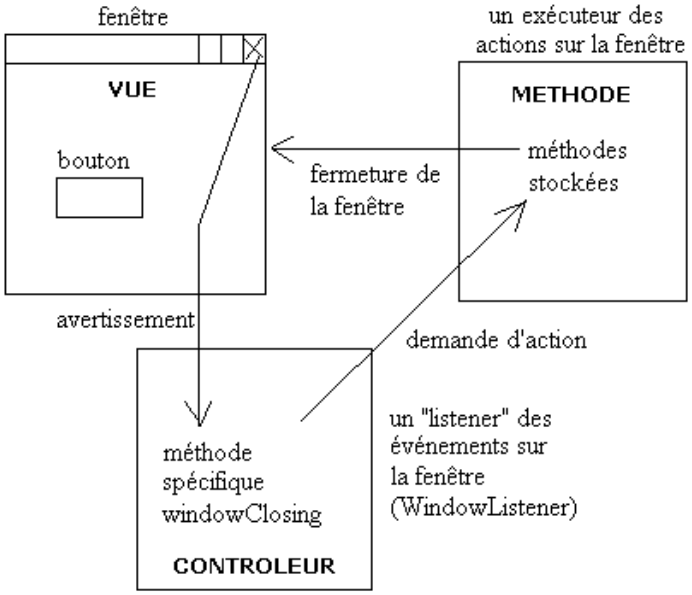
L'**exécuteur** sera un objet d'une classe que l'on appellera classe **Delegue** ; il exécutera les actions demandées par le listener et interagira avec l'interface graphique pour en modifier l'apparence.

Illustration par deux exemples

Clic sur un bouton contenu dans l'interface



Clic sur le bouton de fermeture de la fenêtre



Programmation

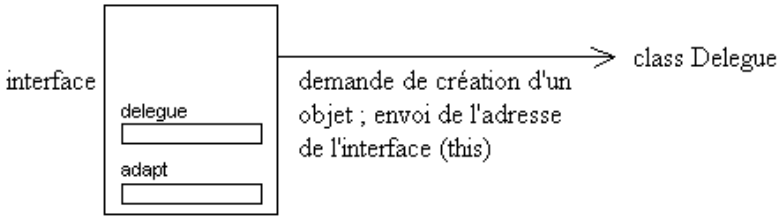
La VUE est l'interface. C'est une instance d'une classe graphique.

Le CONTROLEUR est une instance d'une classe implémentant les Listeners nécessaires ; **on appellera cette classe la classe Adaptateur** : le CONTROLEUR décide comment « adapter » l'apparence de l'interface aux actions demandées par l'utilisateur sur la VUE.

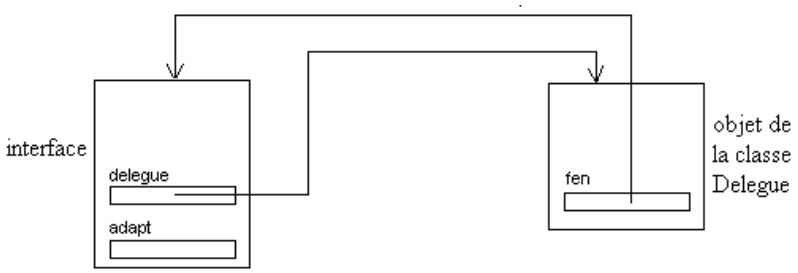
La METHODE est une instance d'une classe contenant toutes les actions à réaliser sur l'interface ; **on appellera cette classe la classe Delegue** : le CONTROLEUR « délègue » à la METHODE la responsabilité des actions à entreprendre.

1) Dans le constructeur de l'interface, on déclare un attribut de nom delegue (de type Delegue) puis on demande à la classe Delegue la création d'un objet Delegue en lui envoyant l'adresse de l'interface :

```
protected Delegue delegue ;
delegue = new Delegue(this) ;          // this = adresse de l'interface
```



L'adresse du délégué ainsi créé est donc stockée dans l'attribut delegue de l'interface.
De son côté, le délégué stocke dans un de ses attributs (appelé ici fen) l'adresse de l'interface pour pouvoir ultérieurement agir sur elle :



Analogie avec le courrier :
Nous écrivons à un destinataire que nous venons de découvrir (à une adresse que nous gardons en mémoire) et nous lui envoyons notre propre adresse (this) pour qu'il puisse lui-même la stocker et nous répondre plus tard.

2) On définit **une classe Adaptateur** qui "écoute" les événements et décide des actions à faire accomplir par le délégué. Pour cela, la classe Adaptateur implémente une ou plusieurs interfaces de la famille des **Listener**.

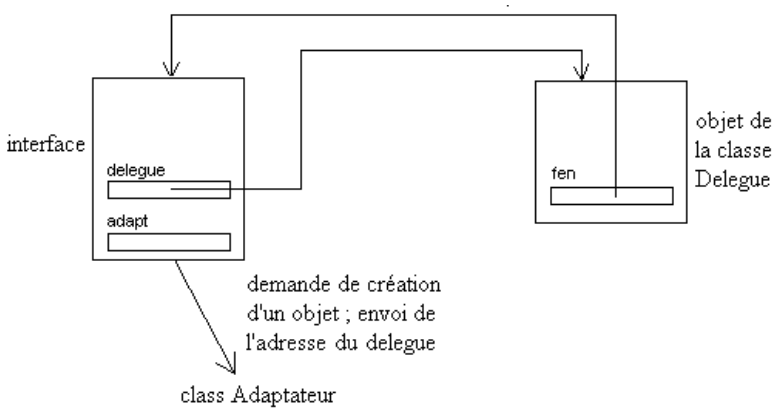
Exemple : pour surveiller les éléments sur un bouton et sur une fenêtre, il faut écrire :

```
class Adaptateur implements WindowListener, ActionListener {
    …
    }
```

puis **définir** obligatoirement dans la classe Adaptateur toutes les méthodes des deux interfaces **WindowListener** et **ActionListener.**

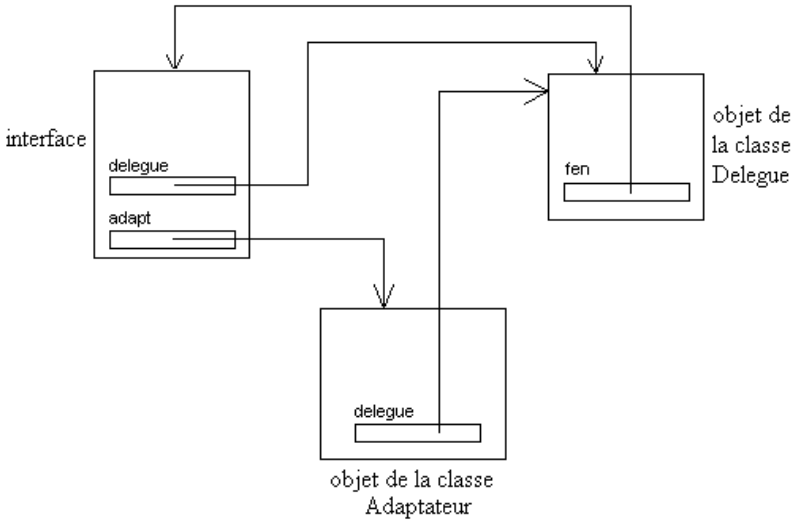
3) On déclare dans le constructeur de l'interface un attribut appelé adapt (de type Adaptateur) et on demande ensuite à la classe Adaptateur de créer un objet de la classe Adaptateur ; on lui envoie l'adresse du délégué, puisque c'est au délégué que l'adaptateur s'adressera :

```
protected Adaptateur adapt ;
adapt = new Adaptateur(delegue) ;
```



Penser encore à **l'analogie avec le courrier** : nous écrivons à un second destinataire (l'Adaptateur adapt), mais nous lui demandons de répondre ultérieurement au premier destinataire (le delegue), dont il n'a pas a priori l'adresse. Il faut donc lui passer cette adresse, pour qu'il puisse la stocker.

L'interface stocke l'adresse de l'adaptateur dans l'attribut adapt, tandis que l'adaptateur stocke celle du délégué dans un attribut delegue :



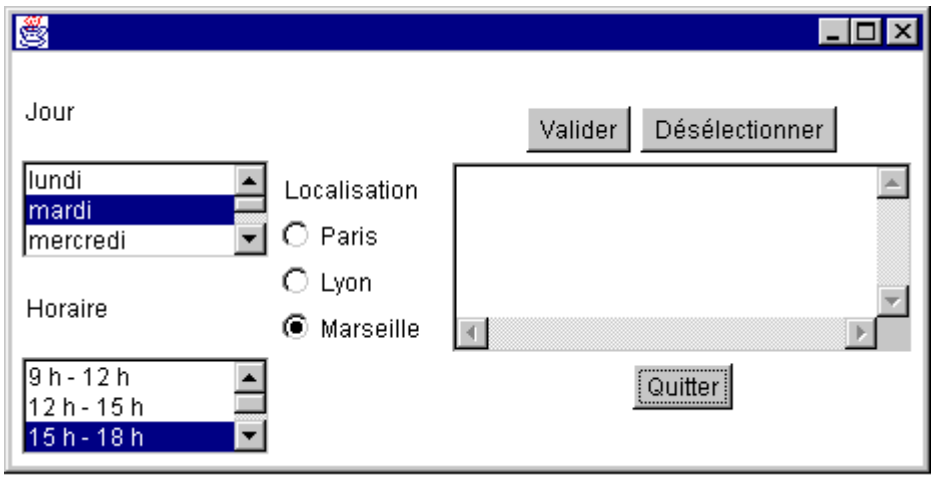
4) Dans le constructeur de l'interface, on "enregistre" l'Adaptateur adapt sur les composants qu'il doit surveiller. Les méthodes d'enregistrement des adaptateurs dépendent des classes de ces composants.
Exemple : si bouton1 est un objet de la classe Button, et si adapt doit surveiller bouton1, on écrira :

```
bouton1.addActionListener(adapt) ;
```

5) On écrit enfin dans la classe Delegue les méthodes nécessaires pour que le délégué puisse effectuer les actions nécessaires sur l'interface.

Application à une interface précédente

Compléter le fichier UtilFenetre.java du dossier UTILFENETRE-FINALE afin de rendre intéractive l'interface suivante.




```
import java.awt.* ;
import …

class Fenetre extends Frame {                    // la classe qui s'occupe de l'interface (la VUE)

        protected Panel p1, p2, p3, p31, p32 ;
        protected TextArea txt ;
        protected Button bouton1, bouton2, bouton3 ;
        protected CheckboxGroup cbg ;
        protected Checkbox chb1, chb2, chb3 ;
        protected List selection1, selection2 ;
        protected Delegue … ;
        protected Adaptateur … ;

        Fenetre() {                …
                // construction des Panel p1, p2 et p3
                …
                // ajouts pour la gestion des événements
                // création du délégué et de l'adaptateur
                delegue = …
                adapt = …

                // enregistrement de l'adaptateur comme listener
                addWindowListener…
                bouton1.addActionListener…
                bouton2.addActionListener…
                bouton3.addActionListener…
                }
        }

// la classe utilisateur
public class UtilFenetre {
        public static void main(String args[]) {
                Fenetre f = new Fenetre() ;
                f.pack() ; f.setVisible(true);
                }
        }
```

```
class Delegue {                                          // partie traitement (la METHODE)
        protected Fenetre fen ;

        Delegue(Fenetre f) {
                fen = f ;
                }
        public void quitter() {
                …
                }
        public void annulle() {
                fen.selection1.deselect…
                fen.selection2.deselect…
                fen.chb1.setState…
                fen.chb2.setState…
                fen.chb3.setState…
                fen.txt.setText…
                }
        public void affichetexte() {
                String res = "Vous avez choisi le " + …
                res=res + "\ndans la tranche horaire " + …
                res=res + "\nà " + …
                fen.txt.setText...
                }
        }

class Adaptateur implements ActionListener, WindowListener {   // aiguillage (le CONTROLEUR)
        protected Delegue delegue ;
        Adaptateur(Delegue d) {
                delegue = d ;
                }

        // méthode indispensable comme ActionListener
        public void actionPerformed(ActionEvent e) {
                Object src = …
                String param = …
                if (param == "Quitter") delegue.quitter() ;
                        else if (param == "Désélectionner")
                                delegue.annulle() ;
                                else delegue.affichetexte() ;
                }

        // méthodes indispensables comme WindowListener
        public void windowClosing(WindowEvent e) {
                delegue.quitter() ;
                }
        public void windowClosed(WindowEvent e) {}
        public void windowIconified(WindowEvent e) {}
        public void windowDeiconified(WindowEvent e) {}
        public void windowActivated(WindowEvent e) {}
        public void windowDeactivated(WindowEvent e) {}
        public void windowOpened(WindowEvent e) {}
        }
```

Lecture/écriture dans des objets graphiques

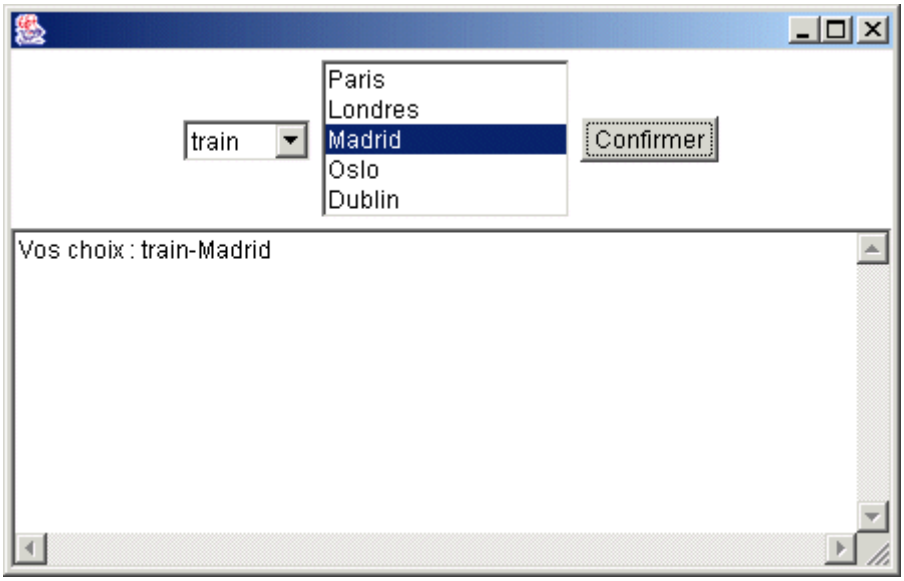
Classe de l'objet	TextField, TextArea	Choice, List
Récupérer de l'information (lire)	`.getText()`	`.getSelectedItem()` (élément) `.getSelectedIndex()` (indice)
Mettre de l'information (écrire)	`.setText(s)`	`.add(s)`
Vider	`.setText("")`	`.removeAll()`

N.B. : `s` est un objet de la classe `String`.

TD6

Compléter le fichier **UtilAwt.java,** créé précédemment dans le dossier **utilawt,** par un objet d'une classe Delegue et un objet d'une classe Adaptateur de telle sorte que le programme traite les événements suivants :

- cliquer sur la **case de fermeture** de la fenêtre ferme la fenêtre ;
- cliquer sur le bouton « Confirmer » permet de faire afficher dans la zone de texte les options choisies :



Démarche à suivre

1) Ajouter un deuxième import :

```
import java.awt.* ;
import ...
```

2) Ajouter des attributs d'instance à l'interface :

```
protected ...
protected ...
```

3) Compléter le constructeur Fenetre() en créant le délégué et l'adaptateur par :

```
delegue = ...
adapt = ...
```

4) Ecrire le début des classes Delegue et Adaptateur :

```
class Delegue {
  protected Fenetre fen ;
  Delegue (Fenetre f) {
        ...
        }
  }
```

```
class Adaptateur implements WindowListener, ActionListener {
  protected Delegue delegue ;
  Adaptateur (Delegue d) {
        ...
        }
  }
```

5) Enregistrer l'adaptateur dans le constructeur Fenetre() sur les éléments à surveiller :

```
add...
suite.add...
```

6) Ajouter à la classe Adaptateur les méthodes indispensables comme Listener :

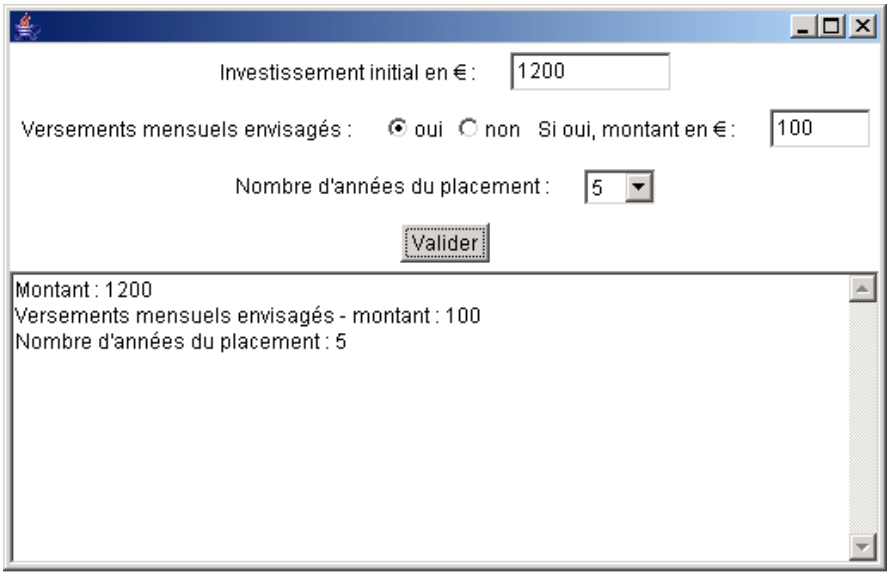
```
// WindowListener
public void windowOpened(WindowEvent e){}
public void windowClosing(WindowEvent e) {
      delegue.quitter() ;
      }
public void windowClosed(WindowEvent e){}
public void windowIconified(WindowEvent e){}
public void windowDeiconified(WindowEvent e){}
public void windowActivated(WindowEvent e){}
public void windowDeactivated(WindowEvent e){}

// ActionListener
public void actionPerformed(ActionEvent e) {
      delegue.afficheres() ;
      }
```

7) Ecrire les méthodes de la classe Delegue réalisant les actions souhaitées :

```
public void quitter() {
      ...
      }

public void afficheres() {
      ...
      }
```

TD7

L'interface graphique ci-dessous a été créée précédemment (fichier **MaFenetre.java** dans le dossier **MAFENETRE)**. Modifier le fichier pour assurer la fermeture de la fenêtre et pour qu'après appui sur le bouton « **Valider** », la zone de texte affiche les renseignements fournis par l'utilisateur :



5. Interfaces et calculs

Les contenus des champs de texte des **TextField** ou **TextArea** sont des **chaînes de caractères** de la classe **String**. Or pour effectuer des calculs il faut travailler sur des **nombres**.

La conversion d'une chaîne de caractères s :

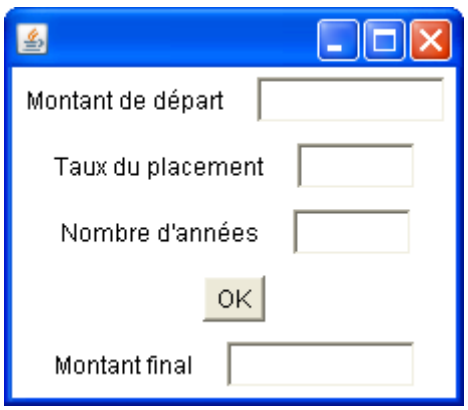
- en un float a se fait par :
  `float a=Float.parseFloat(s);`
  (ou `float a = new Float(s).floatValue();` alternative dépréciée)
- en un double b par :
  `double b=Double.parseDouble(s);`
  (ou `double b = new Double(s).doubleValue();` alternative dépréciée)
- en un int c par :
  `int c=Integer.parseInt(s);`
  (ou `int c = new Integer(s).intValue();` alternative dépréciée)

Inversement, pour afficher un **résultat numérique** dans un champ de texte, il faut le convertir en une **chaîne de caractères**.

- La conversion d'un float a en une chaîne s se fait par :
  `String s=Float.toString(a);`
  (ou `String s = new Float(a).toString();`
- La conversion d'un double b en une chaîne s se fait par :
  `String s=Double.toString(b);`
  (ou `String s = new Double(b).toString();` alternative dépréciée)
- La conversion d'un int c en une chaîne s se fait par :
  `String s=Integer.toString(c);`
  (ou `String s = new Integer(c).toString();` alternative dépréciée)

TD8

On a créé précédemment l'interface suivante dans le dossier **Placement** :



Cette interface comporte :

- un TextField appelé dep pour la saisie du montant de départ
- un TextField appelé tx pour la saisie du taux
- un TextField appelé nb pour la saisie du nombre d'années
- un Button b pour valider
- un TextField appelé res pour l'affichage du montant final

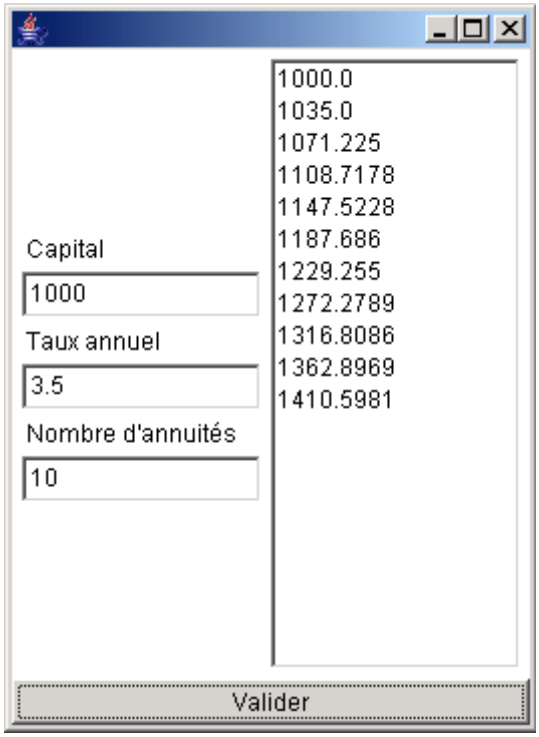
Compléter le fichier **Placement.java** par un objet d'une classe Delegue et un objet d'une classe Adaptateur de telle sorte que le programme traite les événements suivants :

- cliquer sur la **case de fermeture** de la fenêtre ferme la fenêtre ;
- cliquer sur le bouton « OK » permet de faire afficher le montant final du placement.

TD9

Le dossier **utilplan** contient un fichier **UtilPlan.java**. Compiler et faire exécuter cette application.

Le but de l'exercice est de faire afficher les valeurs acquises par le capital, placé à un taux fixe pendant un certain nombre d'années. Ces trois données sont entrées par l'utilisateur dans la partie gauche de la fenêtre. Un clic sur le bouton « Valider » doit provoquer le calcul des valeurs acquises :



Compléter la méthode calcule de la classe Delegue pour qu'elle fasse afficher les résultats du calcul, considérés comme des réels flottants, donc de type primitif float.
On veillera à ce qu'à chaque appui sur le bouton « Valider », les résultats précédents soient effacés.

TD10

Le dossier **calculette** contient un fichier **Calculette.java**. Compiler et faire exécuter cette application. Le but est de faire exécuter les opérations prévues sur les deux nombres donnés (entiers ou réels).

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

class Calc extends Frame {
       protected Panel p1, p2, p3, p10,p11,p12,p13,p20,p21,p22,p23,p24,p31,p32;
       protected TextField f1, f2, f3;
       protected Button b1, b2, b3, b4, b5;
       protected Delegue d1;
       protected Adaptateur a1;

       Calc () {
              setLayout(new BorderLayout());
              p1=new Panel(); p1.setLayout(new GridLayout(1,4));
              p10=new Panel(); p10.add(new Label (" a ")); p1.add(p10);
              p11=new Panel(); f1 = new TextField(8); p11.add(f1); p1.add(p11);
              p12=new Panel(); p12.add(new Label (" b ")); p1.add(p12);
              p13=new Panel(); f2 = new TextField(8); p13.add(f2); p1.add(p13);
              add("North", p1);
```

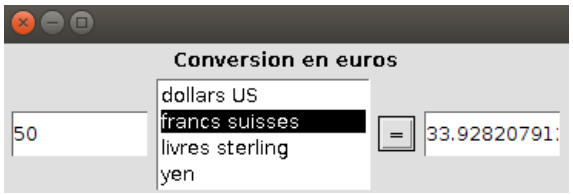
```
            p2=new Panel(); p2.setLayout(new GridLayout(1,5));
            p20=new Panel(); b1 = new Button(" + "); p20.add(b1); p2.add(p20);
            p21=new Panel(); b2 = new Button(" - "); p21.add(b2); p2.add(p21);
            p22=new Panel(); b3 = new Button(" * "); p22.add(b3); p2.add(p22);
            p23=new Panel(); b4 = new Button(" / "); p23.add(b4); p2.add(p23);
            p24=new Panel(); b5 = new Button(" a^b "); p24.add(b5); p2.add(p24);
            add("Center", p2);

            p3=new Panel(); p3.setLayout(new GridLayout(1,2));
            p31 = new Panel(); p31.add(new Label (" result ")); p3.add(p31);
            p32 = new Panel(); f3 = new TextField (20); p32.add(f3); p3.add(p32);
            add("South", p3);

            d1 = new Delegue(this);
            a1 = new Adaptateur (d1);
            this.addWindowListener (a1);
            ...
            }
      }

public class Calculette {
      public static void main (String args [] ) {
            Calc f = new Calc();
            f.pack();
            f.setVisible(true);
            }
      }

class Delegue {
      protected Calc c1;

      Delegue (Calc c) {
            c1=c;
            }
      public void exit() {
            System.exit(0);
            }
      public void somme() {
            }
      public void diff() {
            }
      public void mult() {
            }
      public void divis() {
            }
      public void puis() {
            }
      }

class Adaptateur implements ActionListener, WindowListener {
      protected Delegue d1;

      Adaptateur (Delegue d) {
            d1=d;
            }

      public void actionPerformed (ActionEvent e) {
            // clic sur boutton
            Object src = e.getSource();
            String param = ((Button)src).getLabel();
            if (param.equals(" + ")) {d1.somme();}
            if (param.equals(" - ")) {d1.diff();}
            if (param.equals(" * ")) {d1.mult();}
            if (param.equals(" / ")) {d1.divis();}
            if (param.equals(" a^b ")) {d1.puis();}
            }
      public void windowOpened (WindowEvent e) {}
```

```
	public void windowClosing (WindowEvent e) {  // fermer la fenetre
		d1.exit();
		}
	public void windowClosed (WindowEvent e) {}
	public void windowIconified (WindowEvent e) {}
	public void windowDeiconified (WindowEvent e) {}
	public void windowActivated (WindowEvent e) {}
	public void windowDeactivated (WindowEvent e) {}
	}
```

Attachez l'adaptateur `a1` à tous les boutons, puis complétez les 5 méthodes de calcul `somme()`, `diff()`, `mult()`, `divis()` et `puis()` pour que les calculs se fassent correctement.

TD11 : conversion monétaire

Le dossier **CONVERT** contient le fichier **Convert.java**. Le bouton = est pour l'instant inactif. Le but de l'exercice est de le rendre actif et de faire afficher dans le dernier champ de texte le résultat de la conversion en euros de certaines monnaies.



1) Examiner le fichier **Convert.java** :

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

class FenetreConvert extends Frame {
	static int nb = 4 ;
	static String listemonnaies[] ;
	static double listevaleurs[] ;
	protected Panel p ;
	protected Label lb;
	protected Button bouton ;
	protected TextField res ;
	protected TextField dep ;
	protected List monnaie ;
	protected Delegue d ;
	protected Adaptateur adapt ;

	public static void initmonnaies() {
		listemonnaies = new String[nb] ;
		listevaleurs = new double[nb] ;
		listemonnaies[0] = "dollars US" ; listevaleurs[0] = 0.8939 ;
		// 1 euro = 0.8939 dollar US
		listemonnaies[1] = "francs suisses" ; listevaleurs[1] = 1.4737 ;
		listemonnaies[2] = "livres sterling" ; listevaleurs[2] = 0.6288 ;
		listemonnaies[3] = "yen" ; listevaleurs[3] = 110.49 ;
		}

	FenetreConvert(){
		initmonnaies() ;
		setLayout(new BorderLayout()) ;

		lb = new Label("Conversion en euros",Label.CENTER);
		lb.setFont(new Font("Arial",Font.BOLD,12));
		add("North", lb) ;

		p = new Panel() ;
```

```
                dep = new TextField(10) ;
                p.add(dep) ;
                monnaie = new List(4) ;
                // ajouter les monnaies dans la liste :
                // ...
                p.add(monnaie) ;
                bouton = new Button("=") ;
                p.add(bouton) ;
                res = new TextField(10) ;
                p.add(res) ;
                add("South", p) ;

                d = new Delegue(this) ;
                adapt = new Adaptateur(d);
                this.addWindowListener(adapt);
                bouton.addActionListener(adapt) ;
                }
        }

public class Convert {
        public static void main(String[] args) {
                FenetreConvert f = new FenetreConvert();
                f.pack();
                f.setLocationRelativeTo(null); //the window is placed in the center of the screen.
                f.setVisible(true);
                }
        }

class Delegue {
        protected FenetreConvert fen ;
        Delegue(FenetreConvert f) {
                fen = f;
                }
        public void exit() {
                System.exit(0);
                }
        public void conversion() {
                double valEnEuros;
                // recupere le nombre tap'e dans le premier champ
                // et le convertit en double
                //String s =
                //double quantite =
                // puis l'unit'e mon'etaire s'electionn'ee
                //int choix =
                // convertit le tout en euros
                //valEnEuros =
                // puis en chaine de caracteres
                //s =
                // et affiche le resultat en euros

                }
        }

class Adaptateur implements ActionListener, WindowListener {
        protected Delegue delegue ;
        Adaptateur(Delegue dd) {
                delegue = dd ;
                }
        public void actionPerformed(ActionEvent e) {
           // Compléter :
```

```
    }
public void windowOpened (WindowEvent e) {}
public void windowClosing (WindowEvent e) {
        delegue.exit();
        }
public void windowClosed (WindowEvent e) {}
public void windowIconified (WindowEvent e) {}
public void windowDeiconified (WindowEvent e) {}
public void windowActivated (WindowEvent e) {}
public void windowDeactivated (WindowEvent e) {}
}
```

2) Dans la classe FenetreConvert :
quelles sont les variables de classe ?

……..

quelles sont les méthodes de classe ?

……..

3) Compléter le constructeur de la classe FenetreConvert afin que la liste monnaie affiche les différentes monnaies : dollars US, francs suisses, …

4) Compléter la méthode actionPerformed de la classe Adaptateur.

5) En exploitant les méthodes des classes List et TextField, ainsi que les méthodes de conversion de chaînes en nombres (et réciproquement), compléter la méthode conversion() pour qu'elle fasse afficher dans le deuxième champ de texte le résultat demandé.

TD12 : utilisation de conditions dans une application

Le dossier **impot** contient le fichier **Impot.java**. Compiler ce fichier. Les boutons « Calculer » et « Annuler » sont pour l'instant inactifs. Le but de l'exercice est de les rendre actifs : « Calculer » doit faire afficher le nombre de parts, la valeur de la part et le montant de l'impôt ; « Annuler » doit annuler toutes les données saisies et affichées.

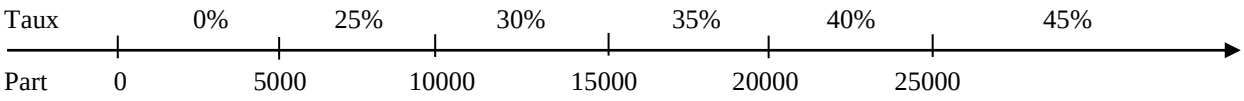
Calcul du nombre de parts :

- si plus de 6 enfants : nombre de parts = nombre d'adultes + 0.5*nombre d'enfants + 1
- sinon, si plus de 3 enfants : nombre de parts = nombre d'adultes + 0.5*nombre d'enfants + 0.5
- sinon, si au moins un enfant : nombre de parts = nombre d'adultes + 0.5*nombre d'enfants
- sinon, nombre de parts = nombre d'adultes.

La part est égale au revenu annuel total divisé par le nombre de parts.

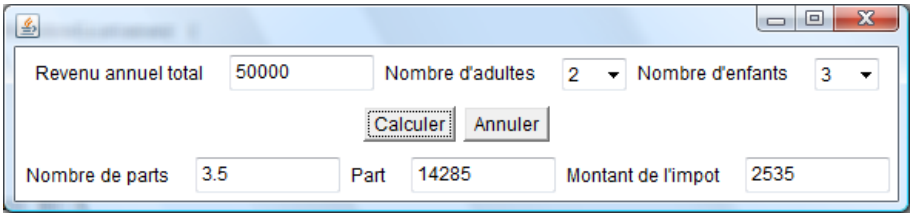
Calcul de l'impôt : six tranches sont définies correspondant aux intervalles suivants, associés aux taux spécifiés :

- tranche 5 : partie au-dessus de 25000 — taux d'imposition : 45 %
- tranche 4 : partie comprise entre 20000 et 25000 — taux d'imposition : 40 %
- tranche 3 : partie comprise entre 15000 et 20000 — taux d'imposition : 35 %
- tranche 2 : partie comprise entre 10000 et 15000 — taux d'imposition : 30 %
- tranche 1 : partie comprise entre 5000 et 10000 — taux d'imposition : 25 %
- tranche 0 : partie en-dessous de 5000 — taux d'imposition : 0 %



Ouvrir le fichier **Impot.java**. Compléter la méthode calcul de la classe Delegue pour qu'elle calcule et fasse afficher le résultat demandé.

Exemple :




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

class FenetreImpot extends Frame {
	protected Panel p1, p2, p3 ;
	protected TextField rev, nbpart, part, impot ;
	protected Choice nba, nbe ;
	protected Button confirmer, annuler ;
	protected Delegue delegue;
	protected Adaptateur adapt;

	FenetreImpot() {
		this.setLayout(new GridLayout(3,1));
		p1 = new Panel();
		p1.add(new Label("Revenu annuel total"));
		rev= new TextField(10); p1.add(rev);
		p1.add(new Label("Nombre d'adultes"));
		nba = new Choice();
		for (int i = 1 ; i <=10 ; i++) nba.add(Integer.toString(i));
		p1.add(nba);
		p1.add(new Label("Nombre d'enfants"));
		nbe = new Choice();
		for (int i = 0 ; i <=10 ; i++) nbe.add(Integer.toString(i));
		p1.add(nbe);
		add(p1);
		p2 = new Panel();
		confirmer = new Button("Calculer"); p2.add(confirmer);
		annuler = new Button("Annuler"); p2.add(annuler);
		add(p2);
		p3 = new Panel();
		p3.add(new Label("Nombre de parts"));
		nbpart= new TextField(10); p3.add(nbpart);
		p3.add(new Label("Part"));
		part= new TextField(10); p3.add(part);
		p3.add(new Label("Montant de l'impot"));
		impot= new TextField(10); p3.add(impot);
		add(p3);
		delegue = new Delegue(this);
		adapt = new Adaptateur (delegue);
		this.addWindowListener (adapt);
		confirmer.addActionListener(adapt);
		annuler.addActionListener(adapt);
		}
	}

public class Impot {
	public static void main(String[] args) {
		FenetreImpot f = new FenetreImpot();
		f.pack();
		f.setVisible(true);
		}
	}

class Delegue {
	protected FenetreImpot fif;

	Delegue (FenetreImpot c) {
```

```
            fif=c;
            }

      public void exit() {
            System.exit(0);
            }

      public void calcule() {
            // récupération des données et conversion en numérique
            // double revenu = ...
            // int nba = ...
            // int nbe = ...
            double nbpart ;
            // calcul et affichage de nbpart
            ...
            double part = 0 ;
            // calcul et affichage de part
            ...
            boolean tranche5 = (part >= 25000) ;
            boolean tranche4 = ...
            boolean tranche3 = ...
            boolean tranche2 = ...
            boolean tranche1 = ...
            boolean tranche0 = (part < 5000);
            double taux5 = 0.45 ;
            double taux4 = 0.40 ;
            double taux3 = 0.35 ;
            double taux2 = 0.30 ;
            double taux1 = 0.25 ;
            double impot ;
            // calcul de l'impot
            if (tranche5) { ... }
            else if (tranche4) { ... }
            else if (tranche3) { ... }
            else if (tranche2) { ... }
            else if (tranche1) { ... }
            else { ... }
            // affichage de impot
            ...
            }

      public void annulle() {
            ...
            }
      }

class Adaptateur implements ActionListener, WindowListener {
      protected Delegue delegue;

      Adaptateur (Delegue d) {
            delegue=d;
            }

      public void actionPerformed (ActionEvent e) {
            // clic sur boutton
            Object src = e.getSource();
            String param = ((Button)src).getLabel();
            if (param.equals("Calculer")) { delegue.calcule(); }
            if (param.equals("Annuler")) { delegue.annulle(); }
            }
      public void windowOpened (WindowEvent e) {}
      public void windowClosing (WindowEvent e) { delegue.exit(); }
      public void windowClosed (WindowEvent e) {}
      public void windowIconified (WindowEvent e) {}
      public void windowDeiconified (WindowEvent e) {}
      public void windowActivated (WindowEvent e) {}
      public void windowDeactivated (WindowEvent e) {}
      }
```

6. Les images dans les interfaces

Le dossier **Referendum** contient les fichiers **Referendum.java** et **vote.jpg.** Cette application ne permet pas la gestion des événements. Il faut donc taper *Ctrl C* dans la fenêtre de commande pour fermer l'interface.

Le code de l'application est donné ci-après. Les instructions nécessaires à l'affichage de l'image sont en gras.

Remarquez les instructions *import* nécessaires. Contrairement aux paquetages *java…* « de base », les paquetages *javax…* sont des extentions.

Dans la classe *LoadImageApp*,

- le constructeur permet de lire le fichier contenant l'image
- la méthode *paint* affiche l'image à partir du point (0,0) : point en haut à gauche. On spécifie un point par ses coordonnées (X,Y). La valeur de X augmente vers la droite et la valeur de Y augmente vers le bas.
- la méthode *getPreferredSize* réserve l'emplacement adéquat en tenant compte de la taille de l'image.

Pour afficher l'image dans l'interface, on y ajoute un objet de la classe *LoadImageApp* (voir *new LoadImageApp()* dans le constructeur de la classe *FenetreReferendum*).

```
import java.awt.* ;
import java.awt.event.* ;
import java.awt.image.* ;
import java.io.* ;
import javax.imageio.* ;

class LoadImageApp extends Component {
   BufferedImage img;

   public void paint(Graphics g) {
      g.drawImage(img, 0, 0, null);
   }

   public LoadImageApp() {
     try {
        img = ImageIO.read(new File("vote.jpg"));
         }
     catch (IOException e) {}
   }

   public Dimension getPreferredSize() {
     if (img == null) { return new Dimension(100,100); }
     else {return new Dimension(img.getWidth(), img.getHeight());}
   }
}

class FenetreReferendum extends Frame {
   protected Panel p ;
   protected Label lb;
   protected CheckboxGroup cbg ;
```

```
    protected Checkbox chb1, chb2 ;

    FenetreReferendum(){
            setLayout(new BorderLayout()) ;

            lb = new Label("Referendum",Label.CENTER);
            lb.setFont(new Font("Arial",Font.BOLD,12));
            add("North", lb) ;

            add("Center", new LoadImageApp());

            p = new Panel() ;
            cbg =new CheckboxGroup();
           chb1=new Checkbox("oui",cbg,true);
            p.add(chb1);
            chb2=new Checkbox("non",cbg,true);
            p.add(chb2);
            add("South", p) ;
            }
     }

public class Referendum {
            public static void main(String[] args) {
                        FenetreReferendum f = new FenetreReferendum();
                        f.pack();
                        f.setLocationRelativeTo(null); //the window is placed in the center of the screen.
                        f.setVisible(true);
                        }
            }
```

TD 13 : insertion d'image dans des interfaces

Il s'agit d'insérer des images dans des applications déjà modifiées en TD.
Au besoin,

- adaptez à l'interface les coordonnées du point à partir duquel l'image est affichée
- changez la couleur du fond de l'interface (voir la méthode **setBackground** de la classe **Frame** dans la documentation)
- modifiez le placement d'objets **Panel** déjà présents.

1. Dans le dossier **convert** se trouve l'image **euro.jpg**. Insérez cette image dans l'application **Convert.java** (déjà modifiée au TD11) pour aboutir à l'interface ci-contre.

2. Dans le dossier **utilawt** se trouve l'image **carte.jpg**. Insérez cette image dans l'application **UtilAwt.java** (déjà modifiée au TD3 et au TD6) pour aboutir à l'interface ci-contre.

TD 14 : applications et images

Partie 1

1) Ouvrir le dossier CHARGEIMAGES comportant le fichier **ChargeImages.java** ainsi que les 19 fichiers images **im1.jpg, im2.jpg, …, im19.jpg**.
Les deux boutons sont pour l'instant inactifs. Un des buts de l'exercice est de les rendre actifs.

2) Examiner le fichier **ChargeImages.java** :

```
import java.awt.* ;
import java.awt.image.* ;
import java.awt.event.* ;
import java.io.* ;
import javax.imageio.* ;

class LoadImage extends Component {
   BufferedImage img;
   public LoadImage(int n) {
     try {
        img = ImageIO.read(new File("im"+n+".jpg"));
         }
     catch (IOException e) {}
   }
   public Dimension getPreferredSize() {
     if (img == null) { return new Dimension(100,100); }
     else { return new Dimension(img.getWidth(), img.getHeight()); }
   }
}
class FenetreChargeImages extends Frame {
          protected Panel p ;
          protected Image image ;
          protected Button suivant, precedent ;
          protected int rang ;
          protected LoadImage ldIm;
          …  // à compléter
```

```
        FenetreChargeImages () {
                setLayout(new BorderLayout()) ;
                setBackground(Color.white);
                rang = 1 ;
                ldIm = new LoadImage(rang);
                add("Center", ldIm);
                p = new Panel() ;
                suivant = new Button("Next") ;
                p.add(suivant) ;
                precedent = new Button("Previous") ;
                p.add(precedent) ;
                add("South", p) ;
                … // à compléter
                }

      public void paint(Graphics g){
                ldIm = new LoadImage(rang);
                g.drawImage(ldIm.img, 0, 0, null);
                }
        }

public class ChargeImages {
        public static void main(String[] args) {
                FenetreChargeImages f = new FenetreChargeImages();
                f.pack();
                f.setLocationRelativeTo(null); //the window is placed in the center of the screen.
                f.setVisible(true);
                }
        }

class Delegue {
        protected FenetreChargeImages ci ;
        Delegue(FenetreChargeImages c) {
                ci = c ;
                }
        public void avancer() {
                … // à compléter
                ci.repaint();
                }
        public void reculer() {
                … // à compléter
                ci.repaint() ;
                }
        public void exit() {
                System.exit(0);
                }
        }

class Adaptateur implements ActionListener, WindowListener {
        protected Delegue delegue ;
        public Adaptateur(Delegue d) {
                delegue = d ;
                }
        public void actionPerformed(ActionEvent e) {
                Object src = e.getSource();
                String param = ((Button)src).getLabel();
                … // à compléter
                }
        public void windowOpened (WindowEvent e) {}
```

```
public void windowClosing (WindowEvent e) { delegue.exit();}
public void windowClosed (WindowEvent e) {}
public void windowIconified (WindowEvent e) {}
public void windowDeiconified (WindowEvent e) {}
public void windowActivated (WindowEvent e) {}
public void windowDeactivated (WindowEvent e) {}
}
```

Quelle est l'image affichée par la méthode paint au lancement de l'application ?

……..

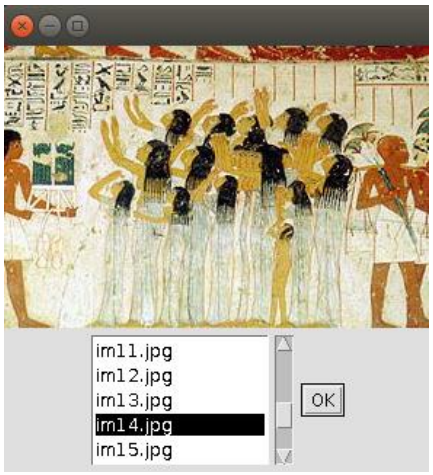
Quel est l'effet de l'instruction suivante ?
ci.repaint() ;

……..

3) Complétez les lignes manquantes pour associer un adaptateur aux deux boutons et à la fenêtre et faire en sorte que les clics sur les boutons soient actifs. On surveillera les numéros des images afin de ne pas demander d'afficher une image inexistante. Quand on aura parcouru toutes les images, on bouclera sur les images déjà vues.

Partie 2

- Créez un dossier CHOIXIMAGE et recopiez-y l'ensemble des fichiers du dossier CHARGEIMAGES.
- Renommez le fichier ChargeImages.java en ChoixImage.java.
- Ouvrez-le et renommez la classe ChargeImages en ChoixImage.
- Modifiez ensuite le fichier ChoixImage.java pour que le changement d'image ne se fasse plus par les deux boutons, mais par une **liste** proposant les différentes images existantes.
- L'image im1.jpg est proposée au démarrage. Lorsque l'utilisateur effectue un autre choix et confirme avec OK, l'image est changée.



7. Applications et dessins

Les objets de type **Canvas** sont conçus pour être dessinés. Pour dessiner dans une application, on y insère donc un objet d'une classe dérivée de Canvas.

La méthode paint associée à chaque composant graphique fournit automatiquement un **contexte graphique** pour ce composant (objet de la classe **Graphics** de java.awt). Sur ce contexte graphique, on peut construire des lignes (drawLine), des polygônes (drawPolygon), des ovales (drawOval), des rectangles (drawRect), insérer des images (drawImage), etc.

TD15 : Dessiner dans une application

Le dossier DESSIN contient le fichier **Dessin.java** dont le contenu est indiqué ci-après.

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

class FenetreDessin extends Frame {
	protected Ardoise ad ;
	protected Delegue delegue ;
	protected Adaptateur adapt ;

	FenetreDessin() {
		setLayout(new BorderLayout()) ;
		ad = new Ardoise(500,500);
		add("Center", ad) ;
		delegue = new Delegue(ad); // le delegue reçoit l'adresse de l'ardoise
		adapt = new Adaptateur(delegue) ;
		// l'adaptateur doit surveiller les mouvements de la souris :
		ad.addMouseMotionListener(adapt);
		ad.addMouseListener(adapt);
		this.addWindowListener(adapt);
		}
	}

class Ardoise extends Canvas { // classe derivant de Canvas
	protected Image im ;		// l'image dessinee
	protected Graphics gr ;		// le contexte graphique de l'image
	protected int x, y, xo, yo ;	// coordonnees de la souris
	protected int largeur, hauteur ;

	Ardoise(int a, int b) {		// constructeur
		largeur = a ;
		hauteur = b ;
		setSize(largeur,hauteur);
		gr = null ;
		}

	public void paint(Graphics g) {	// methode d'affichage
		if (gr == null) {		// initialisation au demarrage
			setBackground (Color.cyan) ;
			im = createImage(largeur, hauteur) ;
			gr = im.getGraphics() ;
			}
		g.drawImage(im,0,0,this) ;	// affichage de im
		}
	}

public class Dessin {
	public static void main(String[] args) {
		FenetreDessin f = new FenetreDessin();
		f.pack();
		f.setLocationRelativeTo(null); //the window is placed in the center of the screen.
		f.setVisible(true);
		}
	}

class Delegue {
	protected Ardoise ard ;

	Delegue(Ardoise a) {
```

```
                ard = a ;
                }

        public void tiresouris(MouseEvent e) {
                ard.x=e.getX() ; ard.y=e.getY() ;
                if (ard.gr != null) {
                        ard.gr.drawLine(ard.xo, ard.yo, ard.x , ard.y) ;
                        ard.xo = ard.x ; ard.yo = ard.y ;
                        }
                ard.repaint();
                }

        public void pressesouris(MouseEvent e) {
                ard.xo=e.getX() ; ard.yo=e.getY() ;
                }

        public void exit() {
                System.exit(0);
                }
        }

class Adaptateur implements MouseMotionListener,MouseListener, WindowListener {
        protected Delegue delegue ;
        public Adaptateur(Delegue d) {
                delegue = d ;
                }
        public void mouseDragged(MouseEvent e) {
                delegue.tiresouris(e) ;
                }
        public void mouseMoved(MouseEvent e) {
                }
        public void mousePressed(MouseEvent e) {
                delegue.pressesouris(e) ;
                }
        public void mouseClicked(MouseEvent e) {
                }
        public void mouseReleased(MouseEvent e) {
                }
        public void mouseEntered(MouseEvent e) {
                }
        public void mouseExited(MouseEvent e) {
                }
        public void windowOpened (WindowEvent e) {}
        public void windowClosing (WindowEvent e) { // ferme la fenetre
                delegue.exit();
                }
        public void windowClosed (WindowEvent e) {}
        public void windowIconified (WindowEvent e) {}
        public void windowDeiconified (WindowEvent e) {}
        public void windowActivated (WindowEvent e) {}
        public void windowDeactivated (WindowEvent e) {}
        }
```

- Indiquer dans quel paquetage se trouvent les interfaces MouseMotionListener et MouseListener :

  ……..

- Indiquer quelles méthodes de la classe Adaptateur appartiennent
  - à l'interface MouseMotionListener :

    ……

- à l'interface MouseListener :

  ……

- Quelle est la différence entre les méthodes mouseDragged, mouseMoved et mousePressed?

  ……..

- Modifier le fichier Dessin.java pour ajouter en bas de l'interface un bouton « Effacer » et faire en sorte qu'un appui sur ce bouton efface effectivement le dessin.

TD16 : Dessiner en couleurs dans une application

Le dossier GEOMETRIQUE contient le fichier **Geometrique.java** dont le contenu est indiqué ci-après.

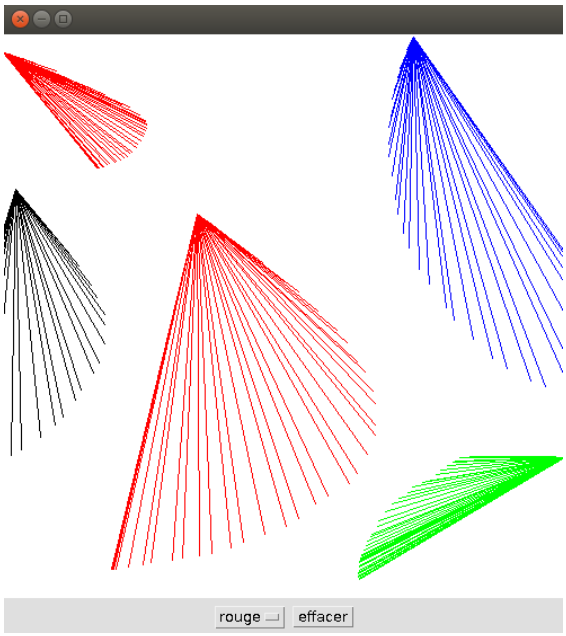
```
import java.awt.* ;
import java.awt.event.* ;

class FenetreGeometrique extends Frame {
        protected Ardoise ad ;
        protected Panel p ;
        protected Button effacer ;
        protected Delegue delegue ;
        protected Adaptateur adaptateur ;

        FenetreGeometrique() {
                setLayout(new BorderLayout());
                ad = new Ardoise(500,500);
                add("Center", ad) ;

                p = new Panel() ;
                delegue = new Delegue(this);
                adaptateur = new Adaptateur(delegue) ;
                effacer = new Button("effacer") ;
                p.add(effacer);
                effacer.addActionListener(adaptateur) ;
                add("South", p) ;

                ad.addMouseMotionListener(adaptateur);
                ad.addMouseListener(adaptateur);
                this.addWindowListener(adaptateur);
                }
}

class Ardoise extends Canvas { // classe derivant de Canvas
        protected Image im ;                    // l'image dessinee
        protected Graphics gr ;                 // le contexte graphique de l'image
        protected int x, y, xo, yo ;            // coordonnees de la souris
        protected int largeur, hauteur ;

        Ardoise(int a, int b) {         // constructeur
                largeur = a ;
                hauteur = b ;
                setSize(largeur,hauteur);
                gr = null ;
          }

        public void paint(Graphics g) {   // methode d'affichage
                if (gr == null) {               // initialisation au demarrage
                        setBackground (Color.white) ;
                        im = createImage(largeur, hauteur) ;
                        gr = im.getGraphics() ;
```

```
                        }
                g.drawImage(im,0,0,this) ;          // affichage de im
                }
        }

public class Geometrique {
        public static void main(String[] args) {
                FenetreGeometrique f = new FenetreGeometrique();
                f.pack();
                f.setLocationRelativeTo(null); //the window is placed in the center of the screen.
                f.setVisible(true);
                }
        }

class Delegue {
        FenetreGeometrique forme ;
        Delegue(FenetreGeometrique t) {
                forme = t ;
                }
        public void boutonEnfonce(MouseEvent e) {
                forme.ad.xo = e.getX();
                forme.ad.yo = e.getY() ;
                }
        public void sourisTiree(MouseEvent e) {
                forme.ad.x=e.getX() ;
                forme.ad.y=e.getY() ;
                if (forme.ad.gr != null) {
                   forme.ad.gr.drawLine(forme.ad.xo, forme.ad.yo, forme.ad.x , forme.ad.y) ;
                   }
                forme.ad.repaint();
                }
        public void nettoyer(ActionEvent e) {
                forme.ad.gr.clearRect(0,0,forme.ad.largeur,forme.ad.hauteur);
                forme.ad.repaint();
                }
     public void exit() {
                System.exit(0);
                }
}

class Adaptateur implements MouseMotionListener, MouseListener, ActionListener, WindowListener  {
        protected Delegue delegue ;
        public Adaptateur(Delegue d) {
                delegue = d ;
                }
        public void mousePressed(MouseEvent e) {
                delegue.boutonEnfonce(e) ;
                }
        public void mouseDragged(MouseEvent e) {
                delegue.sourisTiree(e) ;
                }
        public void mouseMoved(MouseEvent e) {}
        public void mouseClicked(MouseEvent e) {}
        public void mouseReleased(MouseEvent e) {}
        public void mouseEntered(MouseEvent e) {}
        public void mouseExited(MouseEvent e) {}
        public void actionPerformed(ActionEvent e) {
                delegue.nettoyer(e) ;
                }
```

```
        public void windowOpened (WindowEvent e) {}
        public void windowClosing (WindowEvent e) { // ferme la fenetre
                delegue.exit();
                }
        public void windowClosed (WindowEvent e) {}
        public void windowIconified (WindowEvent e) {}
        public void windowDeiconified (WindowEvent e) {}
        public void windowActivated (WindowEvent e) {}
        public void windowDeactivated (WindowEvent e) {}
}
```

On demande d'ajouter un objet de la classe Choice pour qu'il soit possible de modifier pendant l'exécution de l'application la couleur des traits en proposant les options noir (par défaut), rouge, bleu et vert.

Modifier le fichier Geometrique.java pour arriver au résultat suivant.



Quelques indications :

1. A l'aide de la documentation, indiquer dans quel paquetage se trouve la classe Color. Examiner ses attributs.

…….....

2. Ajouter à la classe FenetreGeometrique deux attributs :
   protected Color couleur ;
   protected Choice c ;
3. Construire l'objet c dans le constructeur de la classe FenetreGeometrique. Pour le connecter à l'adaptateur quelle méthode utiliser ?

…….....

4. Ajouter à la classe Delegue une méthode à compléter : public void changeCouleur() {…}
5. La classe Adaptateur doit implémenter une interface supplémentaire. Laquelle ?

…….....

6. Quelle méthode doit être ajoutée à la classe Adaptateur ?

……..

8. Lecture/Ecriture de fichiers

Les classes nécessaires à ce qu'on appelle les « entrées/sorties » appartiennent au paquetage **java.io**.

TD 17 : Lecture de fichiers

Partie 1

Dans le dossier **COMMUNICATION**, figurent le fichier **Communication.java** et différents fichiers texte. Le but est d'afficher le fichier **texte.txt** ligne à ligne à chaque fois que l'on clique sur un bouton " Suite ".

Fichier à lire : le fichier **texte.txt** a le contenu suivant :

```
4
les sanglots longs
des violons de l'automne
bercent mon coeur
d'une langueur monotone
```

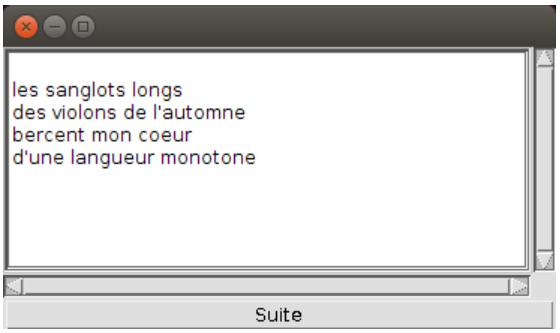
La première ligne du fichier indique le nombre de lignes de texte que l'application va devoir lire.

Fichier Communication.java :

```
import java.awt.* ;
import java.io.* ;
import java.awt.event.* ;

class FenetreCommunication extends Frame {
	protected String [] chaine = new String[15] ;
	protected int nblignes, k ;
	protected TextArea texte ;
	protected Button b ;
	protected Delegue delegue ;
	protected Adaptateur adapt ;

	FenetreCommunication () {
		delegue = new Delegue(this);
		setLayout(new BorderLayout()) ;
		texte = new TextArea();
		add("North", texte) ;
		adapt = new Adaptateur(delegue);
		b = new Button("Suite") ;
		add("South", b) ;
		b.addActionListener(adapt);
		this.addWindowListener(adapt);
		chargement() ;
		k = 0 ;
		}

	public void chargement(){
		BufferedReader fichier ;
		try	{
			fichier =new BufferedReader(new FileReader("texte.txt")) ;
			}
		catch (IOException e) {
			texte.setText("erreur de fichier") ; return ;
			}
		try	{
			nblignes = Integer.parseInt(fichier.readLine()) ;
			for (int i = 0 ; i < nblignes ; i++)
```

```
                                        chaine[i] = fichier.readLine() ;
                                }
                        catch (IOException e) {
                                texte.setText("erreur de transfert") ; return ;
                                }
                        }
        }

public class Communication {
        public static void main(String[] args) {
                FenetreCommunication f = new FenetreCommunication();
                f.pack();
                f.setLocationRelativeTo(null); //the window is placed in the center of the screen.
                f.setVisible(true);
                }
        }

class Delegue {
        protected FenetreCommunication com ;
        Delegue(FenetreCommunication c) {
                com = c ;
                }
        public void traiter() {
                if (com.k < com.nblignes) {
                        …  // à compléter
                        }
                }
        public void exit() {
                System.exit(0);
                }
        }

class Adaptateur implements ActionListener, WindowListener  {
        protected Delegue delegue ;
        public Adaptateur(Delegue d) {
                delegue = d ;
                }
        public void actionPerformed(ActionEvent e) {
                delegue.traiter() ;
                }
        public void windowOpened (WindowEvent e) {}
        public void windowClosing (WindowEvent e) { // ferme la fenetre
                delegue.exit();
                }
        public void windowClosed (WindowEvent e) {}
        public void windowIconified (WindowEvent e) {}
        public void windowDeiconified (WindowEvent e) {}
        public void windowActivated (WindowEvent e) {}
        public void windowDeactivated (WindowEvent e) {}
        }
```

Compléter le fichier **Communication.java** afin que chaque clic sur le bouton « Suite » fasse apparaître une nouvelle ligne du fichier **texte.txt.**



Partie 2

Le but est maintenant d'indiquer le nom du fichier à traiter lors du lancement de l'application.
La commande pour la compilation reste la même :
javac Communication.java
La commande pour l'exécution va varier selon le fichier à traiter.
Exemples :
java Communication texte.txt
java Communication poeme.txt

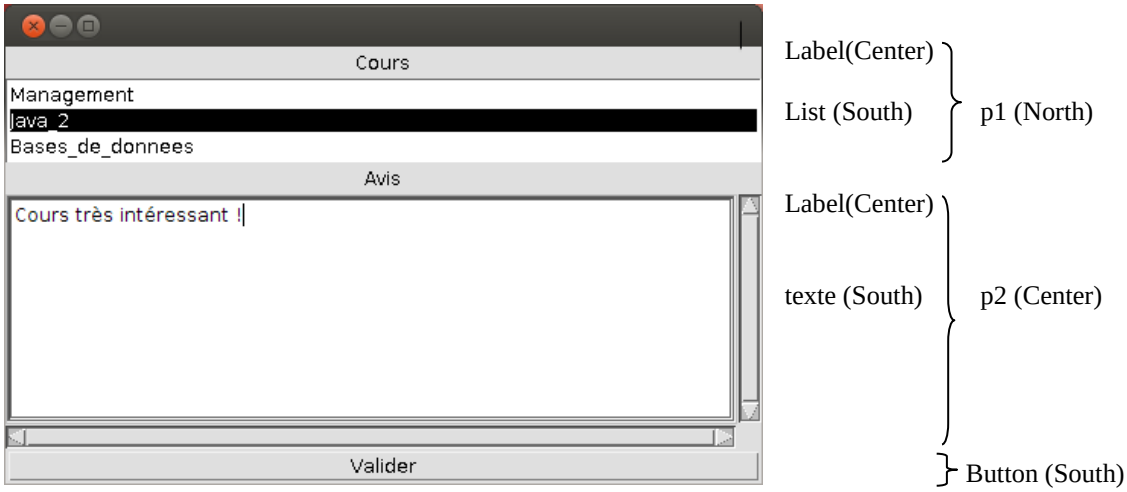
Le nom du fichier ainsi passé en paramètre est le 1er arguement du tableau **args** dans le main : **args[0]**
Modifiez le fichier **Communication.java** pour que le nom du fichier soit pris en compte à l'exécution.
Vérifiez le bon fonctionnement de l'application en testant différents fichiers texte présents dans le dossier **COMMUNICATION.**

TD18 : évaluation des cours

Partie 1

Présentation

On souhaite réaliser une application en Java afin que les étudiants puissent donner un avis sur leurs cours. L'application doit afficher une liste de cours. L'étudiant sélectionne un cours puis rédige un avis. Cet avis va compléter le fichier qui rassemble tous les avis sur ce cours.



Créer un dossier nommé *Eval_cours* dans lequel cette application sera développée. Dans ce dossier, créer un fichier *cours.txt* qui contient le nombre de cours proposés et leur liste. Dans l'exemple présenté ici, seulement 3 cours sont proposés. Le contenu du fichier *cours.txt* est donné ci-contre.

```
3
Management
Java_2
Bases_de_donnees
```

Grâce à un appui sur le bouton *Valider*, l'avis saisi va compléter le fichier adéquat : avis_*Management.txt, avis_Java_2.txt* ou *avis_Bases_de_donnees.txt.*

Questions

Quel est le paquetage nécessaire pour la manipulation de fichiers dans une application Java ?

……

Comment construire un intitulé (objet de la classe Label) avec un alignement « centré » ? (voir les constructeurs de la classe Label)

……

Quel est le constructeur de la classe FileWriter qui permet, à partir d'un nom de fichier, soit de créer un nouveau fichier, soit d'ajouter du texte à un fichier existant ?

……

Squelette de l'application Java

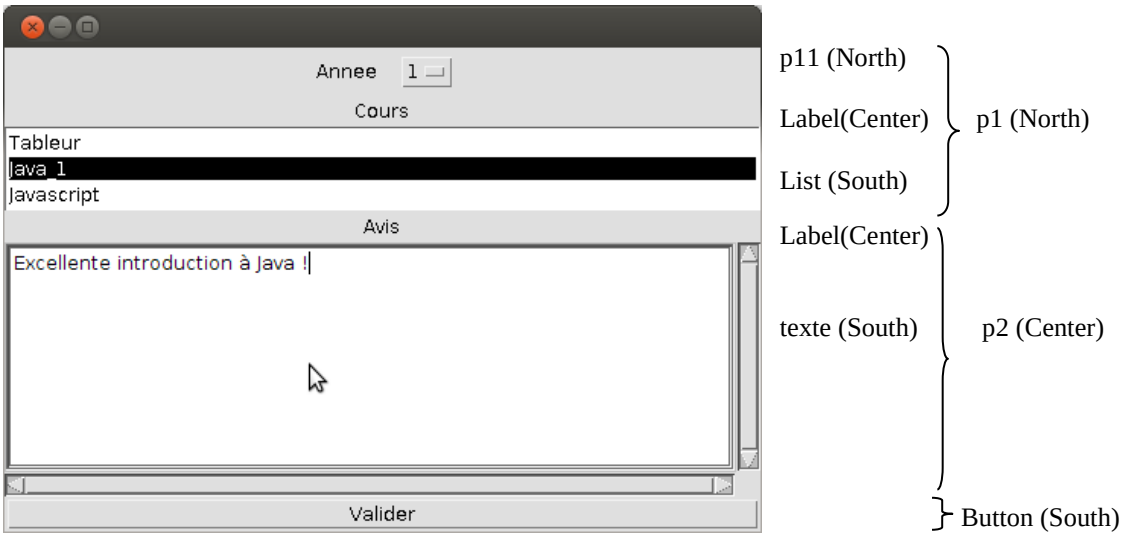
```
import ...                          // les paquetages nécessaires
…
class Traitement extends Frame {
        protected int nbcours; //nb de cours
        protected String [] cours = new String[20] ; //noms des cours (20 au maximum)
        protected Delegue delegue ;
        protected Adaptateur adapt ;
        // déclaration des éléments graphiques nécessaires pour l'interface
        ...
        Traitement() {  //constructeur
                chargement(); //ouverture du fichier cours.txt et récupération des noms des cours
                delegue = new Delegue(this);
                adapt = new Adaptateur(delegue);
                // construction de l'interface
                // prévoir d'attacher l'adaptateur au bouton « Valider »
                …
                }
        public void chargement(){
                BufferedReader fichier ;
                try     {
                        fichier =new BufferedReader(new FileReader("cours.txt"));
                        }
                catch (IOException e) {
                        //afficher dans la zone de texte "erreur de fichier"
                        …
                        return;
                        }
                try     {
                        //grâce au fichier cours.txt, affecter une valeur à nbcours
                        //et les noms des cours au tableau cours
                        …
                        }
                catch (IOException e) {
                        //afficher dans la zone de texte "erreur de transfert "
                        …
                        return;
                        }
                }
        }
public class Eval {
    public static void main(String args[]) {
                //construire et afficher un objet de la classe Traitement
                …
```

```
    }
}
class Delegue {
        protected Traitement ……. ;

        Delegue(Traitement p) {
                …
                }
        public void quitter() {
                …
        }
        public void valider() {
                try
                    {
                        //insertion de l'avis saisi dans le fichier adéquat
                        FileWriter fw = new FileWriter(……..);
                        fw.write (……..);
                        fw.write ("\n-----------------------------------\n"); //pour séparer les différents avis
                        fw.close();
                        }
                catch (IOException e)
                    {
                        //afficher dans la zone de texte "erreur de lors de la lecture"
                        …
                        return ;
                    }
                //nettoyage de la zone de texte en vue d'une prochaine saisie
                ….
                }
        }
class Adaptateur implements …….. {
        protected Delegue delegue ;
        Adaptateur(Delegue d) {
                …
                }
        //méthodes permettant la gestion des événements liés au bouton et à la fenêtre
        …
        }
```

Partie 2

Présentation

Le but est maintenant d'ajouter un objet de la classe Choice qui permet de sélectionner les cours de 1e ou de 2e année. Les cours proposés vont varier selon l'année choisie, et ceci à chaque changement de l'année sélectionnée.



Créer un dossier nommé *Eval_cours_annee* dans lequel cette application sera développée. Dans ce dossier, créer deux fichiers (*cours1.txt* et *cours2.txt*) qui contiennent pour chaque année le nombre de cours proposés et leur liste.

Questions

La classe adaptateur doit implémenter une interface supplémentaire. Laquelle ?

……

Quelle méthode doit être ajoutée à la classe adaptateur ?

……

Quelle méthode doit être utilisée pour connecter l'objet de la classe Choice à l'adaptateur ?

……

Squelette de l'application Java

```
import ...                          // les paquetages nécessaires
…
class Traitement extends Frame {
        protected int nbcours; //nb de cours
        protected String [] cours = new String[20] ; //noms des cours (20 au maximum)
        protected Delegue delegue ;
        protected Adaptateur adapt ;
        //déclaration des éléments graphiques nécessaires pour l'interface
        ...
        Traitement() {
                delegue = new Delegue(this);
                adapt = new Adaptateur(delegue);
                // construction de l'interface
                // prévoir d'attacher l'adaptateur au bouton « Valider » et à l'objet de la classe Choice
                …
                }
        }
```

```
public class Eval {
    public static void main(String args[]) {
            //construire et afficher un objet de la classe Traitement
            …
    }
}
class Delegue {
        protected Traitement …….. ;
        Delegue(Traitement p) {
            ……
            }
        public void quitter() {
            …
        }
        public void changeAnnee() {
            BufferedReader fichier;
            //nettoyage de la liste à chaque changement d'année
            …
            try {
            //ouverture du fichier adéquat
            if (……)
                fichier=….
            else
                …
            //grâce au fichier adéquat, affecter une valeur à nbcours
            //et les noms des cours au tableau cours, puis remplir la liste
            …
            }
            catch (IOException e) {
                //afficher dans la zone de texte "erreur de transfert"
                …
                return;
                }
            }
        public void valider() {
            //insertion de l'avis saisi dans le fichier adéquat et
            //nettoyage de la zone de texte en vue d'une prochaine saisie
            …
            }
        }
class Adaptateur implements …{
        protected Delegue delegue ;
        Adaptateur(Delegue d) {
            …
            }
        //méthodes permettant la gestion des événements liés au bouton, à l'objet de la classe Choice
        //et à la fenêtre
        …
        }
```

Annexe

Gestion des caractères accentués dans les fichiers java

Au besoin compiler avec :

javac –encoding UTF-8 *fichier.java*

L'éditeur NotePad++ utilise le codage UTF-8 par défaut.