Présentation DELPHI

I - H - M

Qu'est ce que Delphi ?

- Delphi est le nom d'un logiciel actuellement largement employé pour créer des logiciels.

- Delphi permet d'utiliser le langage Pascal. Pascal et Delphi NE SONT PAS une seule et même chose :
  - Pascal est un langage informatique,
  - Delphi est un logiciel destiné à créer des logiciels avec ce langage.

- Delphi n'est qu'un enrobage, une enveloppe de confort autour de Pascal, c'est-à-dire qu'il simplifie de nombreuses tâches liées à la programmation en langage Pascal.

- Delphi est destiné à écrire des programmes fonctionnant exclusivement sous Windows.

Qu'est ce que Delphi ?

- Delphi est un environnement graphique de développement et de maquettage rapide.

- La notion de projet est essentielle : un projet permet de regrouper le programme principal ainsi que toutes les *fiches* (fenêtres) et *unités* utilisées.

  ⇒ Notions étudiées dans la suite du cours

- Un *projet Delphi* contient au minimum un programme principal. A la création du projet le programme principal est automatiquement accompagné d'une fiche (une fenêtre vide) et d'une unité (le code associé à la fiche).

Qu'est ce que Delphi ? Un peu de vocabulaire

- **« Programme »** : texte écrit dans un langage informatique, comportant dans notre cas des instructions structurées. Il est destiné à être « converti » par Delphi en un logiciel utilisable sous Windows.

- **« Développer en Delphi »** : écrire des programmes en utilisant Pascal.

- **« Application »** : Logiciel fonctionnant sous Windows.

- **« Projet »** : c'est la base d'une application. Sous Delphi, pour créer une application, on constitue d'abord un projet, constitué de divers morceaux

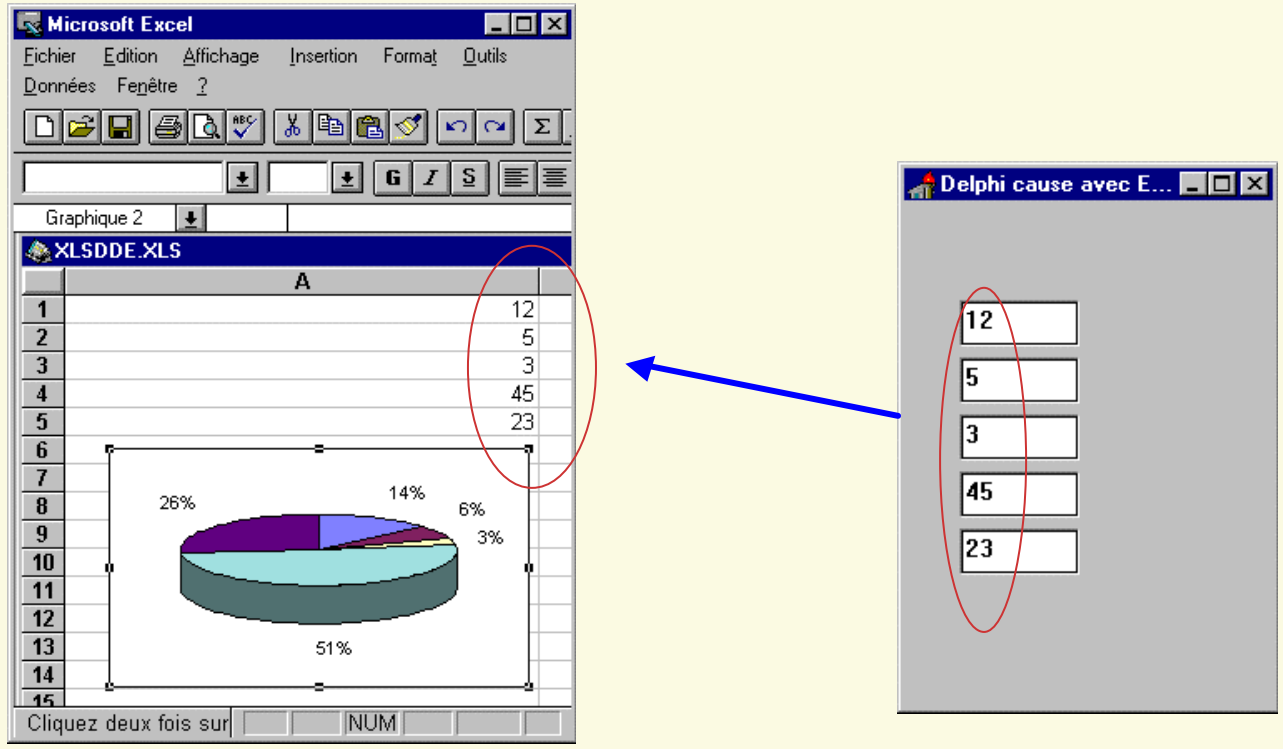
- **« Code », « Code Pascal », « Code source »** : morceau de programme, texte d'un programme écrit en Pascal

- **« Interface (utilisateur) »** : la partie d'un logiciel qui est visible par l'utilisateur, à l'opposé du code source, invisible à l'utilisateur.

- **« Fiche »** : fenêtre à l'état non compilé. Les fiches sont les alter egos sous Delphi des fenêtres sous Windows.

Pourquoi Delphi ? (1)

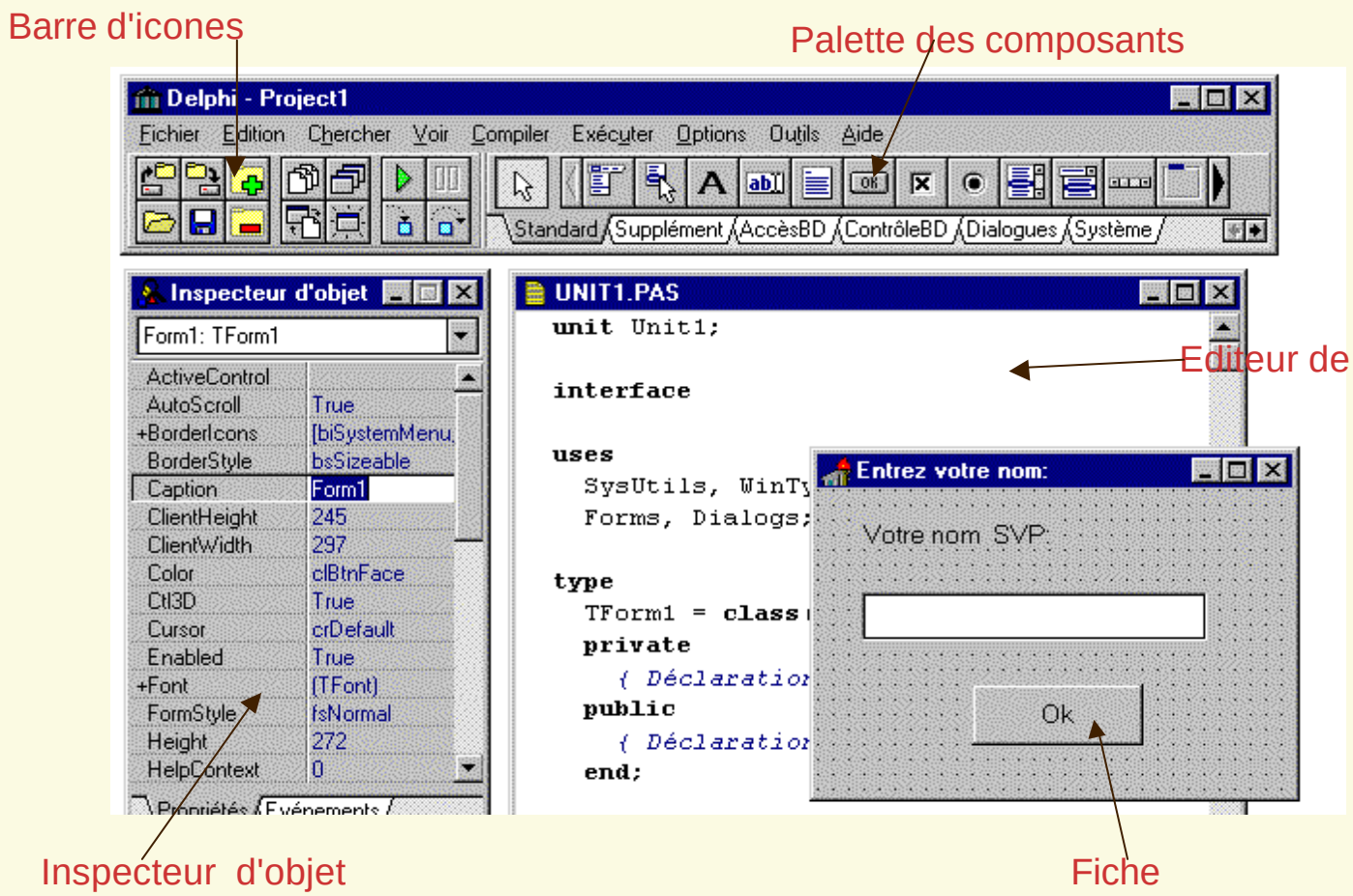
- Amélioration de l'interface utilisateur
- Intégration totale à l'environnement Windows
- Réutilisation et communication avec d'autres programmes Windows



Pourquoi Delphi ? (2)

- Utiliser un environnement de développement visuel similaire aux produits industriels (Visual Basic, Visual C++, Visual Java...mais aussi les formulaires d'Access).
- Conserver le langage Pascal et découvrir ses extensions objets (classes, héritage, polymorphisme).
- Programmer proprement par événements et par exception.
- Accéder facilement aux ressources Windows sans programmation API , ni pointeurs...

L'environnement de Delphi



L'environnement de Delphi

Les logiciels crées sous Windows pourront contenir les effets visuels présents dans les autres applications fonctionnant sous Windows :

- les fenêtres,
- les cases à cocher,
- les boutons,
- les menus,
- les barres d'outils,
- les infos-bulles :

Tout ce qui fait d'une application une application Windows.

L'environnement de Delphi

Delphi permet de créer simultanément les deux aspects interdépendants d'une application :

1. le coté visible (l'interface, pour les utilisateurs du logiciel)
2. le coté invisible (là ou se situe l'intelligence du logiciel) constitué en grande partie de programmes

Quelques notions de base – PLAN de la section

Notions de base de la programmation sous DELPHI

- La notion de projet
- La notion de composant
- Propriétés et événements

Aperçu de l’interface DELPHI

- La barre de menus
- La barre d'outils
- La palette des composants
- L'inspecteur d'objets
- L'éditeur de code

Quelques notions de base (1)

La notion de projet

1. Delphi permet de créer une seule application (un futur logiciel) à la fois, ouverte en tant que projet.
2. Un projet est l'état non compilé d'une application.
3. Chaque projet compilé devient une application.
4. Un projet se compose d'un certain nombre de fichiers et d'options :

Il est utile de consacrer complètement un répertoire (dossier) à chaque projet

Ce répertoire contiendra tous les fichiers constituant le projet (le nombre de fichiers augmentera au fur et à mesure que le projet s'étoffera).

Quelques notions de base (2)

La notion de projet

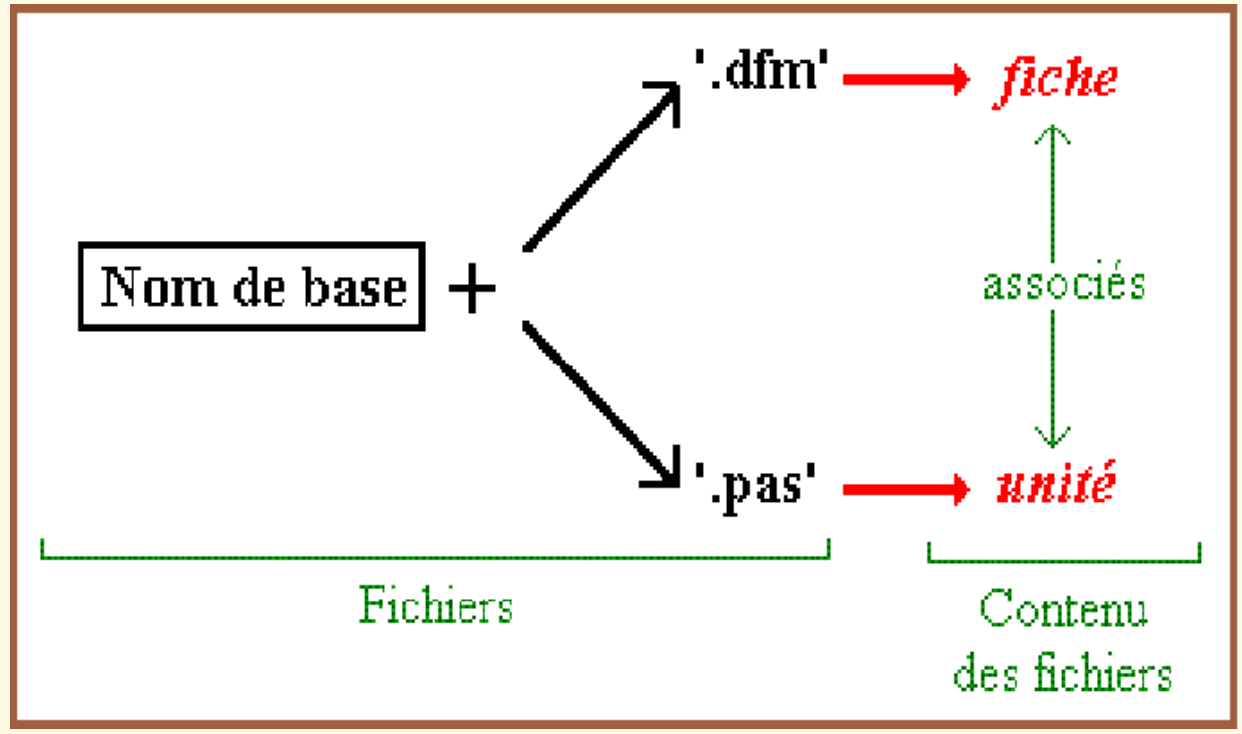
1. Une application Windows est constituée exclusivement de fenêtres. Les logiciels simples peuvent ne contenir qu'une fenêtre ; les plus compliqués peuvent en contenir des dizaines (rarement plus).

2. Un projet non compilé contient ces fenêtres : les fiches.

3. A chaque fiche correspond une (et une seule) unité, c'est-à-dire un texte écrit en langage Pascal, qui contiendra tout ce qui se rapporte à cette fiche : ce qu'elle contient :
   - boutons, menus, cases à cocher, …
   - ce qui doit se passer dans certaines situations (lorsqu'on clique sur un bouton par exemple)
   -…

Quelques notions de base (3)

La notion de projet



Quelques notions de base (4)

La notion de projet : remarques

1. Il est possible d'utiliser des unités qui n'ont pas de fiche associée : pour rassembler des morceaux de programme qui n'ont aucun rapport avec une quelconque fiche : les algorithmes de calcul mathématique, les structures de données et leurs procédures associées…

2. Un projet sous Delphi est constitué d'un fichier-projet ('DPR'), d'unités et de fiches.

3. Le fichier qui contient une unité porte l'extension 'PAS‘.

4. A la compilation du projet, d'autres fichiers seront créés :
   - des fichiers DCU (forme compilée des 'PAS' et 'DFM‘)
   - un fichier 'EXE' si la compilation est aboutie.

Quelques notions de base (5)

Extension du fichier	Description et Commentaires
DPR	(**D**elphi **PR**oject) Contient l'unité principale du projet
PAS	(**PAS**cal) Contient une unité écrite en Pascal. Peut avoir un .DFM correspondant
DFM	(**D**elphi **F**or**M** : fiche Delphi) Contient une fiche (une fenêtre). Le .PAS correspondant contient toutes les informations relatives au fonctionnement de cette fiche, tandis que le .DFM contient la structure de la fiche (ce qu'elle contient, sa taille, sa position, …).

Quelques notions de base (5)

Extension du fichier	Description et Commentaires
DCU	(**D**elphi **C**ompiled **U**nit : Unité compilée Delphi) Forme compilée et combinée d'un .PAS et d'un .DFM optionnel
~???	Tous les fichiers dont l'extension commence par ~ sont des fichiers de sauvegarde, pouvant être effacés pour faire place propre.
EXE	Fichier exécutable de l'application. Ce fichier est le résultat final de la compilation et fonctionne sous Windows exclusivement.
RES	(**RES**source) Fichier contenant les ressources de l'application, tel son icône..
DOF **DSK** **CFG**	Fichiers d'options : suivant les versions de Delphi, ces fichiers contiennent les options du projet, les options d'affichage de Delphi pour ce projet, ...

PLAN de la section

Notions de base de la programmation sous DELPHI

- La notion de projet
- La notion de composant
- Propriétés et événements

Aperçu de l’interface DELPHI

- La barre de menus
- La barre d'outils
- La palette des composants
- L'inspecteur d'objets
- L'éditeur de code

Les composants de base

- Delphi permet de créer des programmes et d'utiliser des éléments prédéfinis :

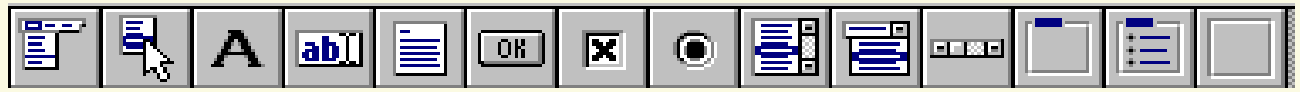
  Les fiches ,
  les boutons,
  les cases à cocher,
  les zones d'édition,
  les menus,
  les barres d'outils,
  les listes, …

- Chaque élément, à l'exception des fiches est accessible dans la palette des composants.

- Ces composants seront placés, dimensionnés, réglés un par un sur les fiches pour constituer une interface utilisateur.

Les composants de base

- la palette standard propose les composants d'interfaces Windows les plus courants:



•TmenuBar	Menu principal d'une fiche
•TPopUpmenu	Menu surgissant
•Tlabel	Etiquette (zone d'affichage de texte)
•TEdit	Zone de saisie de texte
•TButton	Bouton
•Tmemo	Zone de saisie multiligne
•TCheckBox	Case à cocher
•TradioButton	Bouton radio
•TListBox	Liste d'éléments textuels
•TComboBox	Zone d'édition plus liste d'éléments
•TScrollBar	Barre de défilement horizontale ou verticale
•TGroupBox	Boite de regroupement de contrôles
•TradioGroup	Groupe de boutons radio
•TPanel	Panneau, zone d'écran encadrée

PLAN de la section

Notions de base de la programmation sous DELPHI

- La notion de projet
- La notion de composant
- Propriétés et événements

Aperçu de l’interface DELPHI

- La barre de menus
- La barre d'outils
- La palette des composants
- L'inspecteur d'objets
- L'éditeur de code

Propriétés et événements

- L'inspecteur d'objets : « Propriétés » et « Evénements ».

(définis pour tout « composant » accessible dans la barre de composants ainsi qu'aux fiches)

- Chaque fiche, chaque composant possède une liste de propriétés et une liste d'évènements.

- Les propriétés sont des paramètres réglables pour un composant :

les dimensions,
les couleurs,
les polices,
le titre d'une fenêtre,
le texte d'un bouton...

Propriétés et événements

- Les évènements : provoqués à chaque action sur le logiciel

clics, mouvements de souris,
touches frappées au clavier …

- D'autres sont provoqués lorsque :

une fenêtre devient visible, invisible
une case à cocher est cochée,
un élément est sélectionné (liste)...

- La presque totalité des composants déclenchent des évènements pendant l'exécution du logiciel.

PLAN de la section

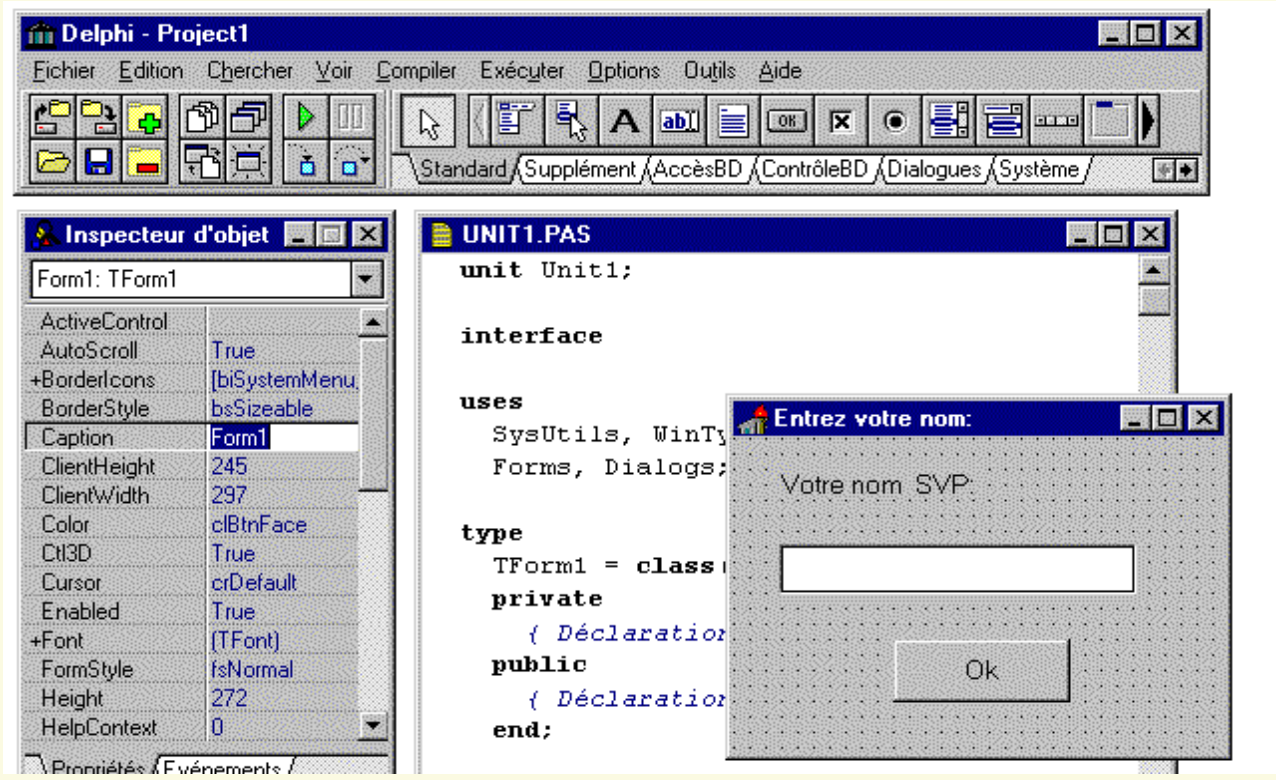
Notions de base de la programmation sous DELPHI

- La notion de projet
- La notion de composant
- Propriétés et événements

Aperçu de l’interface DELPHI

- La barre de menus
- La barre d'outils
- La palette des composants
- L'inspecteur d'objets
- L'éditeur de code

Barres de menus, outils et palette de composants



PLAN de la section

Notions de base de la programmation sous DELPHI

- La notion de projet
- La notion de composant
- Propriétés et événements

Aperçu de l’interface DELPHI

- La barre de menus
- La barre d'outils
- La palette des composants
- L'inspecteur d'objets
- L'éditeur de code

L’inspecteur d’objet

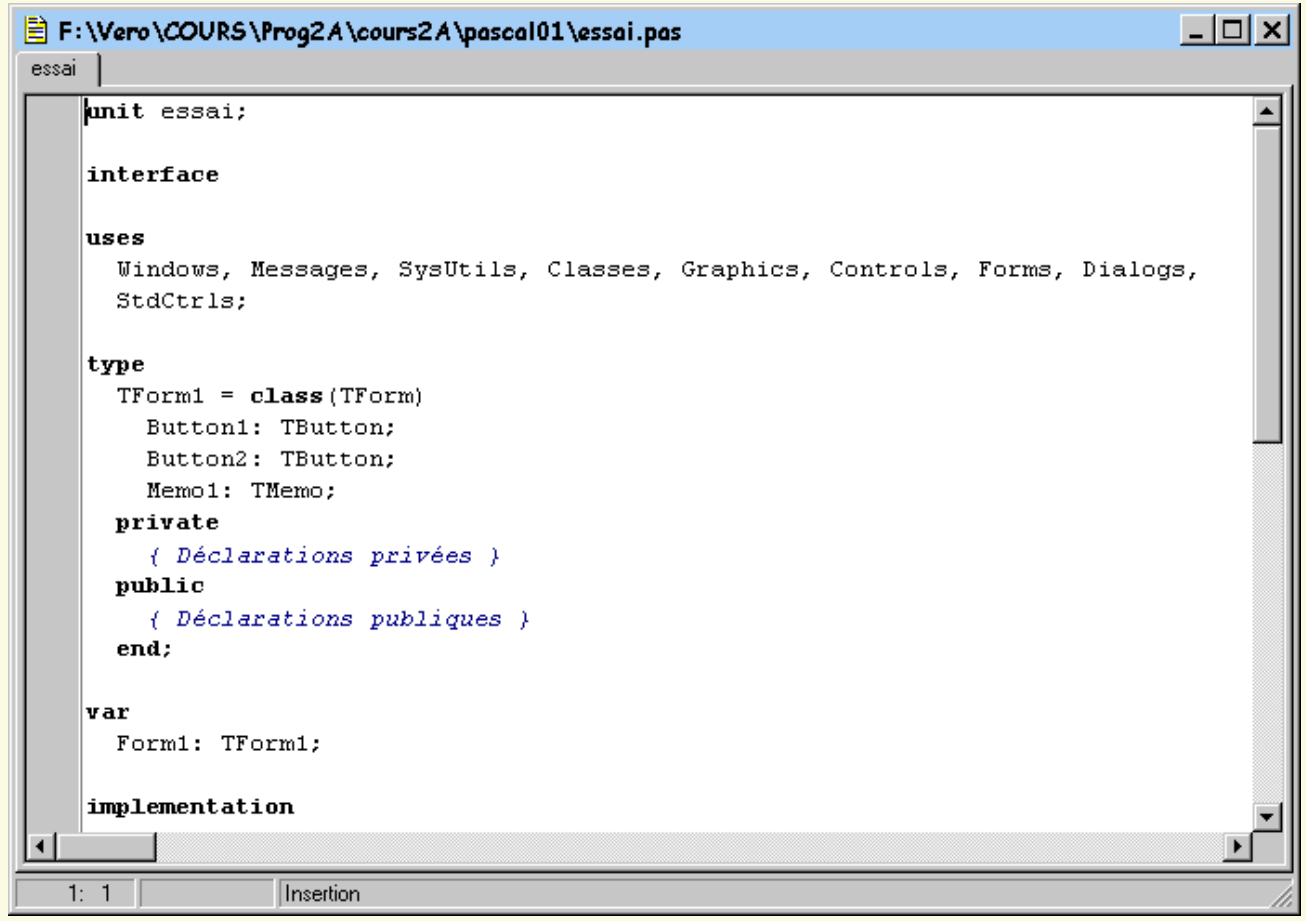
PROPRIETES REMARQUABLES

- Il permet de modifier les propriétés et d'accéder aux événements des composants et des fiches.
- Utilisé lors de la conception de l'interface des applications
- Se compose d'une liste déroulante (combo), listant les composants présents sur une fiche, ainsi que cette fiche.
- Les propriétés et événements d'un élément sont classés dans les deux onglets 'Propriétés' et 'Evénements'.
- Pour éditer un composant ou la fiche dans l'inspecteur d'objets, il suffit de cliquer dessus dans la fiche correspondante

L’éditeur de code

- L'éditeur de code contient les instructions Pascal.
- Il présente une série d'onglets, qui donnent en titre le nom de l'unité éditée actuellement.
- La relation entre une fiche et son unité est non modifiable.

L’éditeur de code



PLAN de la section

Manipulation de la syntaxe OBJET

- Sous Delphi, la syntaxe est objet
- Exemple sur les complexes
- Application aux objets graphiques de DELPHI

La programmation par événement

- Principe général sur un exemple
- « Mode d'emploi »

Sous DELPHI, la syntaxe est objet

- Un objet est une donnée possédant une *structure* complexe.

- Un objet peut utiliser des variables Pascal, comme cela se fait déjà avec les autres données classiques telles les chaînes de caractères ou les pointeurs.

  - Ces variables contiennent des données et du code Pascal permettant de les traiter.
  - On peut y stocker des *données* mais aussi des *instructions Pascal*
  - Les données consistent en des variables de n'importe type (y compris des objets).
  - Le code Pascal Objet est réparti dans des procédures et fonctions nommées *méthodes*. (cf. les méthodes liées aux composants de base).

Sous DELPHI, la syntaxe est objet

- Les méthodes et variables contenues dans les objets sont complétés par d'autres éléments.

  - Parmi ces éléments particuliers, figurent les propriétés et deux méthodes aux dénominations un peu barbares puisqu'on les appelle constructeur et destructeur.

Sous DELPHI, la syntaxe est objet

- Une classe est une représentation informatique d'un objet du monde réel ou d'un concept (classe = extension type)
- Elle modélise:
  - ses caractéristiques (couleur, taille, visible, actif...) par des variables ou propriétés ou attributs.
  - son comportement par des méthodes (procédures, fonctions ou gestionnaires d'événements).
- On dit qu'une classe ENCAPSULE données et procédures
- On crée un objet en l'instanciant tout comme une variable classique (objet = extension variable)

```
Var MonBouton: c_Button;
```

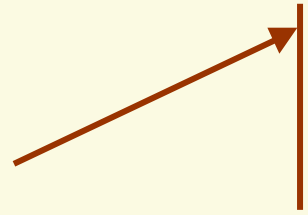
- N.B. Les informations d'une classe peuvent être:
  - Nouvelles pour cet objet (à déclarer)
  - Héritées directement d'un ancêtre (à ne pas déclarer)
  - Surchargées pour modifier le comportement de l'ancêtre (à redéclarer)

Sous DELPHI, la syntaxe est objet

- Les variables manipulées dans un programme sont des objets. Les *classes* sont les *types* permettant de déclarer ces variables.

Là où auparavant une variable était d'un type donné, un objet sera dit d'une classe donnée.

```
var
    S: string;
    Button1: TButton;
```



Classe TButton (cf. palette des composants)

Déclarer un objet se fait de la même manière que pour une variable, à ceci près qu'on le déclare en utilisant une classe et non un type classique

Sous DELPHI, la syntaxe est objet

- Une classe détermine entièrement la structure des objets de cette classe.
- La classe définit les méthodes et les données (mais pas leurs valeurs) contenues dans les futurs objets de cette classe.
- Exemple du moule à gâteaux.

- *Les objets sont les gâteaux qu'on peut réaliser à partir de ce moule.*
- *Le moule définit entièrement la forme du gâteau, c'est la même chose pour les classes : elles déterminent entièrement la structure des objets de cette classe.*
- *Le moule ne contrôle pas le contenu (les données : tarte aux poires ou aux pommes !!). Ces valeurs sont indépendantes de la classe.*

Sous DELPHI, la syntaxe est objet

- **Utilisation des objets**

  Les objets ne s'utilisent pas exactement comme les autres variables. Ils nécessitent des opérations spécifiques :

- *La Construction et destruction*

  Les objets nécessitent deux étapes particulières avant et après leur utilisation. Un objet se construit, s'utilise puis se détruit.

  La construction et la destruction se font par deux instructions.

- Illustration sur une classe proposée par Delphi : la classe TStringList.

  Pour la manipulation de chaînes de caractères.

Sous DELPHI, la syntaxe est objet

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    Lst: TStringList;
  begin
    Lst := TStringList.Create;
    Lst.Destroy;
  end;
```

- La procédure ci-dessus construit l'objet Lst.
- « create » est le constructeur de la classe TStringList (sans paramètre).
- A ce moment,l'objet est utilisable.
- La destruction est effectuée pour cette fois immédiatement après la construction.
- La destruction est un simple appel au destructeur.

Sous DELPHI, la syntaxe est objet

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    Lst: TStringList;
  begin
    Lst := TStringList.Create;
    Lst.Add('Bonjour !');
    Lst.Destroy;
  end;

**************************************************

procedure TForm1.Button1Click(Sender: TObject);
  var
    Lst: TStringList;
  begin
    Lst := TStringList.Create;
    ShowMessage(IntToStr(Lst.Count));
    Lst.Add('Bonjour !');
    ShowMessage(IntToStr(Lst.Count));
    Lst.Destroy;
  end;
```

Sous DELPHI, la syntaxe est objet

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    Lst: TStringList;
  begin
    Lst := TStringList.Create;
    Lst.Add('Bonjour !');
    ShowMessage(Lst[0]);
    Lst.Destroy;
  end;
```

Sous DELPHI, la syntaxe est objet

```
procedure TForm1.Button1Click(Sender: TObject);
  var
    Lst: TStringList;
    nb_ligne: integer;
  begin
    Lst := TStringList.Create;
    Lst.Add('Bonjour !');
    Lst.Add(‘Comment allez-vous ?');
    if Lst.Count > 0 then
      for nb_ligne := 0 to Lst.Count - 1 do
        ShowMessage(Lst[nb_ligne]);
    Lst.Destroy;
  end;
```

Exemple sur les Complexes

- Traitement en Pascal "classique"

```
Type t_complexe = Record
                    r : real;
                    i : real;
                  End;
Procedure Ajouter (C1, C2 : t_complexe; VAR C3 : t_complexe);
Procedure Afficher (C : t_complexe);
….
VAR C1, C2, C3 : t_complexe;
….
Ajouter (C1, C2, C3);
Afficher (C3);
…..
```

Exemple sur les Complexes

- Traitement en Pascal "Objet" = Delphi

```
c_complexe = Class
    r : real;
    i : real;
    Procedure Ajouter (C1: c_complexe; VAR C3 : c_complexe);
    Procedure Afficher ;
End;
....
VAR C1, C2, C3 : c_complexe;
....
C1.Ajouter ( C2, C3);
C3.Afficher;
.....
```

Les ATTRIBUTS de la classe

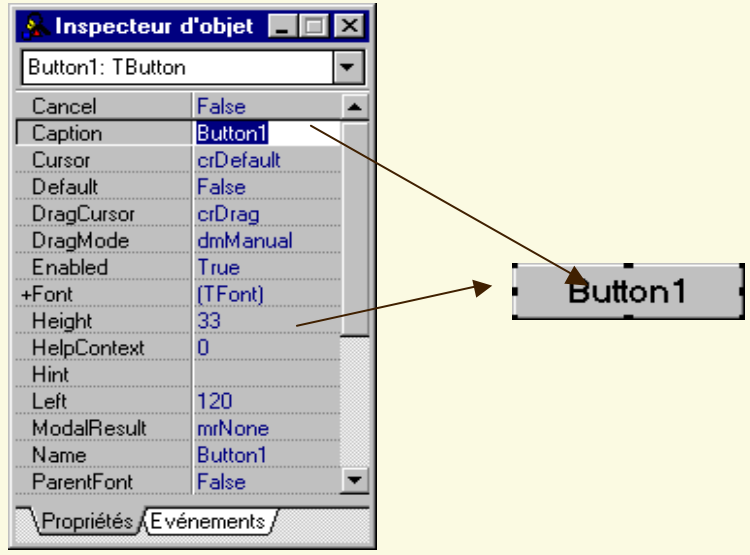
Les METHODES de la classe

Instanciation de 3 objets de la classe

Envoi de message :
Application d'une méthode à un objet

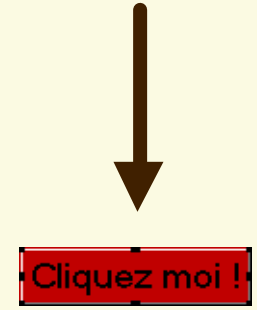
Application aux objets graphiques de Delphi

- Modification des propriétés à la CONCEPTION



- Modification des propriétés à l' EXECUTION

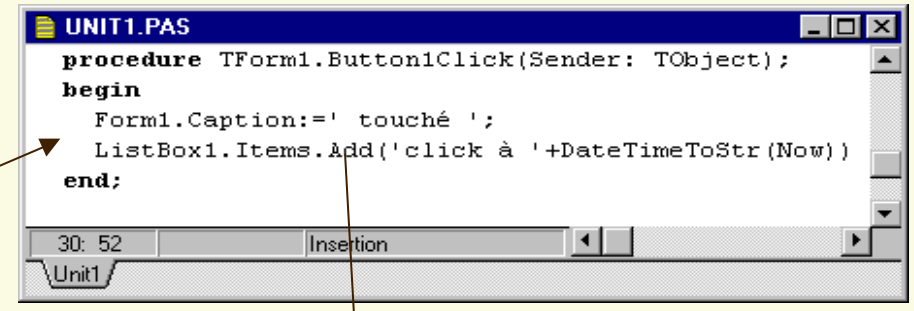
```
VAR Bouton : tButton;

Bouton.Color:=clRed;
Bouton.Caption:='Cliquez moi !';
Bouton.Font.Size:=14;
Bouton.SetBounds(1,1,40,10);
Bouton.Click (Self);
```

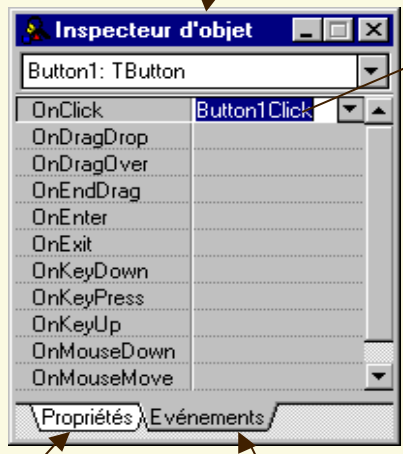


La programmation par événements (1)

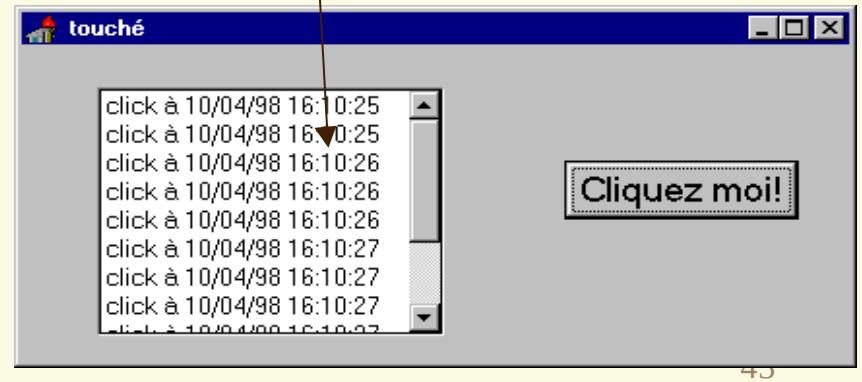
(1) A la conception on définit dans l'inspecteur d'objets que faire sur un événement comme un click de la souris....

(2) Delphi génère automatiquement la procédure





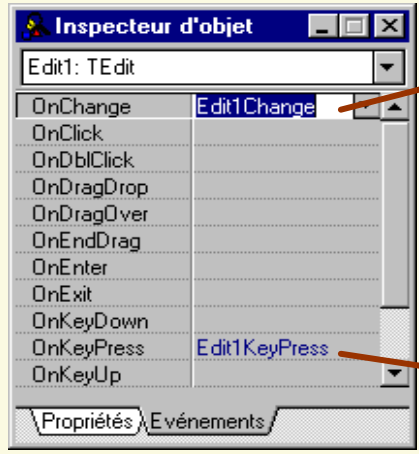
(3) que l'on remplit avec le code des actions à réaliser à l'exécution

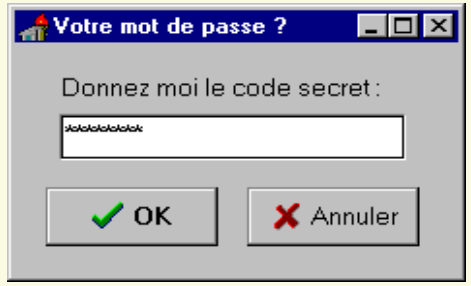


les propriétés de l'objet Button1

les événements de l'objet Button1

La programmation par événements (2)






```
procedure TForm1.Edit1Change(Sender: TObject);
begin
  ButOk.Enabled:=Edit1.Text <>''
end;

procedure TForm1.Edit1KeyPress(Sender: TObject; var
Key: Char);
begin
  (* seulement les touches numériques *)
  If Not (Key in ['0'..'9',#8]) then key:=char(0);
end;

procedure TForm1.BitBtn2Click(Sender: TObject);
begin
  Close
end;
```

Comment on fait ?

- Etape 1: Dessiner l'interface utilisateur (fiches, contrôles…)
  - Déposer les composants à partir de la palette
  - Définir les propriétés en mode conception avec l'inspecteur d'objets

- Etape 2: Pour chaque fiche définir un comportement
  - Est elle modale (bloquante) ou non ? ShowModal ou Show
  - que faire à l'ouverture de la fiche ? événement OnCreate
  - que vérifier à la fermeture de la fiche ? événement OnClose
  - que faire sur un clic souris ? événement OnClicK

- Etape 3: Pour chaque fiche définir ses propriétés
  - Publique ou privée, lecture et/ou écriture

- Etape 4: Insérer du code TurboPascal standard (procédures, fonctions unités...)

Structure d'un programme DELPHI (1)

- **Structure d'un fichier projet (.dpr)**

L'ensemble du texte du fichier est généré par Delphi.

Exemple de fichier :

PremierEssai.dpr

```
program PremierEssai;

  uses
    Forms,
    Principale in 'Principale.pas' {Form1};

  {$R *.RES}

  begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
  end.
```

Structure d'un programme DELPHI (2)

- **Structure d'un fichier projet : illustration sur un exemple**

- Le second bloc sert de connexion entre les différentes unités et le fichier- Projet:

```
Uses
      Unit1,
      Unit2 in 'Unit2.pas',          { liste d'unités privées du « .pas »}
      Unit3 in 'Unit3.pas'  {Form1};
```

Unit2 et Unit3 font partie du projet ; Unit1 non

Unit3 est l'unité de la fiche dont le nom est Form1 (propriété *Name* de la fiche)

- Ces unités peuvent faire partie du projet ou pas.

Structure d’un programme DELPHI (2)

- **Structure d’un fichier projet : illustration sur un exemple**

- Les directives de compilation : « *{$R *.RES}* »

- Le BEGIN…END. : permet l'exécution de l'application.
  Il y a trois instructions qui sont :

```
1. Application.Initialize
2. Application.CreateForm(TForm1, Form1)
3. Application.Run
```

Les Unités en Delphi

- Squelette général

```
UNIT NomUnité;
INTERFACE
    Déclaration des objets publiques (utilisables par les autres unités)
    – constantes, types, classes, variables
    – entête des procédures et fonctions
IMPLEMENTATION
    – Déclaration des objets privés (utilisables dans cette seule unité)
    – Code de toutes les procédures et fonctions (publiques et privées)
END;
```

Les Unités en Delphi

- **Squelette général : un exemple**

```
unit Principale;
  interface
  uses
    Windows, Messages, SysUtils, Classes, Graphics,
    Controls, Forms,Dialogs, StdCtrls;

  type
    TForm1 = class(TForm)
    Button1: TButton;
    private     { Private declarations }
    public      { Public declarations }
    end;

  var
    Form1: TForm1;
  implementation
  {$R *.DFM}
  end.
```

Utilisation d'Unités

- Syntaxe

  Uses NomUnité;

- Place de la déclaration

  UNIT NomUnité;

  INTERFACE

  Uses Liste1_d_Unites;

  – Unités utiles dans les déclarations

  IMPLEMENTATION

  Uses Liste2_d_Unites;

  – Unités utiles dans les instructions

  END;

- ATTENTION

  A la récursivité croisée dans les utilisations

Utilisation d'Unités

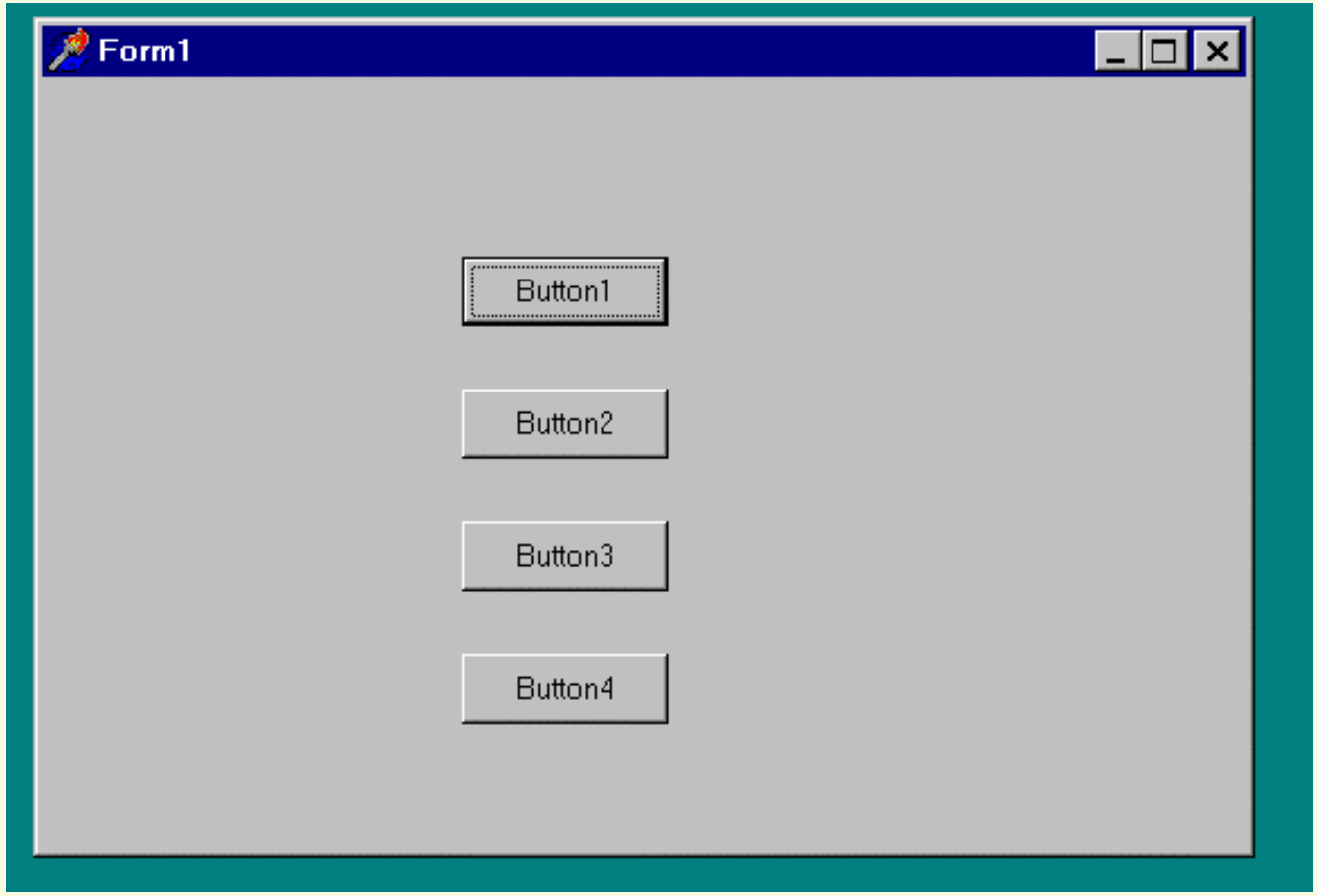
```
UNIT Unit1;
INTERFACE
    Uses Unit2;
    VAR F1 : …
IMPLEMENTATION
    Uses Unit2;
    F1:= ...
    F2:= …
END.
```

```
UNIT Unit2;
INTERFACE
    Uses Unit1;
    VAR F2 : …
IMPLEMENTATION
    Uses Unit1;
    F1:= ...
    F2:= …
END.
```

- Ou mettre les Uses ?

Un exemple

- Fiche principale attendue



Un exemple

- Les déclarations équivalentes

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
```

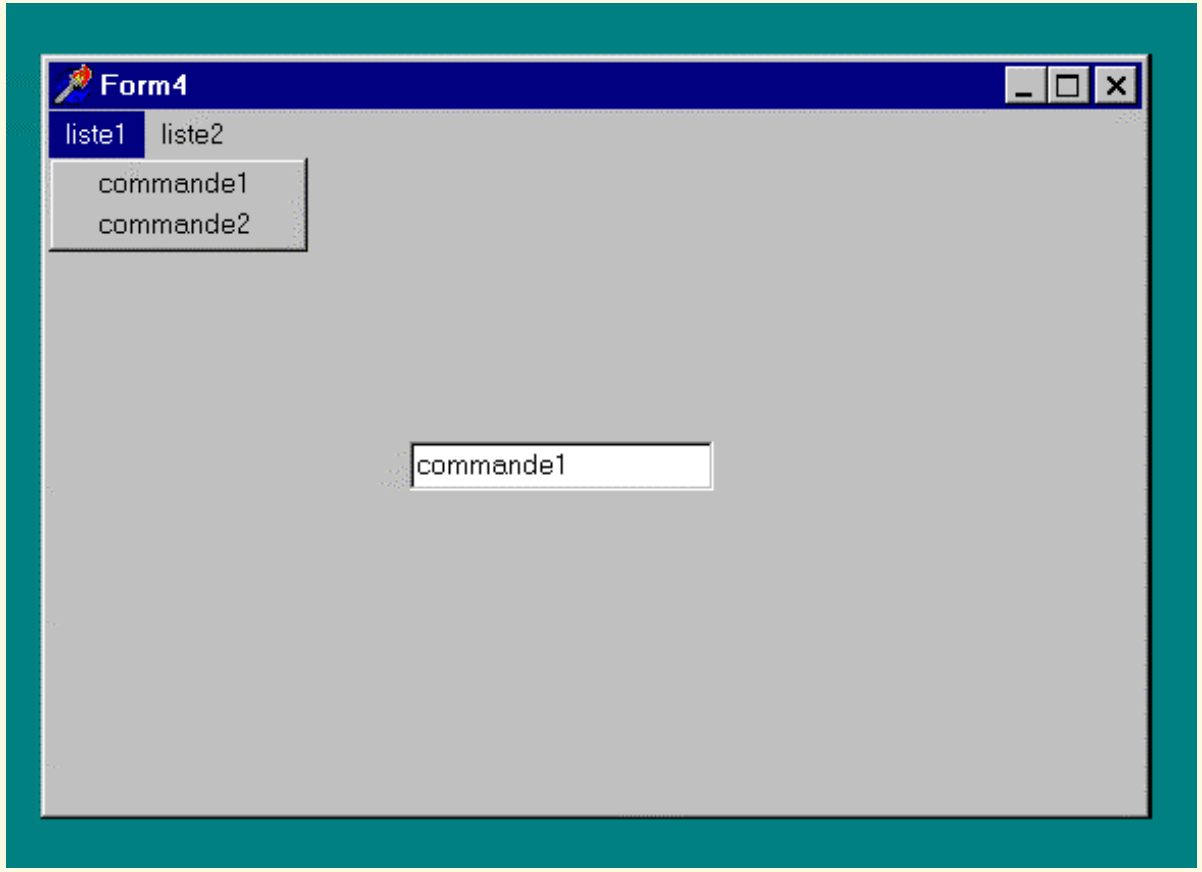
Un exemple

- Le code équivalent du clic sur bouton1

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    Form4.show
 {Form4 devient visible et peut être utilisée en même temps que
    Form1}
end;
```

Un exemple

- L'écran attendu pour la Form4



Un exemple

- Les déclarations équivalentes

```
type
  TForm4 = class(TForm)
    MainMenu1: TMainMenu;
    menu11: TMenuItem;
    commande11: TMenuItem;
    commande21: TMenuItem;
    liste21: TMenuItem;
    commande31: TMenuItem;
    commande41: TMenuItem;
    Edit1: TEdit;
    procedure commande11Click(Sender: TObject);
    procedure commande21Click(Sender: TObject);
    procedure commande31Click(Sender: TObject);
    procedure commande41Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
```

Un exemple

- Le code équivalent

```
procedure TForm4.commande11Click(Sender: TObject);
begin
   edit1.text:='commande1'
end
procedure TForm4.commande21Click(Sender: TObject);
begin
   edit1.text:='commande2'
end;
procedure TForm4.commande31Click(Sender: TObject);
begin
   edit1.text:='commande3'
end;
procedure TForm4.commande41Click(Sender: TObject);
begin
   edit1.text:='commande4'
end;
```

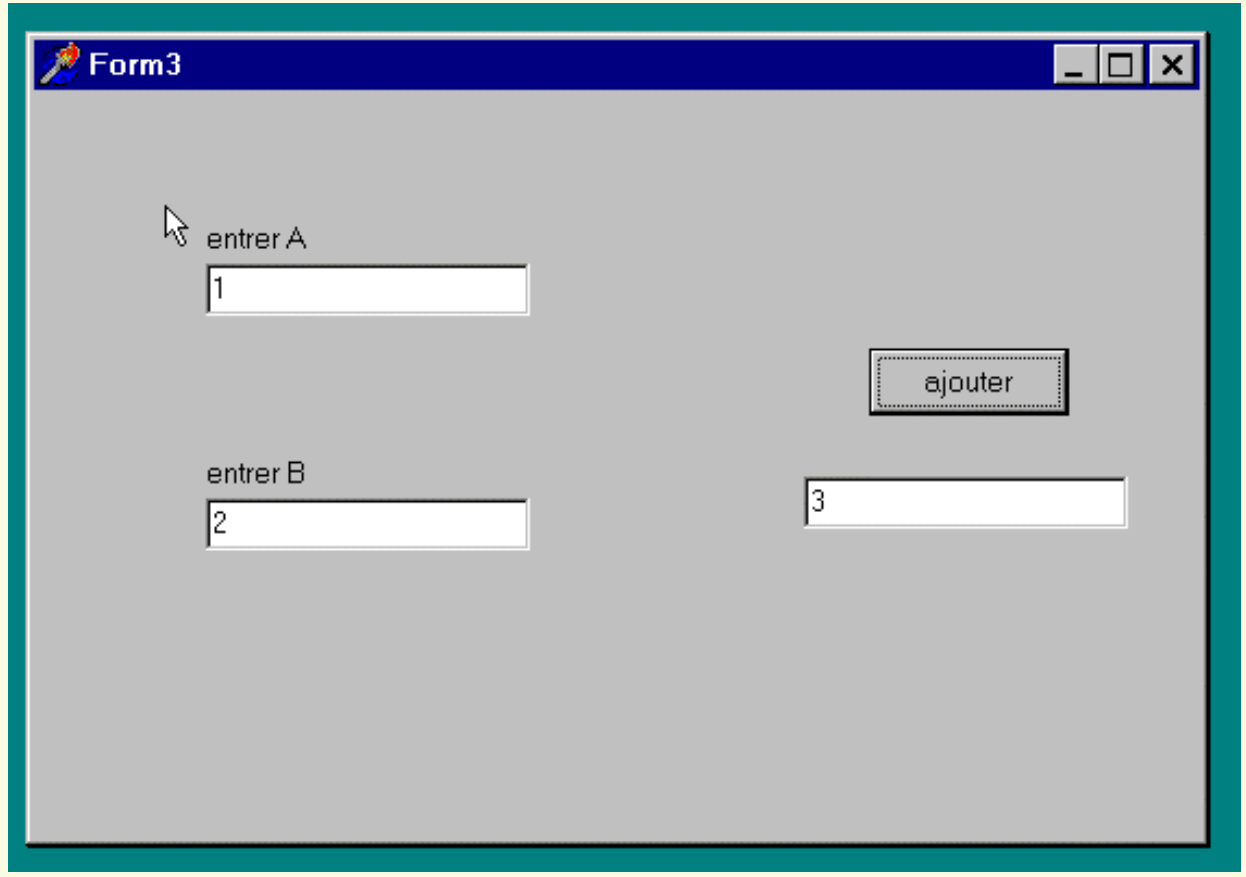
Un exemple

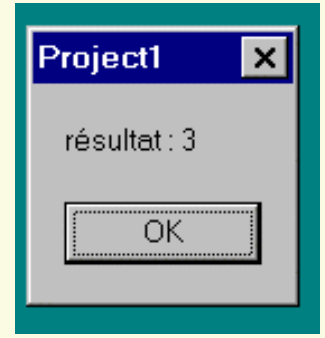
- Le code équivalent du clic sur bouton2

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   Form3.showModal;
{  Form3 devient visible et peut être utilisée.
   Mais Form1 ainsi que les autres fenêtres de l'application déjà
   ouvertes ne peuvent plus être utilisées tant que Form3 n'a pas été
   fermée}
{  l'exécution ne reprend ici que lorsque Form3 a été fermée}
   showMessage('résultat : '+form3.edit3.text)}
end;
```

Un exemple

- L'écran attendu par Form3





Un exemple

- Les déclarations équivalentes

```
type
  TForm3 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
```

Un exemple

- Le code équivalent

```
procedure TForm3.Button1Click(Sender: TObject);
begin
   edit3.text:=FloatToStr(strToFloat(edit1.text)+strToFloat(edit2.text))
end;
```

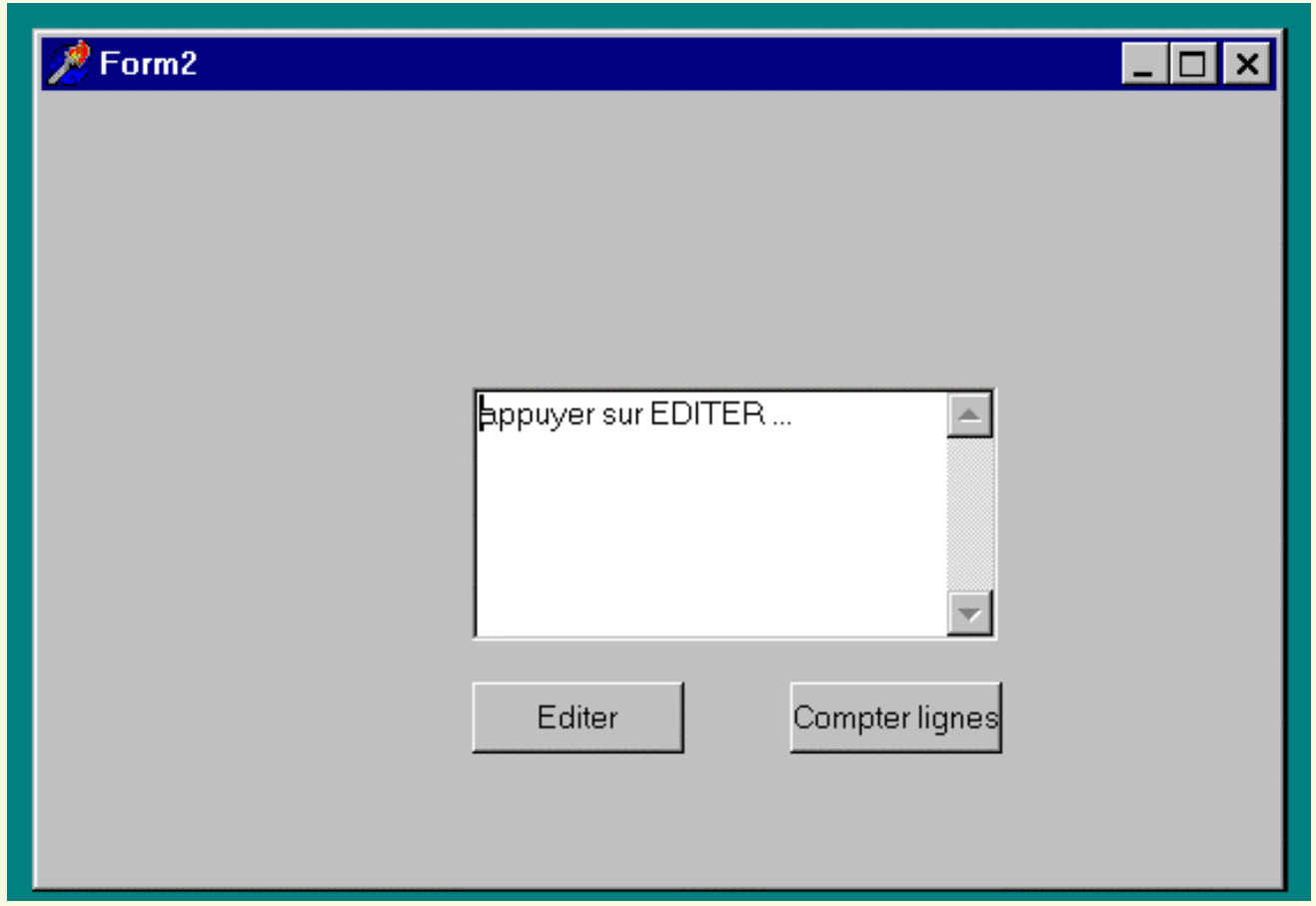
Un exemple

- Le code équivalent du clic sur bouton3

```
procedure TForm1.Button3Click(Sender: TObject);
begin
   form2.memo1.text:='appuyer sur EDITER ...';
   form2.show
end;
```

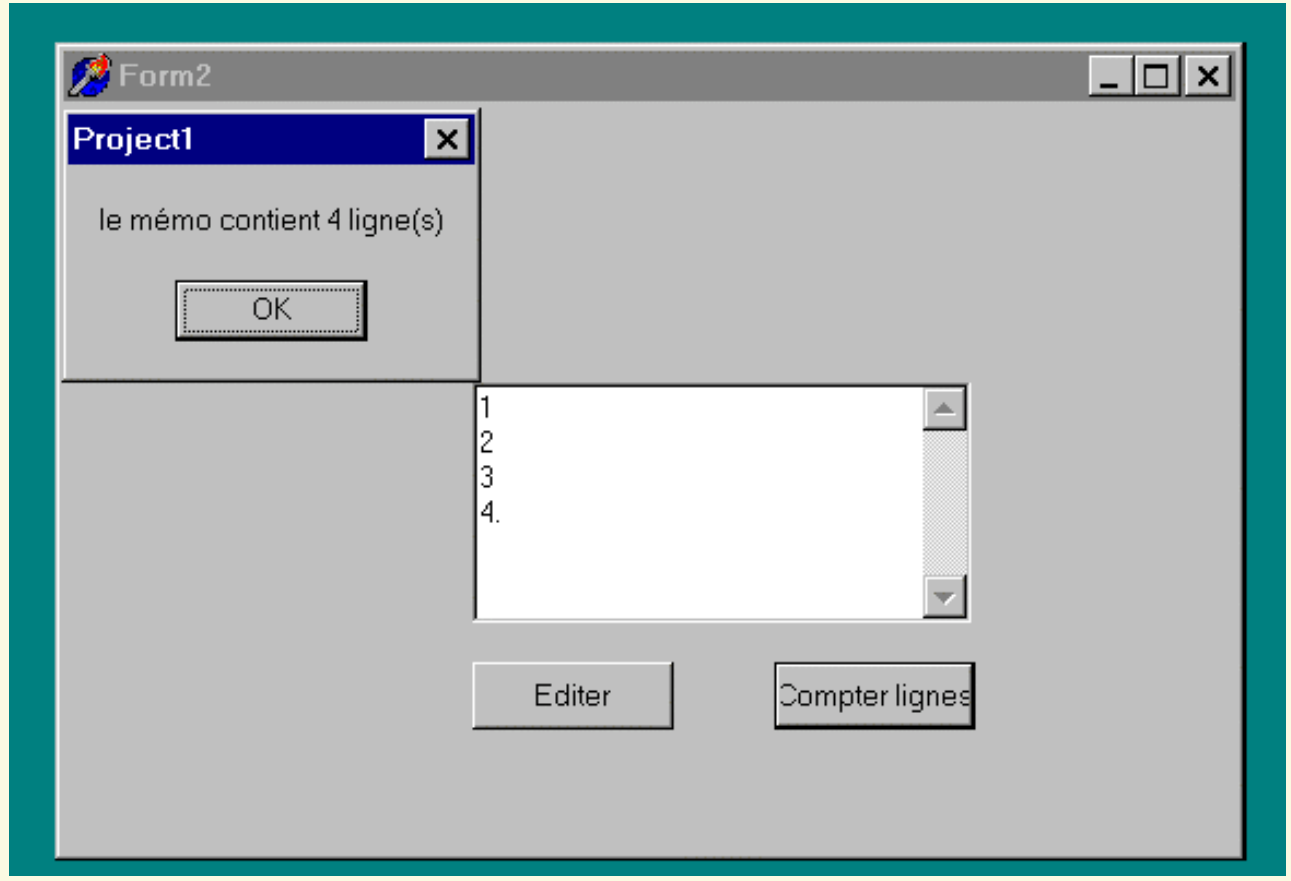
Un exemple

- L'écran attendu pour Form2 (initialisation)



Un exemple

- L'écran attendu pour Form2 (utilisation)



Un exemple

- Les déclarations équivalentes

```
type
  TForm2 = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
```

Un exemple

- Le code équivalent

```
procedure TForm2.Button1Click(Sender: TObject);
begin
    memo1.ReadOnly := False;
end;
procedure TForm2.Button2Click(Sender: TObject);
var i, nbLignes:integer;
begin
    if length(memo1.text)=0 then nbLignes:=0
    else begin
      nbLignes:=1;
      i:=1;
      while i<length(memo1.text) do begin
           if ord(memo1.text[i])=13 then nbLIgnes:=nbLignes+1;
           i:=i+1
       end;
    end;
    showMessage('le mémo contient '+intToStr(nbLignes)+' ligne(s)')
end;
```

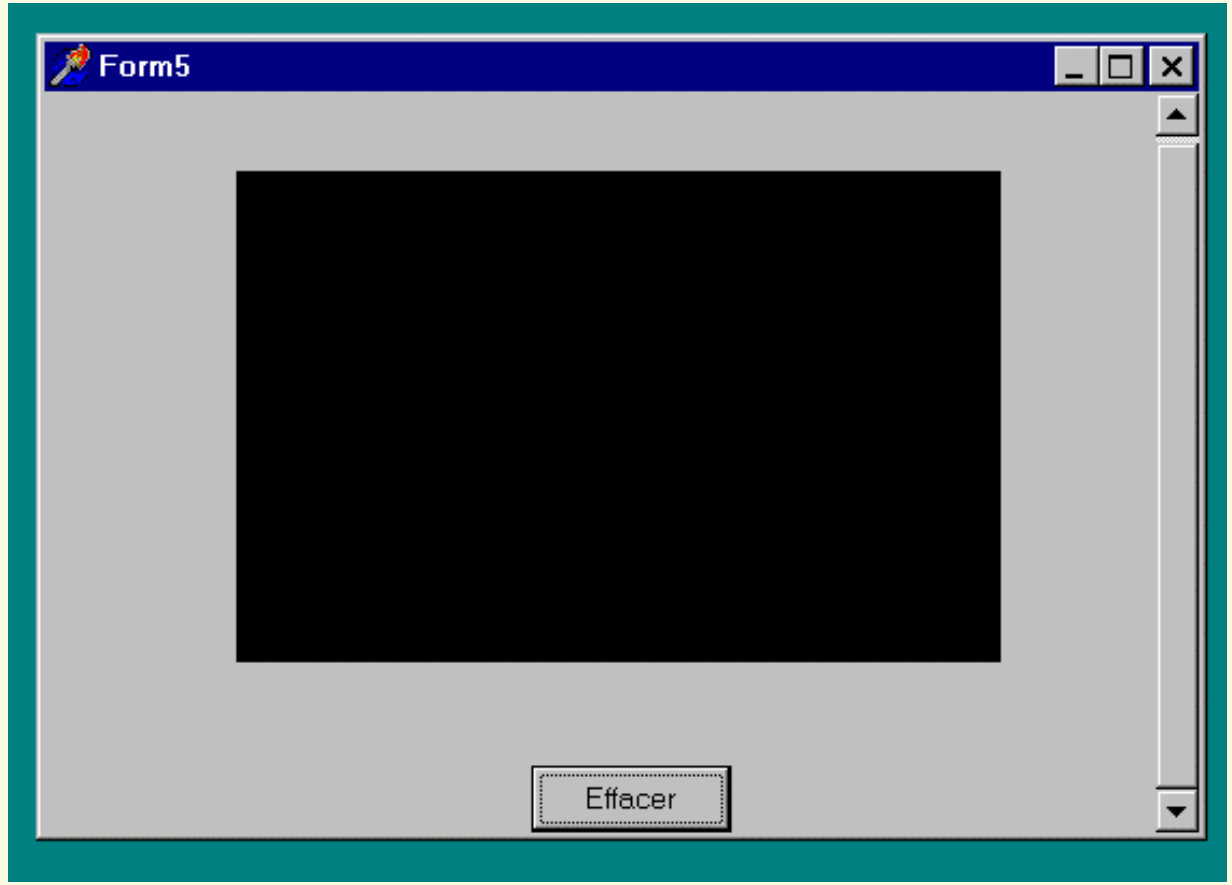
Un exemple

- Le code équivalent du clic sur bouton4

```
procedure TForm1.Button4Click(Sender: TObject);
begin
    Form1.hide;        {avant de rendre Form5 visible, on cache Form1}
    Form5.showModal;
    Form1.show         {puis on montre de nouveau Form1}
end;
```

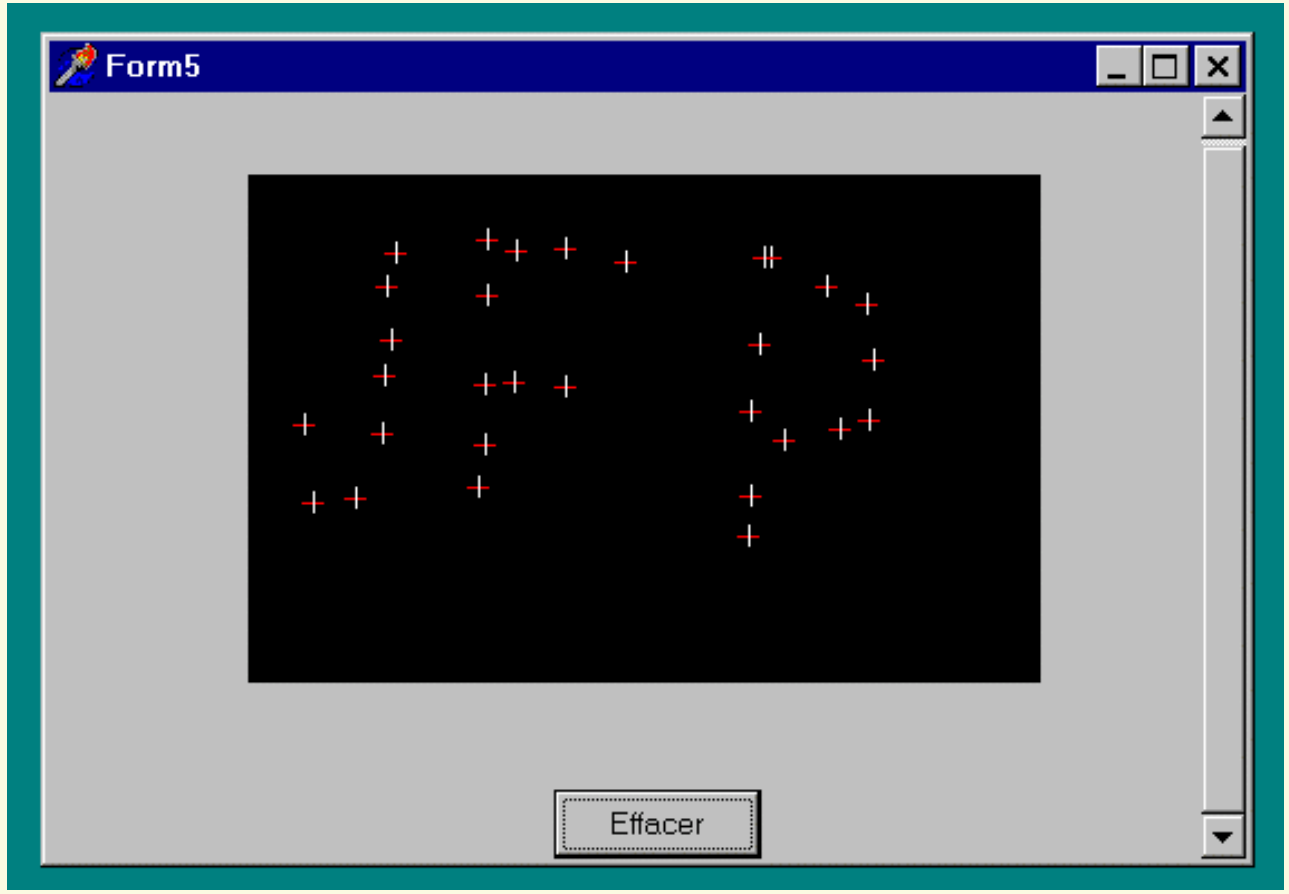
Un exemple

- L'écran attendu pour Form5 (initialisation)



Un exemple

- L'écran attendu pour Form5 (utilisation)



Un exemple

- Les déclarations équivalentes

```
type
  TForm5 = class(TForm)
    Image1: TImage;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure Image1MouseDown(Sender: TObject; Button:
      TMouseButton; Shift: TShiftState; X, Y: Integer);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
```

Un exemple

- Le code équivalent

```
procedure effacer;
begin
   Form5.Image1.Canvas.brush.color:=clBlack;
   Form5.Image1.Canvas.FillRect(Form5.Image1.Canvas.clipRect);
end;
procedure TForm5.Button1Click(Sender: TObject);
begin
   effacer
end;
procedure TForm5.FormShow(Sender: TObject);
begin
   effacer
end;
```

Un exemple

- Le code équivalent

```
procedure TForm5.Image1MouseDown(Sender: TObject; Button:
    TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    Form5.Image1.Canvas.Pen.Color:=ClRed;
    Form5.Image1.Canvas.MoveTo(X-5,Y);
    Form5.Image1.Canvas.LineTo(X+5,Y);
    Form5.Image1.Canvas.Pen.Color:=ClWhite;
    Form5.Image1.Canvas.MoveTo(X,Y-5);
    Form5.Image1.Canvas.LineTo(X,Y+5);
end;
```