Les classes

Définition, implémentation, instanciation sous DELPHI

Création des interfaces graphiques

Introduction à la programmation évènementielle

Ricco Rakotomalala
Université Lumière Lyon 2

Classes ?

Classe – Un type permettant de regrouper dans la même structure : les informations (champs, propriétés, attributs) relatives à une entité ; les procédures et fonctions permettant de les manipuler (méthodes).

Il est dès lors possible de modéliser l'organisation des applications : le diagramme de classes (UML)

Delphi

```
TYPE  NOM_CLASSE = CLASS
            nom_champ_1 : type de données ;
            nom_champ_2 : type de données ;
            …
            fonction_1(paramètres…) : type de données;
            fonction_2(paramètres…) : type de données;
END;
```

Programmer une nouvelle classe

DÉCLARATION ET IMPLÉMENTATION

Exemple : définir la classe TPersonne

UPersonne.pas

```
unit UPersonne;
interface

TYPE    TPersonne = class
            //propriétés
            nom: shortstring;
            age: integer;
            salaireBase: double;
            //méthodes
            procedure saisie();
            function  salaire(avecAnciennete: boolean): double;
            procedure affichage(avecAnciennete: boolean);
        end;

implementation

procedure TPersonne.affichage(avecAnciennete: boolean);
begin
 writeln('nom = ', nom);
 if (avecAnciennete = true)
  then writeln('age = ', age);
 writeln('salaire = ',salaire(avecAnciennete));
end;

procedure TPersonne.saisie();
begin
 write('nom : '); readln(nom);//ou readln(self.nom);
 write('age : '); readln(age);//ou readln(self.age);
 write('salaire de base : '); readln(salaireBase);
end;

function TPersonne.salaire(avecAnciennete: boolean): double;
var tmp: double;
begin
 tmp:= salaireBase;
 if (avecAnciennete = true)
  then tmp:= tmp + 25.0 * age;
 result:= tmp;
end;

end.
```

18: 1 Insertion Code

Pour une meilleure organisation, mieux vaut associer une classe à une unité. TPersonne est décrite dans **interface**, donc visible à l'extérieur de l'unité.

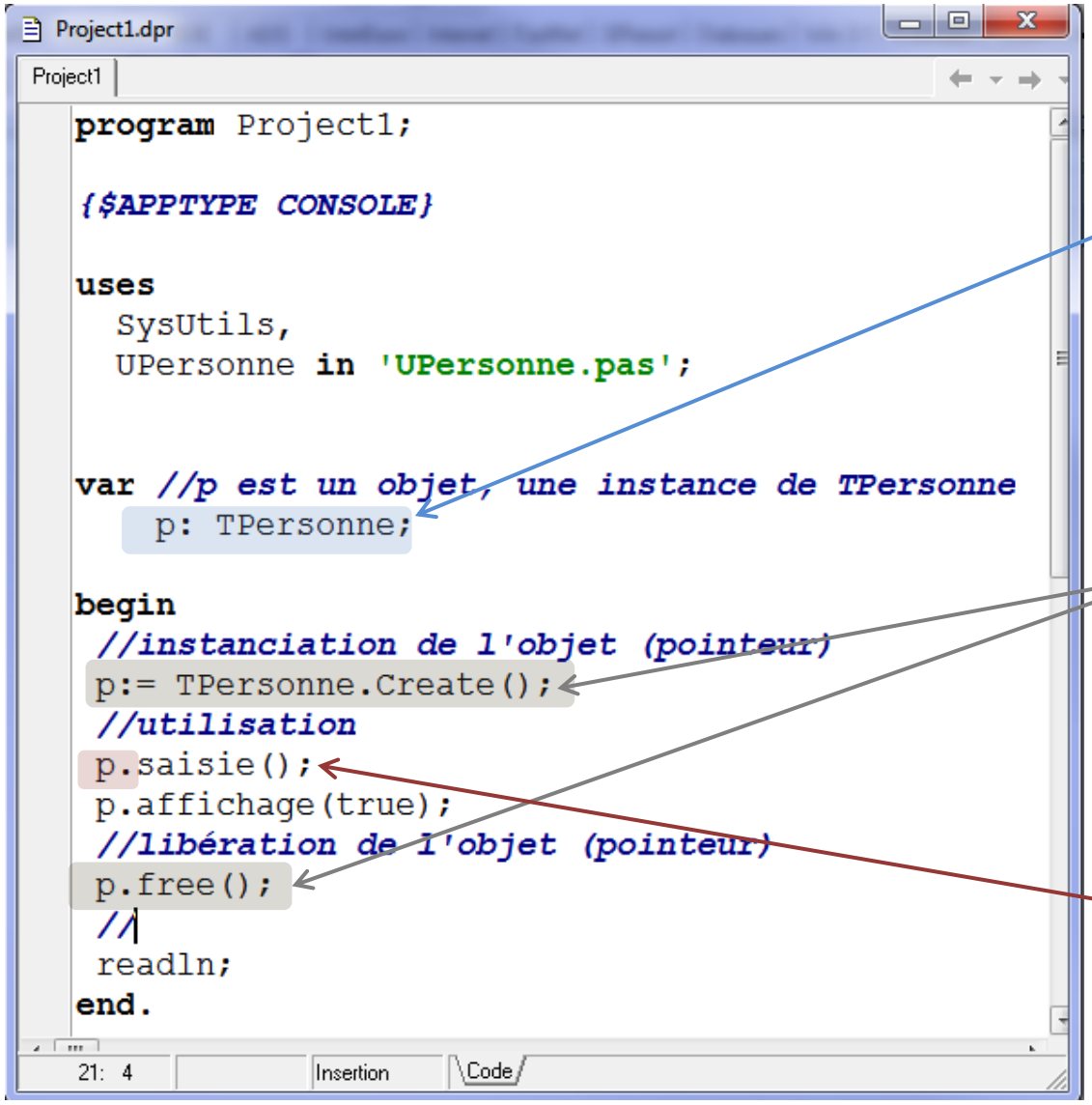
Dans implémentation, lorsqu'on souhaite programmer une méthode, il faut la préfixer par le nom de la classe.

Dans une méthode de classe, les champs et autres méthodes sont directement visibles et utilisables (pour l'instant, cette année). Un peu comme des variables globales à la classe.

Utiliser un objet dans le programme principal

INSTANCIATION

Exemple : utiliser la classe TPersonne



Déclaration de la variable (objet, instance)

Nous n'aurons pas à initialiser et à détruire explicitement les objets cette année.

Le « . » permet d'accéder aux méthodes de l'objet. Il en est de même en ce qui concerne les propriétés. On aurait pu faire :

readln(p.nom);

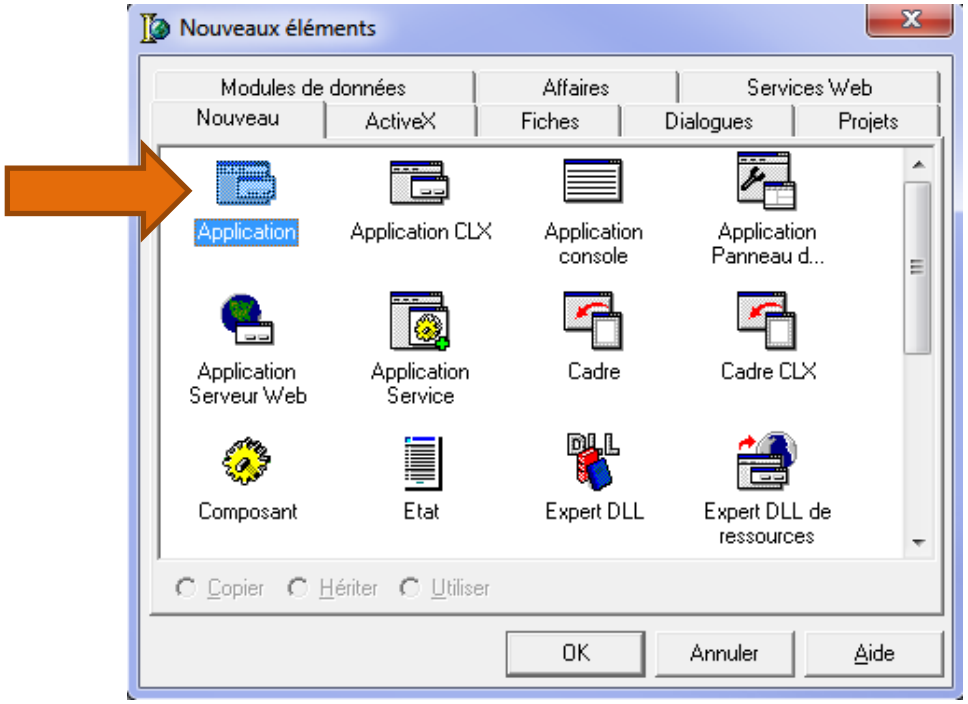
Création des interfaces graphiques sous Delphi – Utilisation des classes

APPLICATIONS FENÊTRÉES

Création d'une application fenêtrée

Après avoir tout fermé dans DELPHI
Faire FICHIER / NOUVEAU / AUTRE

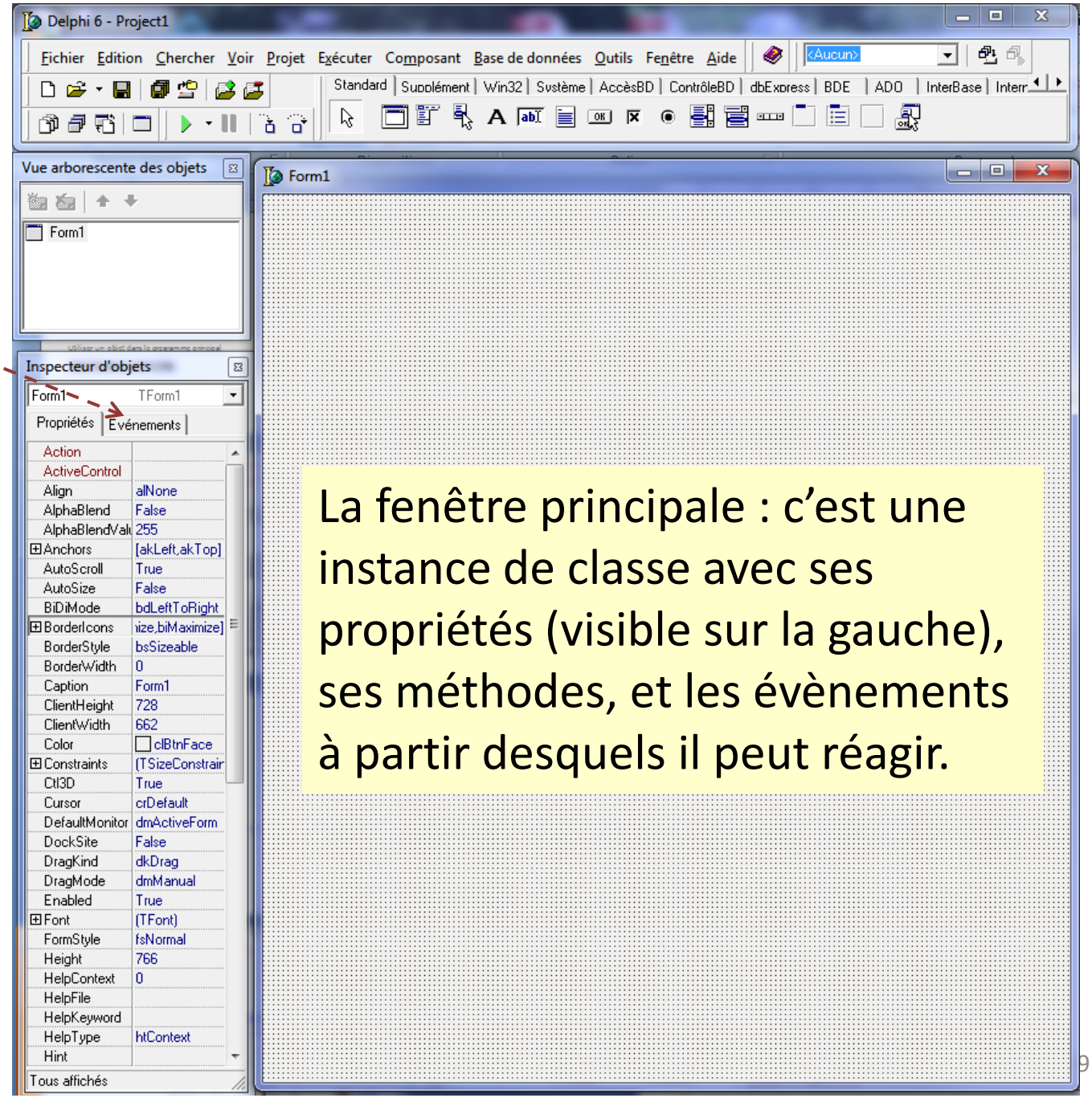
Application fenêtrée WINDOWS



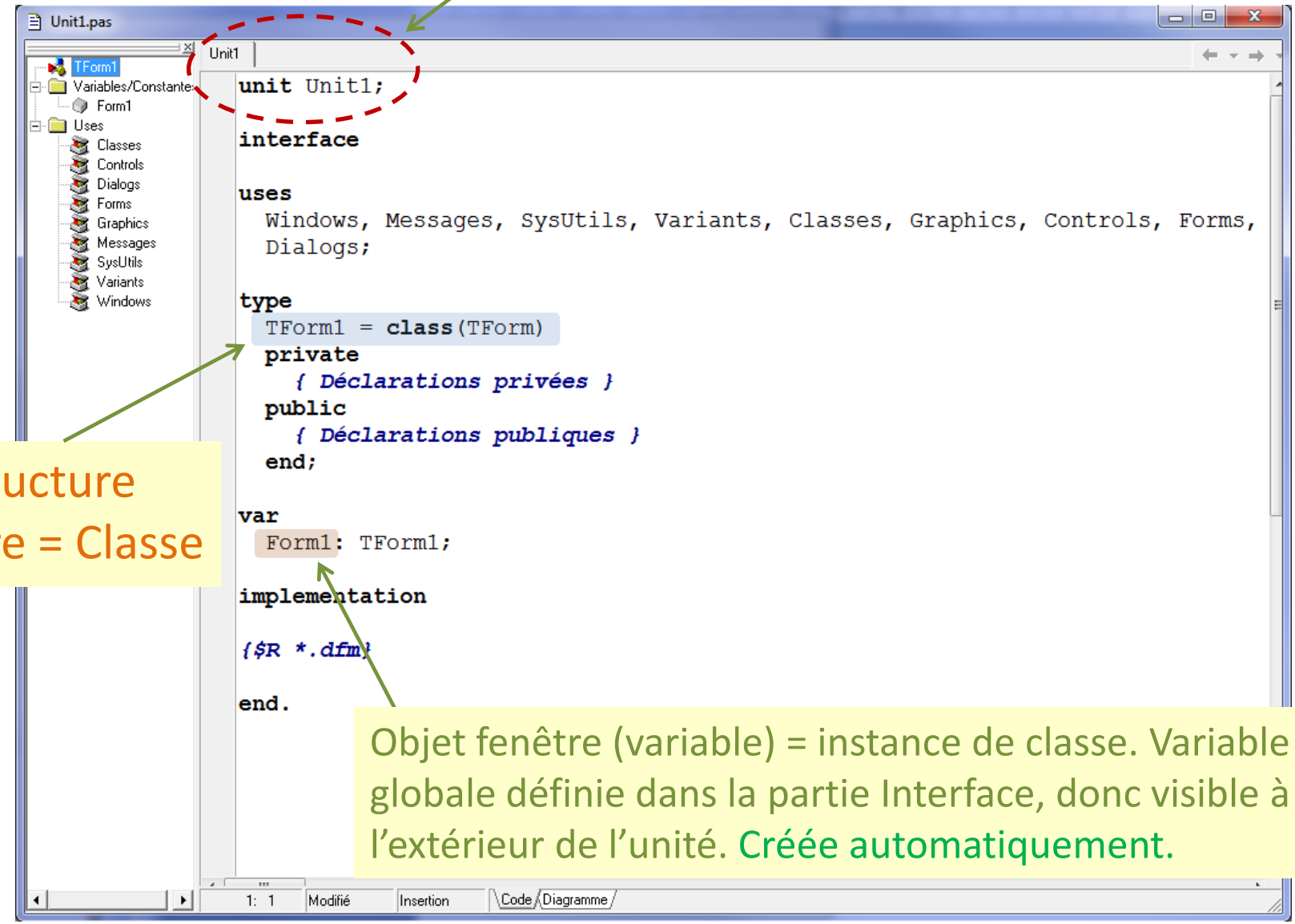
1ère fenêtre = fenêtre principale

Liste des évènements que peut gérer l'objet « Fenêtre »

Propriétés de l'objet fenêtre



La classe est définie dans une unité (bien évidemment). **Fichier « .pas »**.

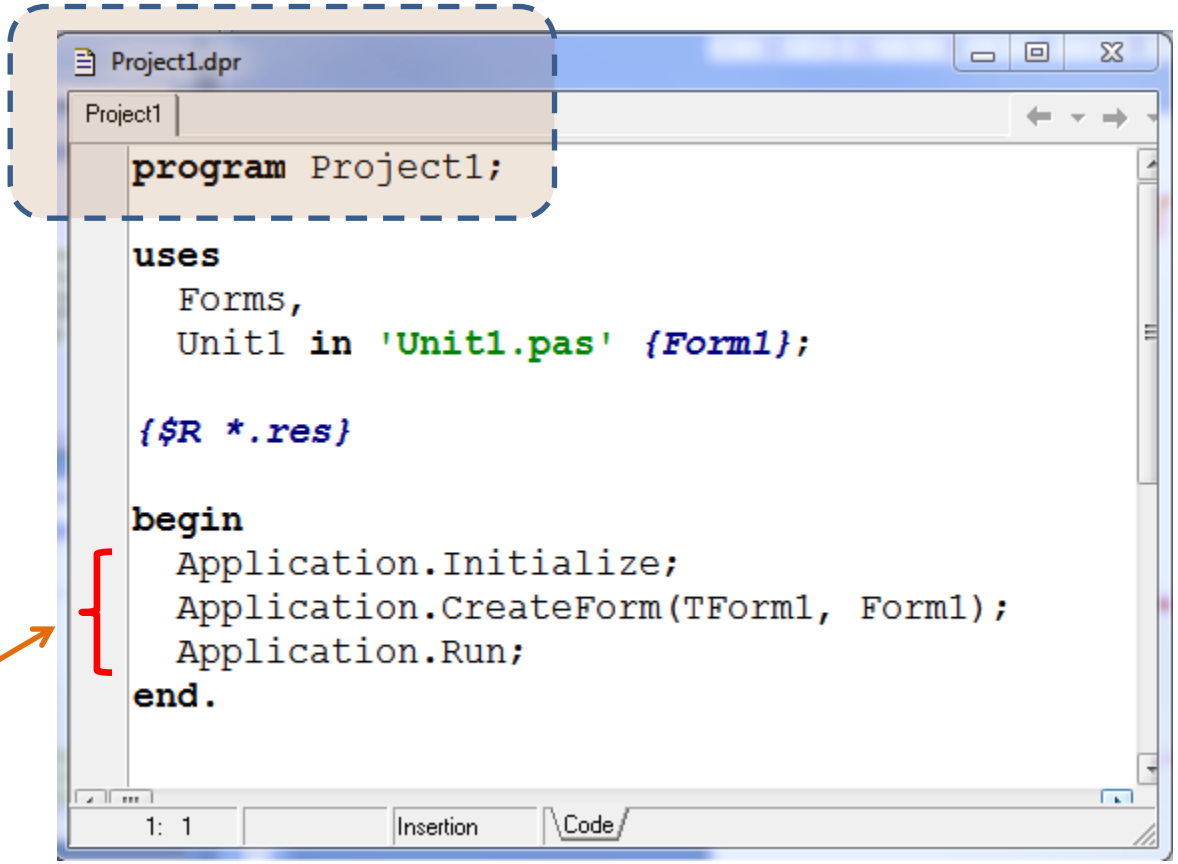


Structure fenêtre = Classe

Objet fenêtre (variable) = instance de classe. Variable globale définie dans la partie Interface, donc visible à l'extérieur de l'unité. Créée automatiquement.

Et le programme principal ?

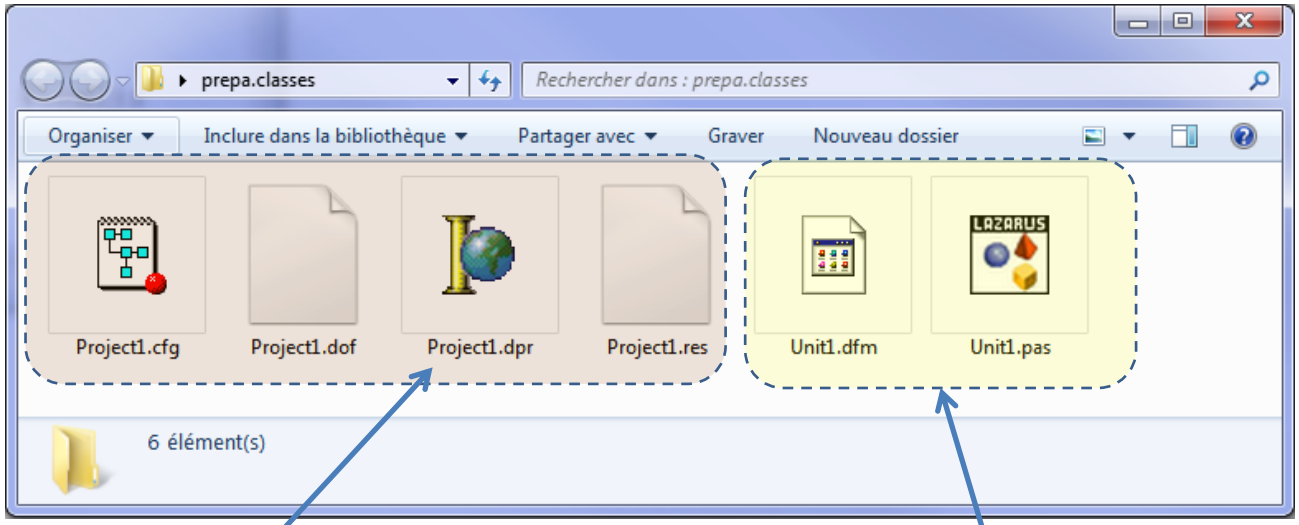
Fichier « .dpr ». Un seul par projet.



Se charge de créer les fenêtres et de lancer l'application.

Dans 99% des cas, nous n'avons pas à effectuer des modifications dans le programme principal dans une application fenêtrée.

Quels sont les fichiers liés au projet ?

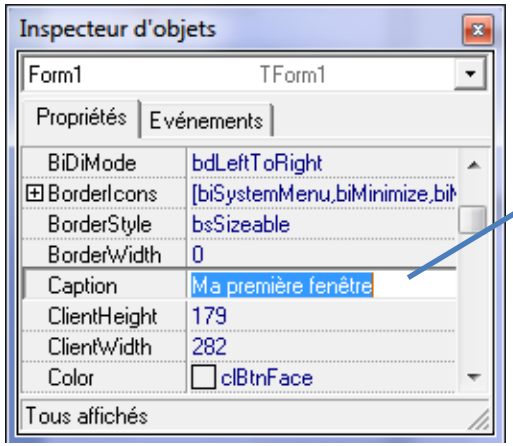


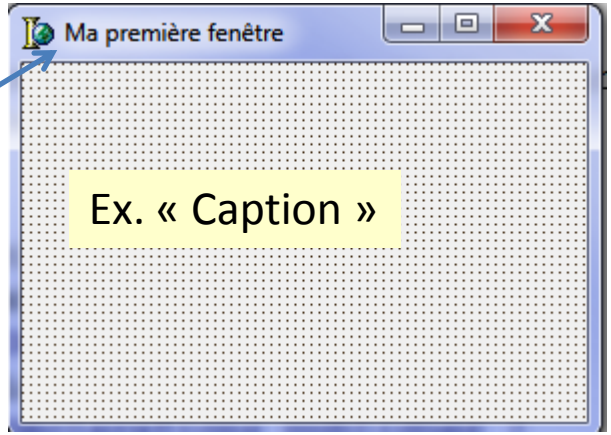
1. Un seul fichier « **.DPR** » par projet.
A ne pas modifier dans 99% des cas.
Les autres sont des fichiers de configurations qui lui sont associées.
Ne pas y toucher.

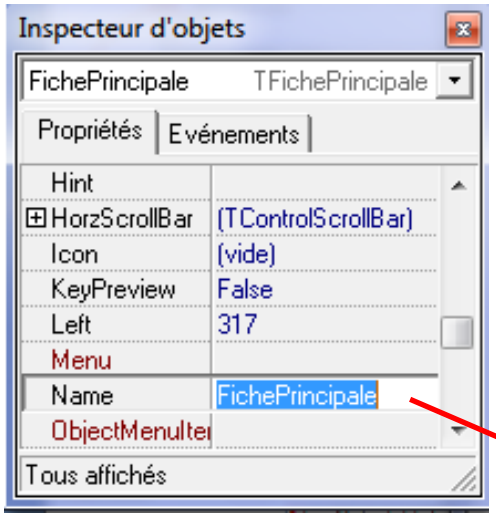
2. Un couple de fichiers « .DFM » et « .PAS » par fenêtre créée.
« **.DFM** » décrit le dessin « graphique » de la fenêtre. Non modifiable directement.
« **.PAS** » est l'unité associée à la classe fenêtre. Nous y programmerons.

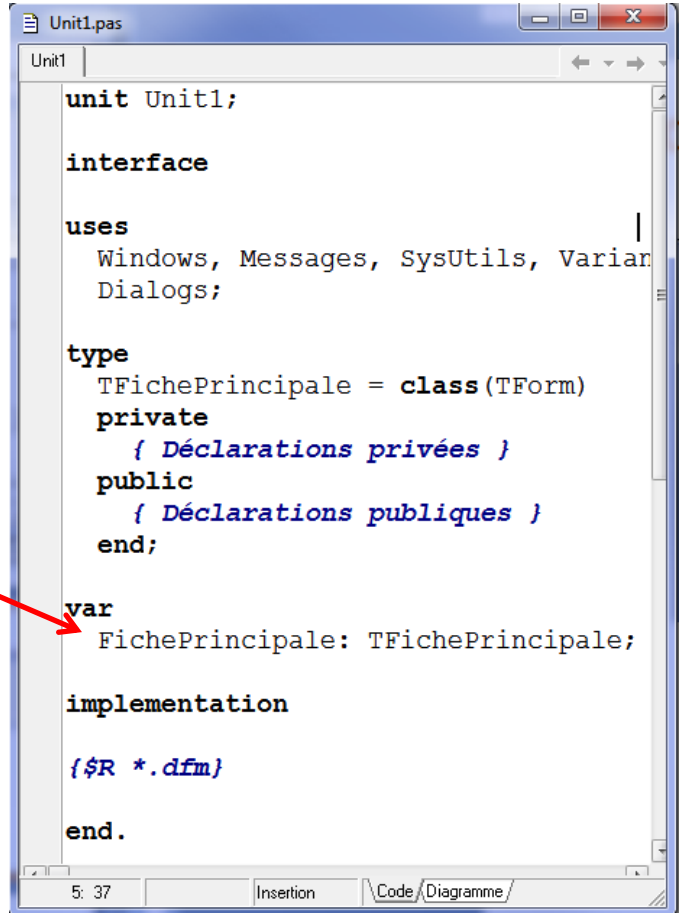
3. Les unités de calcul classiques. 1 fichier « **.PAS** » par module (ex. pour les types enregistrements, les fichiers, les tableaux, etc.).

Modifier les propriétés associés à un objet







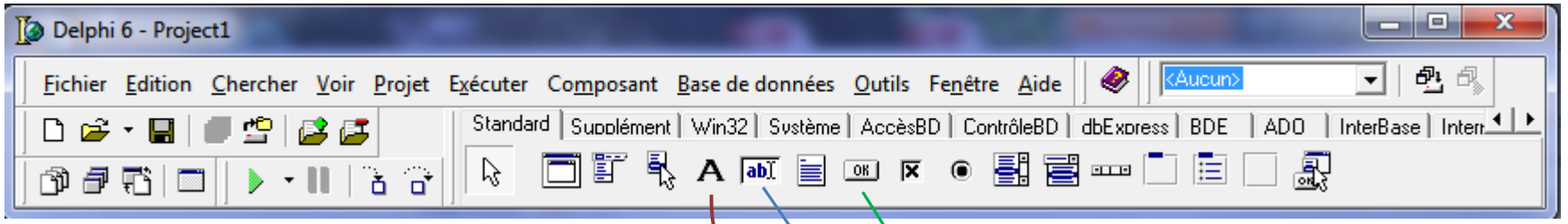


La propriété « **Name** » est très importante. Elle définit le nom de variable associé à l'objet visuel.

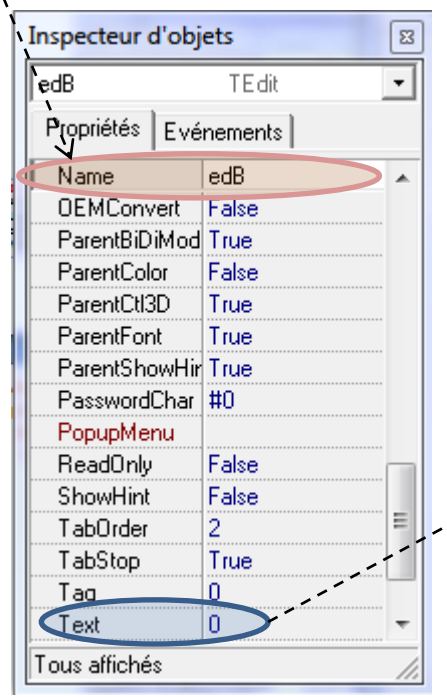
Introduction à la programmation évènementielle

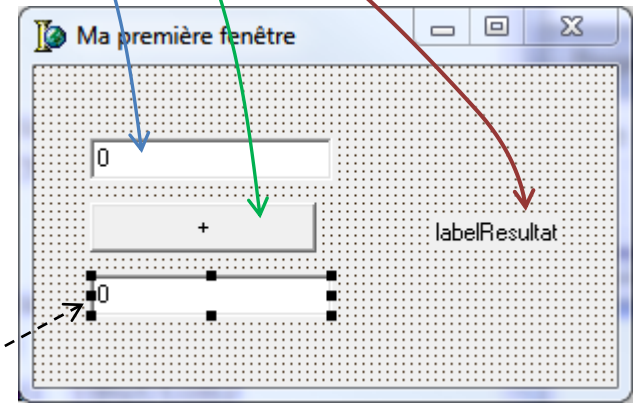
PROGRAMMATION DES OBJETS VISUELS

Ajouter des composants dans la fenêtre à partie de la palette des objets visuels



« Name » a un rôle non visuel, mais primordial pour la programmation.

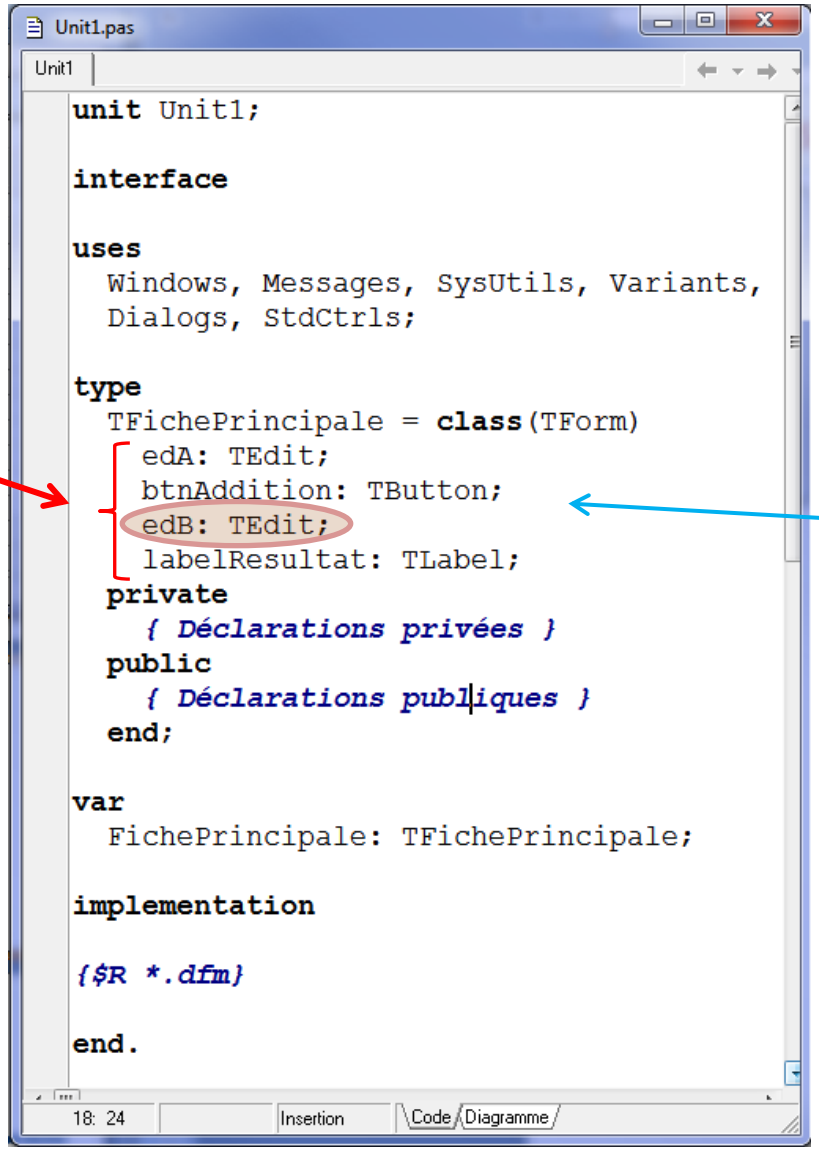




« Text » permet de gérer le contenu du TextBox.
C'est une propriété de type STRING (chaîne de carac.)

Les objets visuels deviennent des champs de la fenêtre.

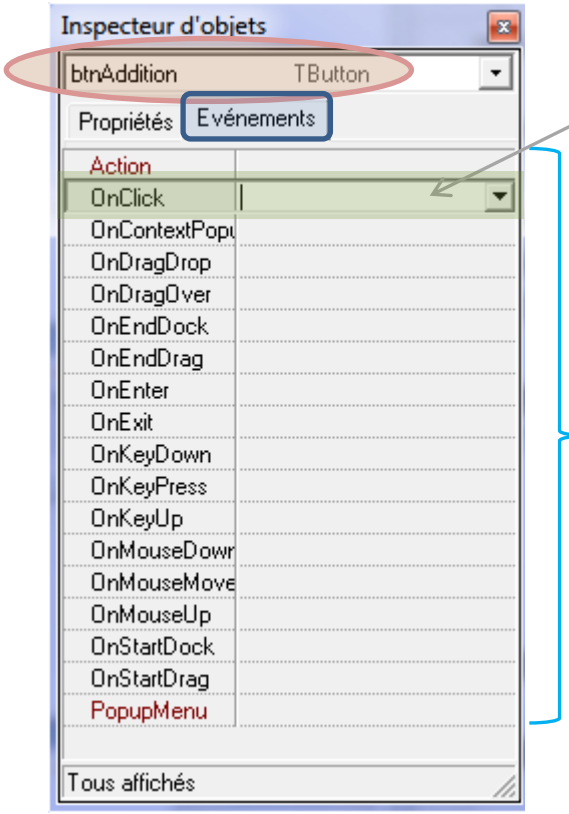
Le nom du champ est défini par la propriété « Name » de l'objet visuel.

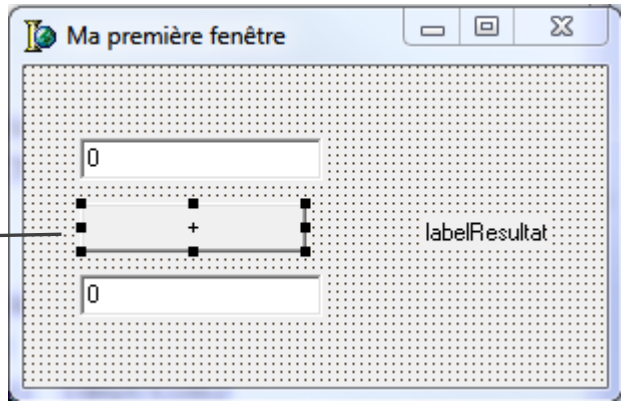


Chaque objet visuel est une instance d'une classe prédéfinie dans DELPHI. Il n'est pas nécessaire de les initialiser explicitement. DELPHI s'en charge tout seul.

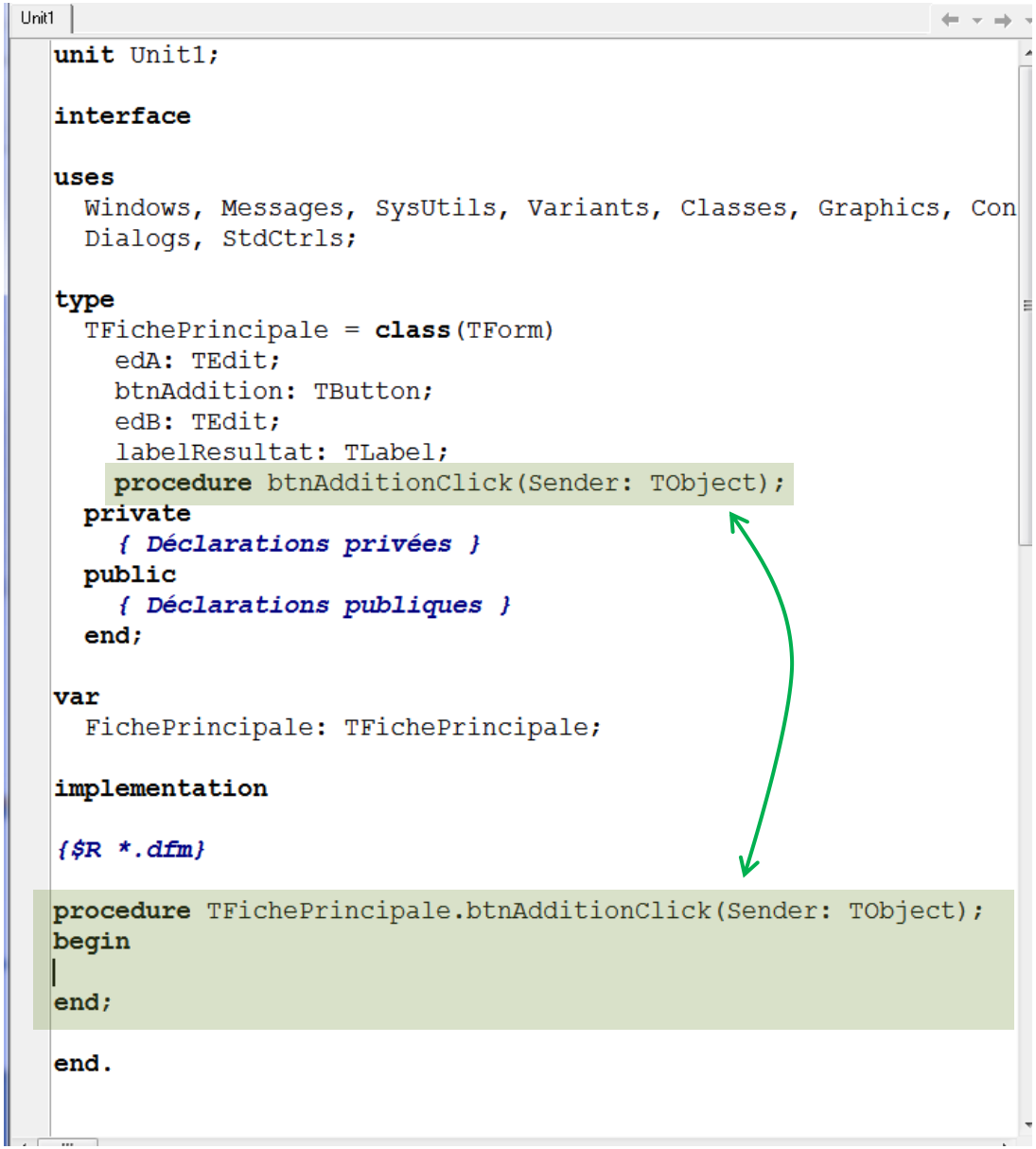
Programmer un évènement

On souhaite par exemple programmer l'evènement « click » sur le bouton.





A chaque objet visuel peuvent être associés des évènements qu'il peut gérer. Ex. clic de la souris sur l'objet, survol de la souris, touche du clavier enfoncée, glisser-déposer à partir ou à destination de l'objet…



Un gestionnaire d'évènement est une méthode de la classe « Fenêtre ».

On peut y manipuler tous les champs de la fenêtre, y compris les objets visuels qui y ont été déposés !

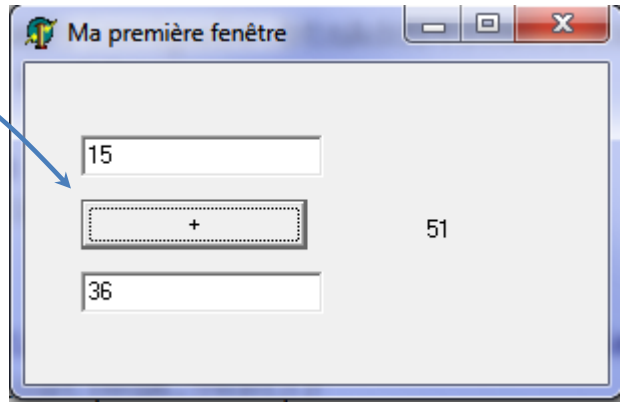
Unit1.pas

Unit1

```
type
  TFichePrincipale = class(TForm)
    edA: TEdit;
    btnAddition: TButton;
    edB: TEdit;
    labelResultat: TLabel;
    procedure btnAdditionClick(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;

var
  FichePrincipale: TFichePrincipale;

implementation

{$R *.dfm}

procedure TFichePrincipale.btnAdditionClick(Sender: TObject);
var a, b, res: double;
begin
 //vérification
 if (edA.Text <> '') and (edB.Text <> '')
  then
   begin
    //récupération des valeurs - oublier les "readln"
    a:= StrToFloat(edA.Text);
    b:= StrToFloat(edB.Text);
    //calcul
    res:= a + b;
    //restitution - oublier les "writeln"
    labelResultat.Caption:= FloatToStr(res);
   end
  else labelResultat.Caption:= 'A ou B non conformes';
end;

end.
```

47: 1 | Insertion | Code | Diagramme

« clic »



FIN…

Les mêmes concepts sont – à peu de choses près – présents dans tous les langages de programmation…