FICHE SAVOIRS

Fiche savoirs : Programmation événementielle (partie 2)

1. Logique événementielle

Présentation des fondements de la programmation événementielle.

1A. Déroulement d'un programme

La programmation événementielle est dépendante des interfaces graphiques. Le programme ne se déroule pas dans un ordre figé, fixé par le développeur, mais dans l'ordre de sollicitation de la part de l'utilisateur (par exemple en cliquant sur un bouton).

Au chargement d'une interface, il est possible d'exécuter automatiquement du code (on parle d'événement "chargement" ou "load"). Ensuite, c'est l'utilisateur qui a la main et peut déclencher des fonctions événementielles en sollicitant des objets graphiques.

La signature d'une fonction événementielle ne peut pas être modifiée. Il est fortement conseillé d'utiliser la création automatique de fonction événementielle car Visual Studio va générer automatiquement du code pour relier cette fonction événementielle à l'événement correspondant (par exemple l'événement clic sur un bouton).

Il est aussi possible d'écrire des fonction ou fonctions non événementielles, comme vous l'avez appris en programmation procédurale, lorsque vous voulez optimiser le code de certaines fonctions événementielles ou vous créer des outils.

1B. Objets graphiques

Un objet graphique est un objet visuel qui peut être intégré dans une interface graphique et qui peut interagir avec l'utilisateur.

Quelques exemples :

- **TextBox** : zone de saisie ;
- **Label** : zone d'affichage d'une information ;
- **Button** : bouton de commande permettant de lancer automatiquement l'exécution d'une fonction événementielle ;
- **ListBox** : liste d'informations permettant la sélection simple ou multiple ;
- **ComboBox** : même principe que ListBox mais une seule ligne est visible à la fois (la ligne sélectionnée) ;
- **CheckBox** : case à cocher (plusieurs peuvent être sélectionnés) ;
- **RadioButton** : bouton radio (un seul sélectionné à la fois) ;
- **Picture** : zone d'affichage d'une image ;
- **GroupBox** : zone de regroupement (utilisée par exemple pour regrouper plusieurs boutons radio) ;
- **DateTimePicker** : sélection d'une date ;

— **NumericUpDown** : sélection d'un entier entre deux bornes.

Important :

Au moment de la création d'un objet graphique, il faut lui donner un nom (name). Ce nom sera son identification unique et pourra ensuite être utilisé dans le code.

Par défaut, lors de la création d'un objet graphique, Visual Studio donne un nom. Il est préférable de le changer et de personnaliser le nom.

On suggère la notation camelCase et de constituer le nom de 2 parties :

— **Les 3 premiers caractères rappellent le type d'objet (exemple : txt pour TextBox) ;** — **Les caractères suivants permettent de clairement identifier le rôle de l'objet.**

Exemples de noms d'objets graphiques : txtNom, txtPrenom, dtpDateNaiss…

1B1. Propriétés

Chaque objet graphique possède des propriétés qui peuvent être exploitées soit directement dans la construction de l'interface (elles permettent de décider de l'aspect visuel de l'objet graphique, dès le début du programme), soit dans le code (elles permettent de gérer l'aspect visuel de l'objet graphique, au fur et à mesure de l'exécution du programme).

Les propriétés sont en fait des variables liées à l'objet. Elles permettent généralement de changer son aspect visuel (contenu, position, taille, couleur…).

Ces propriétés sont nombreuses et variées. Certaines se retrouvent dans quasiment tous les types d'objets, certaines sont très spécifiques.

EXEMPLES

```
// vide le texte contenu dans txtNom
txtNom.Text = "";
// empêche l'accès à txtNom
txtNom.Enabled = false;
// affecte la couleur jaune à la couleur de fond de txtNom
txtNom.BackColor = Color.Yellow;
```

1B2. Méthodes

Chaque objet graphique possède des méthodes qui ne peuvent être exploitées que dans le code. Les méthodes sont en fait des modules, liés à l'objet ou à des propriétés de l'objet. Elles permettent d'agir sur l'objet ou sur l'une de ses propriétés.

Comme pour des modules, vous pouvez repérer les méthodes par les parenthèses.

EXEMPLES

```
// se positionne sur txtNom txtNom.Focus(); //
ajoute un item (une ligne) à la liste
lstProfession
lstProfession.Items.Add("Fonctionnaire");
// compare la date comprise dans dtpDateNaiss avec
aujourd'hui if (dtpDateNaiss.Value.Equals(DateTime.Now)){
};
```

1B3. Événements

Chaque objet graphique possède des événements qui ne peuvent être exploités que dans le code. Un événement permet de déclencher une fonction événementielle. Cette fonction possède une signature spécifique qui ne peut pas être changée et elle est liée à l'événement.

Il suffit ensuite de mettre dans la fonction événementielle, le code que vous voulez exécuter au moment du déclenchement de l'événement.

EXEMPLE

```
/// <summary>
/// Fonction événementielle sur le clic du bouton btnEnregistrer
/// </summary>
/// <param
name="sender"></param> ///
<param name="e"></param>
private void btnEnregistrer_Click(object sender,
EventArgs e) {
   // si la zone du nom est
vide
   if (txtNom.Text.Equals(""))
   {
      // afficher un message d'avertissement
      MessageBox.Show("Le nom doit être
rempli");
      // redonne le focus à la zone de texte du
nom
      txtNom.Focus();
   }

else
   {
      // appel de la fonction d'enregistrement de la personne
      enregistrer();
   }
}
```

L'ordinateur sait que le clic du bouton `btnEnregistrer`, entraine l'exécution de la fonction car elle est liée à l'événement, dans la partie de code (le "Designer") qui se génère automatiquement et que vous ne devez pas toucher. Voici la ligne de code correspondante :

EXEMPLE

```
this.btnEnregistrer.Click += new
System.EventHandler(this.btnEnregistrer_Click);
```

C'est EventHandler qui ajoute une fonction événementielle liée au Click sur `btnEnregistrer`.

1C. Sécurisation et contrôle des accès aux objets graphiques

Le but est de sécuriser l'interface. Cela peut se faire à plusieurs niveaux :

1C1. Limiter les accès aux objets graphiques

Sachant que l'utilisateur peut exploiter les objets graphiques quand il veut, certaines manipulations peuvent parfois poser problème.

Par exemple, on veut peut-être éviter qu'il puisse cliquer sur le bouton d'enregistrement s'il n'a pas rempli les champs obligatoires. Il est possible de rendre non accessible n'importe quel objet graphique avec la propriété booléenne Enabled :

EXEMPLE

```
// rend le bouton btnEnregistrer inaccessible (impossible de cliquer dessus)
btnEnregistrer.Enabled = false;
```

Il est aussi possible de rendre invisible un objet graphique.

EXEMPLE

```
// rend le bouton btnEnregistrer invisible
btnEnregistrer.Visible = false;
```

Il est important de bien gérer ces propriétés dans le code, pour éviter les manipulations non souhaitées.

1C2. Positionner le focus au bon endroit

L'utilisateur peut cliquer sur un objet graphique (lorsque celui-ci est accessible) comme il veut.

Cependant, c'est toujours plus pratique pour lui si l'objet qu'il doit naturellement utiliser à un moment précis, soit déjà "actif", c'est-à-dire qu'il n'ait même pas à cliquer dessus. Il suffit pour cela de donner le "focus" à cet objet.

Par exemple, une fois qu'il a saisi le nom et validé, ce serait bien que le curseur se positionne directement dans la zone du prénom pour la saisie du prénom. Voici comment donner le focus à un objet :

EXEMPLE

```
txtPrenom.Focus();
```

1C3. Contrôler le contenu des objets graphiques

Dans le code, il faut contrôler le contenu des objets graphiques lorsque cela est nécessaire. Par

exemple, sur la saisie du nom, vérifier que sa longueur est au moins de 2 caractères.

1C4. Éviter les erreurs

Pensez aussi, comme pour la programmation procédurale, à éviter les manipulations qui pourraient déclencher une erreur d'exécution.

Par exemple, ne forcez pas le positionnement dans une liste si la liste est vide. Il faut d'abord contrôler si la liste contient au moins un élément :

EXEMPLE

```
// Si la liste n'est pas vide
if (lstProfession.Items.Count
> 0)
{
   // sélectionner par défaut le premier élément
   lstProfession.SelectedIndex = 0;
}
```

Il y a aussi toujours la possibilité de capturer les erreurs avec un try/catch.