Fiche savoirs : programmation événementielle en C#

Dans une application de bureau vous mettez en oeuvre une programmation événementielle qui est différente de la programmation en mode console.

L'interface

Exemple avec cette application graphique de bureau :

L'interface est graphique et l'utilisateur est maître de l'ordre d'exécution :

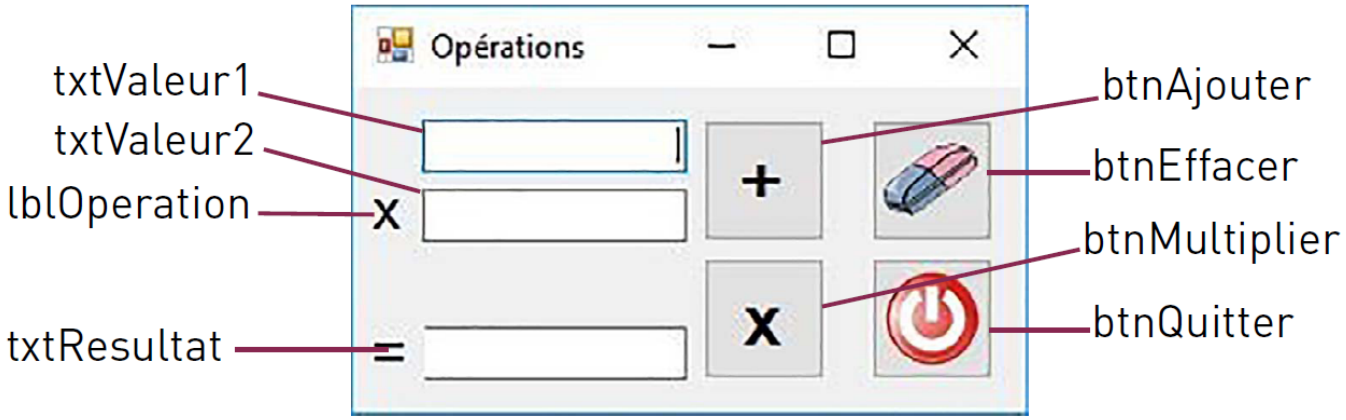
- il peut décider de saisir 2 nombres, puis de cliquer sur l'opération voulue,
- éventuellement de cliquer sur la seconde opération sans modifier les 2 nombres ou en n'en modifiant qu'un seul,
- etc.

C'est l'utilisateur qui choisit à quel moment doit s'exécuter une commande en fonction des objets graphiques qu'il va solliciter.

De plus, l'aspect visuel est nettement plus attractif et intuitif :

- l'opération est **posée** pour être plus claire
- la ligne permet de bien voir la séparation entre la saisie des nombres et l'affichage du résultat,
- les dessins (signes d'opérations) sont plus parlants que le texte...

Voici les caractéristiques des objets graphiques :



Nom (name)	Type	Autres propriétés
txtValeur1	TextBox	TextAlign : Right
txtValeur2	TextBox	(idem txtValeur1)
txtResultat	TextBox	(idem txtValeur1)
lblOperation	Label	Font : Size : 16
btnAjouter	Button	Font : Size : 20
Font : Bold : True		
btnMultiplier	Button	(idem btnAjouter)
btnEffacer	Button	Image : image fournie de la gomme ou image de votre choix redimensionnée
btsQuitter	Button	(idem btnEffacer avec une autre image)

En l'état, l'application peut déjà être testée : les boutons ne sont pas encore actifs mais il est déjà possible de saisir des valeurs dans les zones de saisie.

Le code événementiel

Voici le code de chaque événement :

- **Clic sur btnEffacer : Vider les 3 zones de texte et le label de l'opération. <code c#> private void btnEffacer*Click(object sender, EventArgs e) { txtValeur1.Text = “”; txtValeur2.Text = “”; txtResultat.Text = “”; lblOperation.Text = “”; } </code> * Clic sur btnQuitter : Quitter l'application. <code c#> private void btnQuitter*Click(object sender, EventArgs e) { Application.Exit(); } </code> * Clic sur btnAjouter : Afficher le signe + dans le label de l'opération.
  Faire la somme des 2 valeurs saisies (si c'est possible) et la transférer dans txtResultat <code c#> private void**

btnAjouter*Click(object sender, EventArgs e) { try { txtResultat.Text = (float.Parse(txtValeur1.Text) + float.Parse(txtValeur2.Text)).ToString(); lblOperation.Text = “+”; } catch { }; } </code> * Clic sur btnMultiplier : Afficher le signe “x” dans le label de l'opération.*
***Faire la multiplication des 2 valeurs saisies (si c'est possible) et la transférer dans txtResultat. <code c#> private void btnMultiplier*Click(object sender, EventArgs e) { try { txtResultat.Text = (float.Parse(txtValeur1.Text) * float.Parse(txtValeur2.Text)).ToString(); lblOperation.Text = “x”; } catch { }; } </code> * Changement de texte dans txtValeur1 : Vider l'affichage du résultat et le label de l'opération. <code c#> private void txtValeur1*TextChanged(object sender, EventArgs e) { txtResultat.Text = “”; lblOperation.Text = “”; } </code> * Changement de texte dans txtValeur2 : Vider l'affichage du résultat et le label de l'opération. <code c#> private void txtValeur2*TextChanged(object sender, EventArgs e) { txtResultat.Text = “”; lblOperation.Text = “”; } </code> ===== Le code non événementiel ===== Il est possible de créer des modules non événementiels, comme dans la programmation procédurale classique, pour optimiser le code. Par exemple, on remarque que le code est identique dans les 2 foncions événementielles sur le changement de texte dans txtValeur1 et txtValeur2. On peut alors créer un module isolé et l'appeler dans les deux procédures événementielles. * Isoler le code : Créer un module non événementiel <code c#> private void AnnuleOperation() { lblOperation.Text = “”; txtResultat.Text = “”; } </code> * Appeler le module : Appeler le module dans les procédures événementielles (par exemple pour txtValeur1). <code c#> private void txtValeur1_TextChanged(object sender, EventArgs e) { AnnuleOperation(); } </code>**