TP1 SLAM2 : PREMIERE INTERFACE GRAPHIQUE
INTERFACE DISTRIBUTEUR

Objectifs :

Interface graphique sous VisualStudio2019 (en Windows Form)
Pratique des premiers objets graphiques en C# : zone de texte, cadre, cases à cocher, bouton à option, bouton, liste déroulante, Boîte de dialogue, DateTimePicker.

Documents joints : Fiche Technique MessageBox + Fiche Technique GestionSaisies KeyPress Leave
Durée : 1 à 2 séances

Objectifs de l'application

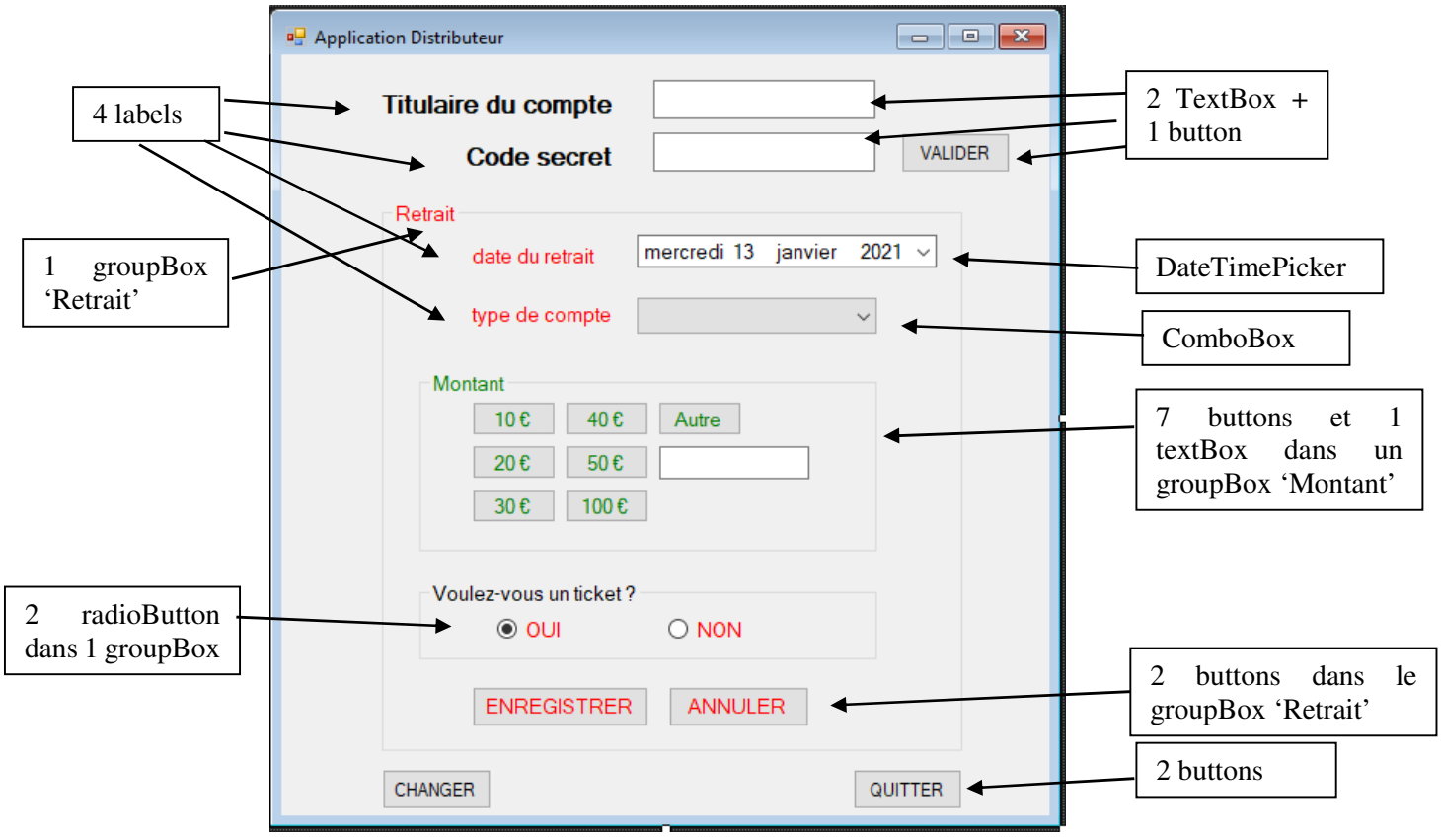
L'objectif va être de créer une interface simple de simulation d'un distributaire bancaire.

Sous VisualStudio2019, pour cette application :

- créer un nouveau projet **Application Windows Form (.NET Framework)** (bien le renommer).
- **enregistrer-le dans votre répertoire personnel**
- renommer la fenêtre créée (propriété **Name** : FormDistributeur, propriété **Text** : Application Distributeur)
- Bien avoir la fenêtre des propriétés ouverte (en général en bas à droite) pour pouvoir changer les propriétés par défaut de chaque contrôle graphique (si ce n'est pas le cas, l'ouvrir avec le menu « Affichage », « Fenêtre Propriétés »).

PARTIE 1 : Mise en place de l'interface avec les contrôles graphiques

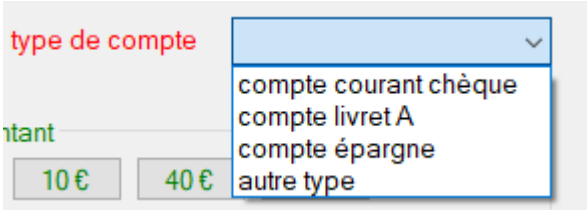
Dans un premier temps, mettre en forme l'interface en glissant les **contrôles graphiques** voulus et en définissant les propriétés par défaut (**toujours mettre à jour au moins les propriétés Name et Text**).



- Bien changer les propriétés Name (nom du contrôle) et Text (texte affiché à l'écran) de tous les contrôles posés et la propriété FONT si vous voulez changer la taille/la couleur/la typographie du texte.

- La comboBox est une liste déroulante, ici nous souhaitons qu'elle ne soit pas modifiable donc mettre sa propriété DropDownStyle à DropDownList.

- Pour compléter la liste déroulante des différents types de comptes : propriété Items, cliquer sur Collections. Remplir un item par ligne.



- Les 2 radioButton doivent fonctionner l'un avec l'autre : quand l'un est coché, l'autre ne l'est pas, pour cela, bien les mettre dans le même conteneur qu'est le groupBox. Pour cocher le OUI par défaut, mettre la propriété Checked à true.

➢ Tester votre interface en démarrant (flèche verte) votre projet.

- Le groupBox 'Retrait' ne doit pas être visible tant que le titulaire et le code ne sont pas correctement saisis : mettre la propriété Visible à False pour ce groupBox.

PARTIE 2 : Mise en place des évènements

Nous allons désormais coder les évènements (les actions) que l'utilisateur peut effectuer sur cette interface.

■ Bouton Valider : Evènement Click

Ce bouton doit permettre de remettre de vérifier si le titulaire et le code sont corrects. Par souci de simplification ici, vous allez décider dans le programme « en dur » d'un nom de titulaire et d'un code (à 4 chiffres).
Pour gérer l'évènement au click de ce bouton, double-cliquer sur le bouton, la fenêtre de code apparaît alors avec l'entête de la méthode déjà écrite. Il ne vous reste plus qu'à compléter avec les actions voulues :

Si le titulaire et le code saisis sont corrects (vérifier les propriétés Text de vos 2 textBox)
alors rendre visible le groupBox 'Retrait'.
Sinon envoyer un message d'erreur (voir la fiche technique MessageBox jointe au TP pour la syntaxe).

Cela doit donner cela :

```
if (tbTitulaire.Text == "nomChoisi" && tbCode.Text == "codeChoisi")
        gbRetrait.Visible = true;
else
        MessageBox.Show("Erreur, veuillez saisir un nom de titualire et un code corrects",
"Erreur", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

■ Bouton Autre : Evènement Click

Ce bouton doit permettre de rendre Visible le textBox pour la saisie d'un autre montant, ou de le rendre à nouveau invisible. Générer l'évènement Click comme précédemment et compléter le code :

Si (le textBox est visible)
alors le rendre invisible
Sinon le rendre visible.

Pour savoir si un contrôle est visible : `if (nomControle.Visible == true)`

■ Bouton Annuler : Evènement Click

Ce bouton doit permettre de remettre la partie 'Retrait' dans son état initial.
Générer l'évènement Click comme précédemment et compléter le code :

- vider les zones de texte de leur contenu (appeler la méthode Clear() sur la zone de texte : par exemple `tbAutre.Clear();`
- Remettre la date du jour : propriété Value du dateTimePicker : `dtRetrait.Value = DateTime.Today;`
- Réinitialiser la liste déroulante : `cbType.SelectedIndex = -1 ;`
- Recocher par défaut la case à cocher (propriété Checked à true) : `rbOUI.Checked = true;`
- Rendre invisible la textBox pour le montant Autre

■ Bouton Changer : Evènement Click

Ce bouton doit permettre de remettre l'application à son état initial.
Générer l'évènement Click comme précédemment et compléter le code :

- vider les zones de texte de leur contenu (appeler la méthode Clear() sur la zone de texte
- Rendre invisible le goupBox 'Retrait'
- le réinitialiser (en appelant le bouton Annuler_click déjà fait : `BtnAnnuler_Click(sender, e);`

■ Boutons des montants : Evènement Click

Les 6 boutons avec un montant précis doivent permettre de récupérer un montant pour le retrait.
Pour cela, d'abord déclarer en haut de votre formulaire de code une variable globale de type chaine à cette form (type chaîne car on a mis le sigle €) :

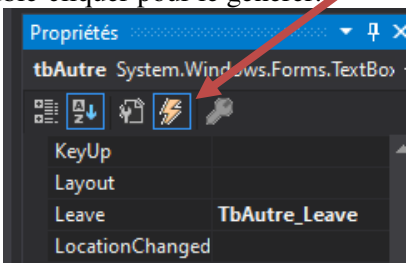
```
public partial class FormDistributeur : Form
 {
     string montant = "";
```

Puis sélectionner tous les 6 boutons ensemble (avec la touche Ctrl) pour créer un évènement Click commun à ces 6 boutons. Générer l'évènement Click et compléter par :

```
montant = ((Button)sender).Text;
// pour récupérer le texte du bouton qui a généré l'évènement
```

■ Zone de texte libre pour montant autre : Evènement Leave

Quand on va quitter cette zone de texte, il nous faut récupérer le montant saisi : sur l'évènement Leave.
Pour générer cet évènement, se mettre sur la zone de texte, du côté de la liste des propriétés, cliquer sur l'éclair pour lister tous les évènements possibles et choisir l'évènement Leave, double-cliquer pour le générer.

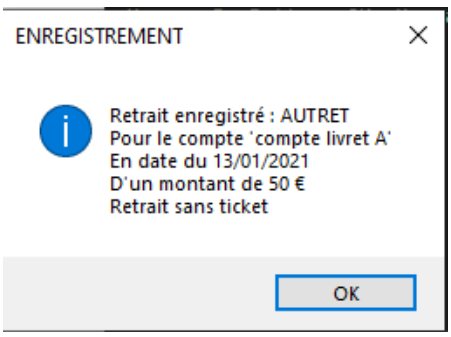


Il ne vous reste plus qu'à compléter la mise à jour de la variable globale montant (déclarée plus haut) avec le contenu saisi : `montant = tbAutre.Text + " €";`

■ Bouton Enregistrer : Evènement Click

Ce bouton doit permettre de résumer le choix de l'utilisateur pour son retrait.
Sur l'évènement Click de ce bouton, une **boite de dialogue** va afficher toutes les informations saisies (voir la fiche Technique C# MessageBox, avec bouton OK) :

- Vérifier si toutes les zones obligatoires pour valider un retrait (montant, type de compte).
- Faire une phrase avec tous les éléments de l'interface du style :

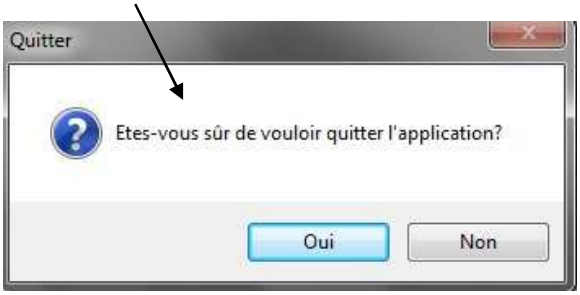


Pour récupérer le texte du zone de texte : NomZoneTexte.Text
Pour récupérer l'élément sélectionné d'une liste : NomListe.SelectedItem
Pour récupérer la date sélectionnée (que la date, sans l'heure):

```
nomDateTimePicker.Value.ToShortDateString()
```

Pour savoir si un radioButton est coché : nomRadio.Checked à true
N'oubliez pas d'utiliser votre variable globale montant pour récupérer le montant choisi (qui a été mis à jour sur les 6 boutons ou sur la zone de texte libre).

■ Bouton Quitter : Evènement Click
Ce bouton doit permettre de quitter l'application à tout moment.
Sur l'évènement Click de ce bouton, avant de quitter, une **boîte de dialogue** demandera confirmation de quitter (voir en bas de la fiche Technique C# MessageBox).



Si la réponse est OUI, l'application sera fermée, l'instruction a générée est : `Application.Exit();`
Sinon la fenêtre restera active et viendra dans son état initial (rappeler l'évènement Click sur Changer revient à : `BtnChanger_Click(sender, e);`

PARTIE 3 : Vérifier les saisies

L'objectif de cette partie va être de vérifier toutes les saisies afin de s'assurer de leur validité.
A l'aide de la Fiche Technique jointe « GestionSaisies KeyPress Leave » et de ses exemples, il faut vérifier que :

- Le titulaire ne comporte que des lettres, tiret et espace (+ caractère BACK pour autoriser l'effacement)
- Le Code ne comporte que des chiffres et est composé de 4 chiffres exactement
- Le montant Autre ne comporte que des chiffres (on pourra aussi vérifier en bonus s'il s'agit d'un montant compatible pour une retrait donc un multiple de 10).

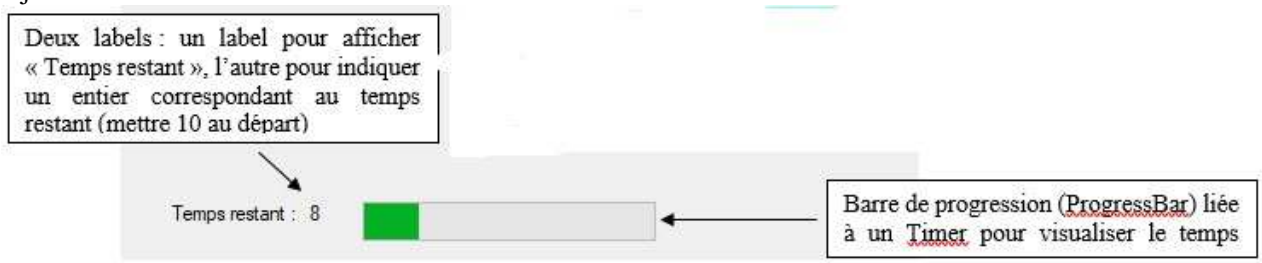
Vous avez 4 évènements KeyPress ou Leave à générer et à compléter ! (les générer avec l'éclair dans la partie des propriétés de chaque contrôle voulu).

BONUS : Gestion d'un temps limité pour le retrait

Contrôles : ProgressBar et Timer

Pour le fun ! Ajoutons à cette partie, une barre de progression pour limiter le temps de retrait à 10 secondes par exemple.
On quittera l'application si l'enregistrement du retrait n'est pas fait à temps ou on permettra de recommencer.

Ajouter sur votre form :



Contrôle PROGRESSBAR : nommé pbRetrait
Selon le temps souhaité maximum pour faire un retrait, mettre à jour les propriétés step, min et max (par exemple, si vous voulez donner 10 secondes pour faire un retrait, mettre min à 0, max à 10, step à 1).
Au-dessus de la barre de progression, un label (nommé labelTemps) indiquera en clair les secondes restantes.

Contrôle TIMER :
Ajouter un timer sur la feuille (il ne s'affiche pas sur la feuille mais en bas), mettre la propriété Interval à 1000 pour être en mode seconde (en millisecondes par défaut).

C'est l'événement *Tick* du timer qui va faire progresser la ProgressBar seconde par seconde :

```
private void timer1_Tick(object sender, EventArgs e)
 {
    // mise à jour du libellé des secondes
     labelTemps.Text = Convert.ToString(Convert.ToInt16(labelTemps.Text) - 1);

     pbRetrait.Increment(1);    // pour faire progresser la ProgressBar

     // vérifier que le temps écoulé n'est pas dépassé (ici si on arrive à 0)
     if (Convert.ToInt16(labelTemps.Text) == 0)
     {
         timer1.Stop();         // arrêt du timer
         // A compléter :
         // - afficher un message pour avertir l'utilisateur du temps écoulé
         // - quitter l'application : Application.Exit() ;
            // ou appeler le bouton ANNULER pour permettre de recommencer
     }
 }
```

Pour lancer la ProgressBar au moment où l'utilisateur commence à se connecter correctement (sur le click du bouton VALIDER par ex), compléter l'évènement `Click` de votre VALIDER :

```
 private void btnValider_Click(object sender, EventArgs e)
 {
      // - Lancer le timer (méthode Start( ))
      //- Mettre à 0 la ProgressBar (propriété Value = 0)
      //- Mettre à 10 le label du temps (propriété Text à "10"
 }
```

Et enfin, si l'utilisateur a bien saisi son retrait dans le temps imparti, il faut arrêter le timer. Ajouter l'arrêt du timer (`timer1.Stop()`) sur l'évènement Click de votre bouton ENREGISTRER.
Vous pouvez rendre invisible la barre de progression si vous le souhaitez alors.

BONUS : Gestion des types de Billets voulus

Contrôle : CheckBox ou CheckListBox (Mettre les choix des options dans Items → Collection)
Plusieurs choix possibles en même temps.

Permettre à l'utilisateur de choisir le type de billets voulus : que des billets de 10€ et/ou des billets de 20€, 50€.
Pour cela proposer les 3 types de billets avec un contrôle de type 'case à cocher' / CheckBox ou CheckListBox qui permettra de cocher 0, 1 ou plusieurs types.

Récupérer ces types de billets voulus dans le récapitulatif lors de l'enregistrement. Penser à décocher tout lors de l'annulation.

Aide :

- Pour savoir si un CheckBox est coché : propriété Checked à true.
- Pour décocher un CheckBox : propriété Checked à false

- Pour récupérer les options cochées dans la `CheckedListBox` (appelée ici `clbBillets`) : parcourir ce tableau en ne sélectionnant que les cases cochées (contenues ici dans `clbBillets.CheckedItems`).

Par exemple :
```
// bien comprendre cet exemple avant de l'adapter à votre code.
int i;
string options = "";
for (i = 0; i < clbBillets.CheckedItems.Count; i++)
{
    options = options + " " + clbBillets.CheckedItems[i].ToString();
}
```

- Dans le cas d'un CheckListBox : Pour décocher toutes les cases cochées :

```
int i;
for (i = 0; i < clbBillets.Items.Count; i++)
    clbBillets.SetItemChecked(i, false);
```