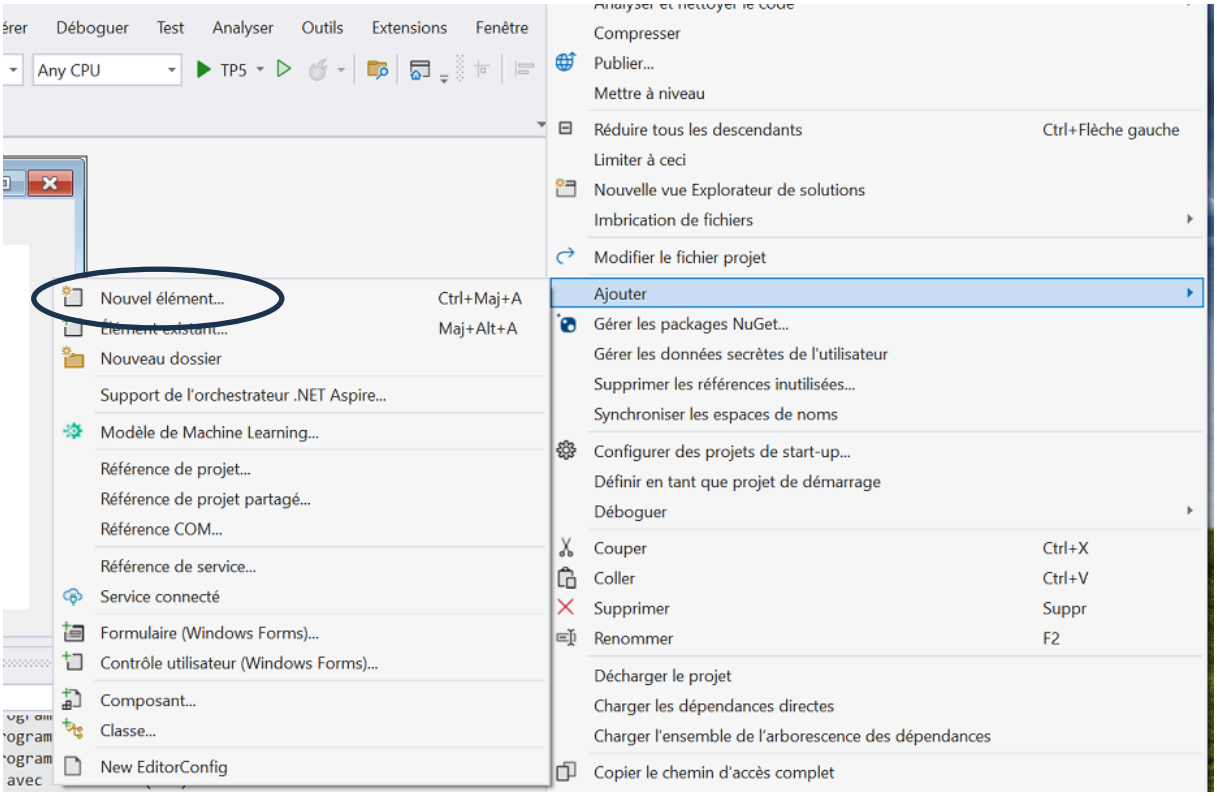
Programmation des Interfaces Interactives Avancées

Travaux pratique numéro 6 - correction

Rappel : La documentation sur Windows Forms pour C# est sur **learn.microsoft.com/fr-fr/dotnet**. Il faut choisir l'**API** pour **Windows Desktop version 8**.

Ce TP et le suivant ont pour but de vous familiariser avec la programmation d'application graphique interactive. Vous serez amené à programmer une interface très simple permettant de dessiner des formes. Dans ce premier TP, vous allez créer pour cela un widget personnalisé dérivant de la classe `Control` de Windows Forms.

1. Créée un nouveau projet `TP6` Windows Forms C# sous Visual Studio. Dans l'explorateur de solution, cliquez droit sur le sommet de la hiérarchie (dossier TP6) et sélectionner `Ajouter` puis `Nouvel Element…` comme ci-dessous.



Choisissez `ZoneDessin` comme nom de nouveau fichier. Une nouvelle classe `ZoneDessin` est ajouté à votre projet.

Dans le fichier *ZoneDessin.cs* créé :

- Ajouter `using System.Windows.Forms` à la liste des namespaces inclus en haut du fichier.

- Faites dériver la classe `ZoneDessin` de la classe `Control` qui est la classe de base des widgets sous Windows Forms.
- Pour inclure ce nouveau widget sur la fenêtre de votre projet, il faut ajouter les lignes ci-dessous dans le fichier définissant la fenêtre principale (probablement `Form1.cs` si vous ne l'avez pas encore renommé). Ces lignes doivent être ajouté dans le constructeur après la méthode `InitializeComponent()` :

```
ZoneDessin zoneDessin = new ZoneDessin();
Controls.Add(zoneDessin);
```

- Placez le code ci-dessous dans le corps de la classe `ZoneDessin` :

```
//Coordonnées du point cliqué
private int x;
private int y;

public ZoneDessin1() : base()
{
    this.Location = new System.Drawing.Point(50, 50);
    this.Size = new System.Drawing.Size(400, 400);
}

protected override void OnPaintBackground(PaintEventArgs e)
    //Repeint le widget
{
    SolidBrush myBrush = new SolidBrush(Color.DarkBlue);
    e.Graphics.FillEllipse(myBrush, this.x-2, this.y-2, 4, 4);
}

protected override void OnMouseClick(MouseEventArgs e)
    //Identifie le point cliqué et demande à repeindre le
widget
{
    this.x = e.Location.X;
    this.y = e.Location.Y;
    this.Invalidate();//Déclenche l'appel à "repeindre" le
widget
    //this.Refresh(); //Fonctionne également
}
```

Quelles sont les coordonnées du widget dans la fenêtre et quelle est sa taille ?

Correction : Les coordonnées du widget sont à 50 pixels horizontalement et verticalement du coin en haut à gauche de la fenêtre. Le widget est un carré dont les côtés mesurent 400 pixels.

La méthode `OnPaintBackground()` est appelée par le système de multifenêtrage dès que le widget a besoin d'être mis à jour. On peut aussi utiliser la méthode `OnPaint()`, mais cette dernière effectue plus de traitements et peut entraîner des délais d'affichage avec effet de scintillement. La méthode `OnPaintBackground()` reçoit un objet `PaintEventArgs` en argument qui contient de l'information sur le widget en question. En particulier, cet argument

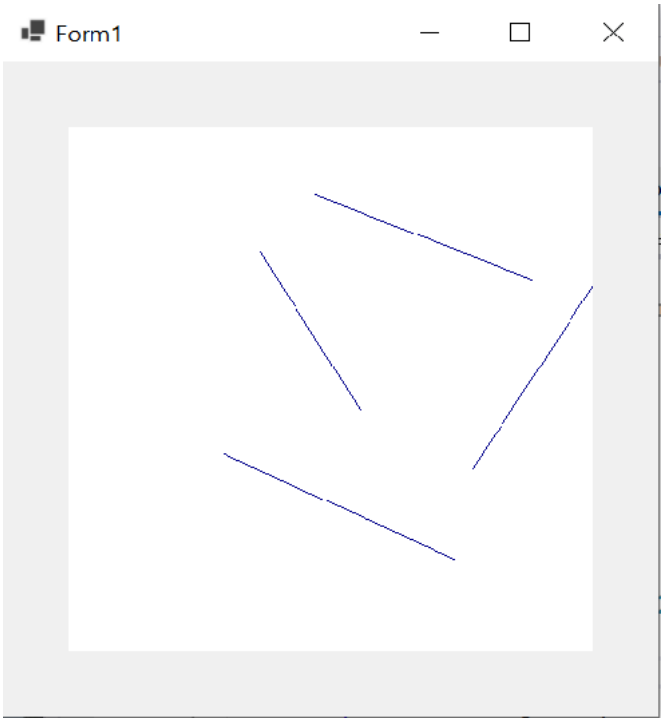
contient un objet `Graphics` qui permet d'effectuer des opérations sur le graphisme du widget.
La méthode `Invalidate()` du widget permet de déclencher une mise à jour du widget.

Dans le code ci-dessous, à quel moment la méthode `OnMouseClick()` est-elle appelé ? Que se passe-t-il alors ?

Correction : La méthode `OnMouseClick()` est appelée quand l'utilisateur clique sur le widget. Un rond de 4 pixels de diamètre apparaît alors à l'endroit du click.

2. Modifier la classe `ZoneDessin` pour créer une fenêtre qui permet de tracer des traits en appuyant sur le bouton souris pour commencer le trait et en relâchant le bouton souris pour terminer le trait. Conseil : Utilisez les méthodes `OnMouseDown()` et `OnMouseUp()` pour récupérer les évènements souris. Pour définir la couleur d'un trait, vous pouvez utiliser la classe `Pen` plutôt que `SolidBrush`.

Exemple de fenêtre :



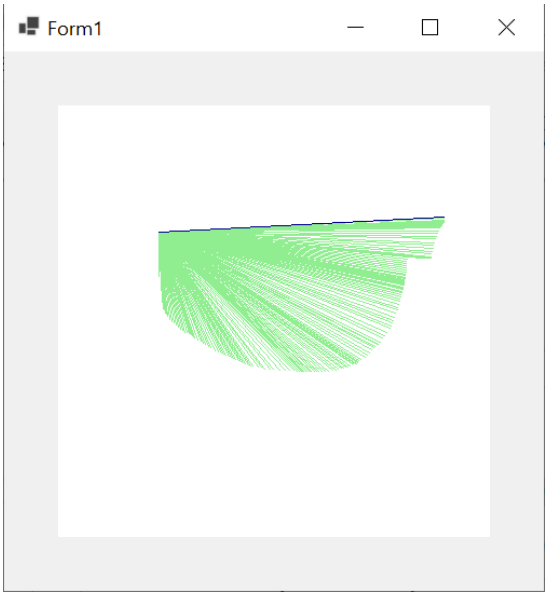
Correction : voir `ZoneDessin2.cs`

Lorsque l'on appuie sur le bouton souris puis qu'on bouge la souris pour choisir le point final du trait, on peut oublier où était exactement le point initial du trait. Pendant ce temps, l'utilisateur n'a donc pas une bonne perception de l'état du système, ce qui contrevient à la première heuristique de Nielsen. Nous allons donc remédier à cela en affichant un trait temporaire qui apparaît, dès qu'on appuie sur le bouton, entre le point initial du trait et le point où se trouve la souris. Ce trait doit varier en même temps que la position de la souris.

3. Ajouter une méthode qui permette de tracer une ligne temporaire lorsque la souris bouge pendant que le bouton est appuyé. Cette ligne doit apparaître dans une autre couleur que celle utilisée pour tracer les traits. Conseil : Utilisez les méthodes `OnMouseMove()`.

Correction : voir `ZoneDessin3.cs`

Comme nous n'avons pas conçu de mécanisme pour effacer la ligne temporaire, celle-ci perdure et créée des traces superflues (voir exemple ci-contre).



Pour remédier à ce problème, nous allons effacer et redessiner l'intégralité des traits dessinés à chaque fois que la souris se déplace, quand le bouton est appuyé. De cette manière, les traces de la ligne temporaire seront effacées dès que celle-ci est modifiée.

1. Créez une classe pour représenter un trait et ajouter un attribut de type *List* pour stocker les traits dans la classe de l'application de dessin. Le trait peut être représenté par un couple de point. Pour information, il existe une classe `System.Drawing.Point` qui permet de représenter des points.

   Changer le(s) méthode(s) responsable(s) de la création des traits pour qu'elle(s) stocke(nt) chaque trait créé. La méthode `OnPaintBackground()` devra effacer toute l'aire de dessin puis réaffiche les traits stockés et, le cas échéant, le trait temporaire. Cette méthode sera appelée à chaque fois que vous créerez un nouveau trait et à chaque fois que vous déplacerez la ligne temporaire. Pour effacer le widget, vous pouvez utiliser la méthode `Clear` objet `Graphics` récupérer par le paramètre `e` de type `PaintEventArgs`. Pour éviter les scintillements, il faut utiliser une mémoire tampon pour créer l'image du widget avant de l'afficher. Pour cela, il faut mettre la propriété `DoubleBuffered` du widget à vrai. Vous devez donc ajouter la ligne suivante au constructeur de `ZoneDessin` :

```
this.DoubleBuffered = true;//Pour éviter le scintillement
```

Correction : voir `ZoneDessin4.cs`