Львівський національний університет імені Івана Франка

Факультет прикладної математики та інформатики

**Паралельні та розподілені обчислення**

**Лабораторна робота №4**

Виконав:

Студент групи ПМі-33

Тимчишин Ярема

Львів 2023

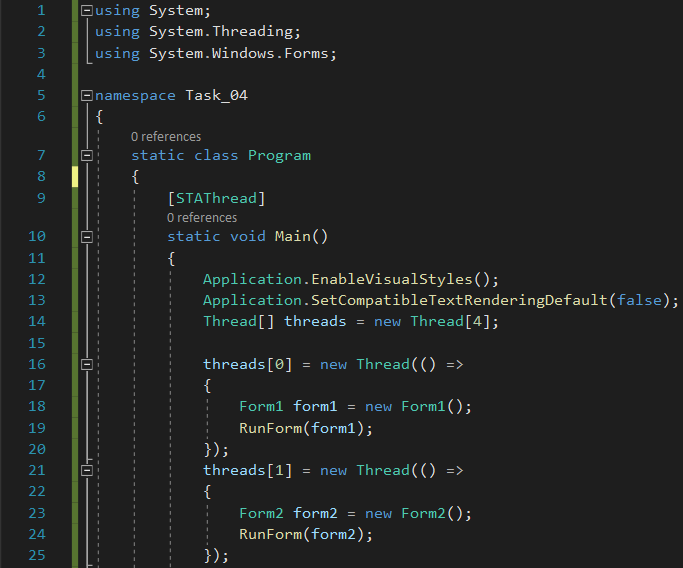
**Тема:** Windows форми.

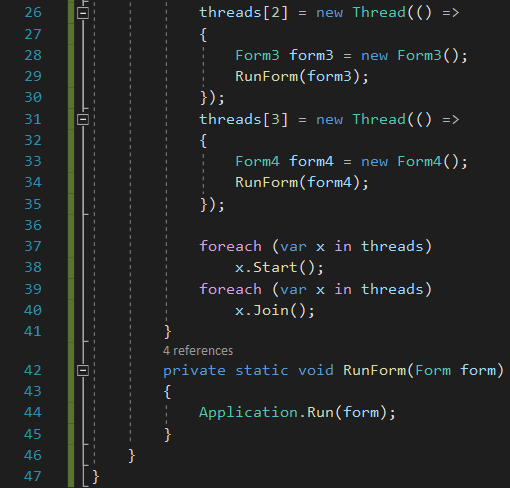
**Завдання:** Створити 4 потоки, в кожному з яких зробити графічне представлення його роботи. Тобто в одному потоці літає кулька по заданій траєкторії, в іншому прямокутник змінює свої розміри, ще в іншому малюється синусоїда тощо. Ці графічні представлення повинні показувати різні (а не однотипні - sin, cos, tg) процеси та чітко відображати безперервну роботу потоку. В кожній з цих 4-х форм створити дві обов’язкові кнопки: одна призупиняє потік (не завершує!), інша – відновлює роботу потоку з точки зупинки. Ще можливі (за бажанням) кнопки завершення потоку та відкривання потоку (у вікні головного потоку).

**Хід роботи**

Використовуючи мову програмування C# написав програму, яка виконує поставлене завдання.

**Program.cs**

****



В цьому файлі знаходиться головний “вхід” в програму.

Почав із трьох “using”. Перше імпортує простір імен *System*, який надає фундаментальні типи та базові класи для програми. Друге імпортує простір імен *System.Threading*, який надає класи та інтерфейси для багатопотоковості. Третє імпортує простір імен *System.Windows.Forms*, який надає класи для створення Windows-додатків.

Статистичний клас *Program* слугує точкою входу для програми.

*[STAThread]* у цій програмі використовується для того, щоб гарантувати, що програма використовує відповідну модель потоків (STA), яка вимагається фреймворком Windows Forms та будь-якими COM-компонентами, з якими вона може взаємодіяти, забезпечуючи належну поведінку та функціональність програми.

Метод *Application.EnableVisualStyles* - це метод класу *System.Windows.Forms.Application* мови C#. Він вмикає візуальні стилі для програми, дозволяючи їй використовувати візуальні стилі, надані операційною системою для відображення елементів керування та елементів інтерфейсу користувача.

Метод *Application.SetCompatibleTextRenderingDefault* - це метод класу *System.Windows.Forms.Application* мови C#. Передаючи сюди *false*, я вказуєте програмі використовувати GDI для рендерингу тексту, забезпечуючи сумісність зі старими програмами і системами.

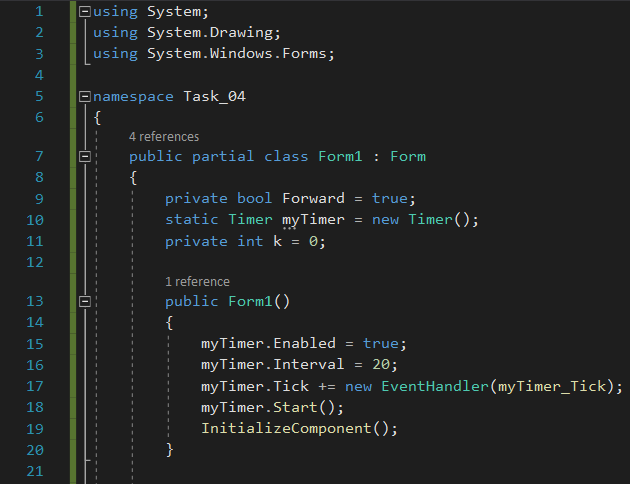
Оголошую та ініціалізую масив з іменем *threads* типу *Thread*, який міститиме чотири об'єкти типу *Thread*.

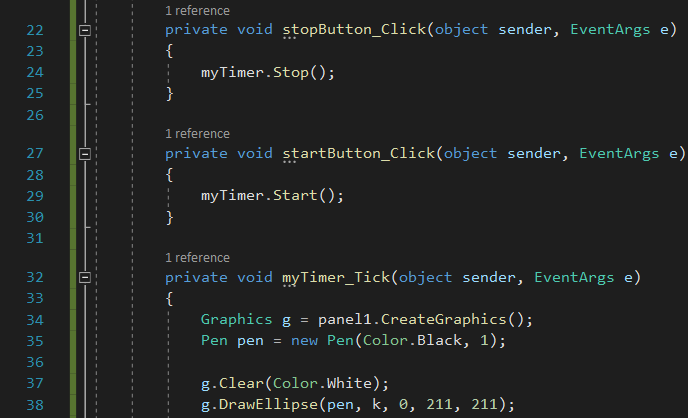
Створюю чотири об'єкти *Thread*, які асоціюються з певними формами (Form1, Form2, Form3, Form4). Кожен блок створює новий потік, який запускає певну форму, створюючи екземпляр відповідного класу форми, а потім викликає метод *RunForm* для запуску форми за допомогою методу *Application.Run*.

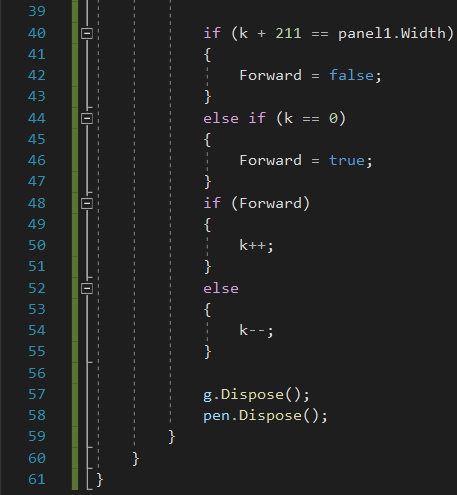
Після чого ці потоки запускаються і об'єднуються за допомогою циклів *foreach*, щоб паралельно виконувати дії, визначені для кожного потоку, ефективно запускаючи кожну форму одночасно.

Метод *RunForm* - це приватний статичний метод, який отримує об'єкт *Form* як параметр і запускає його за допомогою методу *Application.Run*.

**Form1.cs**

****

****

****

Цей файл визначає форму Windows з іменем *Form1* на мові C# з використанням фреймворку Windows Forms.

Знову три “using”. Тут *using System.Drawing;* імпортує простір імен *System.Drawing*, який надає доступ до базової графічної функціональності GDI+.

*public partial class Form1 : Form*: загальнодоступний частковий клас з іменем *Form1*, який успадковується від класу *Form*, роблячи його формою Windows.

Приватна логічна змінна *Forward* з початковим значенням *true*. Ця змінна буде використовуватися для відстеження напрямку руху, де *true* вказує на рух вперед, а *false* - назад.

Статистичний таймер *myTimer*, який буде використовуватися для запуску методу *myTimer\_Tick* через певні проміжки часу.

Приватна цілочисельна змінна *k* з початковим значенням 0. Ця змінна буде використовуватися для відстеження позиції або координати, в якій еліпс малюється або переміщується в межах панелі.

Конструктор класу *Form1*. В ньому виконується декілька важливих дій для ініціалізації форми та налаштування таймера для періодичного оновлення. Спочатку встановлюю властивість *Enabled* таймера *myTimer* у значення *true*. Коли таймер буде увімкнено, він буде генерувати події через певні проміжки часу, визначені властивістю *Interval*. Далі встановлюю властивість *Interval* таймера *myTimer* в 20 мілісекунд. Це означає, що обробник події *myTimer\_Tick* буде запускатися кожні 20 мілісекунд. Потім я “підписую” метод *myTimer\_Tick* на подію *Tick* таймера. Тобто при виникненні події *Tick* (у моєму випадку, це кожні 20 мілісекунд) буде викликано метод *myTimer\_Tick*. Після чого, я запускаю таймер, який після запуску почне генерувати подію *Tick* з вказаним інтервалом (20 мілісекунд). І на кінець, виклик методу *InitializeComponent()* ініціалізує компоненти форми. Загалом, цей блок коду гарантуює, що таймер (*myTimer*) буде налаштовано і він почне цокати з інтервалом у 20 мілісекунд, запускаючи метод *myTimer\_Tick*, який відповідає за малювання та переміщення еліпса на панелі форми.

Метод *stopButton\_Click* є обробником події кліку на кнопці "Стоп" у класі *Form1*. Коли на формі натискається кнопка "Стоп", цей обробник події зупиняє таймер (*myTimer*), фактично призупиняючи періодичне спрацьовування обробника події *myTimer\_Tick*. Ця дія призводить до зупинки руху еліпса на панелі форми.

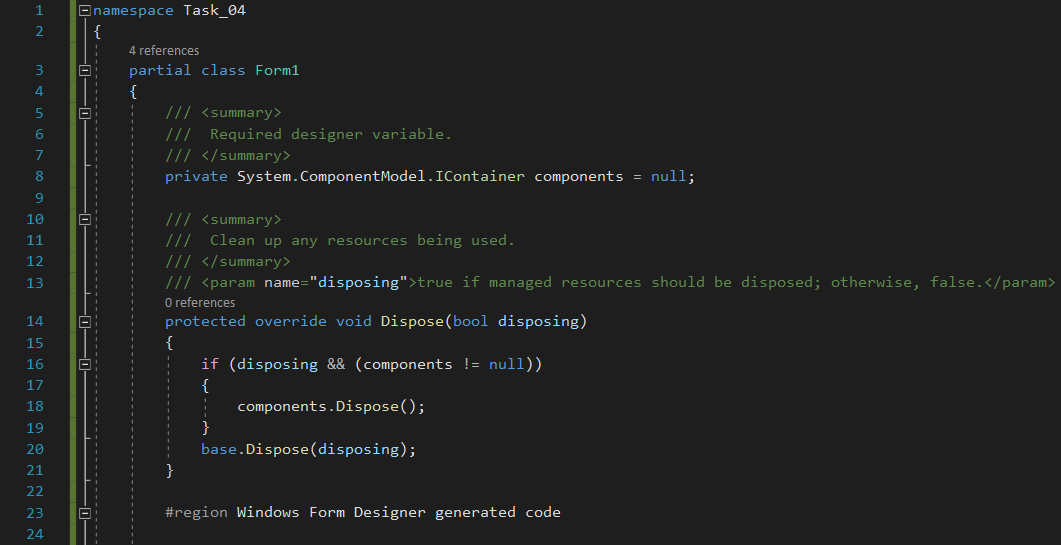
Метод *startButton\_Click* є обробником події кліку на кнопці "Старт" у класі *Form1*. При натисканні кнопки "Старт" на формі, цей обробник події запускає таймер (*myTimer*), що дозволяє періодично запускати обробник події *myTimer\_Tick*. В результаті рух еліпса на панелі форми відновлюється.

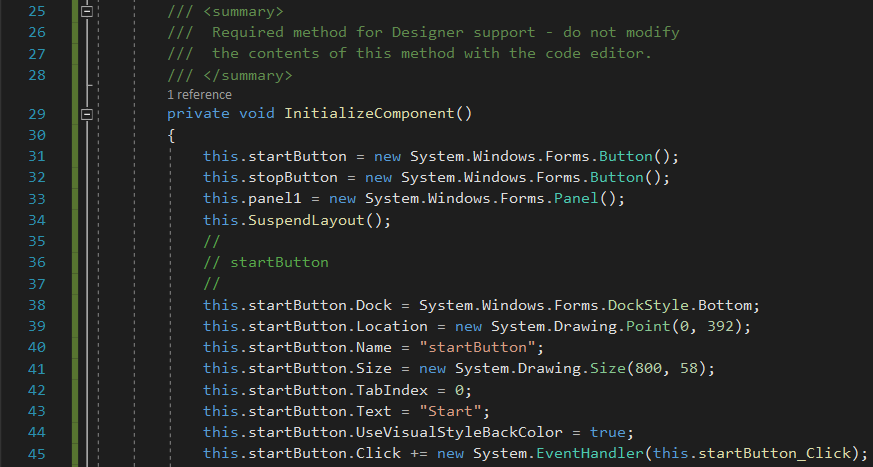
Метод *myTimer\_Tick* є обробником події *Tick* таймера *myTimer*. Цей метод визначає поведінку того, що відбувається на кожному тіку таймера. Спочатку створюю графічний об'єкт з іменем *g*, пов'язаний з панеллю (*panel1*), на якій буде намальовано еліпс. *CreateGraphics()* повертає об'єкт *Graphics*, який надає функції малювання для вказаного елементу управління. Також створюю об'єкт *Pen* з іменем *pen* чорного кольору і шириною 1 одиниця, який буде малювати еліпс. Забарвлюю панель білим кольором. Після чого, методом *DrawEllipse* об'єкту *Graphics (g)* малюю, власне, еліпс, починаючи з координат (*k*, 0), з шириною та висотою 211 одиниць.

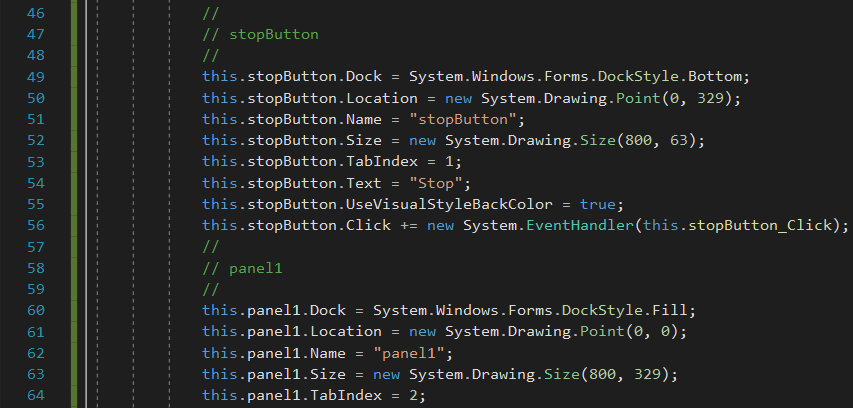
Наступний блок коду відповідає за керування рухом еліпса в межах панелі1. Спочатку два цикли для умов змінення напрямку руху еліпса. Перший *if* перевіряє, чи поточна позиція *k* + 211 (ширина еліпса) дорівнює ширині *panel1*. Якщо *true*, це означає, що еліпс ось-ось досягне правого краю панелі, тому *Forward* встановлюється у *false*, щоб змінити напрямок руху на зворотний. Далі *else if* перевіряє, чи поточна позиція *k* дорівнює 0. Якщо *true*, це означає, що еліпс знаходиться у крайній лівій позиції, тому *Forward* встановлюється у *true*, щоб змінити напрямок руху на “forward”. Наступні два цикли відповідають за оновлення позиції еліпса *k*. *if* збільшує *k* на 1, якщо *Forward* має значення *true*. Це переміщує еліпс вперед. *else* зменшує *k* на 1, якщо *Forward* має значення *false*. Це переміщує еліпс назад.

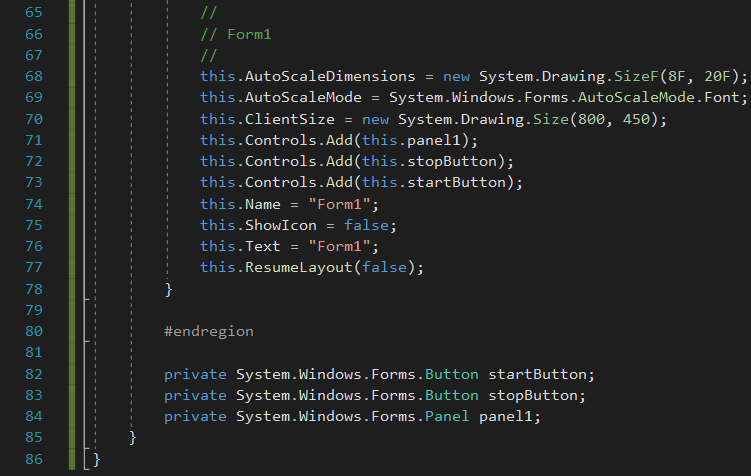
В кінці знищую об’єкт *Graphics* *(g)* та об’єкт *Pen (pen)*.

**Form1.Designer.cs**

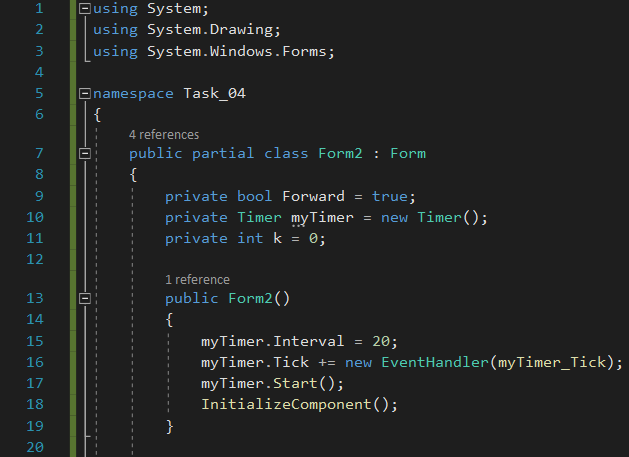


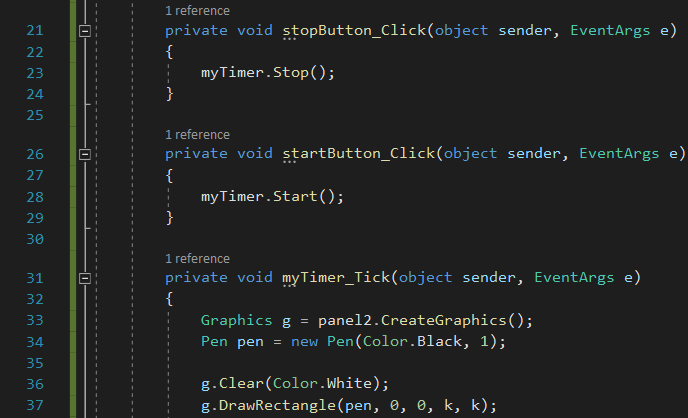


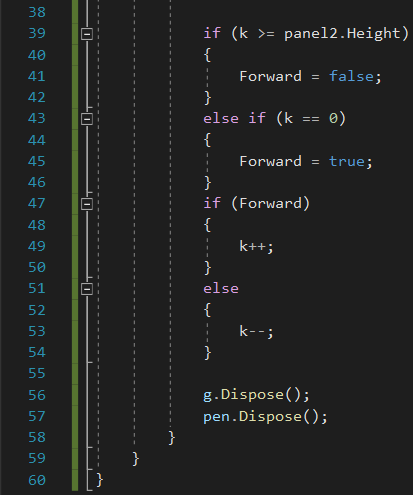




**Form2.cs**



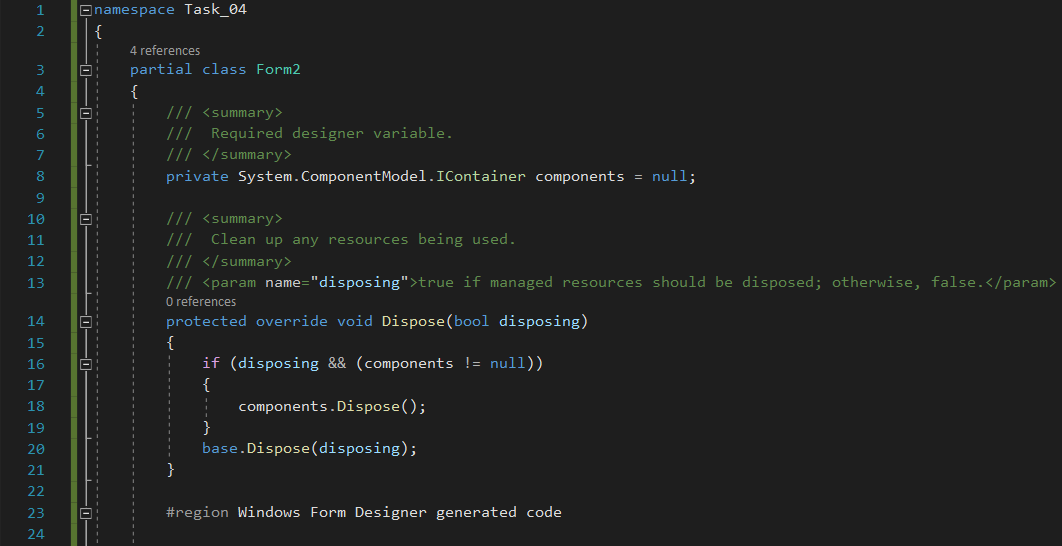


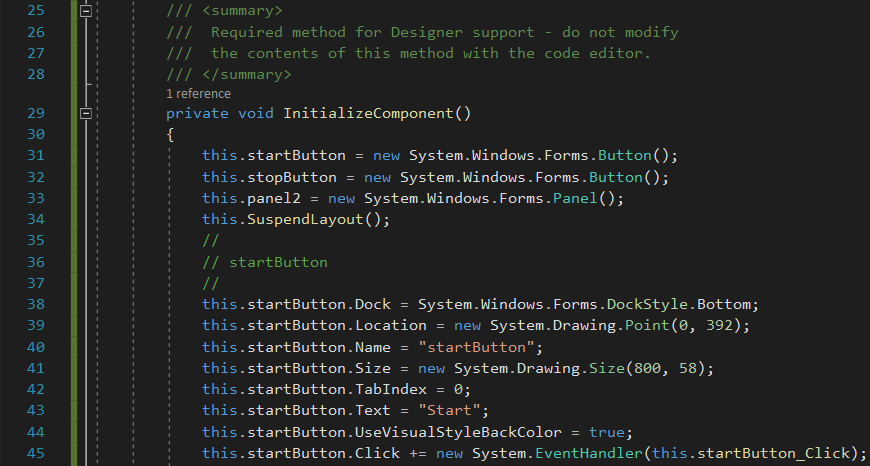


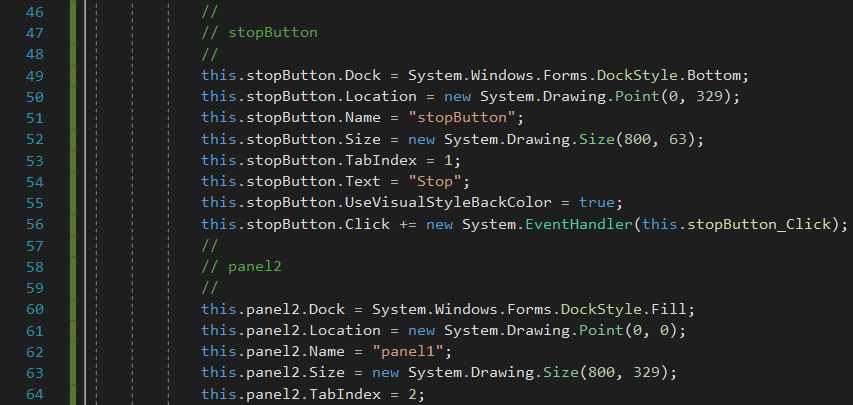
Цей файл визначає форму Windows з іменем *Form2* на мові C# з використанням фреймворку Windows Forms.

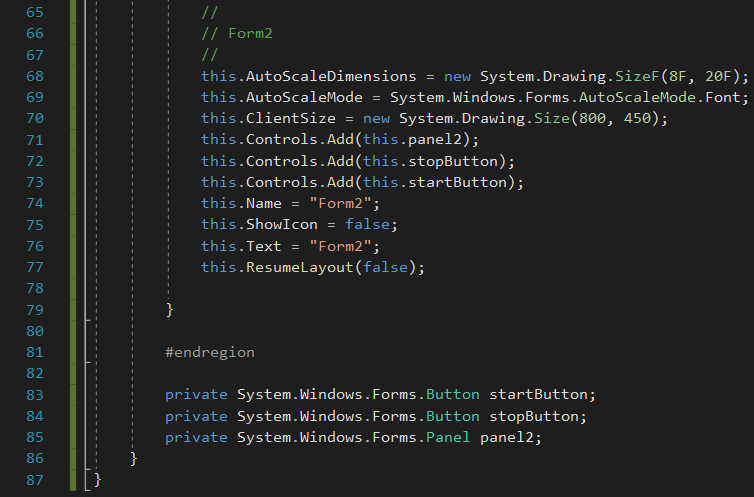
Загалом структура цієї форми є схожою до структури першої форми. Відрізняються вони тим, що в цій формі малюється прямокутник, а не еліпс, і напрямки руху теж різні: еліпс рухається по горизонталі в межах ширини панелі, прямокутник в свою чергу, вертикально зростає в межах висоти панелі (*g.DrawRectangle(pen, 0, 0, k,k)*).

**Form2.Designer.cs**

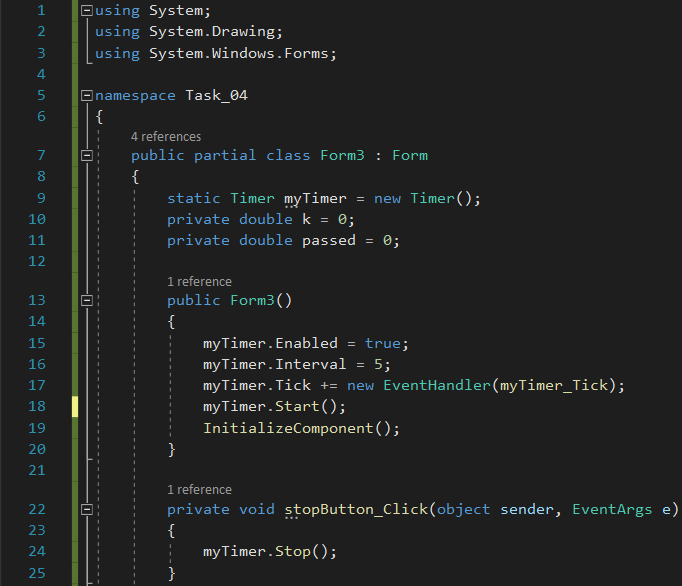


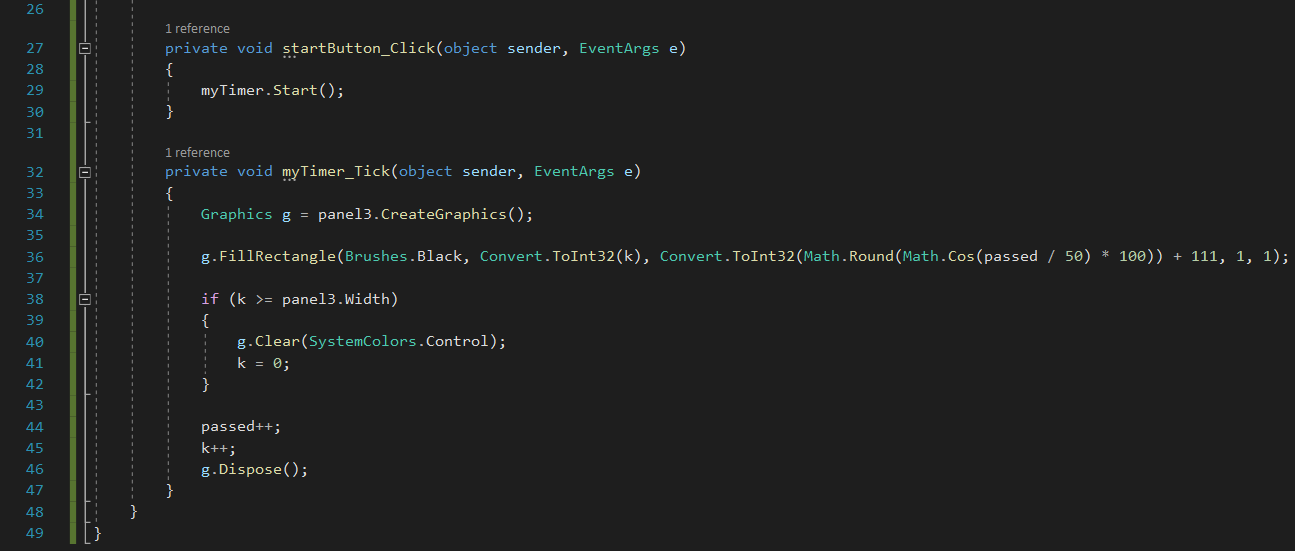






**Form3.cs**



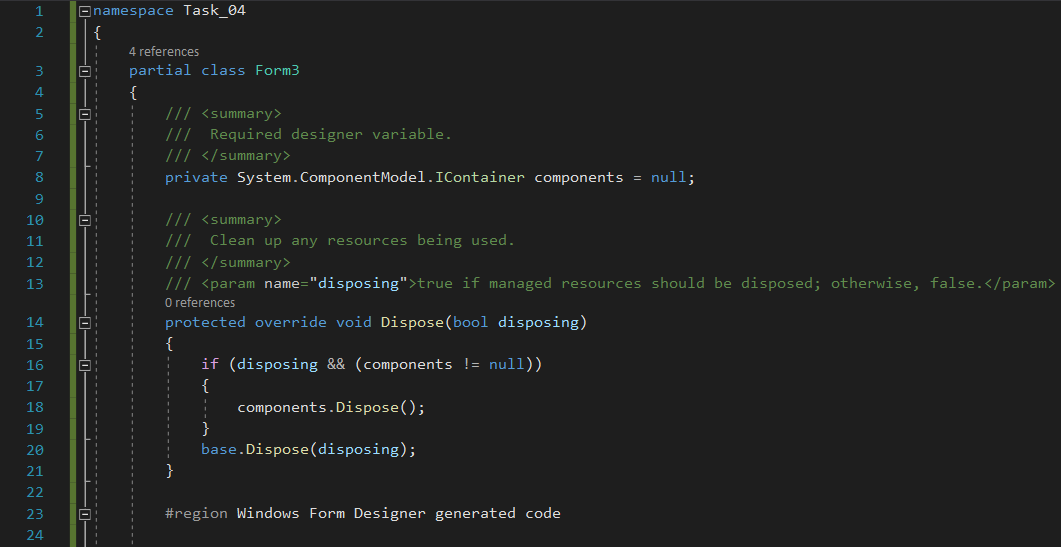


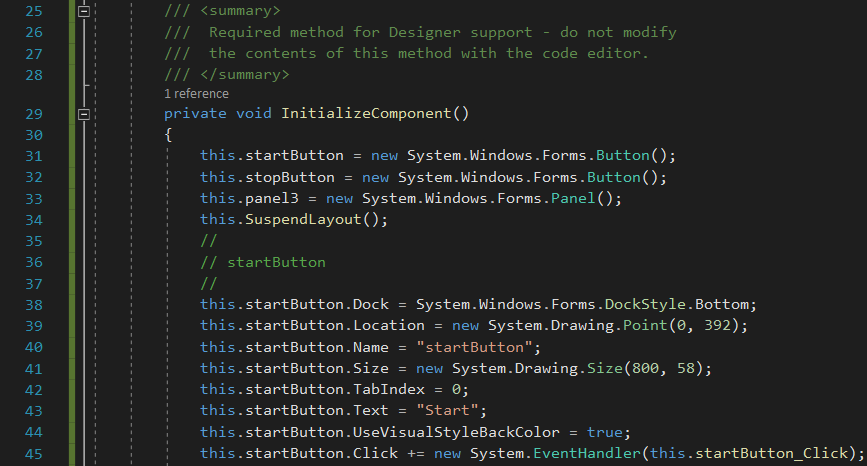
Цей файл визначає форму Windows з іменем *Form3* на мові C# з використанням фреймворку Windows Forms.

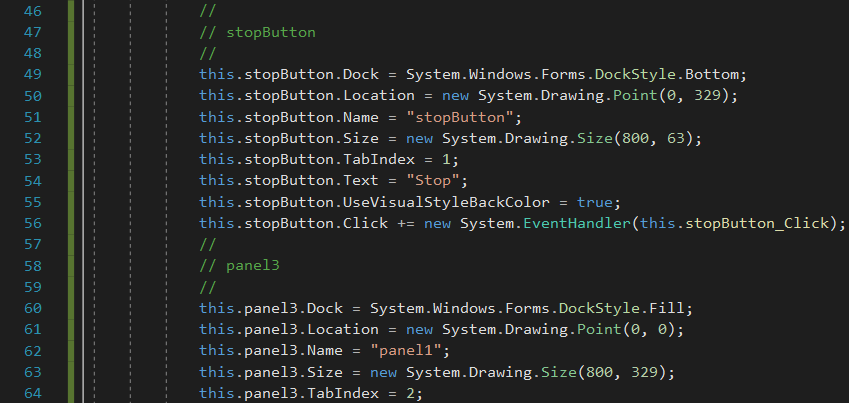
Знову ж таки, структура цієї форми є схожою до структур першої та другої форм, тільки в цій формі, малюється синусоїда. Основні відмінності:

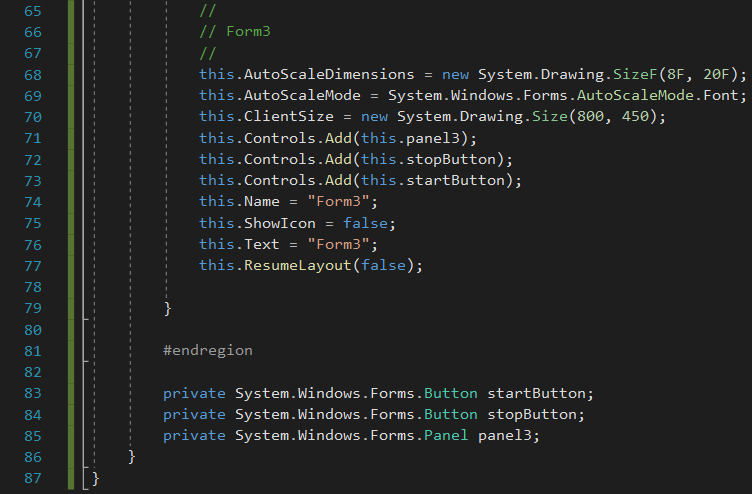
1. Приватна змінна *passed* – використовується для відстеження кількості ітерацій події таймера в класі *Form3*. Її початкове значення дорівнює 0, воно оновлюється і використовується в обробнику події *myTimer\_Tick* для керування анімацією або поведінкою форми.
2. Заповнення прямокутника, або малювання синусоїди – викликаю методу *FillRectangle* об'єкту *Graphics (g)* для заповнення прямокутника вказаним пензлем (у цьому випадку *Brushes.Black*, тобто чорний). Перетворюю значення *k* (яке є double) на ціле число (int). Це ціле значення використовується як X-координата для верхнього лівого кута прямокутника. Після чого обчислюю координату Y для лівого верхнього кута прямокутника. Ось як відбуваються обчислення: ділимо *passed* (змінна, яка представляє кількість тиків таймера) на 50, що вказує на деяке масштабування або регулювання, пов'язане з часом; Обчислюю косинус результату попереднього ділення *passed* на 50; Після чого косинус заокруглюється до цілого числа а також множиться на 100 та додається 111. Операції множення і додавання тут для того, щоб збільшити зображення до більшого діапазону.
3. Цикл if – перевіряє, чи значення *k* не перевищило ширину панелі. Якщо так, то він очищає поверхню малювання панелі і скидає координату X (*k*) на 0, гарантуючи, що прямокутники знову почнуть малюватися з лівого боку панелі, створюючи безперервний цикл анімації.

**Form3.Designer.cs**

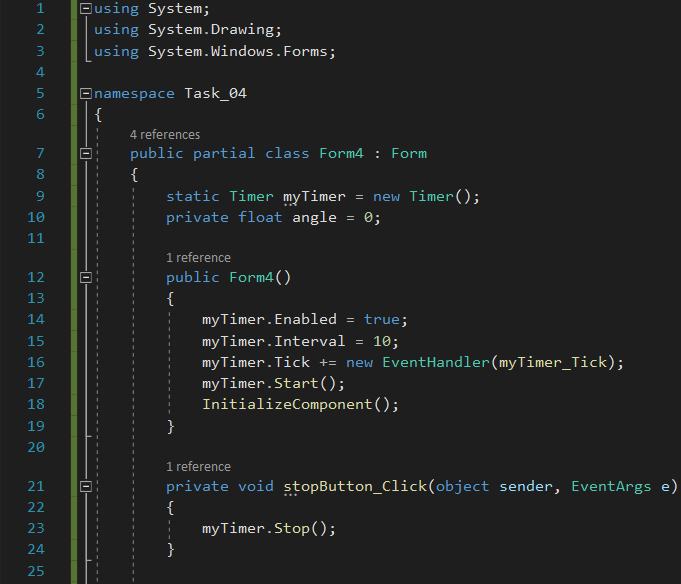


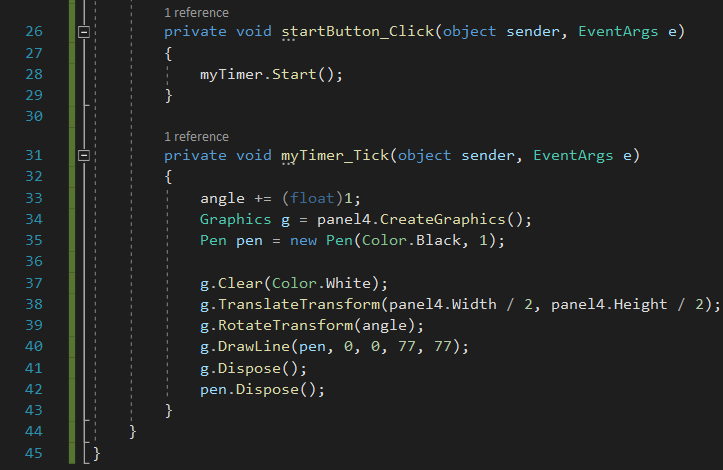






**Form4.cs**



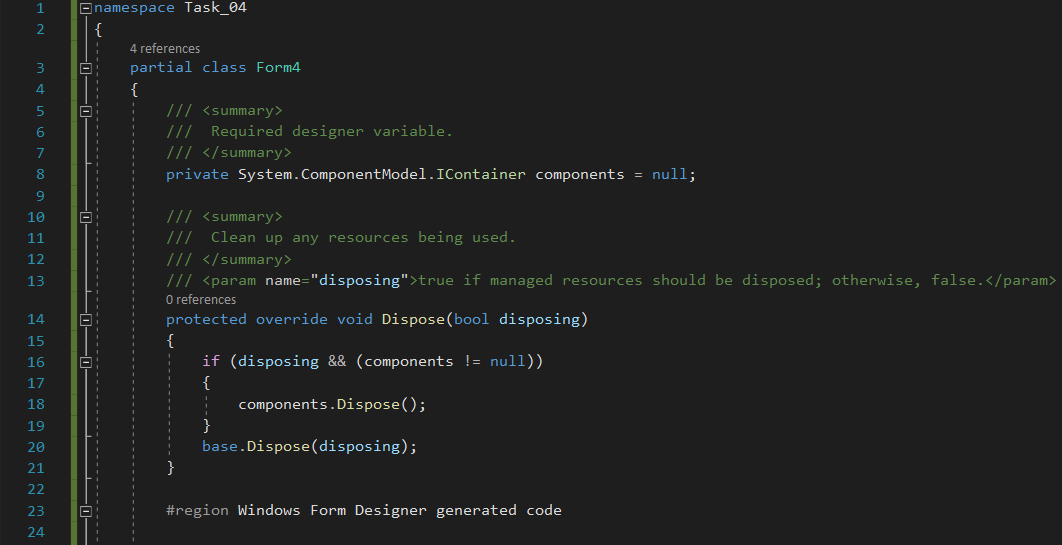


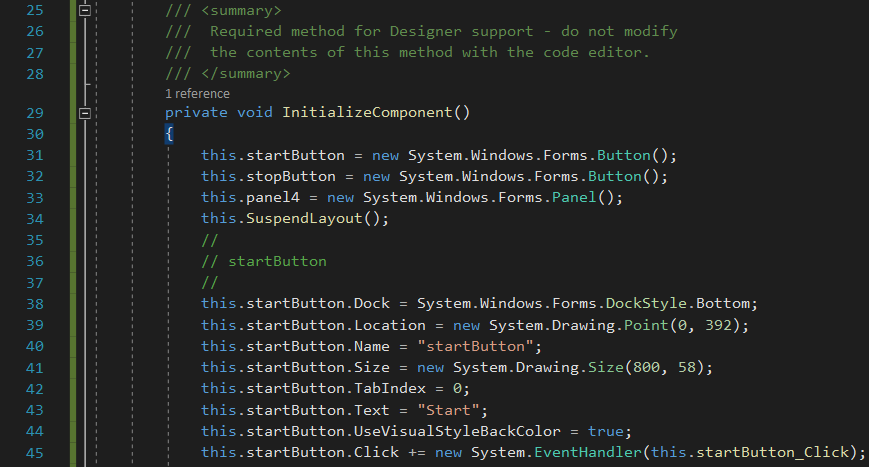
Цей файл визначає форму Windows з іменем *Form3* на мові C# з використанням фреймворку Windows Forms.

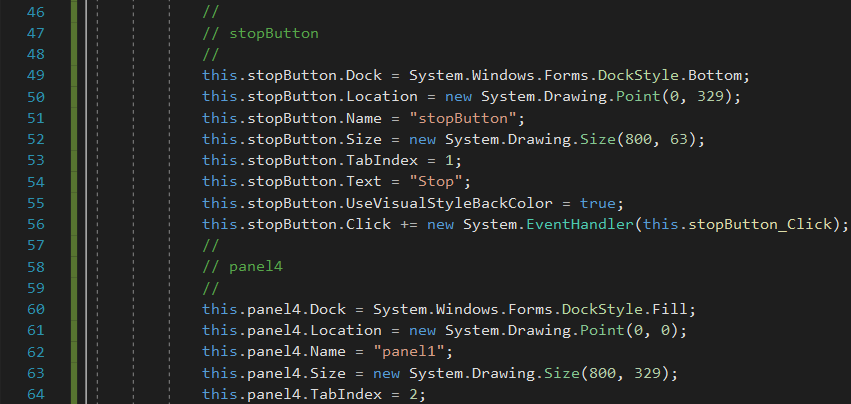
Ну і тут теж, структура цієї форми є схожою до структур першої та другої форм, тільки в цій формі, малюється стрілка яка крутиться. Основні відмінності цієї форми від попередніх:

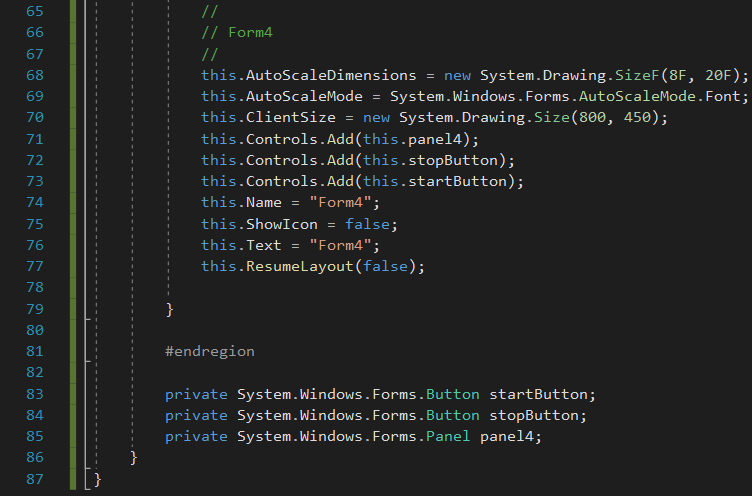
1. Приватна змінна *angle* – використовується для відстеження кута повороту в анімації лінії, що обертається, яка відображається на формі Form4.
2. Малювання стрілки що крутиться – тут все доволі просто, постійно збільшую кут на одиницю, переношу систему координат об'єкта Graphics (g) до нового початку координат, який встановлюється в центрі панелі, Повертає систему координат об'єкта *Graphics* (*g*) на заданий кут (*angle*), малюю лінію у перетвореній системі координат методом *DrawLine* об'єкта *Graphics* (*g*). Лінія проводиться від (0, 0) до (77, 77) у перетвореній системі координат. Після чого знищую об’єкт *Graphics* *(g)* та об’єкт *Pen (pen)*.

**Form4.Designer.cs**









**Результати роботи програми:**

Додаю посилання на Google Drive де Ви зможете знайти запис екрану на якому продемонстровано роботу програми:  
<https://drive.google.com/file/d/1yXyn1EB5Gs6XKSvnkUG7n8Pa419S0RW6/view>