**Розробка текстового опису дизайну програмного проекту “Monitoring Events on a PC”**

***Високорівневий опис реалізації:***

Monitoring Events on a PC – це зручний застосунок для моніторингу подій на комп’ютері, розроблений для платформи Windows з використанням Windows Forms. Застосунок дозволяє користувачам відслідковувати події, що відбуваються на комп’ютері, та зберігати їх у файлі формату .xml. Система також надає можливість перегляду, фільтрації та видалення історії подій.

***Структура програмного рішення:***

Програмне рішення буде реалізовано з використанням архітектури Model-View-Controller (MVC), яка підходить для застосунків на основі Windows Forms. Ця архітектура розділяє інтерфейс користувача (View), бізнес-логіку (Controller) та дані (Model), забезпечуючи чітке розділення обов'язків і спрощуючи підтримку та тестування застосунку.

**Архітектура класів, призначення та їх взаємодія**

*Model:*

* Event: Представляє подію з її властивостями, такими як тип події, назва об’єкта події, дія та час.
* Path: Представляє статичний клас, який зберігає шляхи до файлів та папок.

*Controller:*

* Watcher: Забезпечує логіку для моніторингу подій, включаючи запис дій користувача, створення та оновлення файлу історії, видалення історії та старих записів.

*View:*

* Form1: Головне вікно застосунку, яке містить елементи інтерфейсу користувача для відображення та керування історією подій.

*Watcher -> Form1:*

* StartMonitoring(): Починає моніторинг подій.
* EndMonitoring(): Зупиняє моніторинг подій.
* PrintEvent(string, string, string): Передає інформацію про подію до Form1 для відображення.

*Form1 -> Watcher:*

* Викликає методи StartMonitoring() та EndMonitoring() для керування моніторингом.

*Form1 -> Event:*

* Отримує список подій (Events) для відображення та обробки.

*Event -> Form1:*

* Надає доступ до списку подій (Events).

*Form1 -> Path:*

* Використовує статичні властивості PathDirectory та SelectedFile для роботи з файлами історії.

**Опис функцій**

* Моніторинг подій: дозволяє користувачам відслідковувати зміни у файловій системі та відкриття/закриття програм.
* Запис подій: записує інформацію про події у файл формату .xml.
* Відображення історії: відображає історію подій у табличному форматі.
* Фільтрація історії: дозволяє користувачам фільтрувати історію подій за типом дії та часом.
* Видалення історії: дозволяє користувачам очистити історію подій або видалити файл з історією.
* Видалення старих записів: автоматично видаляє старі записи з історії подій.

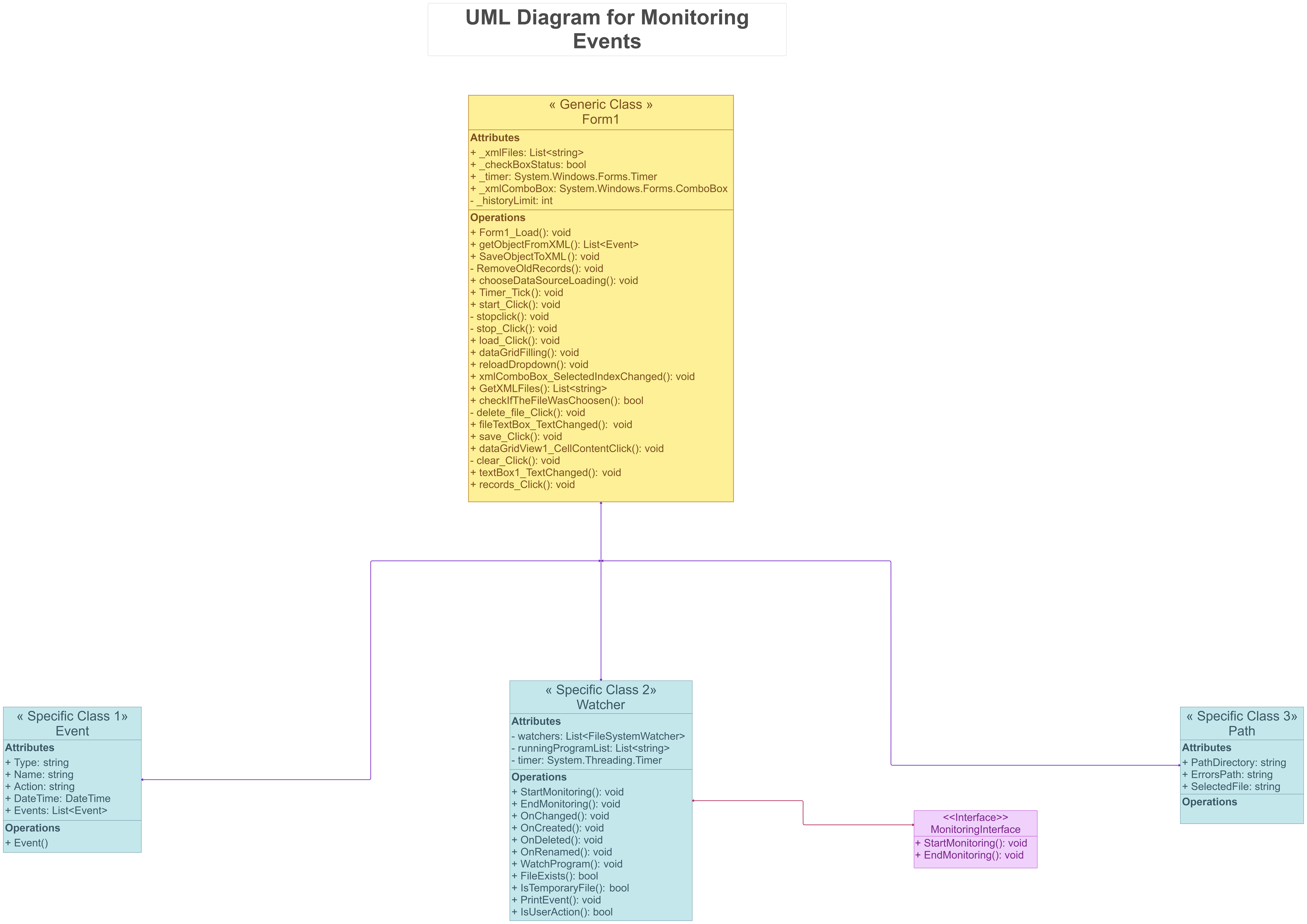
***Перша спроба реалізації каркасу програмного рішення:***

Далі наведено базову реалізацію каркасу форми *Form1* та класу *Watcher* мовою C#. Цей скелет демонструє загальну архітектуру класу та його членів, не включаючи конкретну логіку методів. Цей початковий варіант буде доопрацьовуватися в процесі розробки.

|  |
| --- |
| //Form1.Designer.cs  partial class Form1  {  private System.ComponentModel.IContainer components = null;  protected override void Dispose(bool disposing)  {}  private Button start;  private Button stop;  private Button load;  private Button delete\_file;  private Button save;  private DataGridView dataGridView1;  private DataGridViewTextBoxColumn event\_type;  private DataGridViewTextBoxColumn type;  private DataGridViewTextBoxColumn name;  private DataGridViewTextBoxColumn date;  private TextBox fileTextBox;  private Button clear;  private Button records;  } |

|  |
| --- |
| // Watcher.cs  public class Watcher  {  private List<FileSystemWatcher> watchers;  private List<string> runningProgramList;  private System.Threading.Timer timer;  private static Watcher instance = new Watcher();  private Watcher()  {  runningProgramList = new List<string>();  }  public static Watcher GetInstance()  {  return instance;  }  public void StartMonitoring(string directories) { /\* ... \*/ }  public void EndMonitoring() { /\* ... \*/ }  private void OnChanged(object source, FileSystemEventArgs e) { /\* ... \*/ }  private void OnCreated(object source, FileSystemEventArgs e) { /\* ... \*/ }  private void OnDeleted(object source, FileSystemEventArgs e) { /\* ... \*/ }  private void OnRenamed(object source, RenamedEventArgs e) { /\* ... \*/ }  private void WatchProgram(object temp) { /\* ... \*/ }  private static bool FileExists(string filePath, bool ignoreTemporaryFiles = true) { /\* ... \*/ }  private static bool IsTemporaryFile(string filePath) { /\* ... \*/ }  private void PrintEvent(string type, string fileName, string action) { /\* ... \*/ }  private bool IsUserAction(FileSystemEventArgs e) { /\* ... \*/ }  } |

**Розроблення UML діаграми**



*Рис. 1. UML-діаграма класів програми*