Zespół Szkół Mechanicznych nr 1 im. Franciszka Siemiradzkiego
Technikum nr 10 Mechaniczne z Oddziałami Mistrzostwa Sportowego
Branżowa Szkoła I Stopnia nr 10 Mechaniczna
ul. Św. Trójcy 37, 85-224 Bydgoszcz
zsm1.bydgoszcz.pl

Materiały dydaktyczne

Przedmiot: Programowanie aplikacji desktopowych / Pracownia programowania aplikacji desktopowych

Opracował: Mirosław Miciak

Ćwiczenie: C6

Tematy: Wyszukiwanie danych w listach, kolejkach, tablicach, Framework. WPF w języku C#, Tworzenie prostej aplikacji desktopowej w WPF, Projektowanie interfejsu użytkownika: okna dialogowe, Projektowanie interfejsu użytkownika: przyciski, pola edycyjne, paski narzędziowe, Menu aplikacji desktopowej, Zabezpieczenie aplikacji desktopowej przed błędami użytkownika, Zastosowanie modelu obiektowego: klasy i obiekty, Planowanie aplikacji z zastosowaniem hermetyzacji, Zastosowanie konstruktorów, specyfikatorów dostępu, Zastosowanie konstruktorów kopiujących i destruktorów, Wykorzystanie składowych statycznych klasy, Zastosowanie funkcji zaprzyjaźnionych, Zastosowanie dziedziczenia, Zastosowanie metod wirtualnych, Zastosowanie klas abstrakcyjnych, Zastosowanie przeciążenia metod klas, Zastosowanie szablonów klas, Zastosowanie polimorfizmu w aplikacji, Zastosowanie mechanizmów obsługi wyjątków i zgłaszania wyjątków, Organizacja okien aplikacji: SDI, MDI, Obsługa zdarzeń myszy i klawiatury, Testowanie aplikacji, Komentowanie kodu aplikacji, Tworzenie plików pomocy, Planowanie testów niefunkcjonalnych: użyteczności, wydajnościowe, Planowanie testów: obciążeniowe, zgodności, bezpieczeństwa, Przeprowadzanie testów funkcjonalnych, Stosowanie narzędzi do automatyzacji testowania, Zastosowanie języka np. QML do projektowania interfejsu użytkownika, Wykonanie kompletnej aplikacji desktopowej: notatnik, przeglądarka zdjęć

Temat: Wyszukiwanie danych w listach, kolejkach, tablicach

Zastosowanie konstruktorów

Konstruktor to metoda wywoływana podczas tworzenia obiektu klasy. Jego głównym celem jest inicjalizacja obiektu. W C# konstruktory mogą być przeciążane, co oznacza, że można tworzyć wiele konstruktorów z różnymi parametrami.

Specyfikatory dostępu

Specyfikatory dostępu w C# określają widoczność i dostępność składowych klasy, takich jak pola, właściwości i metody. Wyróżniamy pięć głównych specyfikatorów:

public: Składowa jest dostępna z dowolnego miejsca w kodzie. Przykład:

```
public class Example
{
   public int PublicField;
}
```

private: Składowa jest dostępna tylko w obrębie klasy, w której została zadeklarowana. Jest to domyślny specyfikator dostępu, jeśli nie zostanie podany inny. Przykład:

```
public class Example
{
   private int PrivateField;
}
```

protected: Składowa jest dostępna w obrębie klasy oraz w klasach dziedziczących. Przykład:

```
public class Example
{
   protected int ProtectedField;
}
```

internal: Składowa jest dostępna w obrębie tego samego zestawu (assembly). Przykład:

```
public class Example
{
   internal int InternalField;
}
```

protected internal: Składowa jest dostępna w obrębie tego samego zestawu oraz w klasach dziedziczących. Przykład:

```
public class Example
{
   protected internal int ProtectedInternalField;
}
```

private protected: Składowa jest dostępna w obrębie klasy oraz w klasach dziedziczących, ale tylko w obrębie tego samego zestawu. Przykład:

```
public class Example
{
   private protected int PrivateProtectedField;
}
```

Dzięki specyfikatorom dostępu możesz kontrolować, które składowe klasy są dostępne z zewnątrz, co pomaga w zapewnieniu bezpieczeństwa i enkapsulacji danych.

Konstruktor kopiujący
Konstruktor kopiujący tworzy nowy obiekt jako kopię istniejącego obiektu. Jest używany do głębokiego kopiowania, aby uniknąć problemów związanych z współdzieleniem referencji.

Destruktor
Destruktor jest specjalną metodą wywoływaną, gdy obiekt jest usuwany z pamięci. W C# destruktory są rzadko używane, ponieważ garbage collector automatycznie zarządza pamięcią.

Hermetyzacja (Encapsulation)
Hermetyzacja to koncepcja ukrywania szczegółów implementacji klasy przed zewnętrznymi obiektami. Zwykle realizowana jest za pomocą specyfikatorów dostępu.

Utwórz nowy projekt WPF

1. Otwórz Visual Studio.

2. Kliknij Create a new project.
3. Wyszukaj WPF i wybierz go.
4. Kliknij Next.
5. Nazwij projekt BibliotekaApp i kliknij Create.

Dodaj klasę Ksiazka

1. W Solution Explorer kliknij prawym przyciskiem myszy projekt Biblioteka.
2. Wybierz Add > Class.
3. Nazwij klasę Ksiazka.cs i kliknij Add.
4. Wstaw poniższy kod do Ksiazka.cs:

```
namespace BibliotekaApp
{
   public class Ksiazka
   {
      public string Tytul { get; set; }
      public string Autor { get; set; }
      public int RokWydania { get; set; }

      // Konstruktor domyślny
      public Ksiazka()
      {
         Tytul = "Nieznany";
         Autor = "Nieznany";
         RokWydania = 0;
      }

      // Konstruktor przeciążony
      public Ksiazka(string tytul, string autor, int rokWydania)
      {
         Tytul = tytul;
         Autor = autor;
         RokWydania = rokWydania;
      }

      // Konstruktor kopiujący
      public Ksiazka(Ksiazka innaKsiazka)
      {
         Tytul = innaKsiazka.Tytul;
         Autor = innaKsiazka.Autor;
         RokWydania = innaKsiazka.RokWydania;
      }
   }
}
```

Dodaj klasę Biblioteka

1. W Solution Explorer kliknij prawym przyciskiem myszy projekt Biblioteka.
2. Wybierz Add > Class.
3. Nazwij klasę Biblioteka.cs i kliknij Add.
4. Wstaw poniższy kod do Biblioteka.cs:

```
namespace BibliotekaApp
{
   public class Biblioteka
```

```
    {
        private List<Ksiazka> ksiazki = new List<Ksiazka>();

        // Metoda do dodawania książki
        public void DodajKsiazke(Ksiazka ksiazka)
        {
            ksiazki.Add(ksiazka);
        }

        // Metoda do usuwania książki
        public void UsunKsiazke(Ksiazka ksiazka)
        {
            ksiazki.Remove(ksiazka);
        }

        // Metoda do wyświetlania wszystkich książek
        public List<Ksiazka> PobierzKsiazki()
        {
            return ksiazki;
        }
    }
}
```

Dodaj interfejs użytkownika do MainWindow.xaml

1. Otwórz plik MainWindow.xaml.
2. Wstaw poniższy kod:

```
<Window x:Class="BibliotekaApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="BibliotekaApp" Height="350" Width="500">
    <Grid>
        <StackPanel>
            <TextBox x:Name="TytulTextBox" Margin="10" Text="Tytuł"/>
            <TextBox x:Name="AutorTextBox" Margin="10" Text="Autor"/>
            <TextBox x:Name="RokWydaniaTextBox" Margin="10" Text="Rok wydania"/>
            <Button Content="Dodaj książkę" Click="DodajKsiazke_Click" Margin="10"/>
            <Button Content="Wyświetl książki" Click="WyswietlKsiazki_Click" Margin="10"/>
            <TextBlock x:Name="KsiazkiTextBlock" Margin="10"/>
        </StackPanel>
    </Grid>
</Window>
```

Dodaj logikę do MainWindow.xaml.cs

1. Otwórz plik MainWindow.xaml.cs.
2. Wstaw poniższy kod:

```
using System.Windows;
namespace BibliotekaApp
{
    public partial class MainWindow : Window
    {
        private Biblioteka biblioteka = new Biblioteka();
        public MainWindow()
        {
```

```
        InitializeComponent();
        // Dodawanie przykładowych książek
        biblioteka.DodajKsiazke(new Ksiazka("Czerwony Kapturek", "Bracia Grimm", 1812));
        biblioteka.DodajKsiazke(new Ksiazka("Kopciuszek", "Bracia Grimm", 1812));
        biblioteka.DodajKsiazke(new Ksiazka("Śpiąca Królewna", "Charles Perrault", 1697));
        biblioteka.DodajKsiazke(new Ksiazka("Jaś i Małgosia", "Bracia Grimm", 1812));
        biblioteka.DodajKsiazke(new Ksiazka("Trzy Małe Świnki", "Joseph Jacobs", 1890));
        // zastosowanie konstruktora kopiującego
        Ksiazka ksiazka1 = new Ksiazka("1984", "George Orwell", 1949);
        biblioteka.DodajKsiazke(new Ksiazka(ksiazka1));
    }
    private void DodajKsiazke_Click(object sender, RoutedEventArgs e)
    {
        string tytul = TytulTextBox.Text;
        string autor = AutorTextBox.Text;
        int rokWydania = int.Parse(RokWydaniaTextBox.Text);
        Ksiazka ksiazka = new Ksiazka(tytul, autor, rokWydania);
        biblioteka.DodajKsiazke(ksiazka);
    }

    private void WyswietlKsiazki_Click(object sender, RoutedEventArgs e)
    {
        KsiazkiTextBlock.Text = "Książki w bibliotece:\n";
        foreach (var ksiazka in biblioteka.PobierzKsiazki())
        {
            KsiazkiTextBlock.Text += $"{ksiazka.Tytul} - {ksiazka.Autor} ({ksiazka.RokWydania})\n";
        }
    }
  }
}
```

Zadania do samodzielnego rozwiązania

Zadanie 1 Na podstawie przykładu BibliotekaApp utwórz aplikację KatalogFilmowApp, gdzie zdefiniujemy klasę Film, która ma postać:

```
public class Film
{
  public int Index { get; set; }
  public string Tytul { get; set; }
  public string Rezyser { get; set; }
  public int Rok { get; set; }
  public string Gatunek { get; set; }

  public Film(int index, string tytul, string rezyser, int rok, string gatunek)
  {
    Index = index;
    Tytul = tytul;
    Rezyser = rezyser;
    Rok = rok;
    Gatunek = gatunek;
  }

  public string WyswietlInfo()
  {
    return $"Index {Index} - {Tytul}, reż. {Rezyser}, {Rok}, gatunek: {Gatunek}";
  }
```

```
}
```

Aplikacja ma mieć te same funkcjonalności co przykład BibliotekaApp.

Zadanie 2 W definicji Klasy Książka przykładu BibliotekaApp, dodaj pole Index, Dodaj przycisk "Usuń książkę", który pozwala na usunięcie książki z biblioteki np. po numerze Index. Możesz użyć metody Remove();

Zadanie 3 Dodaj możliwość edytowania danych książki (np. zmiana autora lub roku wydania). Możesz zastosować istniejący przycisk Dodaj książkę i wtedy sprawdzać czy dodawana pozycja literaturowa ma istniejący index, wtedy zapisujesz nowe dane na istniejącym indeksie. Moja propozycja metody wprowadzającej zmiany w klasie Biblioteka wygląda następująco:

```
public void UpdateKsiazka(int index, string tytul, string autor, int rok)
{
   var ksiazka = ksiazki.FirstOrDefault(k => k.Index == index);
   if (ksiazka != null)
   {
      ksiazki[ksiazki.FindIndex(k => k.Index == index)].UpdateKsiazka(index,tytul,autor,rok);
   }
}
```

Natomiast metoda modyfikująca dane książki w klasie może wyglądać następująco:

```
public void UpdateKsiazka(int index, string title, string author, int year)
 {
    Index = index;
    Tytul = title;
    Autor = author;
    RokWydania = year;
 }
```

Oraz część z MainWindow:

```
int index = int.Parse(IndexTextBox.Text);
string tytul = TytulTextBox.Text;
string autor = AutorTextBox.Text;
int rokWydania = int.Parse(RokWydaniaTextBox.Text);
int jest = 0;
foreach (var ksiazka_temp in biblioteka.PobierzKsiazki())
{
   if (ksiazka_temp.Index == index)
   {
      biblioteka.UpdateKsiazka(index,tytul,autor,rokWydania);
      jest = 1;  break;

   }
}
```

Oczywiście nie musisz się sugerować tą propozycją i opracować własne rozwiązanie,