

## Л12. Розробка користувацького інтерфейсу засобами С#

**Контейнер Frame** використовується для оформлення або створення фону для вкладеного елемента. Серед властивостей класу *Frame* слід виділити такі:

- **BorderColor**: колір кордону кадру за допомогою структури *Color*;
- **CornerRadius**: представляє радіус кордону кадру як значення типу *float*;
- **HasShadow**: зберігає значення типу *bool*, яке вказує, чи кадр відкидає тінь.

Створення *Frame* у XAML:

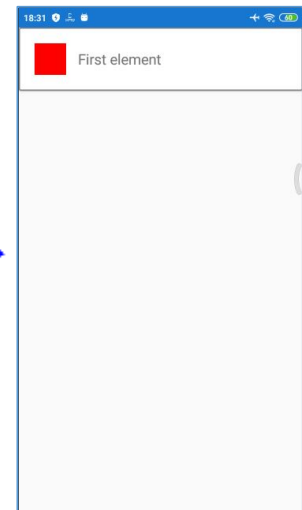
Створення *Frame* у С#:

```
Frame frame = new Frame
{
    Content = new Label { Text = "Hello Xamarin" }
};
```

```
<Frame>
  <Label Text="Hello Xamarin" />
</Frame>
```

Створюємо на базі *Empty* шаблону проєкт зі списку елементів. Кожний елемент списку складається з *BoxView* та *Label*

```
<StackLayout>
  <Frame BorderColor="Gray">
    <StackLayout Orientation="Horizontal" Spacing="15">
      <BoxView Color="Red"></BoxView>
      <Label Text="First element" VerticalOptions="Center"></Label>
    </StackLayout>
  </Frame>
</StackLayout>
```



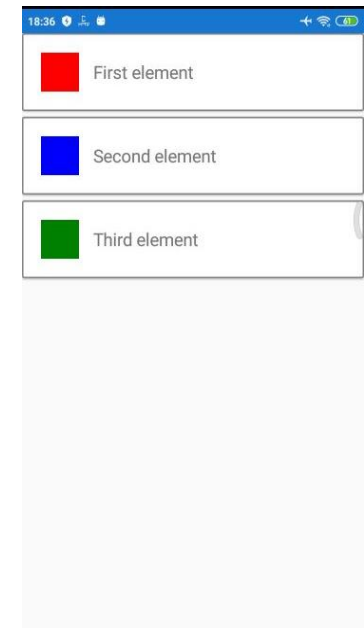
# Л12. Розробка користувацького інтерфейсу засобами С#

## Дублюємо Frame

```
<StackLayout>
  <Frame BorderColor="Gray">
    <StackLayout Orientation="Horizontal" Spacing="15">
      <BoxView Color="Red"></BoxView>
      <Label Text="First element" VerticalOptions="Center"></Label>
    </StackLayout>
  </Frame>

  <Frame BorderColor="Gray">
    <StackLayout Orientation="Horizontal" Spacing="15">
      <BoxView Color="Blue"></BoxView>
      <Label Text="Second element" VerticalOptions="Center"></Label>
    </StackLayout>
  </Frame>

  <Frame BorderColor="Gray">
    <StackLayout Orientation="Horizontal" Spacing="15">
      <BoxView Color="Green"></BoxView>
      <Label Text="Third element" VerticalOptions="Center"></Label>
    </StackLayout>
  </Frame>
</StackLayout>
```

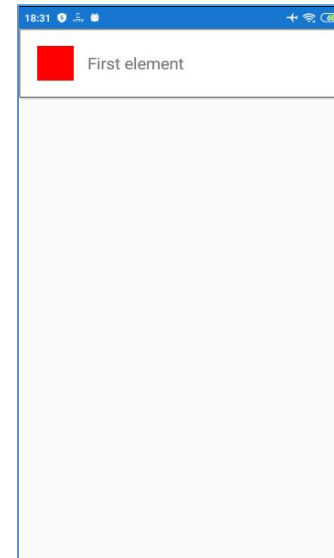


## Л12. Розробка користувацького інтерфейсу засобами С#

Реалізуємо теж саме в коді С#. Видаляємо усе з `ContentPage` та дописуємо код в метод `OnAppearing()`

```
protected override void OnAppearing()
{
    StackLayout _stackLayout1 = new StackLayout();

    Frame _frame1 = new Frame()
    {
        Content = new StackLayout()
        {
            Children =
            {
                new BoxView() { Color = Color.Red},
                new Label() { Text="First element", VerticalOptions=LayoutOptions.Center}
            },
            Orientation = StackOrientation.Horizontal,
            Spacing = 15
        },
        BorderColor = Color.Red
    };
}
```



## Л12. Розробка користувацького інтерфейсу засобами C#

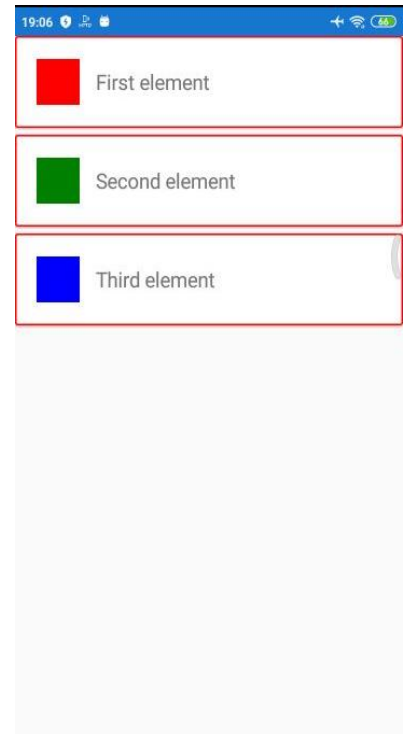
Додаємо, ще 2 Frame, додаємо frame1, frame2, frame3 до stackLayout1 (копіювання Ctrl+D) та Content= stackLayout1

```
Frame frame2 = new Frame()
{
    Content = new StackLayout()
    {
        Children =
        {
            new BoxView() { Color = Color.Green},
            new Label() {Text="Second element", VerticalOptions=LayoutOptions.Center}
        },
        Orientation = StackOrientation.Horizontal,
        Spacing = 15
    },
    BorderColor = Color.Red
};

Frame frame3 = new Frame()
{
    Content = new StackLayout()
    {
        Children =
        {
            new BoxView() { Color = Color.Blue},
            new Label() {Text="Third element", VerticalOptions=LayoutOptions.Center}
        },
        Orientation = StackOrientation.Horizontal,
        Spacing = 15
    },
    BorderColor = Color.Red
};

stackLayout1.Children.Add(frame1);
stackLayout1.Children.Add(frame2);
stackLayout1.Children.Add(frame3);

Content = stackLayout1;
}
```



## Л12. Розробка користувацького інтерфейсу засобами C#

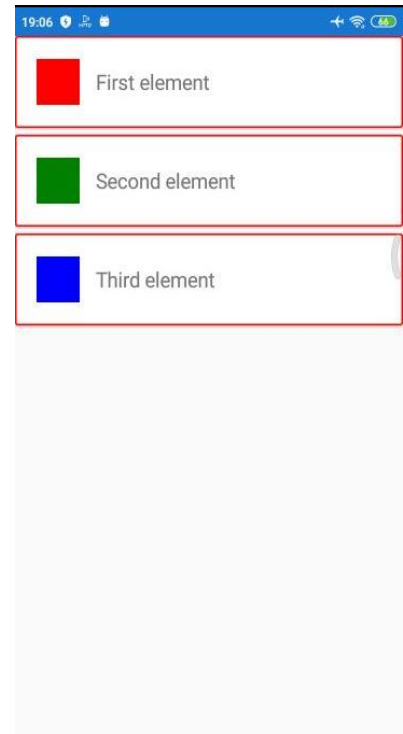
Додаємо, ще 2 *Frame*, додаємо *frame1*, *frame2*, *frame3* до *stackLayout1* (копіювання *Ctrl+D*) та *Content = stackLayout1*

```
Frame frame2 = new Frame()
{
    Content = new StackLayout()
    {
        Children =
        {
            new BoxView() { Color = Color.Green},
            new Label() {Text="Second element", VerticalOptions=LayoutOptions.Center}
        },
        Orientation = StackOrientation.Horizontal,
        Spacing = 15
    },
    BorderColor = Color.Red
};

Frame frame3 = new Frame()
{
    Content = new StackLayout()
    {
        Children =
        {
            new BoxView() { Color = Color.Blue},
            new Label() {Text="Third element", VerticalOptions=LayoutOptions.Center}
        },
        Orientation = StackOrientation.Horizontal,
        Spacing = 15
    },
    BorderColor = Color.Red
};

stackLayout1.Children.Add(frame1);
stackLayout1.Children.Add(frame2);
stackLayout1.Children.Add(frame3);

Content = stackLayout1;
}
```



# Frame

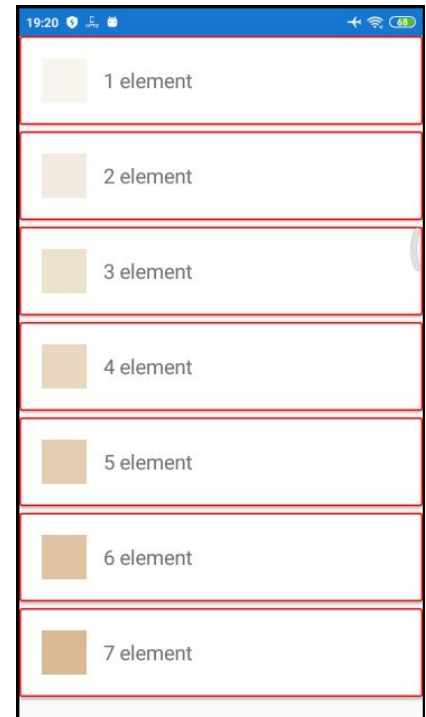
*Розглянемо як додавати елементи в циклі з одним Frame*

```
protected override void OnAppearing()
{
    StackLayout stackLayout1 = new StackLayout();
    Frame frame1 = null; //optimized Memory

    for (int i=1; i<8; i++)
    {
        frame1 = new Frame()
        {
            Content = new StackLayout()
            {
                Children =
                {
                    new BoxView() { Color = Color.FromRgb(i*5, i*10, i*15)},
                    new Label() { Text=${i} element", VerticalOptions=LayoutOptions.Center}
                },
                Orientation = StackOrientation.Horizontal,
                Spacing = 15
            },
            BorderColor = Color.Red
        };

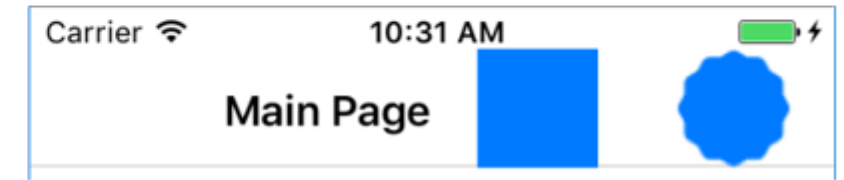
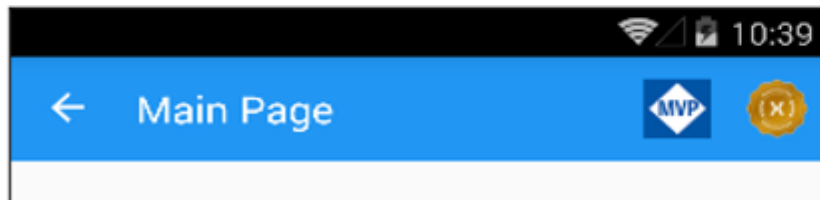
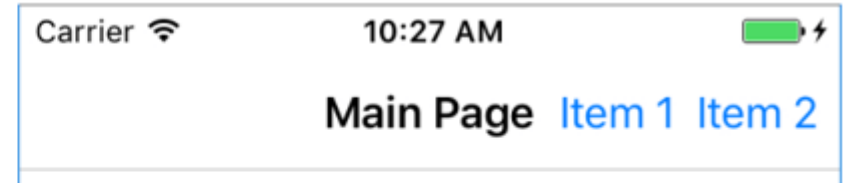
        stackLayout1.Children.Add(frame1);
    }

    Content = stackLayout1;
}
```

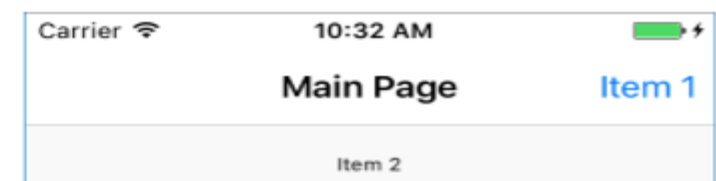


# Клас Xamarin.Forms ToolbarItem

Клас `Xamarin.Forms ToolbarItem` – це особливий тип кнопки, яку можна додати до колекції `Page` об'єкта `ToolbarItems`. Кожен `ToolbarItem` об'єкт відобразиться у вигляді кнопки на панелі навігації програми. Екземпляр `ToolbarItem` може мати значок і відобразиться як основний або додатковий пункт меню. Клас `ToolbarItem` успадковується від `MenuItem`.



*ToolbarItem об'єкти на панелі навігації Android та iOS (обидва Primary)*



*ToolbarItem об'єкти на панелі навігації Android та iOS (один Primary, один Secondary)*

# Клас Xamarin.Forms ToolbarItem

Клас `ToolbarItem` визначає такі властивості:

**Order** (`ToolbarItemOrder`) – це значення переліку, яке визначає, чи відображається `ToolbarItem` екземпляр в основному або додатковому меню;

**Priority** `integer` – це значення, що визначає порядок відображення елементів у `Page` колекції об'єкта `ToolbarItems`

Клас `ToolbarItem` успадковує такі властивості від `MenuItem` класу:

**Command** – це об'єкт  `ICommand`, що дозволяє прив'язувати дії користувача, такі як торкання пальця або клацання, до команд, визначених у моделі подання;

**CommandParameter** `object` – це значення, що вказує параметр, який повинен бути переданий у `Command`;

**IconImageSource** `ImageSource` – значення, що визначає піктограму відображення об'єкта `ToolbarItem`;

**Text** – це об'єкт `string`, що визначає `ToolbarItem` текст, що відображається.

Об'єкт **ToolbarItem** можна створити у XAML. Властивості **Text** можна встановити **IconImageSource**, щоб визначити, як кнопка відображається на панелі навігації. У цьому прикладі показано, **ToolbarItem** як створити екземпляр з деякими загальними наборами властивостей і додати його до колекції `ContentPage.ToolbarItems`:

```
<ContentPage.ToolbarItems>
  <ToolbarItem Text="Example Item"
    IconImageSource="example_icon.png"
    Order="Primary"
    Priority="0" />
</ContentPage.ToolbarItems>
```



## Клас `ToolBarItem`

У **`ToolBarItem`** буде показано об'єкт із текстом, значком та першим в основній області навігаційної панелі. Можна `ToolBarItem` також створити в кодї та додати до колекції `ToolBarItems`:

```
ToolBarItem item = new ToolBarItem
{
    Text = "Example Item",
    IconImageSource = ImageSource.FromFile("example_icon.png"),
    Order = ToolBarItemOrder.Primary,
    Priority = 0
};

// "this" refers to a Page object
this.ToolbarItems.Add(item);
```

Клас `ToolBarItem` успадковує `Clicked` подію від `MenuItem` класу. Обробник подій можна підключити до події `Clicked`, щоб реагувати на торкання або клацати `ToolBarItem` екземпляри в XAML:

```
<ToolBarItem ...
    Clicked="OnItemClicked" />
```

Обробник подій також можна приєднати до коду:

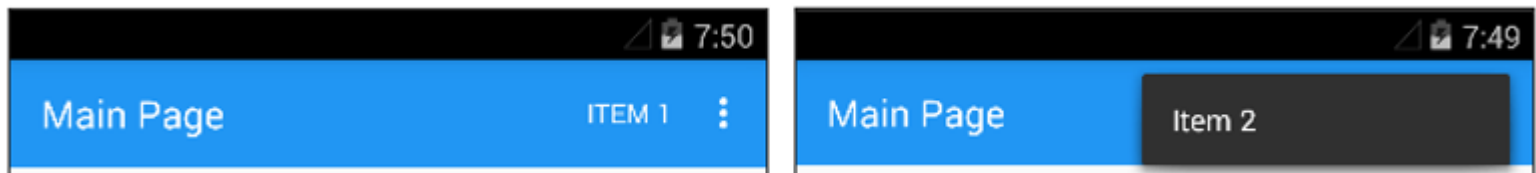
```
ToolBarItem item = new ToolBarItem { ... }
item.Clicked += OnItemClicked;
```

# Клас `ToolBarItem`

У коді показаний приклад реалізації `OnItemClicked` обробника події

```
void OnItemClicked(object sender, EventArgs e)
{
    ToolbarItem item = (ToolbarItem)sender;
    messageLabel.Text = $"You clicked the \"{item.Text}\" toolbar
item.";
}
```

Якщо для властивості `Order` встановлено значення **Primary**, `ToolBarItem` об'єкт буде відображатися на головній навігаційній панелі на всіх платформах. Якщо для властивості `Order` встановлене значення **Secondary**, то поведінка залежить від платформ. У `UWP` та `Android` `Secondary` меню елементів відображається як три точки, які можна торкнутися або клацнути, щоб відобразити елементи у вертикальному списку. У `iOS` `Secondary` меню елементів відображається під навігаційною панеллю у вигляді горизонтального списку



*Secondary* меню елементів для `Android`

# Клас ToolbarItem

## Проект ToolbarMenu

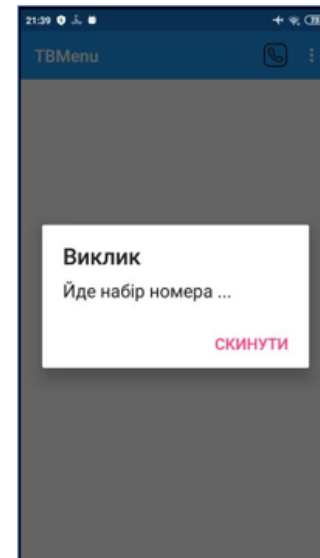
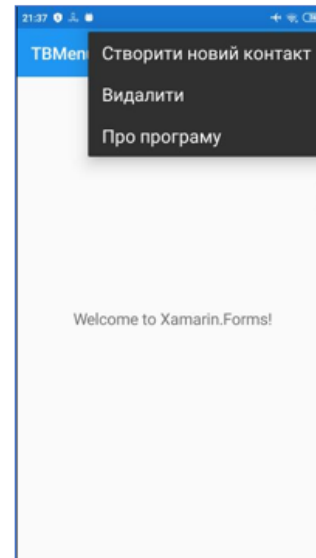
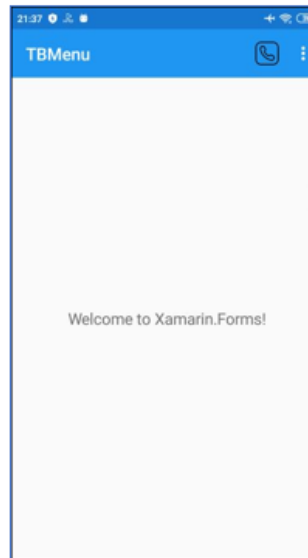
```
public MainPage()
{
    InitializeComponent();
    ToolbarItem tb = new ToolbarItem
    {
        Text = "Зателефонувати",
        Order = ToolbarItemOrder.Primary,
        Priority = 0,
        IconImageSource = "phone.png"
    };

    tb.Clicked += Tb_Clicked;
    ToolbarItem tb1 = new ToolbarItem
    {
        Text = "Створити новий контакт",
        Order = ToolbarItemOrder.Secondary,
        Priority = 1,
    };

    ToolbarItem tb2 = new ToolbarItem
    {
        Text = "Видалити",
        Order = ToolbarItemOrder.Secondary,
        Priority = 2,
    };

    ToolbarItem tb3 = new ToolbarItem
    {
        Text = "Про програму",
        Order = ToolbarItemOrder.Secondary,
        Priority = 3,
    };
    ToolbarItems.Add(tb);
    ToolbarItems.Add(tb1);
    ToolbarItems.Add(tb2);
    ToolbarItems.Add(tb3);
}

private async void Tb_Clicked(object sender, EventArgs e)
{
    await DisplayAlert("Виклик", "Йде набір номера ...",
        "СКИНУТИ");
}
```



## Клас `NavigationPage`

Клас **`NavigationPage`** забезпечує ієрархічну навігацію, коли користувач може переходити по сторінках вперед та назад за своїм бажанням. Цей клас реалізує навігацію на основі стека об'єктів `Page` за методом LIFO (останнім надійшов – першим обслужений). Для переходу з однієї сторінки на іншу програма поміщає нову сторінку у стек навігації, де вона стає активною сторінкою, як показано на рис.



Щоб повернутися до попередньої сторінки, програма витягне поточну сторінку зі стеку навігації, а нова найвища сторінка стане активною сторінкою, як показано на рис.



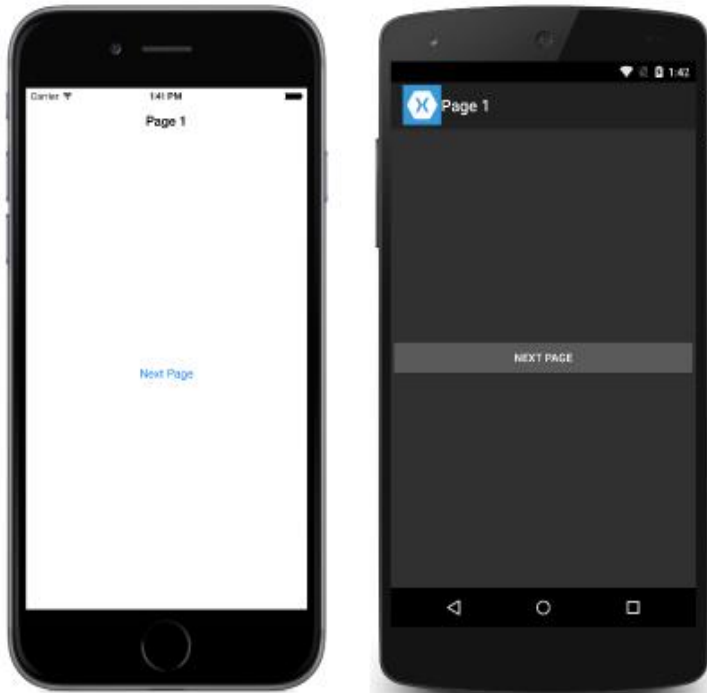
Методи навігації надаються властивістю `Navigation` будь-яких `Page` похідних типів. Ці методи надають можливість надсилати сторінки в стек навігації, витягувати сторінки зі стеку навігації та виконувати маніпуляції зі стеком.

# Клас NavigationPage

Перша сторінка, додана в стек навігації, називається кореневою сторінкою програми, що демонструється в наступному прикладі коду:

```
public App ()
{
    MainPage = new NavigationPage (new Page1Xaml ());
}
```

У результаті в стеці навігації міститься екземпляр *Page1Xaml ContentPage*, де він стає активною сторінкою та кореневою сторінкою програми. Ці дії відображаються на рис.



Відображення кореневої сторінки *Page1Xaml* стеку навігації

Для додавання нової сторінки до стеку навігації використовується код:

```
async void NextPage (object sender, EventArgs e)
{
    await Navigation.PushAsync(new YourPageName());
}
```

Для вилучення нової сторінки зі стеку навігації використовується код:

```
async void BackToPage (object sender, EventArgs e)
{
    await Navigation.PopAsync();
}
```

# Клас NavigationPage

Властивість **Navigation** надає методи **InsertPageBefore** та **RemovePage** керувати стеком шляхом вставки сторінок або їх видалення

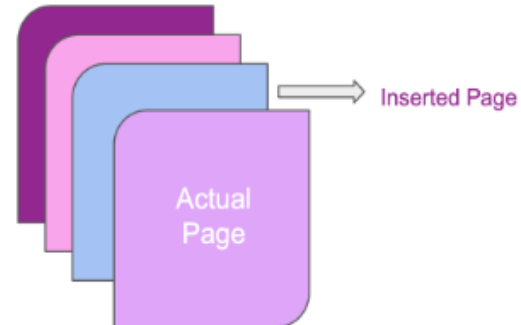
```
Navigation.InsertPageBefore (new MainPage (), this);  
Navigation.RemovePage (new MainPage);
```

InsertBeforePage Method

Original Navigation Stack



With a page inserted  
before actual page

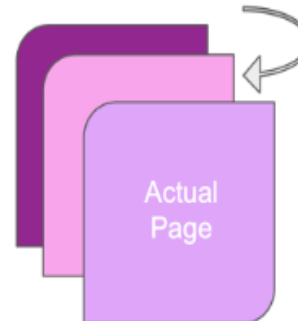


RemovePage Method

Original Navigation Stack



Remove second Page



The result



# Клас NavigationPage

Метод **PopToRootAsync** видаляє всі сторінки, що містяться в навігаційному стеку, крім кореневої сторінки, роблячи її активною сторінкою.

```
async void ClearNavigationStack (object sender, EventArgs e)
{
    await Navigation.PopToRootAsync ();
}
```

Передати дані між сторінками стеку навігації можливо через **constructor** або **BindingContext** об'єкт.

Використовуючи **constructor**, дані можемо отримати в два кроки:

Крок 1: Додайте потрібний параметр на сторінці constructor.

```
public MainPage (string YourName)
{
    InitializeComponent ();
    Name = YourName;
}
```

Крок 2: Передача даних під час виклику сторінки, яка використовувалася раніше. (MainPage)

```
await Navigation.PushAsync(new YourPageName("MariaWhite"));
```

## Клас NavigationPage

Розглянемо приклад використання об'єкта **Binding Context**. Створимо метод **FillDataClicked** та додамо всередину нього дані об'єкта класу **Contact**. Далі створений об'єкт передаємо на сторінку через об'єкт **BindingContext**.

```
async void FillDataClicked (object sender, EventArgs e)
{
    var contact = new Contact {
        Name      = "Merrie",
        LastName  = "White",
        Country   = "USA"
    };

    var contactsPage = new ContactsPage ();
    contactsPage.BindingContext = contact;
    await Navigation.PushAsync (contactsPage);
}
```

Щоб видалити кнопку «Назад», необхідно виконати фрагмент коду у файлі XAML:

```
NavigationPage.HasBackButton="false"
```

Щоб приховати панель навігації, необхідно виконати фрагмент коду у файлі XAML:

```
NavigationPage.HasNavigationBar="false"
```