

МІНІСТЕРСТВО ОСВІТИ Й НАУКИ УКРАЇНИ
Запорізький національний технічний університет

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт з дисципліни
“Мобільні операційні системи та GPS”

для студентів напрямку 6.050103
"Програмна інженерія"

2016

Методичні вказівки до виконання лабораторних робіт з дисципліни “Мобільні операційні системи та GPS” для студентів спеціальності “Інженерія програмного забезпечення”/ Укл.: О.О. Степаненко.– Запоріжжя: ЗНТУ, 2016. – 68 с.

Автори: Степаненко О.О., к.т.н., доцент;

Рецензент: Онищенко В.Ф., к. ф.-м. н., доцент

Відповідальний за випуск: Субботін С.О., д.т.н., проф.

Затверджено
вченою радою інституту
інформатики та
радіоелектроніки

Протокол № 1
від “19” березня 2014 р.

Затверджено
на засіданні кафедри
“Програмні засоби”

Протокол № 7
від “06” березня 2014 р.

Зміст

Лабораторна робота № 1. Розробка програми Silverlight для Windows	5
1. Проектування програми Silverlight	5
1.1. Засобу розробки.....	5
1.2. Metro-Стиль	5
1.3. Елементи й об'єкти Silverlight.....	6
1.4. Панель інструментів і графічний конструктор	8
1.5. Керування іменами елементів в Visual Studio	9
1.6. Властивості в елементах Silverlight.....	9
1.7. Створення методів get і set	10
1.8. Використання властивостей.....	11
1.9. Дизайн сторінки на Silverlight.....	13
2. Мова XAML.....	13
3. Приклад створення додатка Silverlight для Windows Phone.....	16
3.1. Створення додатка	17
3.2. Обчислення результату.....	17
3.3. Події й програми	18
3.4. Події в Silverlight.....	18
3.5. Керування властивостями подій.....	19
4. Короткі підсумки.....	20
5. Вправи	21
Вправа 1. Користувачський інтерфейс програми Калькулятор часу	21
Вправа 2. Синхронізація даних калькулятора часу	21
6. Довідковий матеріал	23
Середовище розробки.....	23
Шаблони додатків	23
Структура простого додатка	24
Основні елементи керування	25
7. Додаткові матеріали.....	26
Лабораторна робота № 2. Модернізація програми Silverlight для Windows Phone	27
1. Поліпшення додатка	27
1.1. Керування властивостями елементів	27
1.2. Редагування Xaml-Коду елементів Silverlight.....	29
1.3. Вивід вікна з повідомленням	31
1.4. Додавання й використання ресурсів	32
2. Зміна й відображення даних	34
Подія Textchanged	34
Прив'язка даних	35
Прив'язка даних через властивість DataContext	40
3. Керування орієнтацією сторінки додатка.....	41
Альбомна й книжкова орієнтація	41

Використання контейнерів для угруповання елементів	44
4. Відображення списків даних	45
4.1. Створення даних для списку	46
4.2. Використання елемента Stackpanel для виводу списку на екран	47
4.3. Використання елемента Listbox для відображення списку елементів	49
5. Навігація по сторінках додатка	53
Навігація по сторінках	53
Передача даних між сторінками	54
Спільне використання об'єктів декількома сторінками	56
6. Використання класів Viewmodel	57
6.1. Створення класу Viewmodel	58
6.2. Навігація по сторінках з використанням методу Goback	60
6.1. Спостережувані колекції	61
7. Короткі підсумки	62
Лабораторная работа № 3. Работа с GPS навигатором	64

Лабораторна робота № 1. Розробка програми Silverlight для Windows

1. Проектування програми Silverlight

Не завжди програмісти при створенні додатків займаються створенням *інтерфейсу*. Часто компанії доручають це завдання дизайнерам. Програмісти ж повинні додати до дизайну програмний код, який виконає необхідну роботу. Silverlight дозволяє розділити процес створення *інтерфейсу* й процес створення програмного коду додатка. Це дозволяє програмістові накидати початковий користувацький *інтерфейс*, який згодом може змінити дизайнер, зробивши його більш привабливим. Програміст також може побрати закінчений дизайн користувацького *інтерфейсу* й потім запрограмувати поведінка кожного елемента.

1.1. Засоби розробки

Проектувальники користувацького інтерфейсу використовують інструменти Expression Blend, а програмісти — Visual Studio для створення коду. Система Silverlight надає можливість об'єднати дизайн користувацького інтерфейсу й код. Гарний підходом для програміста є використання шаблону "заповнювача", який дозволяє запрограмувати поведінка програми й пізніше внести в додаток остаточний дизайн.

Windows Phone SDK (Software Development Kit — набір засобів для розробки програмного забезпечення) включає версії Visual Studio і Expression Blend, які можна використовувати для створення коду й дизайну додатка.

1.2. Metro-Стиль

З погляду дизайну Windows Phone використовує концепцію, яка в керівництвах по проектуванню називається Metro-Стиль. У ньому закладені принципи, які визначають, як виглядають елементи керування, і встановлюють ряд критеріїв, яким повинні відповідати додатки Windows Phone. Ці особливості стилю відбиті у вбудованих компонентах, які можуть використовувати програми.

Додатки, що використовують кнопки, текстові поля й інші елементи, які поставляються з Visual Studio, автоматично будуть дотримуватися вимог стилю. Це більша допомога тим, хто не дуже добре вміє проектувати дизайн. Звичайно, можна повністю перевизначити властивості компонентів, що поставляються, але при створенні додатків для Windows Phone краще цього уникати. Існує посібник з Metro-Стилю, яке доступно за адресою: <http://msdn.microsoft.com/ru-ru/library/hh202915.aspx>.

У цій роботі ми будемо використовувати засоби проектування Silverlight у складі Visual Studio. Вони не дають усіх можливостей створення просунутого графічного інтерфейсу, які надає Expression Blend, але для простих програм цих засобів їх цілком достатньо.

1.3. Елементи й об'єкти Silverlight

З погляду програмування кожний елемент на екрані є програмним об'єктом. Об'єкт включає поведінка й дані. Якби ми створювали додаток для керування банківськими рахунками, то ми могли б створити такий клас для зберігання інформації про рахунок:

```
public class Account
{
    private decimal balance;
    private string name;

    public string Getname()
    {
        return name;
    }

    public bool Setname(string newname)
    {
        // у фінальній версії тут можна перевірити коректність імені
        name = newname;
        return true;
    }

    // інші методи get i set
}
```

У цьому класі зберігається інформація про кількість грошей на банківському рахунку (у властивості `balance`) і ім'я власника рахунку (у властивості `name`). Для створення нового запису можна використовувати ключове слово `new`:

```
Account rob = new Account();
rob.Setname("Rob");
```

Цей код створює новий екземпляр класу `Account` з іменем `rob`. Потім задається ім'я власника рахунку — `"Rob"`. Використання об'єктів — це відмінний спосіб вистави сутностей, інформацію про яких потрібно зберігати й обробляти. За допомогою об'єктів також можна описати елементи на екрані. Область відображуваного на екрані тексту також має властивості: позиція на екрані, колір тексту, сам текст і ін.

На рис. 1 представлений інтерфейс простої програми Суматор, яка знаходить суму двох чисел. Числа вводяться в текстові поля, і при натисканні на кнопку сума результат виводиться в нижнє текстове поле.

Кожний окремий елемент на екрані ставиться до класу `UIElement`, який представляє елементи користувацького інтерфейсу. Екран цього додатка містить сім елементів:

1. Маленький заголовок СУМАТОР. Цей елемент називається назва додатка (Application Title).
2. Великий заголовок додавання. Цей елемент називається назва сторінки (Page Title).

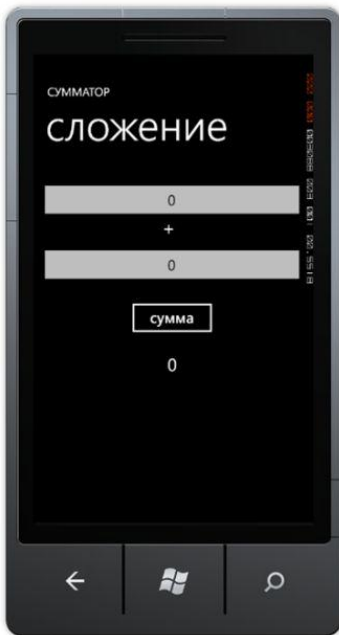


Рис. 1. Экран программы Сумматор

3. Верхнє текстове поле, у яке можна ввести число.
4. Текстовий елемент, що містить знак операції додавання - символ "+".
5. Нижнє текстове поле, у яке можна ввести інше число.
6. Кнопка, при натисканні на яку відбувається додавання.
7. Текстове поле, яке при натисканні на кнопку відображає результат.

Для кожного елемента задається положення на екрані, розмір тексту й інші властивості. Можна змінити колір тексту в текстовім полі, вирівняти його по лівому краю, по правому краю або по центру поля, а також задати інші властивості.

Зазначені елементи мають один з типів:

1. Textbox — дозволяє користувачеві вводити текст.
2. Textblock — текстовий блок, який тільки відображає інформацію.
3. Button — кнопка, при натисканні якої викликається подія.

Властивості цих елементів можна розділити на два типи: властивості, які повинні бути у всіх елементів, наприклад положення на екрані, і властивості, які є специфічними для певного типу елемента. Наприклад, тільки для елемента Textbox можна задати позицію курсору для введення тексту. Можна представити всі елементи в ієрархічній структурі, як показано на рис. 2.

Таку ієрархічну структуру створили проектувальники Silverlight. Клас Uielement є класом, який може відображатися на екрані як візуальний об'єкт і ухвалювати дані, які вводить користувач. На телефоні введення інформації виконується переважно із сенсорного екрана.

Від класу Uielement успадковується клас Frameworkelement, який містить усю інформацію, характерну для всіх відображуваних елементів. Усі інші класи успадковуються від нього. Спадкоємці мають усі властивості батьківського класу й додають власні.

Така ієрархія класів має ряд переваг. Якщо потрібний власний тип текстового поля, можна розширити клас Textbox і додати в нього потрібні властивості й методи. При цьому, система Silverlight буде обробляти всі ці елементи так само, як і звичайні текстові поля, оскільки нові елементи зберігають усі властивості й поведінка вихідного класу.

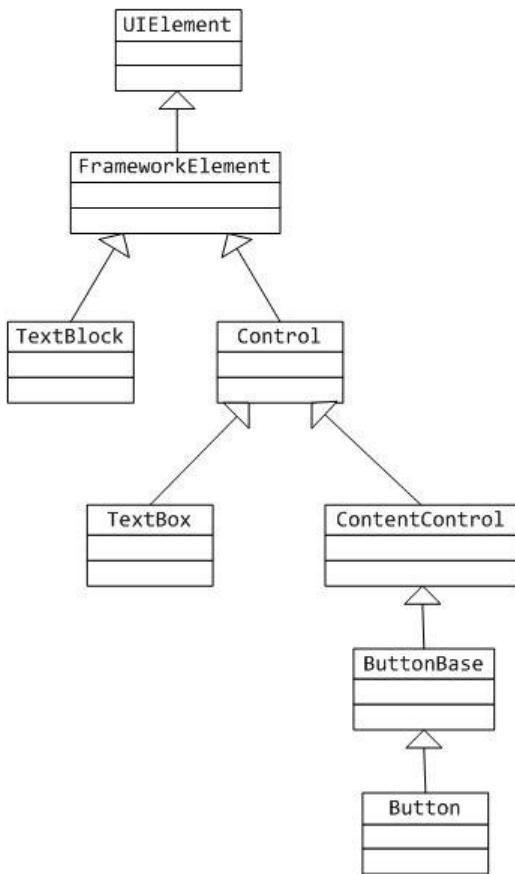


Рис. 2. Структура классов элементов Silverlight

Необхідно пам'ятати, що при зміні елементів на екрані й створенні й керуванні елементами насправді ми змінюємо властивості об'єктів. При проектуванні користувацького інтерфейсу Silverlight ми задаємо значення властивостям елементів, які використовуються для відображення елементів на екрані.

1.4. Панель інструментів і графічний конструктор

Створити додаток для Windows Phone дуже легко. При створенні нового проекту Silverlight відкривається порожня сторінка вікна дизайнера, на яку можна додавати елементи з панелі інструментів. Можна створити інтерфейс програми, просто перетаскуючи елементи з панелі інструментів у вікно дизайнера.

а рис. 3 показаний результат перетаскування елемента TextBox панелі інструментів у вікно сторінки Windows Phone. Процес створення інтерфейсу сторінки додатка для Windows Phone такий як і додатка Windows Forms. Оскільки всі засоби керування розроблені в стилі Metro, створювані додатки автоматично стають схожими на "правильні" MetroДодатки.

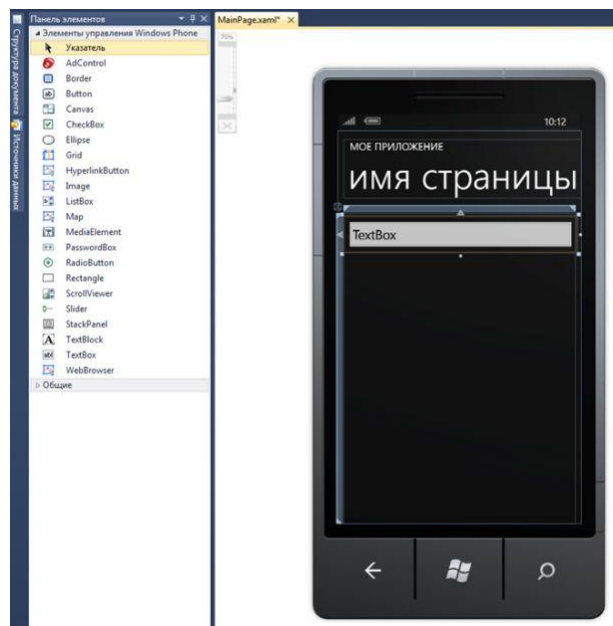


Рис. 3. Окно дизайнера Visual Studio

1.5. Керування іменами елементів в Visual Studio

Після додавання елементів на екран рекомендується дати елементам імена. Середовище проектування дасть кожному елементу імена за замовчуванням, наприклад, Textbox1, які краще змінити на осмислені, підібравши кожному елементу ім'я на основі його призначення.

Для того щоб змінити властивості елементів, потрібно клацнути по елементу у вікні дизайнера й потім знайти потрібну властивість в області властивостей в Visual Studio.

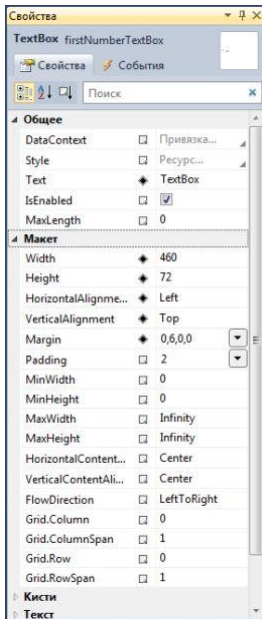


Рис. 4. Область свойств элемента TextBox

на рис. 4 зображена область властивостей для верхнього елемента Textbox на сторінці. Ім'я елемента відображається у верхній частині вікна. Змінимо ім'я цього текстового поля на firstnumbertextbox. При цьому, зміна значення властивості Name перевизначить ім'я змінної, пов'язаної із цим елементом:

```
Textbox firstnumbertextbox;
```

Visual Studio відслідковує оголошення фактичних змінних, які представляють створювані відображувані елементи, і нам не потрібно думати про те, де зберігаються відповідні оголошення цих змінних.

1.6. Властивості в елементах Silverlight

Після перейменування змінн, що представляє текстове поле, можна настроїти її властивості: ширину, відступи від країв і т.д. Значення властивостей відбивають поточну позицію й розмір елемента на екрані. Якщо перетягнути елемент за його край, значення властивостей зміняться.

Якщо змінити значення у вікні Властивості, то елемент у вікні дизайнера автоматично змінить положення. Точно також відбувається й при створенні додатка Windows Forms. Будь-які зміни у вікні дизайнера приводять до зміни вмісту об'єкта.

Властивості в Silverlight і властивості в C#. Коли ми говоримо про властивості елементів Silverlight на сторінці (наприклад, текст у поле Textbox), ми фактично говоримо про значення властивостей у класі, які втримуються в елементі Textbox. Інакше кажучи, якщо програма містить таке оголошення:

```
resulttextblock.Text = "0";
```

запуститься метод Set об'єкта resulttextblock, який установлює текст елемента Textblock у відповідне значення.

Класи й властивості в C#. Класи в C# можуть містити дані (ім'я власника банківського рахунку) і поведінка (методи Getname і Setname, які дозволяють

програмі пізнавати ім'я й змінити це ім'я). Дані в класі представляються у вигляді властивостей, з яким працювати простіше, чим використовувати методи Get і Set. Вони активно використовуються в елементах керуванні Silverlight.

Властивість — це член класу, який містить значення. Можна використовувати змінні для зберігання значень, але потрібно зробити їхніми публічними (public), щоб їх можна було використовувати в додатках. Наприклад, банк може захотіти відслідковувати інформацію про співробітників. Один з параметрів, значення якого може знадобитися відслідковувати, — вік власника банківського рахунку. Можна оголосити в класі публічне поле:

```
public class Account
{
    public int Age;
}
```

Клас Account містить загальнодоступне поле Age. Можна одержати доступ до цього поля по його імені:

```
Account s = new Account();
s.Age = 21;
```

Проблема полягає в тому, що в поле для зберігання віку можна записати неприпустиме значення:

```
s.Age = -100210232;
```

Оскільки поле є загальнодоступним, немає можливості перешкодити подібному дії.

1.7. Створення методів get і set

Щоб контролювати використання полів, можна створити методи get і set і зробити їх загальнодоступними. Вони надають можливість управляти доступом до поля. При цьому, поле Age краще зробити закритим (private), щоб до нього не можна було одержати прямий доступ за межами класу.

```
public class Account
{
    private int age;
    public int Getage()
    {
        return this.age;
    }
    public void Setage(int inage)
    {
        if ((inage > 0) && (inage < 120))
        {
            this.age = inage;
        }
    }
}
```

Тепер можна контролювати доступ до нашого поля, але для цього потрібно написати багато додаткового коду. Для того щоб одержати доступ до значення віку, необхідно викликати створені методи:

```
Account s = new Account();
s.Setage(21);
Console.WriteLine("Вік: " + s.Getage());
```

1.8. Використання властивостей

Властивості дозволяють зробити керування даними більш простим. Властивість Age можна оголосити в такий спосіб:

```
public class Account
{
    private int agevalue;

    public int Age
    {
        set
        {
            if ((value > 0) && (value < 120))
                agevalue = value;
        }
        get
        {
            return agevalue;
        }
    }
}
```

Тут значення віку є властивістю. У властивості оголошені секції для запису й читання його значення. Описані в цих секціях дії еквівалентні описаним раніше методам. При цьому, властивості використовуються так само, як і звичайні поля:

```
Account s = new Account();
s.Age = 21;
Console.WriteLine("Вік: " + s.Age);
```

Коли властивості Age привласнюється значення, викликається код секції set. Ключове слово value позначає значення, яке привласнюється властивості. При читанні значення властивості Age відбувається виклик коду секції get. Такий підхід поєднує переваги використання методів і дозволяє працювати із властивостями так само просто, як і з полями класу.

Перевірка правильності даних у властивостях. При спробі задати неприпустиме значення віку (наприклад, 150), наведений вище код виконає перевірку допустимості й відхилить це значення (ніхто старше 120 років не може мати рахунок у нашому банку), залишивши колишнє значення віку. Єдиний спосіб довідатися, чи було властивості привласнене значення, полягає в перевірці значення властивості після цієї операції:

```
Account s = new Account();
int newage = 150;
s.Age = newage;
if (s.Age != newage)
    Console.WriteLine("Значення віку не було встановлено");
```

Наведений код намагається привласнити віку неприпустиме значення 150, після чого перевіряється, чи було це значення встановлене. Якби для присвоювання значення використовувався метод Set, він міг би повернути значення **false** у випадку невдачі, а при використанні властивості користувач повинен виконати ледве більше додаткової роботи.

Різні способи зчитування значення властивості. Властивості дозволяють виконувати й інші корисні дії.

```
public int Ageinmonths
{
    get
    {
        return this.agevalue * 12;
    }
}
```

Тут описана нова властивість Ageinmonths. Воно призначене тільки для читання, тому що не містить секції set. Воно повертає значення віку в місяцях, використовуючи те ж саме значення, що й властивість Age. Це означає, що можна використовувати кілька різних способів для одержання того самого значення. Можна створити властивості тільки для читання без можливості їх зміни прямо, а також властивості тільки для запису, хоча останні використовуються рідко.

Властивості візуальних елементів. Властивості має сенс використовувати в описі банківського рахунку, де потрібно захистити дані в об'єктах. Але в програмі Silverlight можна ввести будь-який текст в елемент Textblock, і начебто б немає потреби перевіряти допустимість значення, що вводиться. Виконання цього коду сповільнить процес уведення значення. Так, зробивши значення Text публічним рядком, програма містила б менше коду й працювала швидше.

Але при цьому, коли ми змінюємо текст в елементі Textblock, ми прагнемо, щоб текст на сторінці Silverlight також змінився, наприклад, коли програма Суматор виведе на екран результат. Якби програма просто змінила значення поля, система Silverlight ніяк не могла б довідатися, що повідомлення на екрані повинне бути оновлене.

Однак, якщо Text зробити властивістю, при відновленні значення елемента Textblock запуститься відповідний метод, який може оновити збережене значення текстового поля й викликати метод для відновлення екрана, щоб відобразити нове значення. Властивості надають можливість керування об'єктом при зміні його значення. Проста операція:

```
resulttextblock.Text = "0";
```

може привести до виконання декількох сотень операцій C#, оскільки збереження нового значення в елементі Textblock приводить до запуску операцій для відновлення зображення на екрані.

1.9. Дизайн сторінки на Silverlight

Ми можемо завершити сторінку програми Суматор, перетаскуючи потрібні елементи на екран і встановлюючи їх властивості. Першою справою після перетаскування елемента на сторінку рекомендується визначити ім'я цього елемента. Ознакою поганого стилю програми є сторінка, що містить багато елементів з іменами типу Button1, Button2 і т.д.

Розробка сторінки є нескладним завданням, яке зводиться до перетаскування елементів на сторінку й налаштуванню властивостей кожного елемента. Засобу Silverlight дозволяють додати до елементів графічні властивості, які можуть зробити їхніми прозорими, додати зображення до їхнього тла й навіть зробити їх анімованими.

2. Мова XAML

Проектування дизайну на Silverlight складається з маніпуляцій з елементами за допомогою дизайнера Visual Studio і присвоювання значень потрібним властивостям елементів. Виконуючи ці дії, можна створити будь-який користувацький інтерфейс. інтерфейс додатка на Silverlight описується за допомогою мови XAML.

XAML означає "розширювану мову розмітки додатків" (extensible Application Markup Language). Silverlight використовує XAML для опису зовнішнього вигляду сторінки.

XAML визначає чітку границю між зовнішнім виглядом додатка (властивості відображуваних елементів) і поведінкою додатка (які дії виконують елементи додатка).

Такий поділ є гарною ідеєю, оскільки не можна гарантувати, що гарний програміст буде гарним дизайнером. Звичайно над створенням додатків працюють і дизайнери, і програмісти. При цьому виникає проблема поділу роботи між ними: програміст не може нічого робити, поки дизайнер не закінчить створення всіх сторінок додатка.

XAML вирішує цю проблему. Як тільки будуть викладені вимоги користувацького інтерфейсу (скільки текстових полів, скільки кнопок, що буде на кожній сторінці і т.д.), проектувальник може продовжувати роботу над зовнішнім виглядом сторінок, у той час як програміст може зосередитися на написанні програмного коду. Файли XAML містять опис елементів користувацького інтерфейсу, і дизайнер може працювати над їхнім зовнішнім виглядом і розташуванням на сторінці, у те час як програміст доповнює їхнім програмним кодом. Програмістові не потрібно передавати дизайнерові файли з кодом, тому що вони відділені від дизайну, і навпаки.

Уміст файлу XAML. Файл XAML для сторінки містить конструкції, які описують усі елементи на сторінці. Кожний опис містить набір властивостей. Рядок XAML, яка описує елемент Textbox, може виглядати так:

```
<Textbox Height="72" Horizontalalignment="Left" Margin="8,19,0,0"
  Name="firstnumbertextbox" Text="0" Verticalalignment="Top" Width="460"
  Textalignment="Center" />
```

Якщо зрівняти інформацію в області властивостей зі значеннями атрибутів у коді XAML, можна встановити їхню відповідність: ім'я елемента Textbox — firstnumbertextbox, ширина — 460 і т.д. Сторінка й область властивостей синхронізують значення, тому що вони використовують той самий Xaml-Файл сторінки. При переміщенні елементів по сторінці редактор оновлює XAML; область властивостей зчитує дані із цього файлу й оновлює його значення.

XAML є декларативною мовою. Він створений, щоб бути зрозумілим для людей, тому в усіх вище зазначених властивостей зрозумілі імена. Можна відредагувати текст Xaml-Файлу в Visual Studio для зміни зовнішнього вигляду сторінки в дизайнерові й значень властивостей. Безпосереднє редагування коду мовою XAML для досвідчених програмістів часто відбувається набагато швидше, чим перетаскування елементів по екрану й редагування у вікні властивостей. Це особливо корисно, якщо потрібно додати велику кількість однотипних елементів. Для полегшення редагування Xaml-Коду Visual Studio надає підтримку IntelliSense.

При компіляції програми Xaml-Файл перетвориться у файл, що містить низькорівневі інструкції, які створюють необхідні елементи для роботи програми. Оголошення елементів сторінки у файлі XAML означає, що керування ними буде здійснюватися через об'єкти, які може використовувати програма.

Під поняттям розширювана мова розмітки мається на увазі, що можна використовувати правила мови для створення конструкцій, які описують щонебудь. Англійська мова для цього добре походить. Він складається з букв і розділових знаків, які є символами тексту англійської мови. Також у мові є правила (їх називають граматикою), які визначають, як скласти слова й пропозиції. Існують різні види слів. Є іменники, які описують предмети, і дієслова, які описують дії. Коли з'являється щось нове, ми придумуємо нові слова для того, щоб описати це.

Мови, засновані на XML (extensible Markup Language — розширювана мова розмітки), є розширюваними в тому розумінні, що ми можемо придумати нові слова й фрази, які підкоряються правилам мови, і використовувати ці нові конструкції для опису того, що нам потрібно. Їх називають мовами розмітки, тому що вони часто використовуються для опису розташування елементів на сторінці. Слово "розмітка" спочатку використовувалося при печатці. Практично найвідомішою мовою розмітки є HTML — мова розмітки гіпертексту (Hypertext

Markup Language), який використовується у всесвітній павутині для опису формату веб-сторінок.

Програмісти часто створюють свої власні формати зберігання даних, використовуючи XML. Як приклад розглянемо фрагмент XmlaФайлу, у якому представлений список рекордів:

```
<?xml version="1.0" encoding="us-ascii" ?>
<Highscorerecords count="2">
  <Highscore game="Breakout">
    <playername>Rob Miles</playername>
    <score>1500</score>
  </Highscore>
  <Highscore game="Space Invaders">
    <playername>Rob Miles</playername>
    <score>4500</score>
  </Highscore>
</Highscorerecords>
```

У цьому файлі елемент Highscorerecords містить два елементи Highscore: один для гри Breakout і один для гри Space Invaders. Обидва елементи Highscore утримуються усередині елемента Highscorerecords. У кожного елемента є атрибут, який визначає назва гри, і два вкладені елементи: ім'я гравця playername і кількість набраних очок score. Цей текст досить легко зрозуміти.

У найпершому рядку файлу зазначена версія стандарту XML і використовуване кодування символів. XAML використовує правила розширеного мови розмітки, щоб створити мову, яка описує компоненти на сторінці екрана.

```
<Textbox Height="72" Horizontalalignment="Left" Margin="8,19,0,0"
  Name="firstnumbertextbox" Text="0" Verticalalignment="Top" Width="460"
  Textalignment="Center" />
```

Якщо подивитися на опис елемента Textbox, можна побачити, що творці мови XAML просто створили імена полів, які відповідають їхнім вимогам.

XmlxСхема. Стандарт XML також містить опису, як створити схему, яка описує особливий формат документа. Наприклад, у схемі для інформації про рекорди можна вказати, що елемент Highscore повинен містити елемент Playername, атрибут Score і необов'язковий атрибут Date, у якому може зберігатися інформація про дату встановленого рекорду.

Система стандарту формату й схеми означає, що розроблювачі можуть легко створювати формати даних для конкретних цілей. Цьому сприяє факт, що існує багато інструментальних засобів, яких допомагають створювати документи й схеми. Середовище .NET навіть надає програмам можливість збереження об'єктів у вигляді відформатованих XmlnДокументів. Навіть файл рішення Visual Studio зберігається як XmlmДокумент.

Мова XAML на сторінках додатків. В XmlfФайлі можна описати цілу сторінку екрана Windows Phone. При створенні нового проекту Windows Phone автоматично створюється нова сторінка з декількома елементами. При

додаванні на сторінку елементів збільшується й розмір файлу, оскільки для кожного елемента створюється його опис. Деякі елементи є контейнерами — вони можуть містити інші компоненти, наприклад, елемент Grid може містити набір інших елементів і розташовувати їх у вигляді сітки. XamlКод також може включати опису анімації й перетворень, які можна застосовувати до елементів на сторінці для створення вдосконалених користувацьких інтерфейсів.

При цьому, в XAML не існує поняття спадкування: не можна створити елемент XAML, який успадковується від іншого. Елементи Silverlight представляються у вигляді об'єктів, які є частиною ієрархії класів, але коли вони записані у файлі XAML, вони все представляються елементами, які перебувають на одному рівні ієрархії.

За допомогою XAML можна створити досить складний дизайн для програм. Також існує спеціальний інструмент Expression Blend для графічних дизайнерів, який містить розширені засоби для створення графічних інтерфейсів.

3. Приклад створення додатка Silverlight для Windows Phone

Для того щоб додаток виконував якісь дії, потрібно додати код програми мовою C#, який виконає необхідну роботу. Коли Visual Studio створює файл XAML, який описує сторінку додатка Windows Phone, разом з ним створюється файл коду. У цей файл можна помістити код, який змусить додаток працювати. Створений файл уже містить початковий код, який може виглядати так:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Microsoft.Phone.Controls;

namespace Phoneappl
{
    public partial class Mainpage : Phoneapplicationpage
    {
        // Конструктор
        public Mainpage()
        {
            Initializecomponent();
        }
    }
}
```

Більшу частину файлу займають інструкції using, які дозволяють програмі прямо звертатися до класів без необхідності вказувати повний шлях до них.

Наприклад, замість того щоб записати `System.Windows.Controls.Button`, можна написати просто `Button`, тому що файл містить рядок

```
using System.Windows.Controls;
```

Єдиний метод у програмі — конструктор класу `Mainpage`, який викликається при створенні класу. У конструкторові викликається метод `Initializecomponent`, який створює екземпляри візуальних елементів. Цей код автоматично створює Visual Studio на основі Xaml-файлу, який описує структуру сторінки. Код цього методу не рекомендується змінювати прямо, оскільки це може привести до помилок при виконанні програми.



Рис. 5. Экран программы Сумматор с открытой экранной клавиатурой

Не потрібно турбуватися про те, як створюються й виводяться на екран візуальні об'єкти. Досить використовувати високорівневі інструменти або навчитися писати Xaml-код для створення інтерфейсу сторінок.

3.1. Створення додатка

Розглянемо процес створення інтерфейсу для нашої програми для додавання чисел. Після того як ми додамо всі необхідні елементи й запустимо додаток, ми можемо виконати в ньому деякі дії. Ми можемо ввести числа в текстові поля й натиснути на кнопку Сумма (рис. 5).

Якщо доторкнутися до елемента `Textbox`, з'явиться екранна клавіатура, за допомогою якої можна ввести числа в поля. Якщо ми доторкнемося до іншої області екрана, клавіатура зникне. Однак, при натисканні на кнопку сума ми не одержимо результат, оскільки додаток "не знає", що воно повинне виконувати обчислення.

3.2. Обчислення результату

Необхідно створити програмний код для виконання необхідних обчислень і виводу на екран результату:

```
private void calculateresult()
{
    float v1 = float.Parse(firstnumbertextbox.Text);
    float v2 = float.Parse(secondnumbertextbox.Text);

    float result = v1 + v2;

    resulttextblock.Text = result.ToString();
}
```

Об'єкти `Textbox` працюють зі значенням властивості `Text`, доступним для читання й запису. При записі значення у властивість `Text` текст у текстовім полі

зміниться, а при зчитуванні значення цієї властивості програма одержує значення, яке було введено в це поле.

Текст має строковий тип, який потрібно перетворити в числове значення, для того щоб можна було виконати операцію додавання. Для цього використовується метод `Parse`, який ухвалює як параметр рядок і повертає записане в рядку число. Метод `Parse` є в кожного із числових типів (`int`, `float`, `double` і т.д.). Наша програма Суматор може працювати як із цілими, так і із дробовими числами. Метод `Parse` класу `float` перетворить уведений в текстові поля значення в чисельний тип і складає їх.

Після цього, метод перетворить отримане значення в рядок і записує її у властивість `Text` елемента `resulttextblock`. Метод `Tostring` протилежний `Parse`. Він повертає текст, який описує зміст об'єкта. Для типу `float` метод повертає чисельне значення, записане в рядку.

У результаті, ми одержали код, який виконує обчислення й одержує результат. Тепер потрібно, щоб цей код виконувався, коли користувач натискає на кнопку сума.

3.3. Події й програми

У часи консольних додатків програма завжди запускала з самого початку, працювала якийсь час і потім завершувала свою роботу. Але тепер у нашому розпорядженні є багаті користувацькі інтерфейси з більшою кількістю кнопок і інших елементів для взаємодії з користувачами. У цій ситуації дуже важко написати програму, яка перевіряла б послідовно кожний елемент інтерфейсу, щоб побачити, чи спробував користувач його використовувати. Замість цього, система очікує від цих візуальних елементів яких те дій.

Використання подій суттєво полегшує створення програм. Програма не повинна перевіряти кожний елемент на екрані, щоб довідатися, чи виконав користувач над ним якісь дії — замість цього програма просто відповідає на події, що генеруються.

Щоб змусити події працювати, мові програмування потрібний спосіб вистави посилання на метод об'єкта, для чого в C# використовується механізм делегатів. Можна створити тип делегата, який може звернутися до певного методу, і створити екземпляр цього делегата, який звертається до методу в об'єкті. Делегати є потужним і корисним механізмом.

3.4. Події в Silverlight

В C# подія відправляється об'єкту за допомогою виклику методу цього об'єкта. У нашій програмі Суматор коли користувач натискає на кнопку сума, програма повинна викликати потрібний метод, наприклад, метод `calculateresult`.

Можна використовувати редактор Visual Studio: щоб зв'язати метод з подією клацання кнопки миші, потрібно двічі клацнути по цій кнопці у вікні дизайнера. При цьому, Visual Studio установлює властивості кнопки в XAML, для того щоб зв'язати кнопку з методом сторінки. Крім цього, створюється

порожня версія цього методу оброблювача, і середовище переносить курсор до цього методу, щоб можна було відразу почати додавати код, який повинен виконуватися при натисканні кнопки.

Тепер потрібно просто додати наступний код, який виконає необхідні дії:

```
private void calculateresult()
{
    float v1 = float.Parse(firstnumbertextbox.Text);
    float v2 = float.Parse(secondnumbertextbox.Text);

    float result = v1 + v2;
    resulttextblock.Text = result.ToString();
}

private void equalsbutton_Click(object sender, RoutedEventArgs e)
{
    calculateresult();
}
```

Створеному оброблювачеві подій Visual Studio автоматично привласнює ім'я, яке формується з імені елемента, у кінець якого додається текст `_Click`. Таким чином, по цьому імені можна легко визначити, яким образом він викликається.

Тепер програма працює. Однак, якщо користувач уведе в текстове поле нечислове значення, то програма згенерує виключення.

3.5. Керування властивостями подій

Щоб довідатися, як відбувається виклик подій, для початку розглянемо властивості кнопки сума нашого додатка. Для цього потрібно натиснути на кнопку в редакторі Visual Studio, і ми побачимо інформацію про її властивості (рис. 6).

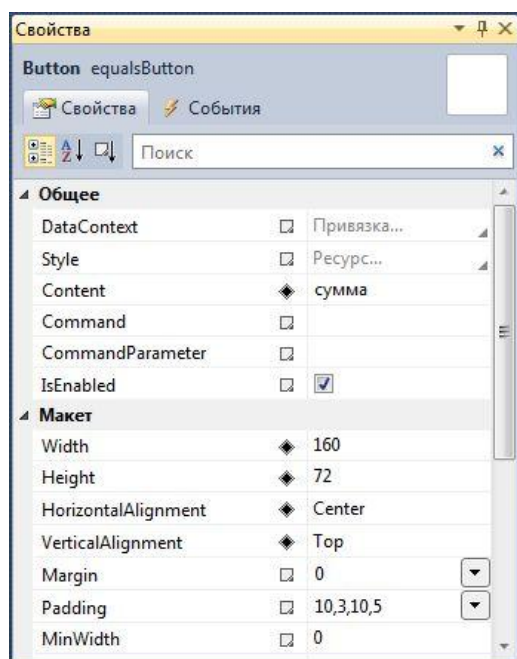


Рис. 6. Область властивостей элемента

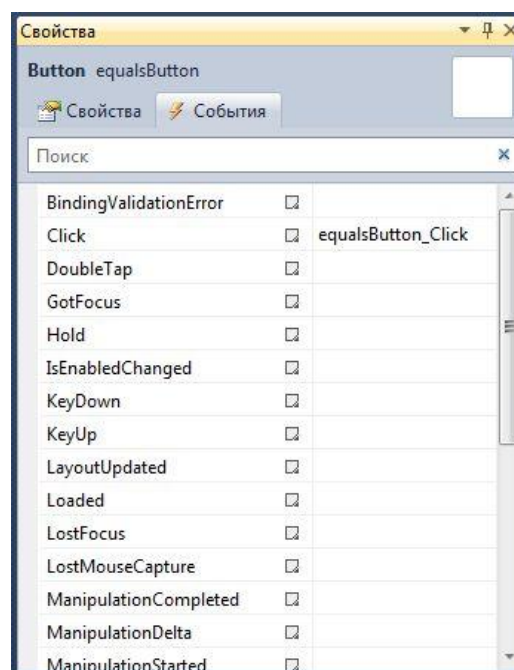


Рис. 7. Події елемента Button

Button

В області властивостей є дві вкладки: Властивості й Події. При натисканні на вкладку Події на екран виводяться події, які може згенерувати цей елемент, і пов'язані з подіями методами (рис. 7).

У цьому вікні подія Click пов'язане з методом equalsbutton_Click. Якщо потрібно припинити зв'язок методу з подією Click, то досить вилучити назва оброблювача в цьому вікні. Якщо спробувати підставити замість зазначеного імені методу довільний рядок, то програма не буде скомпільована, якщо методу із зазначеним іменем не існує.

Дотепер ми керувалися принципом, що відображувані властивості Visual Studio зв'язує з файлом XAML, який описує компоненти на екрані. Ми можемо перевірити, що це так, розглянувши опис кнопки у файлі XAML для сторінки Mainpage:

```
<Button Content="equals" Height="72" Horizontalalignment="Left"
Margin="158,275,0,0"
Name="equalsbutton" Verticalalignment="Top" Width="160"
Click="equalsbutton_Click" />
```

Опис кнопки тепер містить елемент, який зв'язує подію Click з назвою методу. Слід зазначити, що, якщо у файлі XAML зазначене, що кнопка пов'язана з методом, якого не існує, програма не зможе правильно працювати.

4. Короткі підсумки

1. Silverlight надає можливість створювати користувацькі графічні інтерфейси.
2. Користувацький інтерфейс Silverlight складається з візуальних елементів: текстові поля, кнопки й інші.
3. Інтегроване середовище розробки Visual Studio містить вікно дизайнера, яке використовується для додавання елементів користувацького інтерфейсасilverlight на сторінки додатка.
4. З погляду програмного забезпечення, кожний елемент користувацького інтерфейсу представлений особливим типом об'єкта в ієрархії класів, яка є частиною Silverlight.
5. Дизайнери можуть змінювати значення властивостей елементів, змінюючи ці елементи у вікні дизайнера Visual Studio або змінюючи значення властивостей цих елементів.
6. Властивості використовуваних у програмі елементів Silverlight зберігаються в текстових файлах у форматі XAML. Ці файли обновляються засобами проектування й використовуються для створення програмних об'єктів, використовуваних у рішенні.

7. XAML (розширювана мова розмітки додатків) заснований мовою XML і визначає всі властивості елементів дизайну на сторінці. Він дозволяє відокремити дизайн і властивості елементів від програмного коду.
8. XML (розширювана мова розмітки) є основою для створення мов для опису сутностей.
9. Елементи можуть генерувати події, які можуть бути пов'язані з методами програми на C#. Ім'я методу вказується в XmlaОписі для елемента.
10. Методи, пов'язані з подіями, містять бізнес-логікові додатка.

5. Вправи

Вправа 1. Користувацький інтерфейс програми Калькулятор часу

Ваша компанія розв'язала створити додаток для адвокатів. Додаток повинний зберігати й управляти інформацією про час, який витрачає їхній персонал на роботу із клієнтами. У цей час адвокати записують у звіт кількість хвилин, які вони витратили на роботу із клієнтом, і ці звіти обробляються вручну. Після того як адвокатам видали телефони на платформі Windows Phone, було вирішено створити простий Калькулятор часу, який вони зможуть використовувати для розрахунку часу.

Адвокати будуть вводити час запуску й час закінчення (у годиннику й хвилинах), і програма повинна виводить на екран кількість хвилин між цими оцінками часу.

Ваш менеджер попросив Вас розробити користувацький інтерфейс Silverlight для цього додатка:

1. Визначите, які елементи інтерфейсу Silverlight потрібно розмістити на головній сторінці додатка.
2. Намалюйте схему розташування елементів на формі.
3. Створіть «Посібник користувача» для додатка.
4. Спробуйте визначити всі можливі випадки введення неприпустимих значень.
5. Додайте опис дій додатка у випадку введення неприпустимих даних.
6. Створіть набір тестових даних і результатів, які повинна видати програма, і оформите їх у вигляді таблиці. Кількість тестів повинна бути не менш 10.

Вправа 2. Синхронізація даних калькулятора часу

Один з менеджерів по продукції вважає, що телефон повинен відправляти деталізовану інформацію про кожний тимчасовий інтервал у корпоративну систему так, щоб інформація про проведений час могла обновлятися автоматично. Менеджер розв'язав, що потрібно відправляти наступну інформацію:

- дата й час початку роботи;
- дата й час закінчення роботи;
- назва компаніїклієнта;

- місце розташування клієнта, якщо зустріч відбувається не в приміщенні компанії клієнта;
- оцінка якості роботи, поставлена клієнтом — ціле число в діапазоні від одного до п'яти.

На сервер потрібно відправляти звіти, що містять зазначену інформацію про кожну робочу зустріч.

1. Створіть приклад XmlaФайлу, що містить опису трьох зустрічей.
2. Упевніться, що файл містить запису про зустрічі в приміщенні компаніїзамовника, а також записи про зустрічі за межами компанії.

6. Довідковий матеріал

Середовище розробки

Після установки засобів розробки Windows Phone SDK у діалогові вікна Створити проект в Visual Studio з'являться групи проектів для Silverlight for Windows Phone:

Після установки розроблювачеві доступні наступні шаблони додатків Silverlight for Windows Phone:

- Windows Phone Application
- Windows Phone Databound Application
- Windows Phone Class Library
- Windows Phone Panorama Application
- Windows Phone Pivot Application
- Windows Phone Silverlight and XNA Application
- Windows Phone Audio Playback Agent
- Windows Phone Audio Streaming Agent
- Windows Phone Scheduled Task Agent

Шаблони додатків

Серед шаблонів проектів є три шаблони, що представляють собою три основні стилі додатка для Windows Phone:

- Windows Phone Application
- Windows Phone Pivot Application
- Windows Phone Panorama Application



Рис. 8. Шаблони додатків для Windows Phone

Windows Phone Application — це аналог простого діалогового додатка, у якого один основний екран, через який відбувається основна взаємодія з користувачем.

Windows Phone Pivot Application — це якийсь аналог додатка із закладками, де заголовок кожної закладки визначає вміст. Стандартний варіант використання — кожна закладка являє собою ті самі, у цілому, дані, але в різних виставах і/або з різною фільтрацією. Наприклад, календар, поштовий клієнт і налаштування телефону. Шаблон використовує елемент керування Pivot.

Windows Phone Panorama Application — додаток-панорама, у якому зони взаємодії з користувачем також розділені на панелі, але доступні вони через горизонтальне прокручування; фонове зображення встановлене відразу на всю панораму, вона має загальний заголовок, який прокручується повільніше, ніж панелі; контент сусідньої панелі праворуч видний при відображенні поточної. Наприклад, у такий спосіб реалізовані хаби в Windows Phone: Контакти, Marketplace, Фото, Музика+відео й ін. Шаблон використовує елемент керування Panorama.

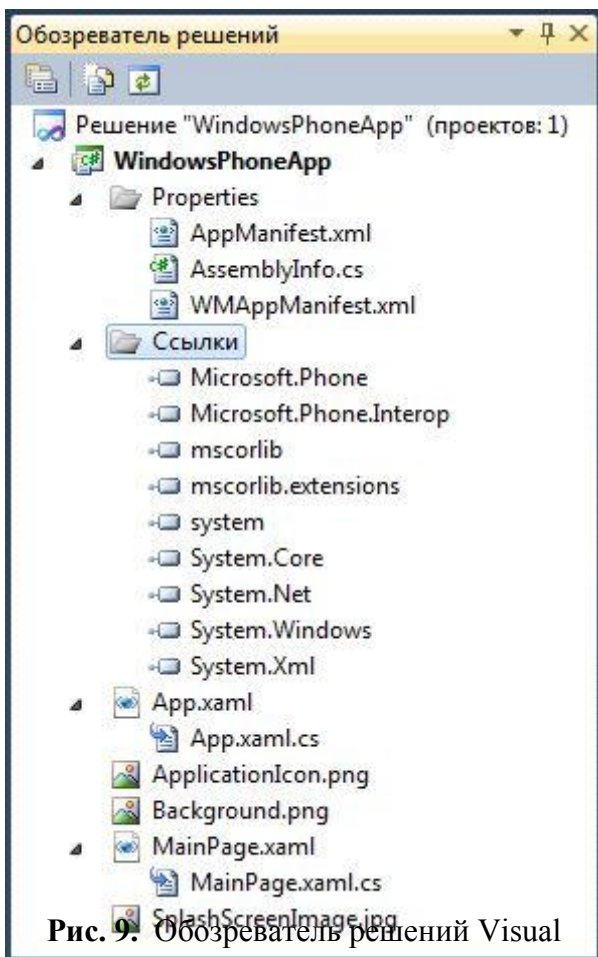


Рис. 9. Обозреватель решений Visual Studio

Структура простого додатка

Розглянемо структуру проекту в оглядачі рішень Visual Studio:

За замовчуванням новий проект містить наступні файли:

- Appmanifest.xml — файл маніфесту, необхідний для генерації XAP файлу, у який упакується додаток для розгортання на телефоні;
- Assemblyinfo.cs — ще один конфігураційний файл, у якому визначаються деякі метадані головного складання додатка;
- Wmappmanifest.xml — файл метаданих, який містить різноманітні налаштування додатка: заголовок, завдання першої сторінки, шляхи до значків, визначення необхідних системних можливостей і т.д.;

- App.xaml — файл ресурсів додатка. Тут розташовуються глобальні ресурси або глобальні події (які відбуваються при запуску додатка). Цей файл також є крапкою входу додатка;
- App.xaml.cs — файл відділеного коду для App.xaml. Тут можна обробляти події й помилки рівня додатка;
- Applicationicon.png — картинка, яка буде значком додатка в телефоні. Цей файл є першим, що побачать користувачі при роботі з додатком;
- Background.png — картинка, яка використовується, коли додаток закріплений на стартовому екрані телефону;
- Mainpage.xaml — головна сторінка додатка. Ця сторінка представляє інтерфейс, який бачить користувач при запуску додатка;
- Mainpage.xaml.cs — файл коду сторінки Mainpage.xaml;
- SplashScreenimage.jpg — картинка, яка відображається під час завантаження додатка.

Файли XAML визначають інтерфейс додатка. Насправді це просто XML файли з мовою розмітки XAML.

Незважаючи на те, що це найпростіший проект, він містить усі ключові елементи, які містять усі інші шаблони й типи проектів.

Зверніть увагу, що частина налаштувань, представлених у вигляді конфігураційних файлів, можуть редагуватися у візуальному інтерфейсі редагування налаштувань додатка.

Основні елементи керування

Щоб створити простий додаток Silverlight для Windows Phone, необхідно розмістити на сторінці додатка візуальні елементи користувацького інтерфейсу. Візуальні елементи можна розділити на кілька груп:

1. Елементи для введення даних користувачем:
 - Textbox — використовується для введення як короткого, так і багаторядкового тексту;
 - Passwordbox — спеціальний елемент для введення пароля, приховує введення користувача;
 - Checkbox — являє собою елемент керування "прапорець". Може бути встановлений або снять, опціонально можна включити підтримку "невизначеного" стану;
 - Radiobutton — дозволяє користувачеві вибрати тільки один варіант із декількох. Згруповані елементи керування (один Groupname) дозволяють вибрати тільки один варіант із групи;
 - Slider — дозволяє користувачеві вибрати з декількох послідовних варіантів. Позиція співвідноситься зі значенням властивості Value;
2. Елементи для вказівки виконання дій:
 - Button — кнопка, при натисканні користувачем генерує подію Click;

- Hyperlinkbutton — кнопка, що відображає посилання. При натисканні переходить на посилання, зазначену у властивості Navigateuri;
3. Елементи для виводу даних на сторінку додатка:
- Textblock — дозволяє відобразити простий текст, без можливості редагування;
 - Image — дозволяє вивести на екран зображення;
 - Listbox — відображає список елементів, які може вибрати користувач;
 - Progressbar — відображає поточний прогрес користувачеві;
4. Елементи для розмітки сторінки:
- Scrollviewer — додає можливість прокручування дочірнім елементам;
 - Border — надає іншому елементу керування рамку й/або фонове зображення;
 - Stackpanel — дозволяє розташовувати дочірні елементи в один ряд по горизонталі або вертикалі з автоматичним вирівнюванням;
 - Grid — є контейнером для інших елементів керування. Дозволяє розташовувати й упорядковувати дочірні елементи керування із прив'язкою до сітки;
 - Canvas — визначає область, у рамках якої можна явно розташувати дочірні елементи за допомогою координат щодо області;
5. Спеціалізовані елементи:
- Map — відображає карту Bing;
 - Mediaelement — дозволяє відтворювати звуки або відео;
 - Webbrowser — відображає веб-сторінку мовою HTML.

7. Додаткові матеріали

1. Windows Phone 7.5 Training Kit:
 Labs\Yourfirstwp7Application\Yourfirstwp7App.html\html\
 Docset_386d610c-7ea0-473f-b375-e10429dfd675.html.
2. Сайт MSDN: <http://msdn.microsoft.com/ru-ru/library/hh202915>.

Лабораторна робота № 2. Модернізація програми Silverlight для Windows Phone

1. Поліпшення додатка

1.1. Керування властивостями елементів

Візуальні елементи Silverlight фактично є класами в коді, і можна настроїти їхнє відображення, змінюючи властивості окремих екземплярів цих елементів. Властивості елементів Silverlight можна задати в області властивостей в Visual Studio або в Xaml-Коді, у якому описуються всі елементи сторінки. Ці налаштування можна задати при створенні програми, а також можна встановлювати властивості елементів і під час роботи програми, наприклад:

```
resulttextblock.Text = result.ToString();
```

У програмі можна управляти будь-якими властивостями візуальних елементів. Розгляд усіх існуючих властивостей елементів виходить за рамки даного курсу, але рекомендується вивчити їх самостійно.

Обробка помилок. На теперішній момент у розробленій на минулих лекціях програмі Суматор не здійснюється обробка помилок. Якщо користувач уведе довільний текст в елемент Textbox, то програма спробує використовувати метод Parse, щоб перетворити цей текст у число, і це може завершитися невдачею (рис. 10).

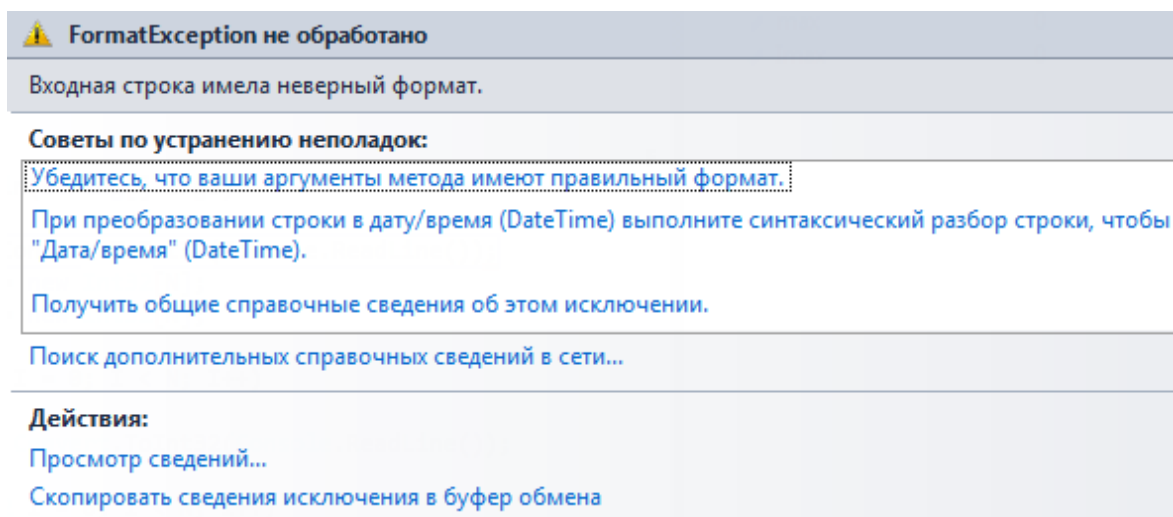


Рис.10. Повідомлення про помилку при введенні неприпустимого значення

Така помилка виникає, оскільки програмний код не робить перевірку на допустимість уведених значень:

```
float v1 = float.Parse(firstnumbertextbox.Text);
```

Якщо в елемент `firstnumbertextbox` увести текст, який не може бути перетворений у число (наприклад, "один"), то виникне виключення. Можна змінити код, використовуючи іншу версію методу `Parse`:

```
float v1 = 0;
if (!int.TryParse(firstnumbertextbox.Text, out v1))
{
    // Код для обробки введення неприпустимого значення
}
```

Цей код використовує метод `Tryparse`, який повертає значення `false`, якщо елемент `firstnumbertextbox` буде містити неприпустимий текст.

Зміна кольору тексту. При виникненні помилки можна виділити текст елемента `Textbox` червоним кольором. `Silverlight` використовує кисті для відрисовки елементів на екрані. Цей потужний механізм дозволяє намалювати текст, використовуючи зображення або текстури, і застосувати деякі додаткові ефекти. В `Silverlight` є набір стандартних квітів, і можна зробити текст елемента `Textbox` червоним, зробивши основну кисть безперервним червоним заливанням:

```
float v1 = 0;

if (!float.TryParse(firstnumbertextbox.Text, out v1))
{
    firstnumbertextbox.Foreground =
        new SolidColorBrush(Colors.Red);
    return;
}
```



Тепер, якщо користувач уведе неприпустиме значення, основний колір текстового поля стане червоним (рис. 11), і виконання методу завершиться без обчислення результату. Однак програма повинна видавати повідомлення про помилку, якщо користувач уведе неприпустимий текст не тільки в перше, але й у друге текстове поле.

Крім цього, у наведеному прикладі є ще один недолік: після того як користувач виправить помилку введення, колір тексту залишиться червоним. Потрібно додати код для повернення кольору тексту в первісний, коли обоє числа будуть введені правильно. Для цього потрібно зберегти вихідну кисть, щоб її можна було використовувати повторно. Повна версія методу `calculateresult` може виглядати так:

Рис. 11. Реакция программы на ошибку при вводе данных

```

private SolidColorBrush errorbrush = new SolidColorBrush(Colors.Red);
private Brush correctbrush = null;

private void calculateresult()
{
    bool errorfound = false;

    if (correctbrush == null)
        correctbrush = firstnumbertextbox.Foreground;

    float v1 = 0;

    if (!float.TryParse(firstnumbertextbox.Text, out v1))
    {
        firstnumbertextbox.Foreground = errorbrush;
        errorfound = true;
    }
    else
        firstnumbertextbox.Foreground = correctbrush;

    float v2 = 0;

    if (!float.TryParse(secondnumbertextbox.Text, out v2))
    {
        secondnumbertextbox.Foreground = errorbrush;
        errorfound = true;
    }
    else
        secondnumbertextbox.Foreground = correctbrush;

    if (errorfound)
        resulttextblock.Text = "Помилка при введенні!";
    else
    {
        float result = v1 + v2;
        resulttextblock.Text = result.ToString();
    }
}

```

Цей код дозволяє обробити можливі помилки при введенні числових даних. Часте створення програм, які одержують дані від користувача, закінчується створенням великого обсягу коду для обчислення результату й обробки помилок при введенні даних.

1.2. Редагування Xaml-Коду елементів Silverlight

XAML є декларативною мовою й використовується для опису й налаштування об'єктів Silverlight. Наприклад, можна настроїти режим введення для елементів Textbox для введення чисел.

Конфігурування елемента Textbox для використання цифрової клавіатури. На теперішній момент програма Суматор використовує стандартну клавіатуру Windows Phone, яка з'являється при виборі елемента Textbox і дозволяє вводити в поле будь-який текст. У нашому випадку переважніше використовувати варіант клавіатури, призначений тільки для введення чисел. Існує кілька конфігурацій екранної клавіатури Windows Phone.

Можна настроїти всі елементи Textbox, щоб при їхньому виборі відображалася клавіатура для введення чисел. Для цього можна змінити відповідна властивість елементів Textbox під час роботи програми, але краще вказати налаштування елемента безпосередньо в XamlФайлі:

```
<Textbox Height="72" Horizontalalignment="Left" Margin="8,19,0,0"
  Name="firstnumbertextbox" Text="0" Verticalalignment="Top" Width="460"
  Textalignment="Center">
  <Textbox.Inputscope>
    <Inputscope>
      <Inputscopename Namevalue="Digits" />
    </Inputscope>
  </Textbox.Inputscope>
</Textbox>
```

Цей опис указує Silverlight положення елемента Textbox на екрані, розмір елемента, його ім'я й деяку інформацію про вирівнювання тексту. Кожне зі значень, укладених у блок <Textbox ... /> є атрибутом текстового поля. Атрибути — це прості елементи, які не є структурованими і являють собою пари "ім'язначення".

Щоб вказати елементу Textbox, яка форма тексту повинна бути введена, у нього доданий вкладений елемент Textbox.Inputscope. Цей елемент є частиною значення Textbox.

Проста інформація про властивості елемента Textbox представляється у вигляді атрибутів, у той час як значення Inputscope є вкладеним елементом, який містить ряд властивостей, які описують тип тексту, що вводиться, наприклад текст, що вводиться, повинен містити цифри й текст, але не знаки пунктуації. Нам досить тільки введення цифр, для чого потрібно додати елемент:

```
<Inputscopename Namevalue="Digits" />
```

При використанні цього опису в обох текстових полях у програмі Суматор при їхньому виклику буде використовуватися клавіатура для введення числових значень, яка є більш зручної, чому клавіатура для введення довільного тексту.

Visual Studio підтримує технологію Intellisense, суть якої полягає в тому, що редактор підказує, яке значення можна ввести в поточне положення курсору. Це суттєво полегшує редагування тексту в програмі й допомагає запобігти можливим помилкам при наборі тексту.

Використання атрибутів і елементів в XAML. У мові XAML атрибути й вкладені елементи є взаємозамінними. Значення властивості елемента може бути лічене як з атрибута, так і з однойменного вкладеного елемента. Можна вказати наступний опис:

```
<Textbox Inputscope="Number" Height="72" Horizontalalignment="Left"
  Margin="144,44,0,0" Name="starthourtextbox" Text="00"
  Verticalalignment="Top"
  Width="104" Textalignment="Center" />
```

Цей опис указує, що текст, що вводиться, повинен бути числом. Такий опис не дозволяє встановити кілька значень властивості, але виявляється більш простим для завдання одного значення.

```
<Textbox Horizontalalignment="Left" Margin="8,175,0,0"
Name="secondnumbertextbox"
    Text="0" Verticalalignment="Top" Width="460" Textalignment="Center">
    <Textbox.Height>72</Textbox.Height>
</Textbox>
```

У цьому описі висота елемента Textbox зазначена не як атрибут, а як вкладений елемент. Ім'я елемента повинне містити назву типу елемента, частиною якого є ця властивість. При цьому не можна вказати в якості імені елемента просто Height, а потрібно вказати Textbox.Height. Це не обмеження XML, а вимога XAML.

З погляду XAML не важливо, яка із цих форм використовується в описі. Для простих пар "ім'язначення" можна використовувати атрибути, але для більш складних структур краще використовувати вкладені елементи.

Настроювання властивостей C#. У коді мовою C# установка значень властивостей виглядає так:

```
// Створення нового елемента Inputscope
Inputscope digitscope = new Inputscope();

// Створення нового елемента Inputscopename
Inputscopename digits = new Inputscopename();
// Установка властивості name у значення Digits
digits.Namevalue = Inputscopenamevalue.Digits;

// Додавання елемента Inputscopename в елемент Inputscope
digitscope.Names.Add(digits);

// Привласнити елементу Textbox створена властивість
firstnumbertextbox.Inputscope = digitscope;
```

Цей код створює всі елементи, представлені в описаному вище коді XAML.

1.3. Вивід вікна з повідомленням

Silverlight містить клас MessageBox, який можна використовувати для повідомлення користувачеві про помилку:

```
MessageBox.Show("Неприпустиме введення");
```

Ця команда виведе у верхню частину екрана повідомлення й видасть звуковий сигнал.

Якщо потрібно відправити користувачеві телефону повідомлення з більшим обсягом тексту, можна вивести на екран повідомлення в кілька рядків:

```
MessageBox.Show("Неприпустиме введення" + System.Environment.NewLine +
    "Будь ласка, повторите введення");
```

Елемент `System.Environment.NewLine` вставить символ нового рядка, прийнятий у цільовій платформі. Клас `MessageBox` можна використовувати, якщо потрібно переконатися в тому, що користувач прийняв повідомлення перед продовженням роботи. При виводі повідомлення на екран виконання програми буде припинено.

Вікно повідомлення з вибором. Можна запропонувати користувачеві зробив вибір у вікні `MessageBox`:

```
if (MessageBox.Show("Ви дійсно прагнете це зробити?", "Виконання дії",
    MessageBoxbutton.Okcancel) == MessageBoxresult.OK)
{
    // Дії, якщо користувач натисне "ok"
}
else
{
    // Дії, якщо користувач натисне "скасування"
}
```

Цей код виводить на екран повідомлення з можливістю скасування дії, представлене на рис. 12. Блок `if` дозволяє виконувати різні дії залежно від відповіді користувача.

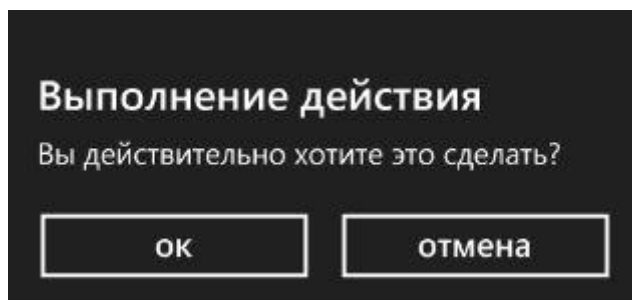


Рис. 12. Спливаюче вікно з питанням користувачеві

1.4. Додавання й використання ресурсів

У додатку можна використовувати різні ресурси, наприклад, зображення й звуки. Усі ресурси розміщаються разом з іншими файлами додатка в `ХарфФайл`, який передається на пристрій.

При використанні зображень у додатку важливо пам'ятати, що цільовий пристрій не містить великий обсяг пам'яті або екран високого дозволу. Найвищий дозвіл для `Windows Phone` становить `800*480` пікселів, тому немає необхідності використовувати фонове зображення більшого розміру.

Посилання на ресурси. Якщо потрібно використовувати ресурси відразу в декількох проектах, у них можна додати не самі ресурси, а посилання на ресурси. У цьому випадку при зміні файлів ресурсів їх нова версія буде автоматично використовуватися в цих проектах при їхній наступній побудові.

Властивість ресурсу "Дія при побудові". При побудові проекту в Visual Studio існує кілька способів додавання ресурсів до програми. Часто використовуються два способи: додавання ресурсу як елемент контенту або як частини складання.

За замовчуванням при побудові проекту ресурс вбудовується в складання. У цьому випадку властивість Дія при побудові має значення Впроваджений ресурс.

Якщо необхідно використовувати ресурс як елемент контенту, потрібно привласнити властивості Дія при побудові значення Зміст. У цьому випадку при побудові програми файл із ресурсом буде скопійований у каталог додатка.

Використання в програмі зображення як частина вмісту. Ресурс, який є елементом контенту, доступний у додатку у вигляді файлу, який перебуває в одній папці із програмою. Щоб вивести на екран зображення з файлу, потрібно додати рядок у код XAML утримуючу шлях до цього файлу із зображенням:

```
<Image Height="611" Horizontalalignment="Left" Margin="8,0,0,0"
    Name="gearbackgroundimage" Stretch="Fill" Verticalalignment="Top"
    Width="472"
    Source="Addinggears.jpg" />
```

Елемент Image виводить на екран сторінки зображення. У ньому описаний атрибут Source, який визначає файл із зображенням. Якщо ввести ім'я файлу як джерела, то при запуску програми зображення буде завантажуватися з файлу й виводитися на екран. Silverlight виводить елементи на екран у тому порядку, у якому вони описані в XamlфФайлі. Якщо цей рядок розмістити перед оголошенням елемента Grid, що містить елементи сторінки, то зображення буде виведено на задньому плані.

Використання в програмі зображення як впроваджений ресурс. Якщо в додатку використовується зображення, вбудоване в складання, XamlКод для звертання до цього зображення може виглядати так:

```
<Image Height="611" Horizontalalignment="Left" Name="gearbackgrounds"
    Stretch="Fill"
    Verticalalignment="Top" Width="472"
    Source="/Addingmachine;component/Images/Addinggears.jpg" />
```

Під час роботи програми зображення буде виглядати однаково, незалежно від способу його використання в програмі. Різниця полягає в розташуванні ресурсу й у коді для звертання до нього.

Вибір способу розміщення ресурсів. При створенні додатків для Windows Phone важливо зрозуміти різницю між елементами контенту й впровадженими ресурсами додатка.

Елемент контенту — це звичайний файл, який зберігається в каталозі додатка. Програма може відкривати й використовувати файли вмісту, коли в цьому є необхідність. Додавання файлів не впливає безпосередньо на розмір програми, хоча при завантаженні вмісту файлів програма буде використовувати більше пам'яті.

Впроваджений ресурс зберігається в складанні програми, і її розмір збільшується. Коли пристрій Windows Phone завантажує програму, воно переглядає все складання програми й виконує деякі перевірки коду. Чим більше розмір програми, тим більше часу займають перевірки. Якщо завантаження програми займає занадто багато часу, її виконання буде перервано операційною системою Windows Phone. Тому потрібно з обережністю впроваджувати більші за обсягом ресурси в складання програми.

Додавання зображення як елемент вмісту дозволяє зменшити розмір програми, і програма буде швидше запускатися. Однак, елементи вмісту завантажуються довше, оскільки перебувають у файловій сховищі. Додавання зображення як вбудований ресурс дозволяє програмі швидше завантажуватися, але збільшує розмір програми, і вона запускається довше.

Рекомендується дотримуватися наступних рекомендацій:

- якщо програма використовує багато більших ресурсів, які не повинні бути завантажені відразу всі, то краще зберігати їх в окремих файлах. Наприклад, можна дозволити користувачеві вибирати фонове зображення з декількох доступних;
- якщо програма використовує багато невеликих по розміру ресурсів, до яких програма часто звертається, то їх краще впроваджувати в складання. Наприклад, можна використовувати власні значки в меню налаштування програми;
- якщо програма занадто довго запускається, рекомендується перенести частина ресурсів зі складання у файли й додати в програму заставку, яка буде відображатися під час запуску програми. Також можна прискорити процес запуску програми, розділивши програму на кілька зв'язаних складань.

2. Зміна й відображення даних

У створеної нами програми Суматор є недолік, пов'язаний з тим, що для одержання результату потрібно щораз натискати на кнопку. Було б краще, якби результат обновлявся автоматично при зміні тексту в одному з текстових полів. Це можна здійснити за допомогою подій.

Подія Textchanged

Раніше ми використовували подію елемента Button, яке відбувається при натисканні на кнопку. З погляду програми подія є викликом методу. Код XAML, що описує кнопку, може виглядати так:

```
<Button Content="сума" Height="72" Horizontalalignment="Left"
        Margin="158,275,0,0" Name="equalsbutton" Verticalalignment="Top"
        Width="160" Click="equalsbutton_Click" />
```

При натисканні на кнопку викликається зазначений оброблювач події. У нашому випадку в ньому викликається метод calculateresult.

```
private void equalsbutton_Click(object sender, RoutedEventArgs e)
{
    calculateresult();
}
```

Можна довідатися, які події може викликати елемент, клацнувши по ньому в редакторі Visual Studio і потім нажавши на кнопку Події в області властивостей. Там же можна призначити, і навіть автоматично створити оброблювачі подій, які потрібно обробити в програмі.

Для елемента `firstnumbertextbox` можна створити оброблювач події `TextChanged`, двічі клацнувши в рядку `TextChanged` у порожньому полі праворуч, і викликати в ньому метод `calculateresult`:

```
private void firstnumbertextbox_TextChanged(object sender, TextChangedEventArgs e)
{
    calculateresult();
}
```

Те ж саме можна зробити для другого текстового поля. Тепер, при зміні тексту в кожному із цих полів результат буде оновлюватися. Отже, тепер можна вилучити кнопку сума.

Однак, виникає проблема з перевіркою допустимості введених значень. Якщо користувач уведе текст замість числа, то програма це виявить і виведе на екран повідомлення. Однак, це повідомлення буде виводитися на екран по два рази при введенні неприпустимого символу. Тобто, подія відбувається двічі — і це відома проблема для додатків Windows Phone. Обійти цю проблему можна, перевіряючи, чи були змінені дані при послідовних викликах подій з тими ж даними:

```
string oldfirstnumber = "";

private void firstnumbertextbox_TextChanged(object sender, TextChangedEventArgs e)
{
    if (firstnumbertextbox.Text == oldfirstnumber)
        return;

    oldfirstnumber = firstnumbertextbox.Text;

    calculateresult();
}
```

Цей оброблювач викликає метод `calculateresult`, тільки якщо текст в елементі `Textbox` дійсно змінився.

Прив'язка даних

Прив'язка даних дозволяє зв'язати візуальні елементи з методами, які працюють із цими елементами. Це дозволяє зв'язати властивість візуального елемента прямо з об'єктом. Усі що потрібно для цього зробити — це створити

особливий об'єкт, значення якого будуть автоматично передаватися елементам Silverlight.

Прив'язка даних може працювати в будь-якому напрямку. Можна зв'язати текст елемента Textbox з додатком таким чином, щоб при зміні тексту в цьому текстовім полі додаток був про це сповіщене. Також можна зв'язати елемент Textblock з об'єктом так, щоб при зміні властивості об'єкта вміст елемента Textblock відповідно обновлялося. Це приклад двунправленої прив'язки, коли програма може сама змінити елемент і відреагувати на зміни в цьому елементі. Можна задати односпрямовану прив'язку, щоб програма тільки виводила на екран результати при зміні властивості об'єкта. Нарешті, можна зв'язати будь-які властивості об'єктів так, щоб програма могла перемістити елемент Silverlight по екрану при зміні властивостей X і Y об'єкта гри.

Створення об'єкта для прив'язки. Спробуємо створити версію програми Суматор, що використовує прив'язку даних. Почнемо із класу, який зробить усю роботу. Потрібно визначити властивості класу, які будуть зв'язані із властивостями візуальних елементів. Наш об'єкт буде містити три властивості:

- текст верхнього елемента Textbox;
- текст нижнього елемента Textbox;
- текст елемента Textblock.

Можна створити клас Adderclass, що має наступну структуру:

```
public class Adderclass
{
    private int topvalue;
    public int Topvalue
    {
        get
        {
            return topvalue;
        }
        set
        {
            topvalue = value;
        }
    }

    private int bottomvalue;
    public int Bottomvalue
    {
        get
        {
            return bottomvalue;
        }
        set
        {
            bottomvalue = value;
        }
    }

    public int Answervalue
    {
        get
        {
```

```

        return topvalue + bottomvalue;
    }
}

```

У цього класу три властивості. Перші два призначені для читання й запису, щоб можна було одержати текст верхнього й нижнього текстового поля й занести в них нові значення. Третя властивість призначена тільки для одержання останнього результату без можливості його зміни прямо. Програма може створити екземпляр цього класу, установити значення властивостей Topvalue і Bottomvalue і потім одержати результат.

Додавання функцій повідомлення. Для того щоб клас можна було зв'язати з об'єктом Silverlight, у ньому повинен бути реалізований інтерфейс Inotifypropertychanged:

```

public interface Inotifypropertychanged
{
    // Подія відбувається при зміні значень властивості
    event PropertyChangedeventhandler PropertyChanged;
}

```

Клас, який реалізує цей інтерфейс, повинен містити делегат події, яка буде використовувати клас Adderclass для повідомлення про зміну значення властивості.

```

public event PropertyChangedeventhandler PropertyChanged;

```

Клас PropertyChangedeventhandler використовується елементами Silverlight для керування повідомленнями подій. Він описаний у просторі імен System.ComponentModel. Якщо компонент Silverlight потрібно зв'язати з якою-небудь властивістю класу, можна додати до цієї події делегати.

Клас Adderclass буде використовувати цей делегат для того, щоб повідомити про зміну одного із властивостей у класі. При цьому відбудеться відновлення екрана. Пов'язані із цими властивостями об'єкти Silverlight зв'язуються з делегатом, щоб вони могли одержати повідомлення якщо буде потреба. Остаточна версія класу може виглядати так:

```

public class Adderclass : Inotifypropertychanged
{
    public event PropertyChangedeventhandler PropertyChanged;

    private int topvalue;

    public int Topvalue
    {
        get
        {
            return topvalue;
        }
        set
        {
            topvalue = value;
        }
    }
}

```

```

        if (Propertychanged != null)
        {
            Propertychanged(this, new Propertychangedeventargs
                ("Answervalue"));
        }
    }

    // те ж саме для властивості Bottomvalue

    public int Answervalue
    {
        get
        {
            return topvalue + bottomvalue;
        }
    }
}

```

Клас Adderclass тепер містить властивість Propertychanged. Щораз при зміні значення властивості він повинен перевірити значення властивості Propertychanged. Якщо воно не рівно null, то воно пов'язане з делегатом. Можна розглядати властивість Propertychanged як список розсилання. Процес може підписатися на розсилання й одержувати повідомлення про події, що цікавлять, які відбуваються в Adderclass. Якщо ніхто не підписаний на список розсилання (тобто значення Propertychanged рівно null), те немає крапки виклику методу для опису змін. Однак, якщо воно не рівно null, це означає, що є об'єкт, який цікавлять зміни, що відбуваються в Adderclass.

Виклик методу, який генерує подія, виглядає в такий спосіб:

```
Propertychanged(this, new Propertychangedeventargs("Answervalue"));
```

Перший параметр є посиланням на об'єкт, який генерує подія. Другим параметром є значення аргументу, яке містить ім'я властивості, значення якого змінилося. Елемент Silverlight використовує технологію відбиття (тобто він буде зчитувати публічні властивості класу Adderclass), щоб довідатися, які властивості доступні. Якщо він одержить повідомлення, що значення властивості Answervalue змінилося, то він буде оновлювати всі властивості візуальних елементів, зв'язані із властивістю цього класу.

Коли програма згенерує подію, на екран буде виведено оновлене значення властивості Answervalue. При виклику коду секції get властивості Answervalue він обчислює й повертає результат.

```

public int Answervalue
{
    get
    {
        return topvalue + bottomvalue;
    }
}

```

Наступним кроком буде з'єднання цього об'єкта з користувацьким інтерфейсом. Користувацький інтерфейс повідомить клас Adderclass, коли

значення двох текстові полів будуть змінені користувачем. Коли будуть вводитися нові значення полів, екземпляр класу Adderclass повинен повідомити користувацький інтерфейс, що результат змінився, і його потрібно вивести на екран.

Додавання простору імен на сторінку MainPage. Для того щоб зв'язати створений об'єкт із даними, потрібно зв'язати код об'єкта з ХамлКодом, що описують користувацький інтерфейс. Необхідно додати в код XAML простір імен, що містить клас Adderclass:

```
xmlns:local="clr-namespace:Addingmachine"
```

Цей атрибут кореневого елемента в ХамлФайлі для сторінки MainPage.xaml робить доступними будь-які класи простору імен Addingmachine для використання в Хамл-Коді. Як і в програмі на С# на початку файлу повинні вказуватися всі директиви using, так і на початку файлу XAML також повинні вказуватися всі використовувані простори імен.

Після додавання простору імен потрібно оголосити ім'я класу, який описаний у цьому просторі імен і використовується в якості ресурсу:

```
<phone:Phoneapplicationpage.Resources>
  <local:Adderclass x:Key="Adderclass" />
</phone:Phoneapplicationpage.Resources>
```

Додавання класу до елемента на сторінці. Після того як простір імен став доступним, можна з'єднати потрібний клас із цього простору імен з елементами сторінки. У нашому випадку таким елементом буде елемент Grid, який містить усі візуальні елементи сторінки. Додавання класу до елемента автоматично робить його доступним для будь-яких вкладених елементів, і таким чином, елементи Textbox і Textblock можуть використовувати цей клас.

```
<Grid x:Name="Layoutroot" Background="Transparent"
  DataContext="{Staticresource Adderclass}">
```

Цей код указує елементу Grid використовувати клас Adderclass, зазначений у значенні атрибута DataContext. У нашому додатку він є статичним ресурсом, у якому клас є частиною програми.

Прив'язка елемента Silverlight до властивості об'єкта. Тепер клас Adderclass доступний у контексті даних елементам на сторінці. При цьому створюється зв'язок між сторінкою Silverlight і об'єктом, у якому описана поведінка.

Тепер потрібно прив'язати властивості кожного елемента до властивостей об'єкта. Для цього можна використовувати область властивостей для кожного елемента. Потрібно зв'язати текст елемента firstnumbertextbox із властивістю Topvalue класу Adderclass. Якщо клацнути по елементу Textbox у редакторі Visual Studio, відкриється область властивостей для цього елемента, у якій для властивості Text у контекстному меню потрібно вибрати пункт Застосувати

прив'язку даних.... Після цього відкриється вікно, у якому потрібно вказати, з якою властивістю якого об'єкта потрібно зв'язати обрана властивість елемента.

Вибравши необхідний елемент зі списку доступних елементів і їх властивостей, можна прив'язати властивість елемента Silverlight до властивості об'єкта. Зверніть увагу, що в розділі Параметри обраний режим Twoway, при якому при зміні користувачем значення в елементі Textbox об'єкт автоматично був про це сповіщений.

При зв'язуванні з елементом Textblock доступний тільки режим Oneway, оскільки не можна передати в програму дані з об'єкта Textblock.

Після установки прив'язки програма буде працювати. При цьому вміст файлу MainPage.xaml.cs з кодом програми не зміниться. Уся робота тепер виконується в класі, який з'єднаний з візуальними елементами. Зміна вмісту елементів Textbox, використовуваних для введення значень, приведе до виникнення події класу Adderclass, які приведуть до зміни вмісту класу Answervalue, пов'язаного з елементом Resulttextblock.

Прив'язка даних через властивість DataContext

Для установки прив'язки до даних треба було проробити більшу роботу. Однак, існує спосіб спростити виконання цих дій.

Установка прив'язки даних в XAML. Можна просто зв'язати властивість об'єкта із властивістю елемента в Xaml-Коді, і Silverlight виконає прив'язку.

```
<Textbox Height="72" Horizontalalignment="Left" Margin="8,19,0,0"
  Name="firstnumbertextbox" Text="{Binding Topvalue, Mode=Twoway}"
  Verticalalignment="Top" Width="460" Textalignment="Center" />
```

У цьому коді зазначена прив'язка тексту елемента до властивості Topvalue об'єкта. Режим прив'язки встановлений у значення Twoway, щоб зміни в програмі (при обчисленні результату) відображалися на екрані. Якщо потрібно використовувати візуальний елемент тільки для того, щоб показати значення (наприклад, результат обчислення), можна використовувати режим прив'язки Oneway.

Установка властивості DataContext. Тепер залишається створити екземпляр класу Adderclass і привласнити його властивості DataContext елемента, що містить елементfirstnumbertextbox. Це можна зробити в конструкторі для основної сторінки.

```
// Конструктор
public MainPage()
{
    Initializecomponent();

    Adderclass adder = new Adderclass();
    Contentgrid.DataContext = adder;
}
```

Властивість DataContext елемента Silverlight ідентифікує об'єкт, який містить усі властивості, пов'язані з елементом і з тими елементами, які в ньому втримуються. Елемент Contentgrid містить елементи Textbox і Textblock, і тепер

кожна прив'язка, яку ці елементи містять, буде відображатися на створений екземпляр класу `Adderclass`.

У цьому прикладі вся важка робота з областю властивостей і ресурсами замінена на невелику кількість Xaml-Коду й один рядок коду.

3. Керування орієнтацією сторінки додатка

Альбомна й книжкова орієнтація

Існує два варіанти орієнтації пристрою: альбомна (пристрій розташований горизонтально) і книжкова (пристрій розташований вертикально). Телефон може визначити орієнтацію, і деякі вбудовані додатки правильно обробляють зміну орієнтації телефону.

Можна створювати додатки, які зможуть коректно працювати в обох режимах орієнтації. При цьому, буде потрібно витратити вдвічі більше часу на створення користувацького інтерфейсу й на написання програмного коду, що обробляє зміну орієнтації телефону, що й змінює відповідно орієнтацію додатків.

Вказівка орієнтації у файлі `Mainpage.xaml`. Сторінка може повідомити `Silverlight`, яку орієнтацію вона підтримує. Для цього використовуються властивості класу `Phoneapplicationpage`:

```
Supportedorientations="Portrait" Orientation="Portrait"
```

Ці налаштування `Visual Studio` використовує за замовчуванням при створенні сторінки. Вони вказують на те, що сторінка використовує тільки портретну орієнтацію. Якщо буде потреба ці налаштування можна змінити:

```
Supportedorientations="Portraitorlandscape" Orientation="Portrait"
```

Із цими налаштуваннями при зміні орієнтації телефону додаток також буде змінювати орієнтацію (рис. 13).

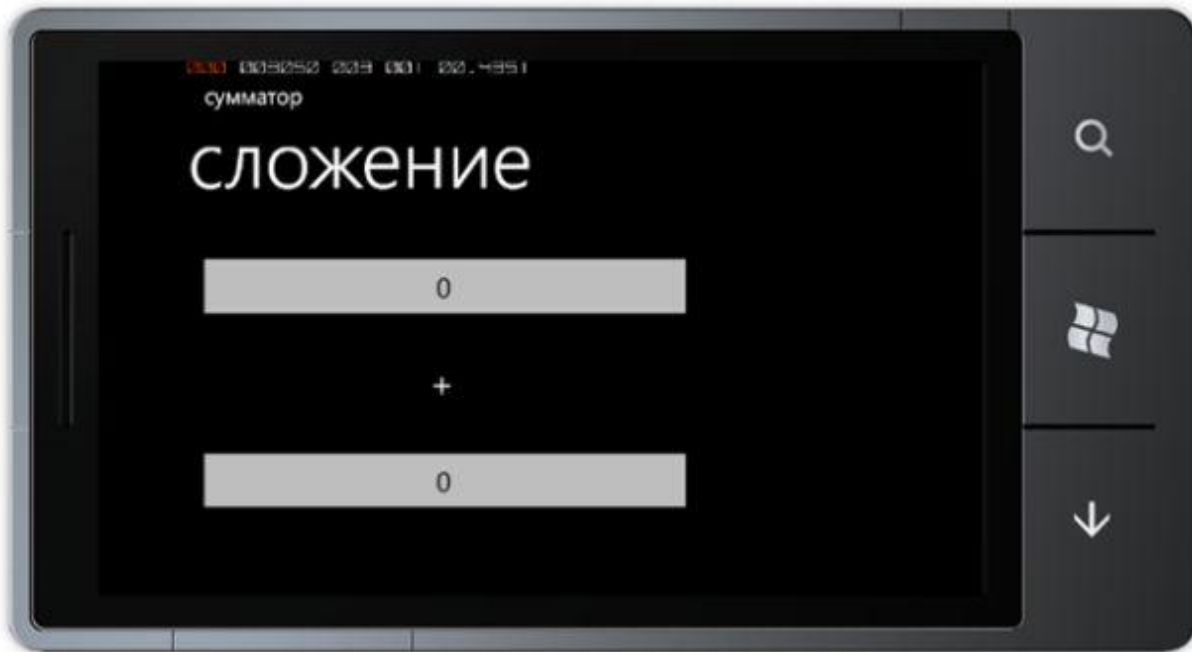


Рис. 13. Додаток Суматор в альбомній орієнтації

Тепер програма змінює вміст екрана при зміні орієнтації пристрою, але оскільки розташування всіх елементів задане фіксованими значеннями, деякі з них перебувають не в тому місці, у якому хотілося б їх бачити, а поле для результату не видне на екрані.

```
<Textbox Height="72" Horizontalalignment="Left" Margin="8,19,0,0"
  Name="firstnumbertextbox" Text="0" Verticalalignment="Top" Width="460"
  Textalignment="Center" />
```

Цей елемент використовує значення відступу від країв екрана, які використовуються для точної вказівки розташування елементів на екрані. При цьому, якщо в елемента зазначена позиція, яка перебуває за межами екрана, то Silverlight просто не буде відображати цей елемент на екрані.

Існує два способи виправлення подібної помилки. Один спосіб полягає в зміні параметрів розташування елементів при зміні орієнтації телефону.

Подія `Orientationchanged`. Клас `Phoneapplicationpage` включає подія `Orientationchanged`, яке відбувається при зміні орієнтації сторінки. Як і для інших подій, для нього можна створити методбоброблювач, у який потрібно додати код для обробки зміни орієнтації екрана:

```
private void Phoneapplicationpage_Orientationchanged(object sender,
  Orientationchangedeventargs e)
{
  if (e.Orientation == Pageorientation.Portraitup)
  {
    setportrait();
  }
  else
  {

```

```

        setlandscape();
    }
}

```

Цей код використовує параметр, значення якого визначає нову орієнтацію, і викликає відповідний метод. Методи змінюють значення відступів від країв елементів, щоб розмістити їхнім підходящим для певної орієнтації образом:

```

private void setlandscape()
{
    firstnumbertextbox.Margin = new Thickness(8, 19, 0, 0);
    firstnumbertextbox.Width = 207;
    secondnumbertextbox.Margin = new Thickness(252, 19, 0, 0);
    secondnumbertextbox.Width = 207;
    plustextblock.Margin = new Thickness(221, 35, 0, 0);
    resulttextblock.Margin = new Thickness(538, 35, 0, 0);
}

```

Код методу `setportrait` аналогічний методу `setlandscape`. Елементи розташовуються у відповідності зі значеннями властивості `Margin`. Відступи від країв задаються значенням властивості `Thickness`, яке містить величину відступу й ширину границі. У нашому випадку ширина границі встановлена в 0, тому границя не буде відображатися. Після додавання цих методів додаток буде працює в обох режимах орієнтації (рис. 14).



Рис. 14. Додаток Сумматор, настроєний під різні режими орієнтації

Часто додатка при зміні орієнтації екрана використовують анімацію переміщення елементів. Для створення подібних ефектів рекомендується використовувати інструмент Expression Blend.

Використання контейнерів для угруповання елементів

Серед елементів Silverlight є елементи-контейнери, які можуть містити інші елементи. Існує кілька різних типів контейнерів, серед яких є елемент Stackpanel.

Елемент Stackpanel містить послідовність елементів Silverlight. Він дозволяє розташовувати вкладені в нього елементи в певному порядку. Елементи в контейнері можуть розташовуватися вертикально (зверху вниз) або горизонтально (ліворуч праворуч). Замість того, щоб набудовувати для візуальних елементів відступи від країв екрана, можна доручити цю роботу елементу Stackpanel:

```
<Stackpanel>
  <Textbox Inputscope="Digits" Height="72" Horizontalalignment="Center"
    Name="firstnumbertextbox" Verticalalignment="Top" Width="460"
    Textalignment="Center" />
  <Textblock Height="56" Horizontalalignment="Center" Name="plustextblock"
Text="+"
    Verticalalignment="Top" Fontsize="32" Width="25" />
  <Textbox Inputscope="Digits" Height="72" Horizontalalignment="Center"
    Name="secondnumbertextbox" Verticalalignment="Top" Width="460"
    Textalignment="Center" />
  <Textblock Height="46" Horizontalalignment="Center" Name="resultttextblock"
    Verticalalignment="Top" Fontsize="30" Width="160" Textalignment="Center"
/>
</Stackpanel>
```

Елементи Stackpanel вирішують самі, як будуть відображатися елементи на екрані. У цьому коді лише зазначене вирівнювання всіх елементів по центру виділеної їм області.

При такому підході виникає дуже корисний побічний ефект, який полягає в тому, що контейнер Stackpanel автоматично перешикує елементи при зміні орієнтації екрана (рис. 15).



Рис. 15. Додаток Суматор, що використовує елемент Stackpanel

Елемент Stackpanel можна додати до проекту, перетягнувши його з панелі інструментів Visual Studio у вікно дизайнера або вводячи текст XAML вручну. Якщо елемент stackpanel повинен групувати елементи горизонтально, потрібно вказати відповідний атрибут:

```
<Stackpanel Orientation="Horizontal">
```

Також можна помістити один Stackpanel в іншій, що дозволяє використовувати інше угруповання вкладених елементів.

Існують і інші типи контейнерів. Один з них — елемент Grid. Він дозволяє створити прямокутну сітку з елементів, у кожному з яких може розміщатися один або кілька елементів. Фактично в додатку Суматор екран містить сітку, що полягає з одного елемента, який займає всю доступну область екрана.

4. Відображення списків даних

Часто в додатках необхідно виводити списки яких-небудь об'єктів. Для цього необхідні спеціальні засоби, що передбачають відображення облікових даних.

Як приклад розглянемо створення списку клієнтів компанії. Інформація про клієнтів включає: ім'я клієнта, адреса й ідентифікатор. Клас Customer може виглядати так:

```
public class Customer
{
    public string Name { get; set; }
    public string Address { get; set; }
    public int ID { get; set; }

    public Customer(string inname, string inaddress, int inid)
    {
        Name = inname;
        Address = inaddress;
        ID = inid;
    }
}
```

Цей клас має єдиний метод, який є конструктором, що ухвалюють три параметри.

Тепер потрібно зберегти список клієнтів. Кращий спосіб для зберігання списків об'єктів — використання класу `List`, який входить до складу мови C#. Можна створити клас `Customers`, що містить ім'я списку клієнтів і сам список:

```
public class Customers
{
    public string Name { get; set; }

    public List<Customer> Customerlist;

    public Customers(string inname)
    {
        Name = inname;
        Customerlist = new List<Customer>();
    }
}
```

Клас `Customers` містить єдиний метод, який є конструктором класу. Він визначає ім'я списку й створює порожній список. Використовуючи цей клас, можна створити новий список у такий спосіб:

```
Customers mailcustomers = new Customers("Список клієнтів");
```

Тепер потрібно заповнити створений порожній список інформацією про клієнтів.

4.1. Створення даних для списку

Для нашого прикладу не має значення, які імена будуть у клієнтів — головне, щоб інформація про клієнтів була різною. Звичайно таку інформацію зручно зберігати в базі даних, але ми поки згенеруємо список на основі невеликої кількості даних. Створимо метод, який буде повертати готовий список:

```
public static Customers Maketestcustomers()
{
    string[] firstnames = new string[] { "Rob", "Jim", "Joe", "Nigel", "Sally",
        "Tim" };
    string[] lastsnames = new string[] { "Smith", "Jones", "Bloggs", "Miles",
```

```

        "Wilkinson", "Brown" });
Customers result = new Customers("Тестовий список");
int id = 0;

foreach (string lastname in lastsnames)
{
    foreach (string firstname in firstnames)
    {
        // формування імені клієнта
        string name = firstname + " " + lastname;
        // додавання клієнта в список
        result.Customerlist.Add(new Customer(name, name + "'s House", id));
        // збільшення значення ідентифікатора для наступного клієнта
        id++;
    }
}
return result;
}

```

У цьому коді створюється 36 клієнтів комбінуванням усіх заданих імен і прізвищ. У кожного клієнта свій унікальний ідентифікаційний номер. Метод `maketestcustomers` є статичним, щоб не потрібно було створювати список перед одержанням тестових даних. Тепер можна використовувати отриманий список у програмі:

```
customerlist = Customers.Maketestcustomers();
```

4.2. Використання елемента `Stackpanel` для виводу списку на екран

Наступним кроком буде вивід отриманого списку клієнтів на екран Windows Phone. Для цього можна помістити ім'я кожного клієнта в окремий елемент `Textblock`. Потрібно, щоб елементи могли розміщатися на екрані незалежно від їхньої кількості. Для цього можна використовувати елемент `Stackpanel`:

```

<Stackpanel Horizontalalignment="Left" Margin="0,0,0,0"
Name="customersstackpanel"
Verticalalignment="Top" />

```

Елементи `Textblock` можна створювати в програмі за допомогою наступного коду:

```

customers = Customers.Maketestcustomers();

foreach (Customer c in customers.Customerlist)
{
    Textblock customerblock = new Textblock();
    customerblock.Text = c.Name;
    customersstackpanel.Children.Add(customerblock);
}

```

У цьому коді для кожного клієнта створюється елемент `Textblock`, у якому у текстове поле записується ім'я клієнта, після чого цей елемент додається `customersstackpanel`. Як видно із прикладу, візуальні об'єкти можна створювати як в Xaml-Коді, так і в коді програми на C#.

Отримана програма може виводити на екран список будь-якого розміру (рис. 16).



Рис. 16. Список клієнтів

Однак, деякі елементи не видні на екрані, оскільки перебувають за його межами. Це є істотним недоліком. Розв'язати проблему можна, додавши в програму можливість прокручування екрана нагору й униз, щоб користувач міг побачити іншу частину списку. Таку можливість надає елемент `Scrollviewer`, якщо в нього помістити елемент `Stackpanel`:

```
<Scrollviewer>
  <Stackpanel Horizontalalignment="Left" Margin="9,6,0,0"
    Name="customersstackpanel" Verticalalignment="Top" />
</Scrollviewer>
```

Тепер у додатку відображається смуга прокручування, і користувач може прокручувати екран нагору й униз, щоб переглянути список повністю.

4.3. Використання елемента Listbox для відображення списку елементів

У представленому прикладі нам довелося самостійно створювати елементи й привласнювати їм значення, після чого розміщати ці елементи на сторінці. Замість виконання цих дій можна використовувати прив'язку даних і спеціальні елементи для розміщення на сторінці списків даних.

Створення шаблону даних. Необхідно зв'язати клас Customer з візуальним елементом, але не можна зробити це прямо, як це було зроблено в програмі Суматор, тому що екземпляр класу Customer містить кілька значень: ім'я клієнта, його адреса й ідентифікатор. Необхідно створити шаблон даних, який визначає, як інформація про клієнта буде виглядати на екрані. Шаблон може мати такий вигляд:

```
<Datatemplate>
  <Stackpanel>
    <Textblock Text="{Binding Name}" />
    <Textblock Text="{Binding Address}" />
  </Stackpanel>
</Datatemplate>
```

Цей шаблон даних визначає, що інформація повинна виводитися в елемент Stackpanel, який містить два елементи Textblock з іменами Name і Address. Відповідно до цього шаблону будуть виводитися два рядки: у першому рядку вказується ім'я клієнта, у другий — його адресу. При цьому в шаблоні не зазначене значення ідентифікатора, оскільки нам не потрібно виводити це значення на екран.

Використання шаблону Datatemplate для списку. Шаблон даних указує Silverlight як потрібно виводити на екран властивості об'єкта. Його можна використовувати для виводу списку клієнтів на екран. Для цього потрібно помістити шаблон Datatemplate у секції Itemtemplate елемента Listbox, який виведе на екран інформацію про клієнтів:

```
<Listbox Name="customerlist">
  <Listbox.Itemtemplate>
    <Datatemplate>
      <Stackpanel>
        <Textblock Text="{Binding Name}" />
        <Textblock Text="{Binding Address}" />
      </Stackpanel>
    </Datatemplate>
  </Listbox.Itemtemplate>
</Listbox>
```

Цей код набудовує зовнішній вигляд списку даних, вивід якого на екран здійснює елемент customerlist. Тепер потрібно вказати список даних, які повинні виводитися на екран:

```
customers = Customers.Maketestcustomers();
customerlist.Itemssource = customers.Customerlist;
```

Цей код одержує тестовий список клієнтів, який привласнюється властивості Itemssource елемента customerlist. Елемент Listbox виведе на екран кожний об'єкт у наборі як елемент списку (рис. 17).



Рис. 17. Список клієнтів з адресами

У коді програми необхідно вказати властивості Itemssource елемента Listbox список введених даних, а всі інші дії виконуються автоматично. Елемент Listbox також автоматично відображає смуги прокручування, якщо список не міститься на екрані цілком. Необхідно тільки розробити шаблон для відображення елементів списку й задати прив'язку до даних.

Додавання стилю відображення. У нашому прикладі у виведеному списку інформація виводиться не дуже наочно, тому що важко візуально відрізнити ім'я клієнта від його адреси. Для підвищення зручності списку можна задати різні розміри й стилі шрифту для імен і адрес, але може бути зручніше використовувати вбудовані стилі, які надаються у вигляді ресурсів для додатків Windows Phone.

Перевага використання вбудованих стилів полягає в тому, що вони використовують різні кольорні схеми Windows Phone. До елементів Textblock у шаблону Datatemplate можна додати назви використовуваних стилів:

```
<Datatemplate>
  <Stackpanel>
    <Textblock Text="{Binding Name}"
      Style="{Staticresource Phonetextralargestyle}" />
    <Textblock Text="{Binding Address}"
      Style="{Staticresource Phonetextsubtlestyle}" />
  </Stackpanel>
```

```
</Datatemplate>
```



У цьому коді використовуються вбудовані стилі Extralargestyle для імені клієнта й Subtlestyle для адреси. Тепер список візуально сприймається набагато краще (рис. 18).

Вибір елементів списку. Наступним кроком буде додавання в програму можливості редагування даних. При виборі одного з елементів списку повинен відображатися екран для редагування обраного елемента (рис. 19).

Рис. 18. Список клієнтів з адресами, що використовує вбудовані стилі



Рис. 19. Экран додатка для редагування інформації про обраного клієнта

При виборі елемента списка элемент Listbox генерує подія Selectionchanged, для якого можна створити оброблювач. Оброблювач може виглядати так:

```
private void customerlist_Selectionchanged(object sender,
Selectionchangedeventargs e)
{
    // одержання вмісту обраного елемента
    Customer selectedcustomer = customerlist.Selecteditem as Customer;

    MessageBox.Show("Обраний элемент " + selectedcustomer.Name);
}
```

Клас `customerlist` містить властивість `Selecteditem`, який є посиланням на обраний об'єкт. У цьому коді обраний об'єкт приводиться до типу `customer`, і програма виводить на екран вікно з повідомленням, у якому зазначено ім'я обраного клієнта.

Тепер необхідно додати можливість перехід до нової сторінки додатка, яка дозволить змінювати інформацію про обраного клієнта. Для цього необхідно вивчити, як у додатках Silverlight виконується перехід до іншої сторінки додатка.

5. Навігація по сторінках додатка

Дотепер усі розглянуті приклади програм використовували тільки одну сторінку Silverlight. Однак, більшість додатків містять кілька сторінок. У нашому прикладі необхідна додаткова сторінка для виводу й редагування інформації про обраного клієнта.

Для додавання нової сторінки в додаток потрібно вибрати в головному меню Visual Studio пункт Проект -> Додати новий елемент.... Відкриється вікно зі списком шаблонів, які можна додати в проект. Виберемо в списку пункт Сторінка Windows Phone у книжковій орієнтації й укажемо ім'я нової сторінки `Customerdetailpage.xaml`. Після натискання на кнопку Додати в проект буде додана нова сторінка додатка Windows Phone. Робота з новою сторінкою нічим не відрізняється від роботи зі сторінкою `Mainpage`.

Навігація по сторінках

Модель переміщення по сторінках Silverlight більше схожа на навігацію в Інтернеті, чому на додаток Windows Forms. У кожній сторінці є своя адреса, яка вказується у вигляді URI (універсальний ідентифікатор ресурсу). Об'єкт `NavigationService` надає методи для навігації.

```
// переміщення до сторінки Customerdetailpage.xaml
NavigationService.Navigate(new Uri("/Customerdetailpage.xaml",
    UriKind.Relativeorabsolute));
```

Цей код викличе перехід додатка на сторінку `Customerdetailpage`. Зверніть увагу, що метод `Navigate` ухвалює значення URI, яке містить назву сторінки й тип URI.

При завданні URI необхідно вибрати, буде зазначений відносний або абсолютна адреса. Установка типу в значення `Relativeorabsolute` забезпечує гнучкість, при завантаженні ресурсів. Значення URI є звичайним рядком, тому необхідно уважно стежити за правильністю вказівки її значення, щоб уникнути помилок під час виконання програми.

Використання кнопки Назад. Користувачі Windows Phone очікують, що при натисканні на апаратну кнопку Назад відбудеться перехід до попередньої сторінки. Ця функціональність автоматично підтримується у всіх додатках для

Windows Phone. Якщо користувач натисне кнопку Назад на сторінці Mainpage, то додаток завершить роботу.

Іноді необхідно перевизначити цю поведінку, наприклад, щоб користувач не переходив на іншу сторінку, якщо не зберіг введені дані. Перевизначити поведінку кнопки Назад можна в оброблювачі події Backkeypress, у якому можна при необхідності скасувати перехід:

```
private void Phoneapplicationpage_Backkeypress(object sender,
    System.ComponentModel.CanceledEventArgs e)
{
    e.Cancel = true;
}
```

Цей метод оброблювач події Backkeypress скасовує перехід до попередньої сторінки при натисканні на кнопку Назад.

Якщо потрібно, щоб користувач підтвердив намір перейти на іншу сторінку, можна використовувати клас MessageBox:

```
private void Phoneapplicationpage_Backkeypress(object sender,
    System.ComponentModel.CanceledEventArgs e)
{
    if (MessageBox.Show("Ви дійсно прагнете перейти на іншу сторінку?",
        "Підтвердження переходу", MessageBoxButton.OKCancel) !=
        MessageBoxResult.OK)
    {
        e.Cancel = true;
    }
}
```

Цей код виводить на екран вікно повідомлення й запитує користувача, чи дійсно він прагне покинути сторінку. Якщо користувач натисне кнопку скасування, перехід буде відмінний.

Передача даних між сторінками

Кожна сторінка Silverlight працює незалежно інших сторінок, і дані кожної сторінки додатки недоступні іншим сторінкам. Але іноді потрібно передавати дані з однієї сторінки в іншу. Якщо дані є простим рядком тексту, найпростіший спосіб передачі — додати рядок до URI цільової сторінки. Цей метод схожий на відправлення запитів на веб-сервер.

У нашій програмі для роботи із клієнтами при виборі користувачем клієнта зі списку потрібно передати його ім'я й адресу на сторінку редагування.

```
private void customerlist_Selectionchanged(object sender,
    SelectionChangedEventArgs e)
{
    // одержання інформації про обраного клієнта
    Customer selectedcustomer = customerlist.SelectedItem as Customer;

    // формування рядка адреси з інформацією про клієнта
    NavigationService.Navigate(new Uri("/Customerdetailpage.xaml?name=" +
        selectedcustomer.Name + "&address=" + selectedcustomer.Address,
        UriKind.Relative));
}
```

```
}
```

Цей метод викликається, коли користувач вибирає клієнта. Метод одержує інформацію про обраного клієнта й формує URI цільової сторінки, у який додає інформацію про обраного клієнта. Частина запиту після символу "?" складається із двох записів виду ім'я=значення, розділених символом "&".

Тепер розглянемо, як цільова сторінка може одержати й використовувати передану інформацію. Для цього необхідно вивчити події, що виникають при навігації по сторінках Silverlight.

Використання подій при навігації по сторінках. При переході на сторінку Silverlight відбувається подія `Onnavigatedto`, а при відході зі сторінки — `onnavigatingfrom`. Для одержання керування при виникненні цих подій потрібно перевизначити відповідні методи в програмному коді сторінки.

Для того щоб додаток одержав керування при переміщенні користувача до сторінки, потрібно перевизначити `Onnavigatedto`:

```
protected override void
Onnavigatedto(System.Windows.Navigation.NavigationEventArgs e)
{
    string name, address;
    if (Navigationcontext.QueryString.TryGetValue("name", out name))
        nametextblock.Text = name;
    if (Navigationcontext.QueryString.TryGetValue("address", out address))
        addresstextblock.Text = address;
}
```

Цей метод намагається одержати із властивості `QueryString` об'єкта `Navigationcontext` значення переданих у рядку адреси параметрів. Щораз, коли користувач переходить до цієї сторінки, цей метод викликається й виводить на сторінку передані значення параметрів. Можна використовувати подію `Onnavigatedto` для настроювання сторінки, коли користувач переходить до неї. При переході на сторінку можна передавати будь-яка кількість параметрів.

Якщо запитуваний на сторінці параметр відсутній, то програма виконується далі без виникнення помилки. Оскільки параметри передаються у вигляді звичайного тексту, потрібно уважно стежити за тим, щоб імена параметрів при передачі даних і їх зчитуванні збігалися.

Наведений вище код можна використовувати при створенні додатка, що полягає із двох сторінок, яке дозволяє користувачеві при виборі клієнта виконати перехід на сторінку з докладною інформацією про цього клієнта.

Однак у цьому коді є проблема, яка виникає при виконанні наступної послідовності дій:

1. Вибір елемента в списку.
2. Перехід до сторінки з докладною інформацією.
3. Натискання кнопки Назад для повернення до списку.
4. Вибір того ж самого елемента в списку.

При виконанні цієї послідовності дій на сторінці не буде відображатися інформація про того ж самого клієнта. Для цього потрібно вибрати іншого клієнта, і потім вибрати першого ще раз. Причина такої поведінки полягає в тому, що програма використовує для переходу подія `Selectionchanged`, яке не відбувається при повторному виборі того ж самого клієнта. Можна усунути цю проблему, скидаючи вибір елемента при переході до цієї сторінки:

```
protected override void
Onnavigatedto(System.Windows.Navigation.NavigationEventArgs e)
{
    customerlist.Selecteditem = null;
}
```

Тепер при повторному виборі того ж самого елемента він буде вважатися "новим" обраним елементом. Однак, установка значення обраного елемента в `null` також розцінюється як вибір елемента, і при спробі звертання до значення цього властивість виникне виключення. Уникнути цього можна, додавши в оброблювач події перевірку, чи був обраний елемент:

```
private void customerlist_Selectionchanged(object sender,
SelectionChangedEventArgs e)
{
    // скасування виклику, якщо елемент не обраний
    if (customerlist.Selecteditem == null)
        return;

    // код методу
}
```

Спільне використання об'єктів декількома сторінками

Часто потрібно, щоб сторінки спільно використовували об'єкти, що містять структуровані дані. У нашому додатку було б зручніше передати на сторінку докладної інформації екземпляр об'єкта.

Можна легко надати спільний доступ до об'єктів для всіх сторінок додатка. Для цього використовується сторінка `App.xaml`, яка автоматично створюється при створенні проекту додатка. Її можна вважати контейнером для додатка Windows Phone. Ця сторінка не виводиться на екран, але файл `App.xaml.cs` містить методи, які виконуються при запуску додатка.

Можна змінити файл `App.xaml.cs` і додати в нього свій код. Якщо потрібно зберігати які-небудь глобальні дані, які повинні бути доступні всім сторінкам додатка, їх оголошення можна розмістити тут.

```
public partial class App : Application
{
    // об'єкт використовується іншими сторінками додатка
    public Customer Activecustomer;
}
```

Цей код містить оголошення об'єкта `Activecustomer`, який буде доступний усім сторінкам. Об'єкт містить посилання на інформацію про обраного клієнта.

Будь-яка сторінка Silverlight може одержати посилання на об'єкт, розміщений на сторінці App:

```
protected override void
Onnavigatedto(System.Windows.Navigation.NavigationEventArgs e)
{
    base.Onnavigatedto(e);

    // одержання посилання на сторінку з інформацією про поточного клієнта
    App thisapp = Application.Current as App;

    // установка контексту даних для відображення в елементі Grid
    customerdisplaygrid.DataContext = thisapp.Activecustomer;
}
```

Властивість Current класу Application є посиланням, по якому можна звернутися до об'єктів у класі Application. При запуску додатка їй привласнюється значення адреси поточної сторінки.

Клас App.xaml.cs визначає дочірній клас App, який розширює батьківський клас Application. Щоб одержати об'єкти, певні в класі App, потрібно одержати значення властивості Current класу Application і привести її до типу App. Наведений вище код запускається на сторінці customerdetailspage, коли до неї відбувається перехід. При цьому викликається об'єкт Activecustomer, який установлюється як контекст даних об'єкта customerdisplaygrid для звертання до обраного клієнта.

Усі об'єкти додатка, які повинні бути доступні іншим сторінкам, має сенс поміщати в клас App.xaml.cs. Тут також можна додати код, який при запуску програми буде завантажувати дані в ці об'єкти зі сховища.

6. Використання класів Viewmodel

На теперішній момент наша програма може відображати інформацію про клієнта, але не дозволяє редагувати цю інформацію. Користувач може змінити текст у полях сторінки, але зміни не будуть зберігатися в програмі.

Щоб внесені на сторінці зміни обновляли дані в списку клієнтів, можна використовувати прив'язку даних. При цьому, прив'язка тексту в елементі Textbox повинна бути двосторонньою, щоб при зміні даних на екрані додатка автоматично обновлялося пов'язане з елементом властивість:

```
<Textbox Name="nametextbox" Text="{Binding Name, Mode=TwoWay}" />
```

Після цього можна додати код до класу Customer класу для керування зв'язаними подіями. Цей код при зміні імені клієнта повинен змінюватися текст на екрані, і якщо текст елемента Textbox зміниться, то пов'язане з елементом також повинне змінюватися автоматично. Але можна зробити це небагато інакше, використавши модель, відому за назвою Model-view-viewmodel.

6.1. Створення класу Viewmodel

Можна додати прив'язку даних із класом Customer, але є деякі причини, чому це не є гарною ідеєю. Існує правило проектування програмного забезпечення: звичайно клас повинен виконувати одна дія, і тільки одне. Клас Customer повинен зберігати дані про клієнтів, але не повинен брати участь у процесі редагування. При зберіганні або передачі даних про клієнтів ми не повинні створювати методи, які стосуються редагування даних. При передачі об'єкта, що представляє клієнта, на іншій пристрій або при здійсненні прив'язки даних будь-яка додаткова властивість або метод буде зайвою витратою простору, тому що не буде використовуватися в цьому контексті.

Можливість перетворення й перевірки допустимості даних. В ідеалі, при введенні даних користувачем завжди повинна здійснюватися перевірка відповідності даних, що вводяться, певним правилам. Наприклад, повинне дотримуватися правило: "Ім'я повинне містити тільки букви й пробіли". Можна помістити ці правила в клас Customer, але буде набагато простіше вказати їх безпосередньо в процесі введення даних, тому що вони повинні використовуватися тільки при введенні даних. Також корисна можливість виконувати перетворення деяких вхідних значень, наприклад дати й часу, з текстового формату, у якому вони вводяться й виводяться, у формат, у якому значення зберігаються.

Можливість скасування дії. Практично завжди при проектуванні форми для редагування яких-небудь значень рекомендується передбачити можливість скасування зроблених користувачем, але ще не збережених змін. Звичайно в таких формах присутні кнопки зберегти й скасування. При використанні прив'язки даних це може виявитися проблемою, оскільки в цьому випадку об'єкт даних відразу змінюється. Щоб відстежити зміни, внесені користувачем, необхідно створити копію даних при відкритті форми, і при скасуванні змін використовувати її, що дуже незручно й вимагає створення додаткового коду.

Створення класу Viewmodel. Можна уникнути цих проблем, створивши клас Viewmodel. Цей клас призначений для вистави даних і зв'язування з компонентами форми й також може виконувати необхідні перевірки допустимості даних.

Створимо клас Viewmodel для вистави дій, які потрібно виконати в користувацькому інтерфейсі. Цей клас з'єднає користувацький інтерфейс із об'єктами даних, містить властивості, які будуть використовуватися в користувацькому інтерфейсі:

```
public class Customerview : InotifyPropertyChanged
{
    private string name;

    public string Name
    {
        get
        {
            return name;
        }
    }
}
```

```

    }
    set
    {
        name = value;

        if (PropertyChanged != null)
        {
            PropertyChanged(this, new PropertyChangedEventArgs("name"));
        }
    }
}

// властивість Address створюється в такий же спосіб

private int id;

public int ID
{
    get
    {
        return id;
    }
}

public event PropertyChangedEventHandler PropertyChanged;

public void Load(Customer cust)
{
    Name = cust.Name;
    Address = cust.Address;
    id = cust.ID;
}

public void Save(Customer cust)
{
    cust.Name = Name;
    cust.Address = Address;
}
}

```

Цей клас генерує події при зміні властивостей у програмі, і елементи Silverlight можуть установлювати нові значення властивостей у класі при їхній зміні користувачем. Зверніть увагу, що властивість ID доступно тільки для читання, і користувач не може змінити його значення. Це дозволяє забезпечити унікальність ідентифікатора клієнта.

Робота з даними в класі Viewmodel. Клас Customerview також містить методи Load і Save, призначені для взаємодії програми із умістом класу Viewmodel. Наступний код виконується перед початком редагування інформації про клієнта:

```

// створення екземпляра класу Viewmodel для сторінки редагування
Customerview view = new Customerview();

protected override void
Onnavigatedto(System.Windows.Navigation.NavigationEventArgs e)
{
    // одержати посилання на сторінку, що містить інформацію про активного
    клієнта
}

```

```

App thisapp = Application.Current as App;

// завантажити об'єкт із інформацією про активного клієнта в клас Viewmodel
view.Load(thisapp.Activecustomer);

// установити контекст даних для відображення
customerdisplaygrid.DataContext = view;
}

```

Коли програма переходить на сторінку для редагування інформації про клієнта, Silverlight викликає метод `Onnavigatedto` для налаштування сторінки. Цей метод одержує інформацію про обраного клієнта й завантажує її в екземпляр класу `Customerview`.

Коли користувач натискає кнопку зберегти, програма повинна вивантажити вміст класу `Viewmodel` у властивість, що редагується, `Customer`:

```

private void savebutton_Click(object sender, RoutedEventArgs e)
{
    // одержати посилання на сторінку, що містить інформацію про активного
    клієнта
    App thisapp = Application.Current as App;

    // скопіювати дані із класу Viewmodel у властивість Activecustomer
    view.Save(thisapp.Activecustomer);

    // повернутися на попередню сторінку
    NavigationService.GoBack();
}

```

6.2. Навігація по сторінках з використанням методу `GoBack`

Клас `NavigationService` містить метод `GoBack`, який переносить користувача до попередньої сторінки. Це ефективніше, чим використання `URI`, тому що при цьому сторінка не оновлюється. Якщо перейти на сторінку, використовуючи метод `Navigate` класу `NavigationService`, буде створена нова сторінка, і вона буде створюватися щораз при переході на неї. Це може сповільнити роботу додатка й привести до створення й звільненню великої кількості елементів Silverlight. Крім того, незручно при кожному переході на сторінку зі списком прокручувати її вниз, оскільки в цьому випадку при переході на сторінку буде відображатися початок списку.

Однак, використання методу `GoBack` для переходу до попередньої сторінки приводить до виникнення деяких проблем. Труднощі полягає в тому, що попередня сторінка не буде містити зміни даних, які внесе користувач. Наприклад, користувач виконує наступні дії:

1. Вибір клієнта зі списку.
2. Зміна імені клієнта.
3. Натискання кнопки Зберегти для виконання змін.
4. Повернення до попередньої сторінки.

Після виконання цих дій у списку будуть утримуватися первісні значення без обліку зроблених користувачем змін. Проблема полягає в тому, що елемент

Listbox не знає, що вміст однієї із записів у списку змінилося, оскільки при виводі списку на екран створюється його копія, і при зміні даних ця копія обновлятися не буде. Рішення цієї проблеми засноване на використанні спостережуваних колекцій Observablecollection.

6.1. Спостережувані колекції

Клас Observablecollection використовується для зберігання колекції елементів і надає підтримку повідомлень так, щоб при зміні колекції елемент Listbox можна було зв'язати з подією зміни. Для створення спостережуваної колекції можна використовувати наявний список:

```
Observablecollection<Customer> observablecustomers;

private void Phoneapplicationpage_Loaded(object sender, RoutedEventArgs e)
{
    // одержати посилання на сторінку, що містить список клієнтів
    App thisapp = Application.Current as App;

    // створити спостережувану колекцію цих клієнтів
    observablecustomers = new Observablecollection<Customer>(
        thisapp.Activecustomerlist.Customerlist);

    // відображення елементів спостережуваної колекції
    customerlist.Itemssource = observablecustomers;
}
```

Цей код виконується при запуску головної сторінки додатка. Він створює новий список observablecustomers на основі наявного списку клієнтів і встановлює цей список як джерело даних елемента customerlist. Тепер, при зміні значення елементів у списку ці зміни будуть відображатися на екрані.

Однак, якщо користувач обновить ім'я клієнта, ця зміна не буде відбита на екрані, тому що Observablecollection реагує на зміну вмісту списку, але не на зміну даних в елементі списку. Це є дуже серйозним недоліком, що приводять до неправильної роботи програми.

Щоб виправити цю помилку, потрібно якось внести зміни в список, щоб він обновлявся на екрані. Один зі способів викликати відновлення полягає в тому, щоб вилучити елемент зі списку й знову його додати. Зміни вмісту спостережуваної колекції генерують події, які обробляє елемент Listbox, який виводить список на екран.

```
protected override void
Onnavigatedto(System.Windows.Navigation.NavigationEventArgs e)
{
    // одержати посилання на сторінку, що містить список клієнтів
    App thisapp = Application.Current as App;

    if (thisapp.Activecustomer != null)
    {
        // знайти потрібний клієнта в списку
        int pos = observablecustomers.IndexOf(thisapp.Activecustomer);
        // вилучити клієнта
        observablecustomers.RemoveAt(pos);
    }
}
```

```

        // повернути клієнта назад
        observablecustomers.Insert(pos, thisapp.Activecustomer);
    }
}

```

Цей код виконується, коли користувач переходить назад до головної сторінки. Якщо обраний який-небудь клієнт, метод видаляє цього клієнта зі списку й потім додає його на те ж саме місце. Це викликає зміна списку, і користувач буде бачити зроблені зміни. При відновленні всього одного елемента відновлення буде виконуватися досить швидко. Це особливо важливо, якщо список містить кілька тисяч елементів.

При використанні класу Viewmodel для редагування інформації про клієнта ми створювали методи для завантаження й збереження інформації, які зтягали інформацію про клієнта в клас Customerview і потім повертали оновлені значення. При використанні класу ObservableCollection необхідно зробити щось схоже. На щастя, існує багато способів одержання списку даних із класу ObservableCollection. Наприклад, можна використовувати розширювальний метод Tolist:

```

thisapp.Activecustomerlist.Customerlist =
observablecustomers.Tolist<Customer>();

```

Використання спостережуваних колекцій не приводять до підвищення витрати пам'яті, оскільки містять набір посилань на вже існуючі об'єкти. Відповідно, потрібно лише невеликий обсяг пам'яті для зберігання додаткових посилань на об'єкти. Однак спостережувані колекції не має змісту використовувати, якщо програма тільки виводить список на екран, але не змінює його.

7. Короткі підсумки

1. Програми можуть змінювати значення властивостей елементів Silverlight для керування відображенням елементів на екрані. До властивостей належать положення елемента на екрані і його колір.
2. Більшість властивостей елементів зручніше задавати, редагуючи прямо ХамлКод. Хамл-опис елемента має певну структуру, у якій одні властивості можуть вкладатися в інші.
3. В елементі Textbox є набір властивостей, які задають тип клавіатури, яка буде використовуватися для введення даних. Можна використовувати для введення спеціальну клавіатуру для введення чисел замість клавіатури для введення довільного тексту.
4. Windows Phone може виводити на екран вікно з повідомленням для користувача. Текст повідомлення може складатися з декількох рядків. Повідомлення можуть також використовуватися для підтвердження або скасування дій користувача.

5. Ресурси можуть бути додані в додаток для Windows Phone як частина вмісту або як впроваджені ресурси. Елемент вмісту копіюється в каталог додатка в окремий файл, який може використовуватися програмою. Впроваджений ресурс вбудовується у файл складання додатка. Елементи вмісту не сповільнюють завантаження складання, але можуть повільніше завантажуватися під час роботи програми. Впроваджені ресурси завантажуються швидше, але збільшують розмір складання програми.
6. Елементи Silverlight можуть генерувати події у відповідь на дії користувача. Наприклад, подія Textchanged елемента Textbox відбувається при зміні тексту елемента.
7. Silverlight надає прив'язку даних, за допомогою якої властивості програмного об'єкта можуть бути з'єднані із властивостями візуального елемента Silverlight. Прив'язка може бути односпрямованою, коли візуальний елемент використовується тільки для виводу значення об'єкта програми, або двунаправленою, коли зміна властивості візуального елемента приводить до відновлення зв'язаного властивості класу.
8. Можна зв'язати колекцію елементів з елементом Listbox для відображення списку елементів. Шаблон даних використовується для опису того, як окремі властивості кожного елемента повинні виводитися на екран.
9. Додатка Silverlight можуть складатися з декількох сторінок. Навігація по сторінках виконується в класі Navigate, який використовує URI сторінки. Прості строкові значення можна передавати між сторінками, розмістивши їх у рядку запиту, приєднаної до URI.
10. Сторінки можуть одержувати події при переміщенні до них і від них. Подія Onnavigatedfrom надає можливість скасування переходу на іншу сторінку, якщо потрібно щоб користувач підтвердив перехід.
11. Більші об'єкти даних можуть спільно використовувати кілька сторінок додатка за допомогою класу App, який є частиною додатка Windows Phone. Будь-яка сторінка додатка Silverlight може одержати посилання на об'єкт додатка, який є частиною цього додатка.
12. Програмісти можуть створити класи Viewmodel, які зв'язують, що редагуються дані з елементами на сторінці Silverlight.
13. Механізм Observablecollection дозволяє відбити на екрані зміни в колекції елементів.

Лабораторна робота №3

Робота з GPS навігатором

Мета роботи:

- ознайомитися з можливостями Gps навігатора;
- навчитися працювати та знаходити координати місця знаходження.

5.2 Основні теоретичні відомості

Навігатор - це пристрій, який дозволяє дізнатися географічні координати будь-якої крапки, місце розташування приладу, траєкторію проходження шляху й т.д.

GPS навігатор - це приймач і комп'ютер в одному корпусі. Приймач приймає сигнали, що передаються супутниками знаходяться на орбіті, а комп'ютер розшифровує сигнал і визначає місце розташування приймача. (Рис.1)



Рисунок 1. - Навігатор GPS

Абсолютно всі моделі відображають на екрані поточне становище, географічні координати крапки, у якій знаходиться прилад, траєкторію пройденого шляху й відмічені крапки. Усі прилади мають кілька сторінок, що відображають різну інформацію: Положення супутників на небосхилі, карту з крапками й пройденими шляхами, сторінку меню з виходом на різні налаштування й пошук, сторінку навігації, де в режимі навігації (прямування до певної крапки) зображений покажчик у вигляді стрілки й сторіночку колійного комп'ютера, де відображаються пройдена відстань, швидкість руху й т.д.

Шлях (трек) - це "слід", пройдений Вами шлях. У пам'ять приладу записується за замовчуванням (заводські налаштування). Алі можна

відключити, якщо треба. Маршрут - (Роут) - це шлях, заздалегідь намічений по крапках. Прилад може провести Вас у режимі навігації як по маршруту, так і по трекові (у режимі трек бек). Маршрут можна побудувати на комп'ютері, потім увести в прилад. Можна побудувати й безпосередньо в приладі.

Місце знаходження

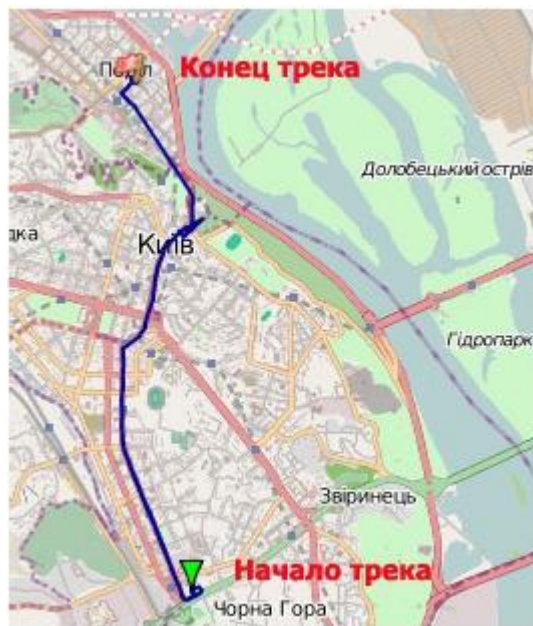
Щоб звернути місце положення для цього достатньо натиснути й деякий час утримувати кнопку. Так саме це можна зробити через головне меню. (Рис. 2)



Малюнок 2. – Координати місця знаходження

Досить вибрати потрібну крапку зі списку й натиснути кнопку "Йти". На сторінці «навігація» з'явиться стрілка з напрямом руху.

А якщо ви знаєте місце призначення, те слід відмітити його на карті.(Рис.3)



Малюнок 3 - Шлях від пункту відправки до місця призначення

Функція автороутінг дозволяє автоматично прокладати маршрут. Для цього потрібно лише поставити крапку закінчення шляху. Шлях від вашого теперішнього місця розташування до кінцевої крапки прилад визначить сам. При цьому протягом проходження по маршруту буде подавати сигнали про відхилення.

За допомогою спеціального програмного забезпечення можна завантажити в навігатора карти, маршрути, треки й т.д. При цьому слід пам'ятати, що у всіх моделях вже предумановленна карта світу, алі з різною деталізацією для різних областей, на якій відображені основні міста, дороги, ріки й т.д.

5.3 Завдання до роботи

5.3.1 Ознайомитися з літературою та основними теоретичними відомостями за темою роботи.

5.3.2 Визначити своє місцезнаходження за допомогою GPS-пристрою.

5.3.3 Визначити відстань між своїм місцезнаходженням та двома найближчими об'єктами на карті.

5.3.4 Оформити звіт з роботи.

5.3.5 Відповісти на контрольні питання.

5.4 Зміст звіту

5.4.1 Тема та позначка роботи.

5.4.2 Короткі теоретичні відомості.

5.4.3 Результати виконання завдання..

5.4.4 Висновки, що містять відповіді на контрольні запитання, а також відображують результати виконання роботи.

5.5 Контрольні запитання

5.5.1 Що таке GPS технологія?

5.5.2 Типи та види GPS пристроїв.

5.5.3 Точність визначення координат. Фактори, що впливають на визначення точності координат та відстаней.

5.5.4 Які умови правильного користування GPS пристроями?

ЛІТЕРАТУРА

1. Шеховцов В.А. Операційні системи. – К: Видавнича група ВНУ, 2005. – 576с.
2. В. Столлингс. Операционные системы. Внутреннее устройство и принципы проектирования. М. Вільямс. 2002.
3. Басс Л., Клементс П., Кацман Р. «Архитектура программного обеспечения на практике», СПб: "Питер", 2005 г, 576 с.
4. Соммервилл Иан «Инженерия программного обеспечения. Изд.6»; М: Диалектика-Вильямс, 2002, 624с.
5. Шоу А. Логическое проектирование операционных систем. Пер. с англ. - М.: Мир, 1981, 256 с.
6. К.Стинсон, К. Зихерт. Windows 200 Professional. Питер. 2003.
7. Основы геоинформатики: Учеб. пособие для студ. Вузов. В 2 кн. Кн. 1 / Е.Г. Капралов, А.В. Кошкарев, В.С. Тикунов и др.; Под ред. В.С. Тикунова. – М.: Издательский центр «Академия», 2004. – 352 с.
8. 2. Цветков. В.Я. Геоинформационные системы и технологии. – М.: Финансы и статистика, 1998. - 288 с. (2 экз., б-ка ИКГ).