Тема 15

<u>Изпитна тема:</u> Приложения с графичен потребителски интерфейс

1. Дефинира понятията: интегрирана среда за разработка (IDE), графичен интерфейс, събитие, обработка, източник. Различава основни контроли на графичния интерфейс и посочва техните свойства.

Средата за програмиране(Integrated Development Environment - **IDE**, интегрирана среда за разработка) е съвкупност от традиционни и нструменти за разработване на софтуерни приложения.

В средата за разработка пишем код, компилираме и изпълняваме програмите.

Средите за разработка интегрират в себе си **текстов редактор** за писане на кода, **език за програмиране**, **компилатор или интерпретатор** и **среда за изпълнение** за изпълнение на програмите, **дебъгер** за проследяване на програмата и търсене на грешки, **инструменти за дизайн на потребителски интерфейс** и други инструменти и добавки.

Графичен интерфейс е разновидност на потребителски интерфейс, в който елементите, предоставени на потребителя за управление, са изпълнени във вид на графични изображения (менюта, бутони, списъци и др.).

Събитие – действие, което протича в реално време и като отговор предизвиква едни или други действия, които се извършват като отговор на дадено събитие.

- събития свързани с клавиатурата
- събития свързани с мишката
- програмни събития

Събитията възникват при действие от страна на потребителя или системата.

Обработчик на събитие – част от програмния код, която съответства на конкретно събитие, наричаме обработчик на събитие (манипулатор на събитие). Като правило всички обекти могат да реагират на събития – едни на повече, други на по-малко.

Източник на събитие – (генератор) на едно събитие може да бъде:

- извършване на определено действие от страна на лицето, което използва (изпълнява) програмата – натискане на клавиш, придвижване или щракване на мишката и др.
- ОС – оповестяване, че текущият сеанс на работа с компютъра завършва и др.
- езиковия процесор – изтичане на зададен период от време.
- програмният код – изпълнение на специфични оператори или прилагане на определени методи към даден екземпляр на обект.

Контроли на графичния интерфейс - Контролите са обекти, които дават възможност за визуализиране на информацията на екрана и за взаимодействието с приложението чрез мишка, клавиатури др. Базов клас за всички контроли е класът System.Windows.Forms.Control

Общи свойства за всички компоненти:

Name – уникално име на компонентата

- клас Button – компоненти button1, button2

Visible – видимост на компонентата - True, False

Enabled – определя достъпността на компонентата -True, False

Size – размер в пиксели

- ширина – width; височина – height;

Location – координати X и Y в пиксели

Екранна форма от клас Form – свойства

StartPosition - място на формата върху екрана

Text – заглавие на прозореца от тип string

BackColor – цвят на вътрешността на формата

ForeColor – цвят за изписване на текстове във всички обекти

ControlBox – активира бутоните за затваряне, минимизиране и оразмеряване

Етикет – клас Label

Предназначение – поставяне на надписи

Основното предназначение на компонентите от класа Label е да се поставят надписи в основния прозорец на програмата и останалите контейнери. В етикета се поставят различни заглавия на прозорците и пояснящи надписи за предназначението на осталите компоненти. В етикета могат да се извеждат и стойности, които трябва само да се покажат в прозореца, но не бива да се променят от потребителя – например, резултати от извършените от програмата пресмятания. Текстът, който ще се изпише в етикета се задава в свойството Text. При поставяне на етикет в екранната форма Text = label1. Исканият от нас текст се въвежда в полето за редактиране на свойството. Друго важно свойство е Font – съставно свойство, в което се задават параметрите на използвания шрифт, с много подсвойства. Най-важните от тях са Name и Size, задаващи вида и размера на шрифта, както и определящите стила - Bold, Italic и Underline.

Свойства – Text, Font, Name, Size

Текстова кутия – клас TextBox

Компонентите на класа TextBox (текстова кутия) са предназначени за въвеждане на данни от клавиатурата по време на изпълнение на програмата. Това е и основното различие между текстовата кутия и етикета. Останалите свойства са почти идентични. Компонентата позволява да се въвеждат данни на един или на много редове, което се управлява от свойството Multiline. След като потребителят е въвел данните в текстовата кутия, те стават достъпни в програмата като съдържание на свойството Text. Като съдържание на това свойство по време на проектиране на формата или програмно може да се постави текст, който ще се покаже при отваряне на прозореца и може да бъде съдържание на кутията по премълчаване или подсказка за потребителя какво да въведе в полето. Компонентите, в които потребителят може да въвежда данни, се наричат активни. Важно събитие за текстовата кутия е TextChanged (текстът е променен). Когато това събитие се случи, програмата би трябвало да го обработи, като съхрани въведеното в кутията в съответна променлива.

Предназначена за въвеждане на данни от клавиатурата

Свойства

- Multiline – въвеждане на повече от един ред
- Text – приема въведените стойности или визуализиран текст при зареждане на формата
- TextChanged – активна компонента

Бутон – клас Button

Компонентите на класа Button са предназначени за подаване на команди от страна на потребителя към изпълняваната програма. Основно свойство на командния бутон е надписът му, което е стойност на свойството Text и е добре да подсказва командата, която ще се стартира, когато потребителят щракне върху него с мишката. За тази компонента е характерно събитието Click, което се генерира при натискане на бутона с левия бутон на мишката. Средата автоматично генерира тяло на метода, с който това събитие да бъде обработено, когато щракнем с бутона. В тази функция програмистът изписва програмния код, който изпълнява свързаната с бутона команда.

Предназначение - за подаване на команди; командни бутони

Свойства-Text(задава текст върху бутона), Click(активира се при натискане)

Всички контроли от Windows Forms дефинират събития, които програмистът може да прихваща. Например контролата Button дефинира събитието Click, което се активира при натискане на бутона. Събитията в Windows Forms управляват взаимодействието между програмата и контролите и между самите контроли помежду им.

2. Дава пример за конструкции за контрол на изпълнението. Демонстрира употребата на конструкции за прихващане и обработка на изключения.

Конструкции за контрол на изпълнението - if / else , вложени if конструкции, **switch-case**

```
if (булев израз)
{
 тяло на условната конструкция;
}
else
{
 тяло на else-конструкция;
}
```

Тук тази конструкция работи по следния начин: в зависимост от резултата на израза в скобите (булевият израз) са възможни два пътя по, които да продължи потока от изчисленията. Ако булевият израз е true, се изпълнява тялото на условната конструкция, а else се пропуска като операторите в него не се изпълняват. В обратния случай се изпълнява else-конструкцията, а се пропуска основното тяло и операторите в него не се изпълняват.

```
switch (селектор)
{
 case стойност-1: код за изпълнение; break;
 case стойност-2: код за изпълнение; break;
 case стойност-3: код за изпълнение; break;
 case стойност-4: код за изпълнение; break;
 // ...
 default: код за изпълнение; break;
}
```

Конструкцията switch-case избира измежду части от програмен код на базата на изчислената стойност на определен израз. Този израз най-често е целочислен, но може да бъде и от тип string или char. Стойността на селектора трябва задължително да бъде изчислена преди да се сравнява със стойностите вътре в switch конструкцията. Етикетите (case) не трябва да имат една и съща стойност. При намиране на съвпадение на селектора с някоя от case стойностите, switch-case конструкцията изпълнява кода след съответния case. При липса на съвпадение, се изпълнява default конструкцията, когато такава съществува. Всеки case етикет, както и етикетът по подразбиране (default), трябва да завършват с ключовата дума break, която приключва работата на

switch-case конструкцията, след като е намерено съвпадение и е изпълнен съответния код.

Пример:

```
x = int.Parse(textBox1Number1.Text);
y = int.Parse(textBox2Number2.Text);
switch (label1Sight.Text) {
case "+": rezultat = x + y; break;
case "-": rezultat = x - y; break;
case "*": rezultat = x * y; break;
case "/": rezultat = x / y; break;
default: { labelBug.Font = new Font(labelBug.Font, FontStyle.Bold);
MessageBox.Show(labelBug.Text); break; }
 } textBox3Sum.Text = rezultat.ToString();
```

В езика C# с оператора try … catch … finally може да се обработват изключения (exception), т.е. ситуации, които ако не бъдат обработени, ще предизвикат аварийно спиране на програмата. Синтаксис:

try { < оператори, при изпълнение на които се очаква изключение > }
catch (< вид на изключението >)
{ < оператори, които ще се изпълнят при възникне на изключение > }
finally { < оператори, които ще се изпълнят винаги > }
Когато програмата достигне този оператор, се опитва да изпълни операторите в блока try. Ако изключението се случи, се изпълнява кодът след catch (хващам, улавям), а след това кодът след finally (накрая). Ако очакваното изключение не се случи, се изпълнява само кодът след finally. Частите catch и finally не са задължителни, може да липсват.

Пример:

```
try {
    a = int.Parse(textBoxA.Text);
 }
catch (FormatException)
{
   a = 0;
   textBoxA.Text = “0”;
 }
```

3. Демонстрира работа с класове, обекти и свързаните с тях събития.

- Библиотеката Windows Forms дефинира:
  - съвкупност от базови класове за контролите и контейнер-контролите
  - множество графични контроли
- Основни базови класове:
  - Component – .NET компонент

- Control – графична контрола (компонента с графичен образ)
- ScrollableControl – контрола, която поддържа скролиране на съдържанието си
- ContainerControl – контрола, която съдържа други контроли и управлява поведението на фокуса

- Класът System.Windows.Forms.Control е основа на всички графични Windows Forms контроли
- Неговите свойства са типични за всички Windows Forms контроли
- По-важните свойства на класа Control:
  - Anchor, Dock – задават по какъв начин контролата се "закотвя" за контейнера си
  - Bounds – задава размера и позицията на контролата в нейния контейнер
  - BackColor – задава цвета на фона
  - ContextMenu – задава контекстно меню (popup menu) за контролата
- По-важните събития на класа Control:
  - Click – настъпва при щракване с мишката върху контролата
  - Enter, Leave – настъпват при активиране и деактивиране на контролата
  - KeyDown, KeyUp – настъпват при натискане и отпускане на клавиш (или комбинация)
  - KeyPress – при натискане на нефункционален клавиш
  - MouseDown, MouseUp, MouseHover, MouseEnter, MouseLeave, MouseMove, MouseWheel – настъпват при събития от мишката, настъпили върху контролата

Поставяне на контроли:

Поставянето на контроли във формата става чрез Controls.Add:

```
Form form = new Form();
Button button = new Button();
button.Text = "Close";
form.Controls.Add(button);
```

Управление на събитията:

Прихващането на събития става така:

```
Form form = new Form();
Button button = new Button();
button.Click += new EventHandler(
   this.button_Click);
...
private void button_Click(
```

```
   object sender, EventArgs e)
{
   // Handle the "click" event
}
```

4. Обяснява моделите бази данни. Разработва модел бази данни, така че да реши поставената задача.

- Модели на базите от данни
  - йерархичен (дървовиден)
  - мрежови
  - релационен (табличен)
  - обектно-релационен
- Релационните бази от данни
  - Представляват съвкупности от таблици и връзки между тях (релации)
  - Ползват здрава математическа основа: релационната алгебра
- RDBMS(**Relational Database Management System**) системите се наричат още
  - сървъри за управление на бази от данни
  - или просто "Database сървъри"

ADO.NET - Набор от класове за работа с данни

- Набор от класове, интерфейси, структури и други типове за достъп до данни през изцяло .NET базирана реализация
- Програмен модел за работа с данни
- Осигурява възможност за работа в несвързана среда
- Осигурява връзка с XML
- Наследник на ADO (Windows технология за достъп до бази от данни)

Data Provider-и в ADO.NET

Data Provider-ите са съвкупности от класове, които осигуряват връзка с различни бази от данни

За различните RDBMS системи се използват различни Data Provider-и

- Различните производители използват различни протоколи за връзка със сървърите за данни

Дефинират се от 4 основни обекта:

- Connection – за връзка с базата
- Command – за изпълнение на SQL
- DataReader – за извличане на данни

- DataAdapter – за връзка с DataSet. Представлява набор от SQL команди и връзка с база данни, които се използват за попълване на DataSet и актуализиране на източника на данни.

Данните се кешират в DataSet обект и връзката се преустановява

- Отваряне на връзка (SqlConnection)
- Пълнене на DataSet (чрез SqlDataAdapter)
- Затваряне на връзката
- Работа със DataSet-а
- Отваряне на връзка
- Нанасяне на промени по данните по сървъра
- Затваряне на връзката

SqlClient Data Provider

- SqlConnection – осъществява връзката с MS SQL Server
- SqlCommand – изпълнява команди върху SQL Server-а през вече установена връзка
- SqlDataReader – служи за извличане на данни от SQL Server-а
    - Данните са резултат от изпълнена команда
- SqlDataAdapter – обменя данни между DataSet обекти и SQL Server
    - Осигурява зареждане на DataSet с данни и обновяване на променени данни
    - Може да се грижи сам за състоянието на връзката с базата данни

```
const string CONNECTION_STRING =
   "Server=localhost; Database=Northwind; " +
   "Integrated Security=true; " +
   "Persist Security Info=false";
// Create the connection
SqlConnection con =
   new SqlConnection(CONNECTION_STRING);
using (con)
{
   // Open connection
   con.Open();
   // Use the connection here
   // ...
}
```

5. Посочва и различава видовете мобилни операционни системи и съответните платформи за създаване на мобилни приложения. Определя възможността за разработване на приложение според възможностите на мобилното устройство.

Мобилни операционни системи (МОС)

- Android е операционна система за мобилни устройства поддържана от Google. Операционната система поддържа различни хардуерни платформи, вкл. телефони, таблети, модни аксесоари (Android Wear), Телевизори (Android TV), автомобилни инфотеймънт системи (Android Auto) и др.
- iOS е мобилна операционна система на компанията Apple Inc. Платформата предлага високо ниво на защита, както на личните данни, така и на самата система, включително и чрез достъп с пръстов отпечатък. Налична и версия за използване в автомобили наречена Apple CarPlay. Платформата не е с отворен код, което ограничава до известна степен разпространението и само до устройства разработка на Apple.
- Windows Phone/Mobile е мобилна операционна система на Microsoft. Тя също не е с отворен код, но е достъпна на хардуерни устройства от множество производители на преносими устройства, като HTC, Samsung, Toshiba, Nokia и др.

Мобилните устройства днес са с все по-голяма изчислителна мощ, в тях са ингерирани различни сензори и с хардуерни параметри почти достигащи настолните компютри. Разширява се и множеството на мобилните платформи, които управляват различните категории мобилни устройства. В тази връзка при разработката на мобилни приложения ясно се очертават тенденции като:

- уеднаквяване на средствата за разработка;
- подход на разработка на „универсални“ приложения;
- разработка на приложения за сега набиращите популярност мобилни устройства от категориите на т.н. „умни дрехи“, автомобилни системи, потребителска електроника и др.

Колкото повече расте броят на мобилните устройства, толкова по-често възниква въпросът какъв подход да изберат разработчиците, за да осигурят определена информационна услуга на мобилни устройства.

Алтернативите са:

- да се разработи уеб приложение, което потребителят да използва чрез уеб браузъра;
- да се разработи т.н. „native“ („нейтив“) приложение;
- разработка на хибридно приложение.

Правилният избор се обуславя от няколко фактора/цели:

- Ако основните цели са да се осигури удобен потребителски интерфейс, висока скорост на работа, използване на геолокация, микрофон или камера, взаимодействие с други приложения на мобилното устройство и др., то изборът трябва да падне върху „нейтив“ приложение.

- И ако се правят чести промени в интерфейса и съдържанието, като се търси ниска цена и кратко време за разработка, ако се изисква поддръжка на много мобилни платформи, малко място за съхранение и лесна актуализация, по-удачно е да се избере мобилно уеб приложение.

Подходи за разработка на мобилни приложения

Мобилен Уеб и RWD

Мобилните уеб приложения не са типични приложения за мобилни устройства. Те представляват уеб приложения, които изглеждат (интерфейс) и функционират като „нейтив“ приложения.

- Те се управляват от браузъра на устройството и обикновено използват HTML5, CSS и JavaScript.
- При разработка на уеб приложения тенденциите показват налагане на два подхода.
- За разработчиците е много по-лесно да създават уеб приложения, които да са достъпни от всякакви устройства или да използват т.н. „Responsive web design“ (RWD) подход за тази цел.
- При мобилни приложения това означава, че визуализацията изцяло зависи преди всичко от големината на дисплея, а съдържанието и функционалността са константни. Използват се работни рамки, като Bootstrap, Zurb Foundation и Skeleton. Известен недостатък е еднотипният изглед на приложенията.

Mobile First

- При този подход при дизайна и разработката се обръща внимание първо на това как да се осигури достъп от мобилни устройства и едва след това, възможно е и на доста по-късен етап, приложението се надгражда и за десктоп системите.
- Този подход е подходящ, ако компанията, която предлага продукта или услугата, е ориентирана именно към динамични, търсещи бързината и с афинитет към иновациите клиенти в области като е-медии, напр. новини, интернет търговията, маркетинга и забавленията.
- Недостатък е необходимостта да се осигури достъп до услугите от множество браузъри и множество различни устройства(различни по големина екрани), което оскъпява разработката.

Хибридни приложения

Хибридните приложения по нищо, което касае функционалността, не се отличават от другите два вида мобилни приложения.

- Инсталират се на мобилното устройство чрез магазините за приложения и с тяхна помощ може да се комуникира в социалните медии, да се играят игри, да се снима и др.
- Разликата с другите две категории се изразява в архитектурата на приложението.

- По същество те са като всички уеб мобилни приложения, създадени със средства като HTML, CSS и Java Script, но използват като хост „нейтив“ приложение.
- Тази особеност позволява на хибридните приложения да имат достъп до всички хардуерни компоненти на устройството (камера, акселерометър, GPS и т.н.), които обикновено са с ограничен достъп от уеб браузъра.

За разработка на уеб приложения се използват стандартни средства като HTML, CSS и JavaScript. Докато за „**нейтив**“ **Андроид** приложения се изисква познания и програмиране на Java, то за **iOs** се изисква Swift или Objective-C, за **Windows Phone** - C++, C# а за Blackberry - Cascades (C++).

Едно от решенията да не се разработва едно приложение на много платформи с различни програмни средства е разработката на **универсални приложения**:

Microsoft представи новата си платформа Windows 10. Тя е достъпна за използване на различни устройства – телефони, таблети, различни електронни устройства и десктоп системи. С нея компанията се стреми да наложи концепцията за т.н. Универсални приложения -Universal Windows Platform (UWP). В основата си концепцията за универсалните приложения е много проста: писане на програмен код само веднъж, и асоцииране с потребителския интерфейс подходящ за всички устройства, работещи под Windows. За разработка могат да се използват: HTML, CSS и JavaScript, XAML и C#, XAML, DirectX и C++.

6. Познава начините за работа с текст и изображения в приложение за мобилно устройство.

- Използване на Xamarin.Forms за въвеждане или показване на **текст**.

Xamarin.Forms има три основни изгледа за работа с текст:

- **Label** - за представяне на единичен или многоредов текст. Може да показва текст с множество опции за форматиране в един и същи ред.
- **Entry** - за въвеждане на текст, който е само един ред. Влизането има режим на парола.
- **Editor** - за въвеждане на текст, който може да отнеме повече от един ред.

Пример за декорация на текст:

XAML КОД:

```
<Label Text="This is underlined text." TextDecorations="Underline"  />
<Label Text="This is text with strikethrough." TextDecorations="Strikethrough" />
<Label Text="This is underlined text with strikethrough." TextDecorations="Underline, Strikethrough" />
```

Еквивалентен код на c#

var underlineLabel = new Label { Text = "This is underlined text.", TextDecorations = TextDecorations.Underline };

var strikethroughLabel = new Label { Text = "This is text with strikethrough.", TextDecorations = TextDecorations.Strikethrough };

var bothLabel = new Label { Text = "This is underlined text with strikethrough.", TextDecorations = TextDecorations.Underline | TextDecorations.Strikethrough };

Други свойства:

TextTransform - трансформира в главни и малки букви текст, съхранен в свойството Text.

CharacterSpacing - знаците в текста, показан от етикета, са разделени с независима от устройството единица.

TextColor – цвят на текста

FontSize – размер на текста

```
<Label TextColor="#77d065" FontSize = "20" Text="This is a green label." />
```

Еквивалентен код на c#

```
var label = new Label { Text="This is a green label.", TextColor = Color.FromHex("#77d065"), FontSize = 20 };
```

Formatted text - позволява представянето на текст с множество шрифтове и цветове в един и същ изглед.

- Използване на Xamarin.Forms за работа с **изображения**.

Изображенията са решаваща част от навигацията на приложенията. Приложенията на Xamarin.Forms трябва да могат да споделят изображения на всички платформи, но също така и да показват различни изображения на всяка платформа.

Xamarin.Forms използва View за показване на изображения на страница. Той има няколко важни свойства:

Source - Екземпляр на ImageSource, като File, Uri или ресурс, който настройва изображението за показване.

ImageSource екземпляри могат да бъдат получени с помощта на статични методи за всеки тип източник на изображение:

- FromFile - Изисква име на файл или файлов път, който може да бъде разрешен на всяка платформа.
- FromUri - Изисква Uri обект, например **new Uri("http://server.com/image.jpg")**

- FromResource - Изисква идентификатор на ресурс към файл с изображение, вграден в приложението или .NET Standard Library, с **Build Action:EmbeddedResource**.
- FromStream - Изисква поток, който предоставя данни за изображения.

Aspect – оразмеряване на изображението в границите, в които се показва (независимо дали да го разтягате или изрязвате).

- Fill - Разтяга изображението, за да запълни напълно и точно областта на дисплея. Това обаче може да доведе до изкривяване на изображението.
- AspectFill - Изрязва изображението, така че да запълни областта на дисплея, като същевременно запазва аспекта (т.е. без изкривяване).
- AspectFit - Поставя в кутия изображението (ако е необходимо), така че цялото изображение да се побере в областта на дисплея, като се добавя празно пространство в горната/долната част или отстрани в зависимост от широчината или височината на изображението.

Изображенията могат да се зареждат от локален файл, вграден ресурс, да се изтеглят или зареждат от поток.

Пример с локален файл:

```
<Image Source="waterfront.jpg" />
```

Еквивалентен код на c#

```
var image = new Image { Source = "waterfront.jpg" };
```

7. Дава пример за създаване на различни екрани в мобилно приложение и връзката между тях.

Няколко термина и концепции са важни за разбиране, за да се поддържат множество екрани.

- **Размер на екрана** – количеството физическо пространство за показване на вашето приложение

- **Плътност на екрана** – Броят на пикселите във всяка дадена област на екрана. Типичната мерна единица е точки на инч (dpi).

- **Резолюция** – общият брой пиксели на екрана. При разработването на приложения разделителната способност не е толкова важна, колкото размерът и плътността на екрана.

- **Независим от плътността пиксел (dp)** – Виртуална мерна единица, позволяваща проектиране на оформления независимо от плътността. Тази формула се използва за преобразуване на dp в екранни пиксели:

px = dp × dpi ÷ 160

- **Ориентация** – Ориентацията на екрана се счита за пейзажна, когато е по-широк, отколкото е висок. За разлика от това, портретната ориентация е, когато екранът е по-висок, отколкото е широк. Ориентацията може да се промени по време на живота на приложението, докато потребителят завърта устройството.

Увеличаването на разделителната способност без увеличаване на плътността ще увеличи размера на екрана. Ако обаче плътността и разделителната способност се увеличат, тогава размерът на екрана може да остане непроменен. Тази връзка между размера на екрана, плътността и разделителната способност бързо усложнява поддръжката на екрана.

За да помогне за справяне с тази сложност, рамката на **Android** предпочита да използва *независими от плътността пиксели (dp)* за оформления на екрана. Чрез използване на пиксели, независими от плътността, елементите на потребителския интерфейс ще изглеждат на потребителя с еднакъв физически размер на екрани с различна плътност.

Android използва мащабиране на чертежите по време на изпълнение до подходящия размер. Възможно е обаче мащабирането да причини замъгляване на растерните изображения.

Решение на проблема при Android

За да заобиколите този проблем, осигурете **алтернативни ресурси за различните плътности**. Когато проектирате устройства за множество резолюции и плътност на екрана, ще се окаже по-лесно да започнете с изображения с по-висока разделителна способност или плътност и след това да намалите.

Декларирането на размера на екрана гарантира, че само поддържани устройства могат да изтеглят приложението. Това се постига чрез задаване на елемента **supports-screens** във файла **AndroidManifest.xml.**

```
<supports-screens android:resizable="true"
                  android:smallScreens="true"
                  android:normalScreens="true"
                  android:largeScreens="true" />
```

8. Демонстрира знания, чрез които осъществява достъп до хардуера на мобилно устройство.

Достъп до хардуера на мобилното устройство се осъществява чрез добавяне на референция Xamarin.Essentials във вашия клас.

Някой хардуерни компоненти и работа с тях са:

Accelerometer - класът на Accelerometer ви позволява да наблюдавате сензора на акселерометъра на устройството, който показва ускорението на устройството в

триизмерно пространство. Функционалността на акселерометъра работи чрез извикване на методите Start и Stop, за да слушате за промени в ускорението. Всички промени се изпращат обратно чрез събитието ReadingChanged.

Compass – наблюдава компаса за промени. Функционалността Compass работи чрез извикване на методите Start и Stop, за да слушате за промени в компаса. Всички промени се изпращат обратно чрез събитието ReadingChanged.

Geolocation - Извличане на GPS местоположението на устройството. Можете да получите последното известно местоположение на устройството, като извикате метода GetLastKnownLocationAsync. Това често е по-бързо от извършването на пълна заявка, но може да бъде по-малко точно и може да върне null, ако не съществува кеширано местоположение.

```
try
{
    var location = await Geolocation.GetLastKnownLocationAsync();

    if (location != null)
    {
        Console.WriteLine($"Latitude: {location.Latitude}, Longitude:
{location.Longitude}, Altitude: {location.Altitude}");
    }
}
```

Пълна заявка за запитване на координатите на местоположението на текущото устройство може да се използва GetLocationAsync.

9. Обяснява принципите на сигурност при проектиране и разработване на мобилни приложения.

Има няколко принципа за подобряване на сигурността на мобилните приложения, включително:

- Бял списък на приложението - списък на защитени приложения

-Осигуряване на сигурност на транспортния слой

-Силно удостоверяване и упълномощаване на потребителя

-Шифроване на данни при запис в паметта

-Предоставяне на достъп на приложение на ниво API

-Процеси, свързани с потребителски идентификатор

-Предварително определени взаимодействия между мобилното приложение и операционната система

-Изисква потребителско въвеждане за привилегирован / повишен достъп

-Правилно боравене със сесията.

10. Избира подходящи методи за тестване на мобилно приложение, така че да реши поставената задача.

Чрез емулатор, който емулира функциите на една компютърна система в друга компютърна система.

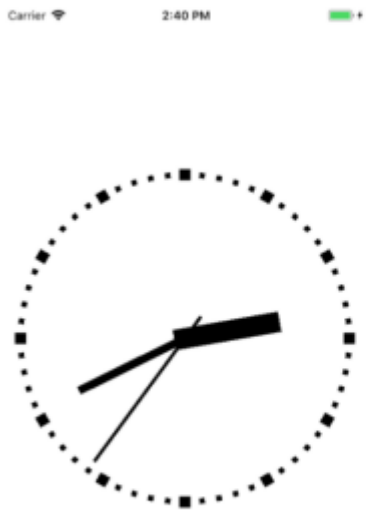
Например тестването на емулаторът на Android симулира устройство с Android на компютъра, така че се тестват приложението на различни устройства и нива на Android API, без да се използва физическо устройство. Във Visual Studio се използва Android Device Manager за създаване и конфигуриране на виртуални устройства с Android (AVD), които се изпълняват в емулатора на Android. Когато стартираме за първи път Android Device Manager, той представя екран, който показва всички текущо конфигурирани виртуални устройства. За всяко виртуално устройство се предоставя, името, ОС (версия на Android), процесор, размер на паметта и разделителна способност на екрана.

11. Открива грешки в програмен код и го модифицира, така че да реши поставената задача

12. Анализира, определя и допълва програмен код, така че да реши поставената задача.

Задача 1: Направете класически аналогов часовник, реализиран изцяло чрез оразмеряване, позициониране и завъртане. Анализирайте и допълнете ако е необходимо програмния код.

Мобилното приложение има следния изглед , виж фиг.1.



Фиг.1

```
using System;
using Xamarin.Forms;
```

```
namespace BoxViewClock
{
    public partial class MainPage : ContentPage
    {

        struct HandParams
        {
            public HandParams(double width, double height, double offset) : this()
            {
                Width = width;
                Height = height;
                Offset = offset;
            }

            public double Width { private set; get; }   // fraction of radius
            public double Height { private set; get; }  // ditto
            public double Offset { private set; get; }  // relative to center pivot
        }

        static readonly HandParams secondParams = new HandParams(0.02, 1.1, 0.85);
        static readonly HandParams minuteParams = new HandParams(0.05, 0.8, 0.9);
        static readonly HandParams hourParams = new HandParams(0.125, 0.65, 0.9);

        BoxView[] tickMarks = new BoxView[60];

        public MainPage()
        {
            InitializeComponent();

            // Create the tick marks (to be sized and positioned later).
            for (int i = 0; i < tickMarks.Length; i++)
            {
                tickMarks[i] = new BoxView { Color = Color.Black };
                absoluteLayout.Children.Add(tickMarks[i]);
            }

            Device.StartTimer(TimeSpan.FromSeconds(1.0 / 60), OnTimerTick);
        }

        void OnAbsoluteLayoutSizeChanged(object sender, EventArgs args)
        {
            Point center = new Point(absoluteLayout.Width / 2, absoluteLayout.Height / 2);
            double radius = 0.45 * Math.Min(absoluteLayout.Width, absoluteLayout.Height);
```

```
        for (int index = 0; index < tickMarks.Length; index++)
        {
            double size = radius / (index % 5 == 0 ? 15 : 30);
            double radians = index * 2 * Math.PI / tickMarks.Length;
            double x = center.X + radius * Math.Sin(radians) - size / 2;
            double y = center.Y - radius * Math.Cos(radians) - size / 2;
            AbsoluteLayout.SetLayoutBounds(tickMarks[index], new Rectangle(x, y, size,
size));
            tickMarks[index].Rotation = 180 * radians / Math.PI;
        }

        LayoutHand(secondHand, secondParams, center, radius);
        LayoutHand(minuteHand, minuteParams, center, radius);
        LayoutHand(hourHand, hourParams, center, radius);
    }

    void LayoutHand(BoxView boxView, HandParams handParams, Point center, double
radius)
    {
        double width = handParams.Width * radius;
        double height = handParams.Height * radius;
        double offset = handParams.Offset;

        AbsoluteLayout.SetLayoutBounds(boxView,
            new Rectangle(center.X - 0.5 * width,
                          center.Y - offset * height,
                          width, height));

        boxView.AnchorY = handParams.Offset;
    }

    bool OnTimerTick()
    {
        DateTime dateTime = DateTime.Now;
        hourHand.Rotation = 30 * (dateTime.Hour % 12) + 0.5 * dateTime.Minute;
        minuteHand.Rotation = 6 * dateTime.Minute + 0.1 * dateTime.Second;

        double t = dateTime.Millisecond / 1000.0;

        if (t < 0.5)
        {
            t = 0.5 * Easing.SpringIn.Ease(t / 0.5);
        }
        else
```

```
            {
                t = 0.5 * (1 + Easing.SpringOut.Ease((t - 0.5) / 0.5));
            }

            secondHand.Rotation = 6 * (dateTime.Second + t);
            return true;
        }
    }
}
```