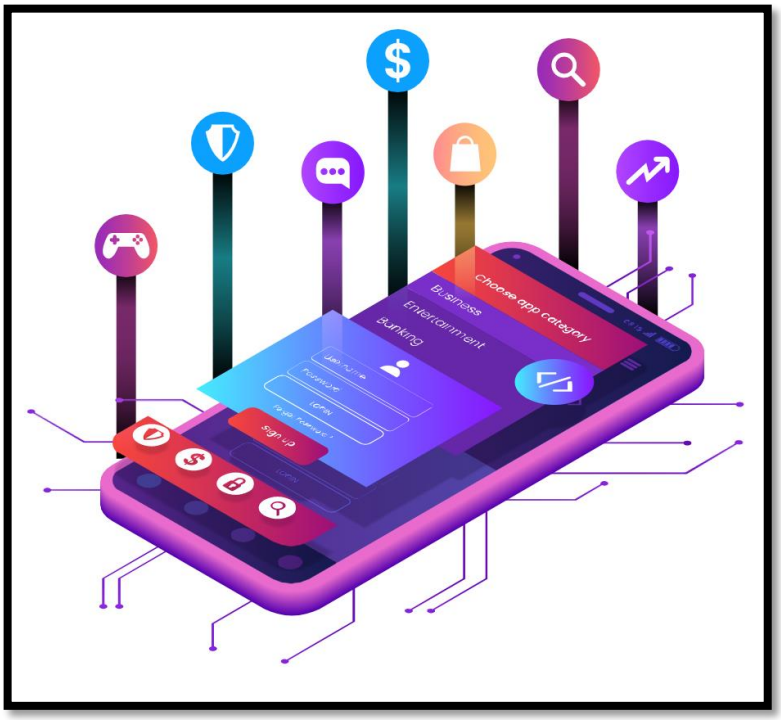
ҚОСТАНАЙ ОБЛЫСЫ ӘКІМДІГІ БІЛІМ БАСҚАРМАСЫНЫҢ
«ҚОСТАНАЙ ЖОҒАРЫ ПОЛИТЕХНИКАЛЫҚ КОЛЛЕДЖІ» КМҚК
КГКП «КОСТАНАЙСКИЙ ПОЛИТЕХНИЧЕСКИЙ ВЫСШИЙ КОЛЛЕДЖ»
УПРАВЛЕНИЯ ОБРАЗОВАНИЯ АКИМАТА КОСТАНАЙСКОЙ ОБЛАСТИ

УЧЕБНО – МЕТОДИЧЕСКИЙ КОМПЛЕКС

Разработка мобильных приложений



Костанай, 2022

УЧЕБНО – МЕТОДИЧЕСКИЙ КОМПЛЕКС

Разработка мобильных приложений

ЦЕЛЬ ДИСЦИПЛИНЫ:

Цель: освоение навыков программирования приложений для мобильных устройств с использованием языка Java и фреймворка Xamarin.

Задачи:

1. Изучение SMART технологии
2. Изучение основ языка Java для программирования
3. Разработка приложений в Android Studio

Разработка приложений с использованием фреймворка Xamarin

Разработка мобильных приложений. Учебно – методический комплекс.

Костанай: типография Костанайского политехнического высшего колледжа, 2022.- с.

Ссылка на электронный ресурс:
https://book.kpvk.edu.kz/courses/pm13-razrabotka-smart-prilozhenij/

Учебно – методический комплекс предназначен для изучения дисциплины «Разработка мобильных приложений»

Оглавление

Раздел 1. Использование SMART технологии

Тема 1.1 Введение в SMART.

SMART / SMARTER — это мнемоническая аббревиатура, используемая в менеджменте и проектном управлении для определения целей и постановки задач.

Существует несколько вариантов расшифровки аббревиатуры SMART.

S - specific, significant, stretching - конкретная, значительная.

Объясняется, что именно необходимо достичь. Например, «увеличить чистую прибыль собственного предприятия».

M - measurable, meaningful, motivational - измеримая, значимая, мотивирующая.

Объясняется, в чем будет измеряться результат. Если показатель количественный, то необходимо выявить единицы измерения, если качественный, то необходимо выявить эталон отношения. Например, «увеличить прибыль собственного предприятия на 25% относительно чистой прибыли текущего года».

A - attainable, agreed upon, achievable, acceptable, action-oriented - достижимая, согласованная, ориентированная на конкретные действия.

Объясняется, за счет чего планируется достичь цели. И возможно ли ее достигнуть вообще. Например, «увеличить прибыль собственного предприятия на 25% относительно чистой прибыли текущего года за счет снижения себестоимости продукции, автоматизации ресурсоемких операций и сокращения штата, занятых на исполнении автоматизируемых операций сотрудников на 80 % от текущего количества». А вот совершить кругосветный круиз на резиновой уточке вряд ли удастся.

R - realistic, relevant, reasonable, rewarding, results-oriented - реалистичная, уместная, полезная и ориентированная на конкретные результаты.

Определение истинности цели. Действительно ли выполнение данной задачи позволит достичь желаемой цели. Необходимо удостовериться, что выполнение данной задачи действительно необходимо. Например, если брать «сокращение штата занятых на исполнении автоматизируемых операций сотрудников на 80 %» в качестве отдельной подзадачи, которая также ставится по SMART, то сотрудников можно не увольнять, а перевести на иные должности, на которых эти сотрудники смогут принести компании доход, а не просто экономию. Если брать страховую компанию, то вместо увольнения сотрудникам можно предложить продолжить работу в качестве агента либо не расходовать средства на автоматизацию, а просто увеличить норму выработки.

T - time-based, timely, tangible, trackable - на определенный период, своевременная, отслеживаемая.

Определение временного триггера / промежутка, по наступлению / окончанию которого должна быть достигнута цель (выполнена задача). Например, «к окончанию второго квартала следующего года увеличить

прибыль собственного предприятия на 25 % относительно чистой прибыли текущего года за счет снижения себестоимости продукции, автоматизации ресурсоемких операций и сокращения штата, занятых на исполнении автоматизируемых операций сотрудников на 80% от текущего количества».

Задание: необходимо составить и описать проект по SMART технологии.

Тема 1.2 Составление проекта SMART технологии

Технология SMART (СМАРТ) — современный подход к постановке работающих целей. Система постановки smart — целей позволяет на этапе целеполагания обобщить всю имеющуюся информацию, установить приемлемые сроки работы, определить достаточность ресурсов, предоставить всем участникам процесса ясные, точные, конкретные задачи.

SMART является аббревиатурой, расшифровка которой: Specific, Measurable, Achievable, Relevant, Time bound. Каждая буква аббревиатуры SMART означает критерий эффективности поставленных целей. Рассмотрим каждый критерий smart цели более подробно.

Specific: Конкретный.

Цель по SMART должна быть конкретной, что увеличивает вероятность ее достижения. Понятие «Конкретный» означает, что при постановке цели точно определен результат, который Вы хотите достичь. Сформулировать конкретную цель поможет ответ на следующие вопросы:

- Какого результата я хочу достичь за счет выполнения цели и почему?
- Кто вовлечен в выполнение цели?
- Существуют ли ограничения или дополнительные условия, которые необходимы для достижения цели?

Всегда действует правило: одна цель — один результат. Если при постановке цели выяснилось, что в результате требуется достичь нескольких результатов, то цель должна быть разделена на несколько целей.

Measurable: Измеримый

Цель по SMART должна быть измеримой. На этапе постановки цели необходимо установить конкретные критерии для измерения процесса выполнения цели. В постановке измеримой цели помогут ответы на вопросы:

- Когда будет считаться, что цель достигнута?
- Какой показатель будет говорить о том, что цель достигнута?
- Какое значение у данного показателя должно быть для того, чтобы цель считалась достигнутой?

Achievable or Attainable: Достижимый

Цели по SMART должны быть достижимы, так как реалистичность выполнения задачи влияет на мотивацию исполнителя. Если цель не является достижимой — вероятность ее выполнения будет стремиться к 0. Достижимость цели определяется на основе собственного опыта с учетом всех имеющихся ресурсов и ограничений.

Ограничениями могут быть: временные ресурсы, инвестиции, трудовые ресурсы, знания и опыт исполнителя, доступ к информации и ресурсам, возможность принимать решения и наличие управленческих рычагов у исполнителя цели.

Relevant: Значимый

Для определения значимости цели важно понимать, какой вклад решение конкретной задачи внесет в достижение глобальных стратегических задач компании. В постановке значимой цели поможет следующий вопрос: Какие выгоды принесет компании решение поставленной задачи? Если при выполнении цели в целом компания не получит выгоды — такая цель считается бесполезной и означает пустую трату ресурсов компании.

Иногда Relevant заменяют на Realictic (реалистичный).

Time bound: Ограниченный во времени

Цель по SMART должна быть ограничена по выполнению во времени, а значит должен быть определен финальный срок, превышение которого говорит о невыполнении цели. Установление временных рамок и границ для выполнения цели позволяет сделать процесс управления контролируемым. При этом временные рамки должны быть определены с учетом возможности достижения цели в установленные сроки.

Примеры SMART целей

Приведем несколько примеров постановки SMART целей в компании:

Направление задачи	Пример цели по SMART	Комментарии автора
Увеличение продаж	Увеличить продажи бренда А на территории России к концу года на 25%	Конкретность цели определяется указанием % роста, региона продаж и названия бренда. Цель ограничена по времени годовым периодом, может быть измерена с помощью статистики продаж компании. Достижимость цели может быть определена только специалистами компании. Но предположим, что бренд получит необходимый уровень инвестиций для роста продаж. Цель

		значима, так напрямую связана с эффективностью бизнеса.
Продвижение товара	Достичь уровня знания товара А среди молодой аудитории на уровне 51% через 3 года, после запуска товара на рынок.	Цель конкретна, так как указана целевая аудитория и название бренда. Цель ограничена во времени и может быть измерена с помощью опроса. Достижимость может быть определена только специалистами компании, но предположим, что компания выделит необходимый уровень инвестиций для достижения планового показателя. Цель значима, так как знание товара имеет прямую корреляцию с продажами продукта.
Увеличение дистрибуции	Завести бренд компании в количестве 3 SKU в ТОП-10 ключевых торговых сетей до июля 2014 года.	Конкретность цели подтверждается указанием количества позиций и списка сетей. Цель имеет четкий срок выполнения и может быть явно измерена с помощью проверки отгрузок компании в данные сети. Достижимость цели может оценить только специалист по продажам, но предположим, что компания обеспечит отдел продаж необходимым бюджетом и рекламными материалами для листинга. Цель значима, так как дистрибуция в ключевые сети имеет прямое влияние на рост продаж.

Задания:

1. Опишите другие варианты расшифровки SMART.
2. Как Вы считаете, как это связано с разработкой приложения?
3. Придумать приложение, которое будет реализовываться в качестве проекта по SMART технологии.

Раздел 2. Знакомство с Android Studio

Тема 2.1 Установка Android Studio. Windows. Устранение проблем. Обзор программы Android Studio. Запуск первого приложения на эмуляторе. Запуск первого приложения на телефоне

1.Java SDK (JDK)

Т.к. разработка приложений ведется на Java, нам нужно скачать и установить соответствующее SDK, называемое еще JDK (если, конечно, оно уже не установлено).

После установки рекомендую перезагрузить компьютер.

2. Среда разработки + Android SDK

В среде разработки мы будем создавать программу и получать на выходе готовое приложение. Сейчас существует несколько сред разработки, мы выберем рекомендуемую гуглом Android Studio.

Учитывайте, что файл может весить до 2 гигов.

Итак, скачали exe-файл. Запускаем его. Жмем Next, пока он не спросит пути.

От нас требуется указать два пути. Первый путь будет использован для установки Android Studio. Второй - для установки Android SDK.

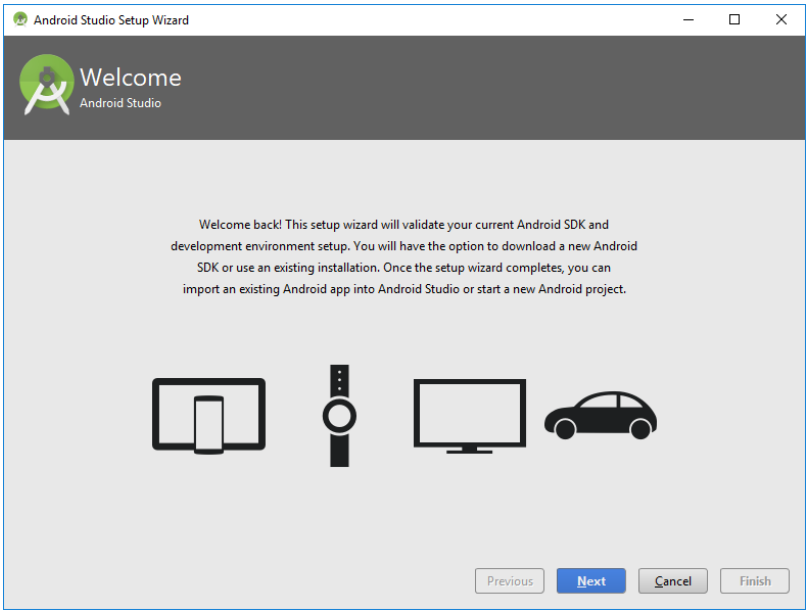
Давайте заменим их на свои. Для этого создадим каталог android. На всякий случай сделайте так, чтобы путь к нему был без пробелов и русских символов. Например - <имя диска>:android. У меня это будет d:android. И этот путь подставим в визард.

Жмем несколько раз Next, пока не начнется установка. Когда установка закончится, запустится Android Studio. Если не запустилась, то ищите ее ярлык в Пуске.

Первым делом она спросит, не надо ли импортировать настройки с прошлой версии.

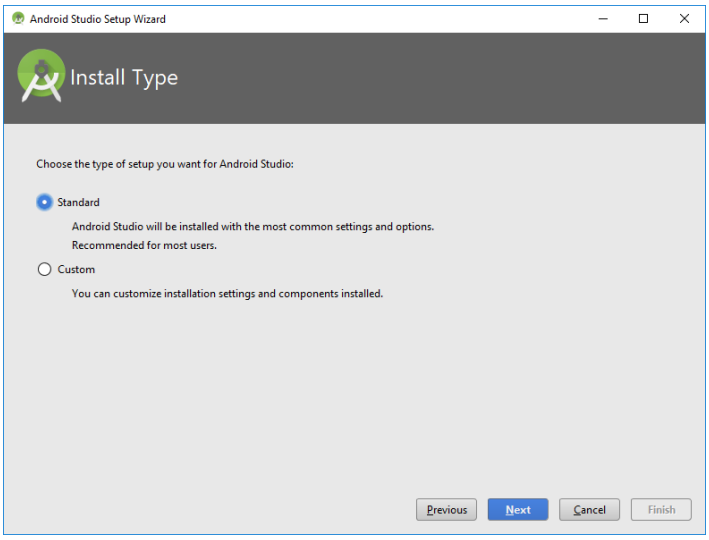
Оставляем выбранным нижний пункт и жмем Ок. У нас пока нет старых настроек.

Далее появится визард установки



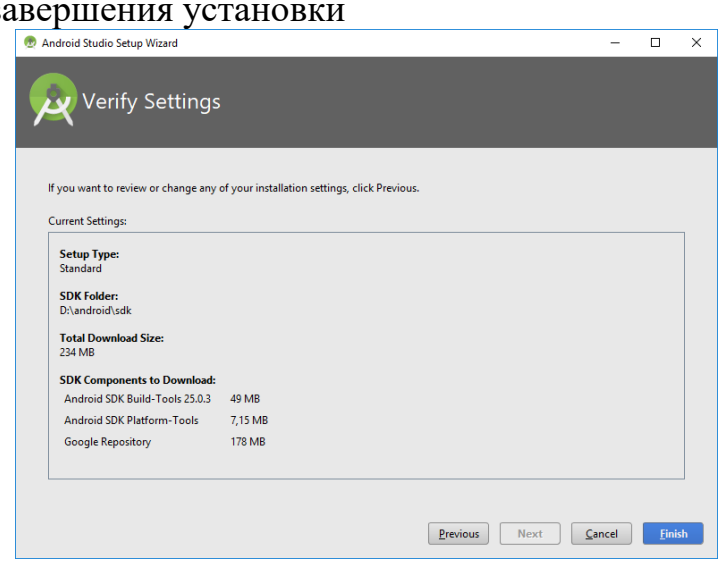
Жмем Next

Тут оставляем Standard

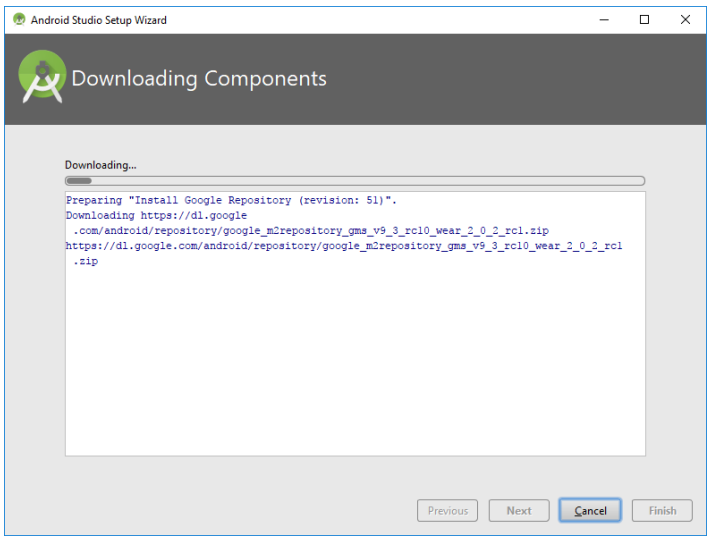


Жмем Next

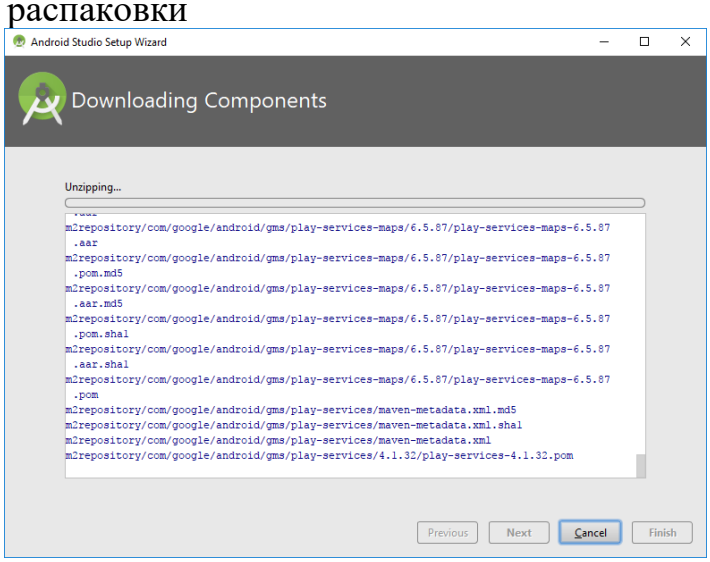
Визард сообщает нам, что ему необходимо загрузить несколько компонентов для завершения установки



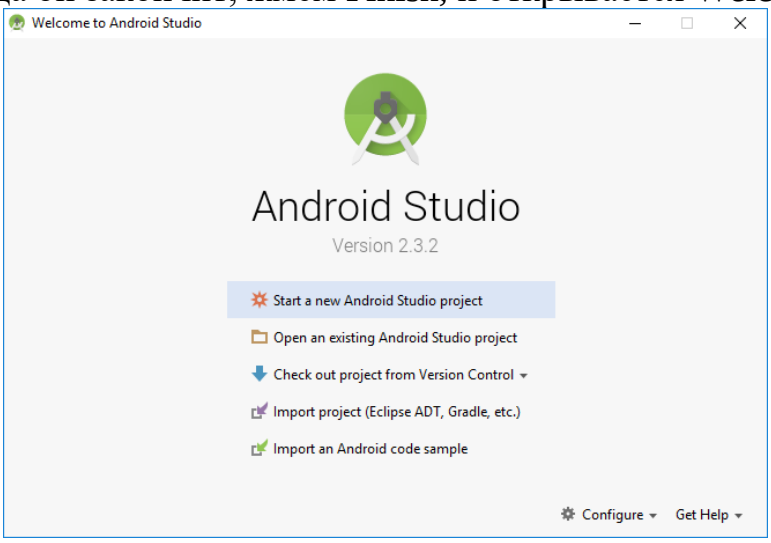
Жмем Next. Пошел процесс загрузки



Затем процесс распаковки



В итоге, когда он закончит, жмем Finish, и открывается Welcome экран.



После выполнения этих шагов мы получили среду разработки, с помощью которой можно кодить Android-приложения. Если что-то не получается или выдает ошибку - попробуйте посмотреть форум или погуглить, наверняка вы не первый сталкиваетесь с такой проблемой и в сети уже есть описание решения.

Android Studio периодически будет клянчить у вас скачать и установить ей обновления. Очень не советую этого делать, пока вы новичок. Вы после обновления вряд-ли заметите какие-то новшества в новой версии студии, а проблемы себе можно создать немалые.

На следующем уроке мы настроим Android Virtual Device (AVD), создадим наше первое приложение и запустим его. AVD – это эмулятор смартфона с операционной системой Android, на котором можно запускать и тестировать приложения. Не подключать же свой смартфон каждый раз.Также мы рассмотрим структуру проекта приложения.

Тема 2.2-2.3 Изучение языка Java

Основным строительным блоком программы на языке Java являются инструкции (statement). Каждая инструкция выполняет некоторое действие, например, вызовы методов, объявление переменных и присвоение им значений. После завершения инструкции в Java ставится точка с запятой (;). Данный знак указывает компилятору на конец инструкции. Например:

```
1       System.out.println("Hello Java!");
```

Данная строка представляет вызов метода System.out.println, который выводит на консоль строку "Hello Java!". В данном случае вызов метода является инструкцией и поэтому завершается точкой с запятой.

Кроме отдельных инструкций распространенной конструкцией является блок кода. Блок кода содержит набор инструкций, он заключается в фигурные скобки, а инструкции помещаются между открывающей и закрывающей фигурными скобками:

```
1       {
2           System.out.println("Hello!");
3           System.out.println("Welcome to Java!");
4       }
```

В этом блоке кода две инструкции, которые выводят на консоль определенную строку.

Выполнение программы. Метод main

Java является объектно-ориентированным языком, поэтому всю программу можно представить как набор взаимодействующих между собой классов и объектов. В первой главе при создании первого приложения программа была определена следующим образом:

```
1       public class Program{
2
3           public static void main (String args[]){
4
5               System.out.println("Hello Java!");
6           }
7       }
```

То есть основу нашей программы составляет класс Program. При определении класса вначале идет модификатор доступа public, который

указывает, что данный класс будет доступен всем, то есть мы сможем его запустить из командной строки. Далее идет ключевое слово class, а затем название класса. После названия класса идет блок кода, в котором расположено содержимое класса.

Входной точкой в программу на языке Java является метод main, который определен в классе Program. Именно с него начинается выполнение программы. Он обязательно должен присутствовать в программе. При этом его заголовок может быть только таким:

```
1       public static void main (String args[])
```

При запуске приложения виртуальная машина Java ищет в главном классе программы метод main с подобным заголовком, и после его обнаружения запускает его.

Вначале заголовка метода идет модификатор public, который указывает, что метод будет доступен извне. Слово static указывает, что метод main - статический, а слово void - что он не возвращает никакого значения. Далее в скобках у нас идут параметры метода - String args[] - это массив args, который хранит значения типа String, то есть строки. При запуске программы через этот массив мы можем передать в программу различные данные.

После заголовка метода идет его блок, который содержит набор выполняемых инструкций.

Комментарии

Код программы может содержать комментарии. Комментарии позволяют понять смысл программы, что делают те или иные ее части. При компиляции комментарии игнорируются и не оказывают никакого влияния на работу приложения и на его размер.

В Java есть два типа комментариев: однострочный и многострочный. Однострочный комментарий размещается на одной строке после двойного слеша //. А многострочный комментарий заключается между символами /* текст комментария */. Он может размещаться на нескольких строках. Например:

```
/*
   многострочный комментарий
   Объявление нового класса,
   который содержит код программы
*/
public class Program{   // начало объявления класса Program

   // определение метода main
   public static void main (String args[]){    // объявление нового метода

       System.out.println("Hello Java!");     // вывод строки на консоль
   }   // конец объявления нового метода
} // конец объявления класса Program
```

Для хранения данных в программе предназначены переменные. Переменная представляет именованную область памяти, которая хранит

значение определенного типа. Каждая переменная имеет тип, имя и значение. Тип определяет, какую информацию может хранить переменная или диапазон допустимых значений.

Переменные объявляются следующим образом:

```
1    тип_данных имя_переменной;
```

Например, определим переменную, которая будет называться x и будет иметь тип int:

```
1    int x;
```

В этом выражении мы объявляем переменную x типа int. То есть x будет хранить некоторое число не больше 4 байт.

В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

имя может содержать любые алфавитно-цифровые символы, а также знак подчеркивания, при этом первый символ в имени не должен быть цифрой

в имени не должно быть знаков пунктуации и пробелов

имя не может быть ключевым словом языка Java

Кроме того, при объявлении и последующем использовании надо учитывать, что Java - регистрозависимый язык, поэтому следующие объявления int num; и int NUM; будут представлять две разных переменных.

Объявив переменную, мы можем присвоить ей значение:

```
1    int x;      // объявление переменной
2    x = 10;     // присвоение значения
3    System.out.println(x);  // 10
```

Также можно присвоить значение переменной при ее объявлении. Этот процесс называется инициализацией:

```
1    int x = 10;     // объявление и инициализация переменной
2    System.out.println(x);  // 10
```

Если мы не присвоим переменной значение до ее использования, то мы можем получить ошибку, например, в следующем случае:

```
1    int x;
2    System.out.println(x);
```

Через запятую можно объявить сразу несколько переменных одного типа:

```
1    int x, y;
2    x = 10;
3    y = 25;
4    System.out.println(x);  // 10
5    System.out.println(y);  // 25
```

Также можно их сразу инициализировать:

```
1    int x = 8, y = 15;
2    System.out.println(x);  // 8
3    System.out.println(y);  // 15
```

Отличительной особенностью переменных является то, что мы можем в процессе работы программы изменять их значение:

```
1    int x = 10;
2    System.out.println(x);  // 10
```

```
3       x = 25;
4       System.out.println(x);  // 25
```

Ключевое слово var

Начиная с Java 10 в язык было добавлено ключевое слово var, которое также позволяет определять переменную:

```
1       var x = 10;
2       System.out.println(x);  // 10
```

Слово var ставится вместо типа данных, а сам тип переменной выводится из того значения, которое ей присваивается. Например, переменной x присваивается число 10, значит, переменная будет представлять тип int.

Но если переменная объявляется с помощью var, то мы обязательно должны инициализировать ее, то есть предоставить ей начальное значение, иначе мы получим ошибку, как, например, в следующем случае:

```
1       var x;      // ! Ошибка, переменная не инициализирована
2       x = 10;
```

Константы

Кроме переменных, в Java для хранения данных можно использовать константы. В отличие от переменных константам можно присвоить значение только один раз. Константа объявляется также, как и переменная, только вначале идет ключевое слово final:

```
1       final int LIMIT = 5;
2       System.out.println(LIMIT);  // 5
3       // LIMIT=57; // так мы уже не можем написать, так как LIMIT - константа
```

Как правило, константы имеют имена в верхнем регистре.

Константы позволяют задать такие переменные, которые не должны больше изменяться. Например, если у нас есть переменная для хранения числа pi, то мы можем объявить ее константой, так как ее значение постоянно.

Тема 2.4-2.5 Изучение языка Java. Построение интерактивных приложений. Приложение с несколькими активностями.

В прошлых темах мы создали первый проект, однако весь проект состоял лишь из примитивного кода, добавляемого по умолчанию. Теперь определим сами приложение, которое также также будет очень простое. Пусть на одной странице приложения мы будем вводить некоторые данные и по нажатию на кнопку будет происходить переход к другой странице приложения, которая будет отображать ранее введенные данные.

Итак, возьмем ранее созданный проект (или создадим новый).

Добавление новой Activity

И по умолчанию у нас уже есть одна activity - класс MainActivity. Теперь добавим еще одну. Для этого нажмем правой кнопкой мыши в структуре проекта на папку, в котором находится класс MainActivity, и затем в контекстном меню выберем New->Activity->Empty Activity:

После этого откроется диалоговое окно создания новой activity:

В этом окне в поле Activity Name введем MessageActivity. После этого в поле Layout Name автоматически должно установиться activity_message (если не устновилось, то введем в это поле activity_message). В остальных полях оставим значения по умолчанию:

Package Name - должен иметь то же название, что и пакет, в котором находится MainActivity

Source Language должен иметь значение Java

И затем нажмем Finish.

После этого в папке с MainActivity должен появиться файл с новой activity - MessageActivity:

Кроме того, в каталоге res/layout должен появиться файл activity_message.xml, который представляет описание графического интерфейса для MessageActivity.

Вначале откроем файл activity_message.xml и изменим его код следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MessageActivity">
    <TextView
        android:id="@+id/messageText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:textSize="18sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Определение интерфейса для MessageActivity содержит только элемент TextView, который выводит некоторую строку на экран. Рассмотрим установленные в нем атрибуты:

android:id="@+id/messageText" - TextView имеет идентификатор "messageText", через который мы сможем обращаться к TextView в коде java. Символ @ указывает XML-парсеру использовать оставшуюся часть строки атрибута как идентификатор. А знак + означает, что если для элемента не определен id со значением header, то его следует определить.

android:layout_width="wrap_content" - ширина текстового поля будет такой, чтобы вместить все его содержимое на экране

android:layout_height="wrap_content" - высота TextView будет такой, чтобы вместить все его содержимое на экране

android:textSize="18sp" - высота текста в TextView составляет 18 единиц (для установки высоты шрифта используется величина sp)

app:layout_constraintBottom_toBottomOf="parent" - нижний край TextView будет выравниваться по нижней стороне контейнера ConstraintLayout

app:layout_constraintLeft_toLeftOf="parent" - левая граница TextView будет выравниваться по левой стороне контейнера ConstraintLayout

app:layout_constraintRight_toRightOf="parent" - правый край TextView будет выравниваться по правой стороне контейнера ConstraintLayout

app:layout_constraintTop_toTopOf="parent" - верхний край TextView будет выравниваться по верхней стороне контейнера ConstraintLayout

Получение данных

Суть MessageActivity будет заключаться в том, что она будет получать некое текстовое сообщение и выводить его на экран (формально в элемент TextView, который был определен выше). Как мы можем получить в MessageActivity (и в любой другой activity) некоторые данные, которые переданы из другой activity?

Возьмем код класса MessageActivity. По умолчанию он выглядит так:

```
package com.example.helloapp;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MessageActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_message);
    }
}
```

И изменим его следующим образом:

```
package com.example.helloapp;

import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.TextView;

public class MessageActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_message);

        // Получаем объект Intent, который запустил данную activity
        Intent intent = getIntent();
        // Получаем сообщение из объекта intent
        String message = intent.getStringExtra("message");
        // Получаем TextView по его id
        TextView messageText = (TextView) findViewById(R.id.messageText);
        // устанавливаем текст для TextView
        messageText.setText(message);
    }
}
```

Также, как и в MainActivity (и в других activity), создание текущей activity происходит в методе onCreate(). Все классы фctivity должны реализовать метод onCreate, так как система вызывает его при создании новой activity. Именно в этом методе задается компоновка нового объекта activity с помощью метода setContentView и именно здесь происходит начальная настройка компонентов.

Каждый объект Activity вызывается объектом Intent. Мы можем в коде Java получить вызывающий объект Intent с помощью метода getIntent:

```
1    Intent intent = getIntent();
```

Но Intent нам важен не сам по себе, а потому что через него в эту activity будут передаваться данные. Эти данные могут представлять различные типы, но в здесь мы предполагаем, что в MainActivity будет передаваться строка. И для получения строки у объекта Intent вызывается метод getStringExtra():

```
1    String message = intent.getStringExtra("message");
```

В метод передается ключ данных. То есть каждый элемент передаваемых данных представлен в формате "ключ-значение". Через ключ мы можем получить значение. И в данном случае ключом будет "message". А значение по этому ключу попадет в переменную String message.

Получив переданные в MessageActivity данные, мы можем передать их в TextView для вывода на экране устройства. Для этого вначале находим виждет TextView по его id и затем вызываем его метод setText(), который устанавливает выводимый в TextView текст:

```
1    TextView messageText = (TextView)
2findViewById(R.id.messageText);
    messageText.setText(message);
```

Таким образом, MessageActivity получит переданное ей сообщение и выведет его в TextView. Теперь рассмотрим, как вызвать эту activity из MainActivity и как передать ей это сообщение.

Задание: создать простое приложение, формирующее вывод Фамилии Имени и названия группы.

Тема 2.6 – 2.7 Изучение языка Java. Пользовательский интерефейс, Создание макетов. ListView, адаптеры и слушатели событий.

Рассмотрите ролик: https://www.youtube.com/watch?v=6keeinbv-Y8

Изучение языка Java. Пользовательский интерфейс, Создание макетов. ListView, адаптеры и слушатели событий.

https://www.youtube.com/watch?v=6keeinbv-Y8

Класс javafx.scene.control.ListView<T> позволяет создавать списки. ListView является обобщенным типом. То есть в зависимости от того, объекты какого типа должен хранить ListView, мы можем его типировать тем или иным типом.

Для создания ListView могут применяться два конструктора:

ListView(): создает пустой список

ListView(ObservableList<T> items): создает список, наполненный элементами items

Создадим простейший список:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;

public class Main extends Application{

    public static void main(String[] args) {

        Application.launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {

        // создаем список объектов
        ObservableList<String> langs = FXCollections.observableArrayList("Java", "JavaScript", "C#", "Python");
        ListView<String> langsListView = new ListView<String>(langs);

        FlowPane root = new FlowPane(langsListView);
```

```
        Scene scene = new Scene(root, 250, 200);

        stage.setScene(scene);
        stage.setTitle("ListView in JavaFX");
        stage.show();
    }
}
```

По умолчанию ListView занимает некоторые стандартные размеры, которые не всегда могут будь удобны. В этом случае мы можем использовать ряд методов для установки длины и ширины:

void setPrefHeight(double height): задает предпочтительную высоту

void setPrefWidth(double width): задает предпочтительную ширину

void setPrefSize(double width, double height): задает предпочтительные ширину и высоту

Получение выбранных элементов

Для получения выбранных в ListView элементов необходимо обратиться в ListView к методу

1 final MultipleSelectionModel<T> getSelectionModel()

Этот метод возвращает объект MultipleSelectionModel - модель, которая отслеживает выбранные элементы. Этот объект определяет метод selectedItemProperty(), который возвращает объект ReadOnlyObjectProperty. И с помощью метода addListener() у ReadOnlyObjectProperty можно установить слушатель, который будет отслеживать изменения выбранных элементов:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;

import javafx.geometry.Orientation;
import javafx.beans.value.ObservableValue;
import javafx.beans.value.ChangeListener;
import javafx.scene.control.MultipleSelectionModel;

public class Main extends Application{

    public static void main(String[] args) {

        Application.launch(args);
    }
```

```
        @Override
        public void start(Stage stage) throws Exception {

            Label selectedLbl = new Label();
            // создаем список объектов
            ObservableList<String> langs =
FXCollections.observableArrayList("Java", "JavaScript", "C#", "Python");
            ListView<String> langsListView = new ListView<String>(langs);
            langsListView.setPrefSize(250, 150);
            // получаем модель выбора элементов
            MultipleSelectionModel<String> langsSelectionModel =
langsListView.getSelectionModel();
            // устанавливаем слушатель для отслеживания изменений
            langsSelectionModel.selectedItemProperty().addListener(new
ChangeListener<String>(){

                public void changed(ObservableValue<? extends String> changed,
String oldValue, String newValue){

                    selectedLbl.setText("Selected: " + newValue);
                }
            });

            FlowPane root = new FlowPane(Orientation.VERTICAL, 10, 10,
selectedLbl, langsListView);
            Scene scene = new Scene(root, 250, 200);

            stage.setScene(scene);
            stage.setTitle("ListView in JavaFX");
            stage.show();
        }
    }
```

Здесь вначале получаем модель выбора элементов - MultipleSelectionModel<String>. Причем поскольку в списке хранятся объекты String, то MultipleSelectionModel тоже типизируется типом String.

Затем прикрепляем слушатель, который будет отслеживать изменения выбора элементов. Слушатель представляет тип, который реализует интерфейс ChangeListener. Причем интерфейс также типизируется типом хранимых в списке объектов - типом String.

В методе changed этого интерфейса с помощью параметра newValue мы можем получить новое выбранное значение и что-то с ним сделать, например, отобразить на метке. Также можно получить ранее выбранное значение через параметр oldValue.

Множественное выделение

По умолчанию ListView позволяет выбирать одномоментно только один элемент, однако мы можем установить выбор нескольких элементов. Для этого применяется метод

final void setSelectionMode(SelectionMode mode)

В метод передается значение из перечисления javafx.scene.control.SelectionMode: SelectionMode.MULTIPLE (множественный выбор) или SelectionMode.SINGLE (одиночный выбор - по умолчанию).

Например, используем множественный выбор:

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Label;
import javafx.scene.control.ListView;
import javafx.scene.control.SelectionMode;
import javafx.scene.control.MultipleSelectionModel;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;

import javafx.geometry.Orientation;
import javafx.beans.value.ObservableValue;
import javafx.beans.value.ChangeListener;

public class Main extends Application{

    public static void main(String[] args) {

        Application.launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {

        Label selectedLbl = new Label();

        // создаем список объектов
        ObservableList<String> langs =
FXCollections.observableArrayList("Java", "JavaScript", "C#", "Python");
        ListView<String> langsListView = new ListView<String>(langs);
        langsListView.setPrefSize(250, 150);
```

```
        // получаем модель выбора элементов
        MultipleSelectionModel<String> langsSelectionModel = langsListView.getSelectionModel();
        langsSelectionModel.setSelectionMode(SelectionMode.MULTIPLE);

        // устанавливаем слушатель для отслеживания изменений
        langsSelectionModel.selectedItemProperty().addListener(new ChangeListener<String>(){

            public void changed(ObservableValue<? extends String> changed, String oldValue, String newValue){

                String selectedItems = "";
                ObservableList<String> selected = langsSelectionModel.getSelectedItems();
                for (String item : selected){
                   selectedItems += item + " ";
                }
                selectedLbl.setText("Selected: " + selectedItems);
            }
        });

        FlowPane root = new FlowPane(Orientation.VERTICAL, 10, 10, selectedLbl, langsListView);

        Scene scene = new Scene(root, 250, 200);

        stage.setScene(scene);
        stage.setTitle("ListView in JavaFX");
        stage.show();
    }
}
```

Чтобы получить все выделенные элементы, мы можем обратиться к методу getSelectedItems() объекта SelectionModel.

Динамическое изменение ListView

После того, как для ListView установлен источник данных в виде коллекции ObservableList, все операции по добавлению новых элементов в ListView или удалению старых производятся только с ObservableList. Поскольку этот интерфейс наследует базовый для всех коллекций интерфейс Collection, то для добавления и удаления в ObservableList мы можем использовать методы add() и remove() соответственно:

```
import javafx.application.Application;
import javafx.stage.Stage;
```

```
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Button;
import javafx.scene.control.TextField;
import javafx.scene.control.ListView;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;
import javafx.geometry.Orientation;

public class Main extends Application{

    public static void main(String[] args) {

        Application.launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {

        TextField textField = new TextField();

        ObservableList<String> langs =
FXCollections.observableArrayList("Java", "JavaScript", "C#", "Python");
        ListView<String> langsListView = new ListView<String>(langs);
        langsListView.setPrefSize(250, 150);

        Button addBtn = new Button("Add");
        Button deleteBtn = new Button("Delete");
        FlowPane buttonPane = new FlowPane(10, 10, textField, addBtn,
deleteBtn);

        addBtn.setOnAction(event -> langs.add(textField.getText()));
        deleteBtn.setOnAction(event -> langs.remove(textField.getText()));

        FlowPane root = new FlowPane(Orientation.VERTICAL, 10, 10,
buttonPane, langsListView);

        Scene scene = new Scene(root, 300, 250);

        stage.setScene(scene);
        stage.setTitle("ListView in JavaFX");
        stage.show();
    }
}
```

В данном случае мы изменяем элементы в источнике данных, к которому привязан ListView. Однако мы можем полностью поменять источник данных с помощью метода setItems(ObservableList collection):

```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.FlowPane;
import javafx.scene.control.Button;
import javafx.scene.control.ListView;
import javafx.collections.ObservableList;
import javafx.collections.FXCollections;
import javafx.geometry.Orientation;

public class Main extends Application{

    public static void main(String[] args) {

        Application.launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {

        ObservableList<String> langs = FXCollections.observableArrayList("Java", "JavaScript", "C#", "Python");
        ListView<String> langsListView = new ListView<String>(langs);
        langsListView.setPrefSize(250, 150);

        Button btn = new Button("Change");

        btn.setOnAction(event -> {

            ObservableList<String> newLangs = FXCollections.observableArrayList("PHP", "Go", "C++");
            langsListView.setItems(newLangs);
        });

        FlowPane root = new FlowPane(Orientation.VERTICAL, 10, 10, btn, langsListView);
        Scene scene = new Scene(root, 300, 250);

        stage.setScene(scene);
        stage.setTitle("ListView in JavaFX");
```

```
        stage.show();
    }
}
```

Тема 2.8 - Изучение языка Java. Проект MyMovies.

Используя полученные знания языка Java и по среде Android Studio разработать мини приложение MyMovies.

Тема 2.9 Получаем данные из интернета

Изучение языка Java. Получаем данные из интернета. JSON

Для работы с форматом json нет встроенных средств, но есть куча библиотек и пакетов, которые можно использовать для данной цели. Одним из наиболее популярных из них является пакет com.google.code.gson.

Для его использования в проекте Android в файл guild.gradle, который относится к модулю app, в секцию dependencies необходимо добавить соответствующую зависимость:

```
1       implementation 'com.google.code.gson:gson:2.8.6'
```

То есть после добавления секция зависимостей dependencies в файле build.gradle может выглядеть следующим образом:

```
dependencies {

    implementation 'com.google.code.gson:gson:2.8.6'

    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

После добавления пакета в проект добавим новый класс User, который будет представлять данные:

```
package com.example.filesapp;

public class User {

    private String name;
    private int age;
```

```
    User(String name, int age){
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    @Override
    public  String toString(){
        return "Имя: " + name + " Возраст: " + age;
    }
}
```

Объекты этого класса мы будем сериализовать в формат json и наоборот десериализовать из файла.

Для работы с json добавим следующий класс JSONHelper:

```
package com.example.filesapp;

import android.content.Context;
import com.google.gson.Gson;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;

class JSONHelper {

    private static final String FILE_NAME = "data.json";

    static boolean exportToJSON(Context context, List<User> dataList) {
```

```
            Gson gson = new Gson();
            DataItems dataItems = new DataItems();
            dataItems.setUsers(dataList);
            String jsonString = gson.toJson(dataItems);

            FileOutputStream fileOutputStream = null;

            try {
               fileOutputStream = context.openFileOutput(FILE_NAME,
Context.MODE_PRIVATE);
               fileOutputStream.write(jsonString.getBytes());
               return true;
            } catch (Exception e) {
               e.printStackTrace();
            } finally {
               if (fileOutputStream != null) {
                  try {
                     fileOutputStream.close();
                  } catch (IOException e) {
                     e.printStackTrace();
                  }
               }
            }

            return false;
         }

         static List<User> importFromJSON(Context context) {

            InputStreamReader streamReader = null;
            FileInputStream fileInputStream = null;
            try{
               fileInputStream = context.openFileInput(FILE_NAME);
               streamReader = new InputStreamReader(fileInputStream);
               Gson gson = new Gson();
               DataItems dataItems = gson.fromJson(streamReader, DataItems.class);
               return  dataItems.getUsers();
            }
            catch (IOException ex){
               ex.printStackTrace();
            }
            finally {
               if (streamReader != null) {
                  try {
                     streamReader.close();
```

```
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            if (fileInputStream != null) {
                try {
                    fileInputStream.close();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        return null;
    }

    private static class DataItems {
        private List<User> users;

        List<User> getUsers() {
            return users;
        }
        void setUsers(List<User> users) {
            this.users = users;
        }
    }
}
```

Для работы с json создается объект Gson. Для сериализации данных в формат json у этого объекта вызывается метод toJson(), в который передаются сериализуемые данные.

Для упрощения работы с данными применяется вспомогательный класс DataItems. На выходе метод toJson() возвращает строку, которая затем сохраняется в текстовый файл.

Для десериализации выполняется метод fromJson(), в который передается объект Reader с сериализованными данными и тип, к которому надо десериализиовать данные.

В итоге проект будет выглядеть так:

Теперь определим основной функционал для взаимодействия с пользователем. Изменим файл activity_main.xml следующим образом:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
android:layout_width="match_parent"
android:layout_height="match_parent">

<EditText
   android:id="@+id/nameText"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:hint="Введите имя"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintBottom_toTopOf="@id/ageText"
   app:layout_constraintTop_toTopOf="parent" />
<EditText
   android:id="@+id/ageText"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:hint="Введите возраст"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintBottom_toTopOf="@id/addButton"
   app:layout_constraintTop_toBottomOf="@id/nameText" />
<Button
   android:id="@+id/addButton"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:text="Добавить"
   android:onClick="addUser"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toRightOf="parent"
   app:layout_constraintBottom_toTopOf="@id/saveButton"
   app:layout_constraintTop_toBottomOf="@id/ageText" />
<Button
   android:id="@+id/saveButton"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
   android:text="Сохранить"
   android:onClick="save"
   app:layout_constraintLeft_toLeftOf="parent"
   app:layout_constraintRight_toLeftOf="@id/openButton"
   app:layout_constraintBottom_toTopOf="@id/list"
   app:layout_constraintTop_toBottomOf="@id/addButton"/>
<Button
   android:id="@+id/openButton"
   android:layout_width="0dp"
   android:layout_height="wrap_content"
```

```
        android:text="Открыть"
        android:onClick="open"
        app:layout_constraintLeft_toRightOf="@id/saveButton"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toTopOf="@id/list"
        app:layout_constraintTop_toBottomOf="@id/addButton"/>
    <ListView
        android:id="@+id/list"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toBottomOf="@id/openButton" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Здесь определены два текстовых поля для ввода названия модели и цены объекта User и одна кнопка для добавления данных в список. Еще одна кнопка выполняет сериализацию данных из списка в файл, а третья кнопка - восстановление данных из файла.

Для вывода сами данных определен элемент ListView.

И изменим класс MainActivity:

```
package com.example.filesapp;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

import java.util.ArrayList;
import java.util.List;

public class MainActivity extends AppCompatActivity {

    private ArrayAdapter<User> adapter;
    private EditText nameText, ageText;
    private List<User> users;
    ListView listView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
```

```
        nameText = (EditText) findViewById(R.id.nameText);
        ageText = (EditText) findViewById(R.id.ageText);
        listView = (ListView) findViewById(R.id.list);
        users = new ArrayList<User>();

        adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, users);
        listView.setAdapter(adapter);
    }

    public void addUser(View view){
        String name = nameText.getText().toString();
        int age = Integer.parseInt(ageText.getText().toString());
        User user = new User(name, age);
        users.add(user);
        adapter.notifyDataSetChanged();
    }

    public void save(View view){

        boolean result = JSONHelper.exportToJSON(this, users);
        if(result){
            Toast.makeText(this, "Данные сохранены",
Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(this, "Не удалось сохранить данные",
Toast.LENGTH_LONG).show();
        }
    }
    public void open(View view){
        users = JSONHelper.importFromJSON(this);
        if(users!=null){
            adapter = new ArrayAdapter<>(this,
android.R.layout.simple_list_item_1, users);
            listView.setAdapter(adapter);
            Toast.makeText(this, "Данные восстановлены",
Toast.LENGTH_LONG).show();
        }
        else{
            Toast.makeText(this, "Не удалось открыть данные",
Toast.LENGTH_LONG).show();
        }
    }
```

}

Все данные находятся в списке users, который представляет объект List<User>. Через адаптер этот список связывается с ListView.

Для сохранения и восстановления данных вызываются ранее определенные методы в классе JSONHelper. Кнопка добавления добавляет данные в список user, и они сразу же отображаются в ListView. При нажатии на кнопку сохранения данные из списка users сохраняются в локальный файл. Затем с помощью кнопки открытия мы сможем открыть ранее сохраненный файл.

Раздел 3. Xamarin

Тема 3.1 Xamarin и кросс-платформенная разработка

Xamarin.Forms представляет платформу, которая нацелена на создание кроссплатформенных приложений под Android, iOS и Windows 10. Зачем использовать именно данную платформу, какие преимущества она несет? Есть определенные статистические данные, что значительная часть мобильных приложений создается более чем для одной платформы, например, для Android и iOS. Однако неизбежно разработчики сталкиваются со следующими трудностями:

различие в подходах построение графического интерфейса так или иначе влияет на разработку. Разработчики вынуждены подстраивать приложение под требования к интерфейсу на конкретной платформе

разные API - различие в программных интерфейсах и реализациях тех или иных функциональностей также требует от программиста учет этих специфических особенностей

разные платформы для разработки. Например, чтобы создавать приложения для iOS нам необходима соответствующая среда - Mac OS X и ряд специальных инструментов, типа XCode. А в качестве языка программирования выбирается Objective-C или Swift. Для Androida мы можем использовать самый разный набор сред - Android Studio, Eclipse и т.д. Но здесь для подавляющего большинства приложений применяется Java или Kotlin.А для создания приложений под Windows используется Visual Studio, а в качестве языков - C#, F#, VB.NET, C++

Такой диапазон платформ, средств разработки и языков программирования не может положительно сказываться на сроках создания приложений, и, в конечном счете, на денежных средствах, выделяемых на разработку. Было бы очень эффективно иметь один инструмент, который позволял легко и просто создавать приложения сразу для всех платформ. И именно таким инструментом и является платформа Xamarin (произносится как "зэмарин").

Xamarin позволяет создавать одну единственную логику приложения с применением C# и .NET сразу для всех трех платформ - Android, iOS, UWP.

Преимущества использования Xamarin.Forms:

В процессе разработки создается единый код для всех платформ

Xamarin предоставляет прямой доступ к нативным API каждой платформы

При создании приложений мы можем использовать платформу .NET и язык программирования C# (а также F#), который является достаточно производительным, и в тоже время ясным и простым для освоения и применения

Xamarin Forms поддерживает несколько платформ. Основные платформы: Android, iOS, UWP, Tizen. Дополнительные платформы, которые развиваются сообществом: MacOS, WPF, GTK#.

5 января 2021 года вышла последняя версия данной платформы - Xamarin Forms 5.0 и собственно она и будет рассматриваться в данном руководстве.

Как работает Xamarin

Работу Xamarin графически можно представить следующим образом:

Xamarin работает поверх фреймворка Mono, который предоставляет opensource-реализацию .NET Framework. Mono может работать поверх разных платформ - Linix, MacOS и т.д.

На уровне каждой отдельной платформы Xamarin полагается на ряд субплатформ. В частности:

Xamarin.Android - библиотеки для создания приложений на ОС Android

Xamarin.iOS - библиотеки для создания приложений для iOS

Эти субплатформы играют большую роль - через них приложения могут напрявлять запросы к прикладным интерфейсам на устройствах под управлением ОС Android или iOS. Вкратце это выглядит следующим образом.

С помощью Xamarin.Android код C# с использованием Xamarin компилируется в Intermediate Language (IL), который затем при запуске приложения компилируется в нативную сборку. Xamarin-приложения запускаются в среде выполнения Mono. Напрямую код не может обращаться к API Android. Для этого надо обратиться к функциональность пространств имен Android.* и Java.*, которые прдоставляются виртуальной машиной Android Runtime (ART). Специальная прослойка Managed Callable Wrappers (MCW) позволяет транслировать вызова managed-кода в нативные вызовы и обращаться к функциональности пространств имен Android.* и Java.*

И наоборот, когда Android Runtime (ART) обращается к приложению с кодом Xamarin, то все вызовы проходят через обертку Android Callable Wrappers (ACW).

Приложения Xamarin.iOS в отличие от Xamarin.Android, который использует JIT-компиляцию, применяют AOT-компиляцию (Ahead-of-Time) кода C# в нативный ARM-код. Xamarin использует промежуточный слой Selectors (селекторы) для трансляции вызовов кода Objective-C в код на C# и слой Registrars (регистраторы) для трансляции кода C# в Objective-C. В итоге слои Selectors и Registrars в целом представляют премежуточный слой, который на иллюстрации выше обозначен как "bindings" и который собственно позволяет взаимодействовать коду Objective-C с кодом C#.

В итоге благодаря этим платформам мы можем создавать отдельно приложения для Android, отдельно для iOS, но наиболее важной особенностью Xamarin является возможность создавать кроссплатформенные приложения - то есть одна логика для всех платформ. Данная возможность представлена технологией Xamarin.Forms и которая работает как бы уровнем выше Xamarin.Android и Xamarin.iOS. То есть с помощью Xamarin.Forms мы один раз можем определить визуальный интерфейс, один раз к нему привязать какую-то логику на C#, и все это будет работать на Android, iOS и Windows. Затем Xamarin.Forms с помощью рендереров (renderer) - специальных

объектов для связи контроллов на XAML/C# с нативными контроллами транслируют визуальные компоненты Xamarin.Forms в графический интерфейс, специфичный для каждой платформы.

Установка Xamarin

Для разработки кроссплатформенных приложений на Xamarin нам нужна среда разработки. Для Windows такой средой является Visual Studio. То есть, если нашей ОС является Windows, то нам вначале надо установить Visual Studio 2019. При этом можно использовать в том числе и бесплатный выпуск Visual Studio 2019 Community.

При установке Visual Studio 2019 в программе для установщика обязательно надо выбрать пункт "Разработка мобильных приложений на .NET":

После инсталляции мы сможем в меню Help ->About Microsoft Visual Studio увидеть отметку о Xamarin:

Если целевой операционной системой является Mac OS X, то в этом случе нам надо установить Visual Studio for Mac.

Кроме того, для разработки на Mac OS X требуется установить XCode, который доступен в AppStore.

Тема 3.2 Графический интерфейс в Xamarin Forms

В Xamarin.Forms визуальный интерфейс состоит из страниц. Страница представляет собой объект класса Page, она занимает все пространство экрана. То есть то, что мы видим на экране мобильного устройства - это страница. Приложение может иметь одну или несколько страниц.

Страница в качестве содержимого принимает один из контейнеров компоновки, в который в свою очередь помещаются стандартные визуальные элементы типа кнопок и текстовых полей, а также другие элементы компоновки.

Возьмем созданный в прошлой теме проект HelloApp (или создадим новый). По умолчанию весь интерфейс создается в классе App, который располагается в файле App.xaml.cs и который представляет текущее приложение:

Его код по умолчанию:

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

namespace HelloApp
{
    public partial class App : Application
    {
        public App()
        {
```

```
            InitializeComponent();
            MainPage = new MainPage();
        }

        protected override void OnStart()
        {
            // Handle when your app starts
        }

        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }

        protected override void OnResume()
        {
            // Handle when your app resumes
        }
    }
}
```

Работа класса App начинается с конструктора, где сначала вызывается метод InitializeComponent(), который выполняет инициализацию объекта, а потом устанавливается свойство MainPage. Через это свойство класс App устанавливает главную страницу приложения. В данном случае она определяется классом HelloApp.MainPage, то есть тем классом, который определен в файлах MainPage.xaml и MainPage.xaml.cs.

Но данный путь не единственный. Xamarin.Forms позволяет создавать визуальный интерфейс как с помощью кода C#, так и декларативным путем с помощью языка xaml, аналогично html, либо комбинируя эти подходы.

Создание интерфейса из кода C#

Добавим в проект HelloApp обычный класс на языке C#, который назовем StartPage.

И определим в этом классе следующее содержимое:

```
using Xamarin.Forms;

namespace HelloApp
{
    class StartPage : ContentPage
    {
        public StartPage()
        {
            Label header = new Label() { Text = "Привет из Xamarin Forms" };
            this.Content = header;
```

```
        }
    }
}
```

Данный класс представляет страницу, поэтому наследуется от класса ContentPage. В конструкторе создается метка с текстом, которая задается в качестве содержимого страницы (this.Content = header).

Чтобы обозначить StartPage в качестве стартовой страницы, изменим класс App:

```
using Xamarin.Forms;

namespace HelloApp
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new StartPage();
        }
        protected override void OnStart()
        {
            // Handle when your app starts
        }

        protected override void OnSleep()
        {
            // Handle when your app sleeps
        }

        protected override void OnResume()
        {
            // Handle when your app resumes
        }
    }
}
```

Теперь свойство MainPage указывает на только что созданную страницу StartPage.

Также стоит отметить, что в Visual Studio есть готовый шаблон для добавления новых классов страниц с простейшим кодом. Так, чтобы добавить новую страницу, надо при добавлении нового элемента выбрать шаблон Content Page (C#):

Данный класс добавляется в главный проект решения (в данном случае это HelloApp).

Добавленный класс страницы будет иметь следующий код:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection.Emit;
using System.Text;

using Xamarin.Forms;

namespace HelloApp
{
    public class Page1 : ContentPage
    {
        public Page1()
        {
            Content = new StackLayout
            {
                Children = {
                    new Label { Text = "Hello Page" }
                }
            };
        }
    }
}
```

Этот класс также будет наследоваться от базового класса ContentPage и будет иметь практически ту же самую организацию, что и выше созданный класс MainPage.

И также в классе приложения мы можем установить эту страницу в качестве стартовой:

```
using Xamarin.Forms;

namespace HelloApp
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();
            MainPage = new Page1();
        }
        //.....
    }
}
```

Тема 3.3 Контейнеры компоновки

В Xamarin мы можем использовать ряд элементов. Их объединяет то, что все они унаследованы от общего класса View и поэтому наследуют ряд общих свойств.

Кроме обычных элементов типа кнопок и текстовых полей, в Xamarin Forms также имеются контейнеры, которые позволяют скомпоновать содержимое, расположить его определенным образом.

Для определения содержимого страницы класс страницы ContentPage имеет свойство Content. По умолчанию этому свойству присваивается один элемент Label.

Но свойство Content имеет ограничение - для него можно установить только один элемент. И чтобы помещать на страницу сразу несколько элементов, нам надо исользовать один из элементов компоновки. Элемент компоновки представляет класс, который наследуется от базового класса Layout<T>:

StackLayout
AbsoluteLayout
RelativeLayout
Grid
FlexLayout

Все элементы компоновки имеют свойство Children, позволяющее задать или получить вложенные элементы.

Например, установка вложенных элементов в коде C#:

```
public class MainPage : ContentPage
{
   public MainPage()
   {
      Label label1 = new Label() { Text = "Первая метка" };
      Label label2 = new Label() { Text = "Вторая метка" };

      StackLayout stackLayout = new StackLayout()
      {
         Children = {label1, label2}
      };

      this.Content = stackLayout;
   }
}
```

Аналогично в XAML мы могли бы написать:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
```

```
            x:Class="HelloApp.MainPage">

  <StackLayout x:Name="stackLayout">
    <StackLayout.Children>
      <Label Text="Первая метка" />
      <Label Text="Вторая метка" />
    </StackLayout.Children>
  </StackLayout>
</ContentPage>
```

Либо даже сократить:

```
<StackLayout x:Name="stackLayout">
    <Label Text="Первая метка" />
    <Label Text="Вторая метка" />
</StackLayout>
```

Поскольку коллекция Children представляет собой обычный список, то он поддерживает операции по управлению элементами. В частности, мы можем динамически добавить новые элементы. Например:

```
stackLayout.Children.Add(new Label { Text = "Третья метка" });
```

Хотя в данном случае коллекция элементов рассматривалась на примере контейнера StackLayout, но тот же принцип работы с элементами будет характерен и для других элементов компоновки.

Тема 3.4 Элементы в Xamarin и их свойства

Отступы

Для установки отступов у элементов применяются свойства Margin и Padding. Свойство Margin определяет внешний отступ элемента от других элементов или контейнера. А свойство Padding устанавливает внутрение отступы - от внутреннего содержимого элемента до его границ. Оба этих свойства представляют структуру Thickness.

Например, зададим оба отступа в XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
    <StackLayout Padding="60">
        <BoxView Color="Blue" Margin="50" HeightRequest="100" />
        <BoxView Color="Red"  Margin="50" HeightRequest="100" />
    </StackLayout>
</ContentPage>
```

Установка в коде C#:

```
public class MainPage : ContentPage
{
   public MainPage()
   {
      var stackLayout = new StackLayout
      {
         Padding = new Thickness(60),
         Children = {
            new BoxView { Color = Color.Blue, Margin = new Thickness (50) },
            new BoxView { Color = Color.Red, Margin = new Thickness (50) }
         }
      };
      Content = stackLayout;
   }
}
```

Кроме того, структура Thickness имеет свойства Left, Top, Right, Bottom, поэтому задать ее определение мы можем тремя способами:

Указать одно значение для отступов со всех сторон

Указать два значения для отступов по горизонтали (слева и справа) и вертикали (сверху и снизу)

Указать четыре значение для отступов для каждой из сторон

Установка в коде:

```
public class MainPage : ContentPage
{
   public MainPage()
   {
      var stackLayout = new StackLayout
      {
         Padding = new Thickness(0, 20, 0, 0),
         Children = {
            new BoxView { Color = Color.Green, Margin = new Thickness (20) },
            new BoxView { Color = Color.Blue, Margin = new Thickness (10, 25) },
            new BoxView { Color = Color.Red, Margin = new Thickness (0, 20, 15, 5) }
         }
```

```
        };
        Content = stackLayout;
    }
}
```

Определение в XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
    <StackLayout Padding="0,20,0,0">
        <BoxView Color="Green" Margin="20" />
        <BoxView Color="Blue" Margin="10, 15" />
        <BoxView Color="Red" Margin="0, 20, 15, 5" />
    </StackLayout>
</ContentPage>
```

В отношении отступ надо учитывать следующую вещь: так как на устройствах с iOS верхняя панель занимает 20 единиц, то для iOS желательно устанавливать отступ сверху размером как минимум в 20 единиц, чтобы текст метки не налазил на панель.

В этом случае мы можем глобально установить отступ сверху для всех платформ в 20 или более единиц. Либо можно задать отступ непосредственно только для iOS:

```
public class MainPage : ContentPage
{
    public MainPage()
    {
        Label header = new Label() { Text = "Привет из Xamarin Forms" };
        this.Content = header;
        if(Device.RuntimePlatform == Device.iOS)
            Padding = new Thickness(0, 20, 0, 0);
    }
}
```

В данном случае с помощью значения Device.RuntimePlatform мы можем получить текстовую метку текущей платформы (например, "iOS", "Android" и т.д.), на которой запущено приложение, и сравнить ее. Константа "Device.iOS" фактически хранит значение "iOS". То есть, если текущая платформа - iOS, установить определенный отступ.

Выравнивание по горизонтали и вертикали

Все элементы, используемые при создании интерфейса, наследуются от класса View, который определяет два свойства HorizontalOptions и VerticalOptions. Они управляют выравнивание элемента соответственно по горизонтали и по вертикали.

В качестве значения они принимают структуру LayoutOptions. Данная структура имеет ряд свойств, которые хранят объекты опять же LayoutOptions:

Start: выравнивание по левому краю (выравнивание по горизонтали) или по верху (выравнивание по вертикали)

Center: элемент выравнивается по центру

End: выравнивание по правому краю (выравнивание по горизонтали) или по низу (выравнивание по вертикали)

Fill: элемент заполняет все пространство контейнера

StartAndExpand: аналогичен опции Start с применением растяжения

CenterAndExpand: аналогичен опции Center с применением растяжения

EndAndExpand: аналогичен опции End с применением растяжения

FillAndExpand: аналогичен опции Fill с применением растяжения

Зададим выравнивание в коде C#:

```
1
2
3

Label header = new Label() { Text = "Привет из Xamarin Forms" };
header.VerticalOptions = LayoutOptions.Center;
header.HorizontalOptions = LayoutOptions.Center;
```

Аналог в xaml:

```
1

<Label Text="Привет из Xamarin Forms" VerticalOptions="Center" HorizontalOptions="Center" />
```

Выравнивание текста внутри элемента

Выравнивание текста по горизонтали и вертикали задается с помощью свойств HorizontalTextAlignment и VerticalTextAlignment соответственно. В качестве значения эти свойства принимают одну из констант перечисления TextAlignment:

Start: текст выравнивается по левому краю по горизонтали или по верху по вертикали

Center: текст выравнивается по центру

End: текст выравнивается по правому краю по горизонтали или по низу по вертикали

Пример на c#:

```
Label header = new Label() { Text = "Привет из Xamarin Forms" };
header.HorizontalTextAlignment = TextAlignment.Center;
header.VerticalTextAlignment = TextAlignment.Center;
```

В xaml:

```
<Label Text="Привет из Xamarin Forms" VerticalTextAlignment="Center" HorizontalTextAlignment="Center" />
```

Работа с цветом

За установку цвета фона и текста элементов отвечают свойства BackgroundColor и TextColor соответственно. В качестве значение они принимают структуру Color:

```
Label header = new Label() { Text = "Привет из Xamarin Forms" };
header.HorizontalTextAlignment = TextAlignment.Center;
header.VerticalTextAlignment = TextAlignment.Center;

header.BackgroundColor = Color.Teal;
header.TextColor = Color.Red;
```

Или в XAML:

```
<Label Text="Привет из Xamarin Forms" HorizontalTextAlignment="Center" VerticalTextAlignment="Center"
    BackgroundColor="Blue" TextColor="Yellow" />
```

Кроме встроенных констант типа Color.Red также для установки цвета мы можем указать и другие значения, используя структуру Color. Для этого нам надо передать значение для одной из компонент красного, зеленого, синего цветов, смесь которых даст финальный цвет. Передаваемое значение имеет тип double и должно находиться в диапазоне от 0.0 до 1.0.

Чтобы передать эти значения, можно использовать один из конструкторов структуры Color:

new Color(double grayShade): устанавливает тон серого цвета

new Color(double r, double g, double b): устанавливает компоненты красного, зеленого и синего

new Color(double r, double g, double b, double a): добавляет еще один параметр - a, который передает прозрачность и имеет значение от 0.0 (полностью прозрачный) до 1.0 (не прозрачный)

Использование конструктора:

```
Label header = new Label() { Text = "Привет из Xamarin Forms" };
header.BackgroundColor = new Color(0.9, 0.9, 0.8); //rgb
```

Также для установки цвета мы можем использовать ряд статических методов:

Color.FromHex(string hex): возвращает объект Color, созданный по переданному в качестве параметра шестнадцатеричному значению

Color.FromRgb(double r, double g, double b): возвращает объект Color, для которого также устанавливаются компоненты красного, зеленого и синего

Color.FromRgb(int r, int g, int b): аналогичен предыдущей версии метода, только теперь компоненты красного, зеленого и синего имеют целочисленные значения от 0 до 255

Color.FromRgba(double r, double g, double b, double a): добавляет параметр прозрачности со значением от 0.0 (полностью прозрачный) до 1.0 (не прозрачный)

Color.FromRgba(int r, int g, int b, int a): добавляет параметр прозрачности со значением от 0 (полностью прозрачный) до 255 (не прозрачный)

Color.FromHsla(double h, double s, double l, double a): устанавливает последовательно параметры h (hue - тон цвета), s (saturation - насыщенность), l (luminosity - яркость) и прозрачность.

Например:

```
Label header = new Label() { Text = "Привет из Xamarin Forms" };
header.TextColor = Color.FromRgba(255, 0, 0, 160);
```

В xaml мы можем задавать цвет с помощью шестнадцатеричных значений также, как в HTML/CSS:

```
<Label Text="Привет из Xamarin Forms" BackgroundColor="#a7a7aa" TextColor="Red" />
```

Тема 3.5 Платформа-зависимый код

Платформа-зависимый код

Несмотря на то, что проект Xamarin Forms позволяет обеспечить единую логику приложения и единый графический интерфейс, иногда возникает необходимость подстроить некоторые аспекты под конкретную операционную систему. Одним из механизмов, который позволяет это сделать, представляет класс Device.

Свойство RuntimePlatform

Прежде всего, свойство RuntimePlatform класса Device позволяет получить платформу, на которой запущено приложение. В качестве значения свойство хранит название платформы. В частности, это может быть одна из встроенных констант класса Device:

Device.Android
Device.iOS
Device.UWP
Device.WPF
Device.Tizen
Device.GTK
Device.macOS

Например, определим в коде метку, текст которой зависит от текущей пдатформы:

```
Label appLabel = new Label { HorizontalTextAlignment = TextAlignment.Center};
switch (Device.RuntimePlatform)
{
   case Device.Android:
      appLabel.Text = "Android";
      break;
```

```
        case Device.iOS:
            appLabel.Text = "iOS";
            break;
        case Device.UWP:
            appLabel.Text = "UWP";
            break;
        case Device.WPF:
            appLabel.Text = "WPF";
            break;
        case Device.GTK:
            appLabel.Text = "GTK";
            break;
        default:
            appLabel.Text = "Undefined";
            break;
    }
```

Свойство Idiom

Свойство Idiom класса Device позволяет проверить тип устройства. В качестве значения оно принимает одно из значений перечисления TargetIdiom:

TargetIdiom.Phone: устройства iPhone, iPod и Android с шириной менее 600 dip

TargetIdiom.Tablet: устройства iPad, Windows и Android с шириной более 600 dip

TargetIdiom.Desktop: настольные компьютеры

TargetIdiom.TV: телевизоры на платформе Tizen

TargetIdiom.Watch: часы на платформе Tizen

TargetIdiom.Unsupported: не поддерживается

В зависимости от результатов проверки значения свойства можно выполнить тот или иной набор инструкций:

```
if(Device.Idiom == TargetIdiom.Phone)
{
    // код для смартфонов
}
else if (Device.Idiom == TargetIdiom.Tablet)
{
    // код для планшетов
}
else if (Device.Idiom == TargetIdiom.Desktop)
{
    // код для настольных компьютеров
}
```

Тема 3.6 Ресурсы и стили. Привязка в Xamarin

Ресурсы и стили. Привязка в Xamarin

Привязка данных (data binding) является одним из ключевых моментов платформы Xamarin Forms.

Привязка данных состоит из двух компонентов: источника (source) и цели (target). Привязка осуществляется от свойства источника к свойству цели. И когда происходит изменение источника, механизм привязки автоматически обновляет также и цель.

Цель привязки должна представлять объект BindableObject, а свойство, к которому осуществляется привязка, должно быть свойством BindableProperty. Поскольку большинство визуальных элементов в Xamarin Forms наследуются от класса BindableObject, то в качестве цели привязки будут, как правило, выступать визуальные элементы.

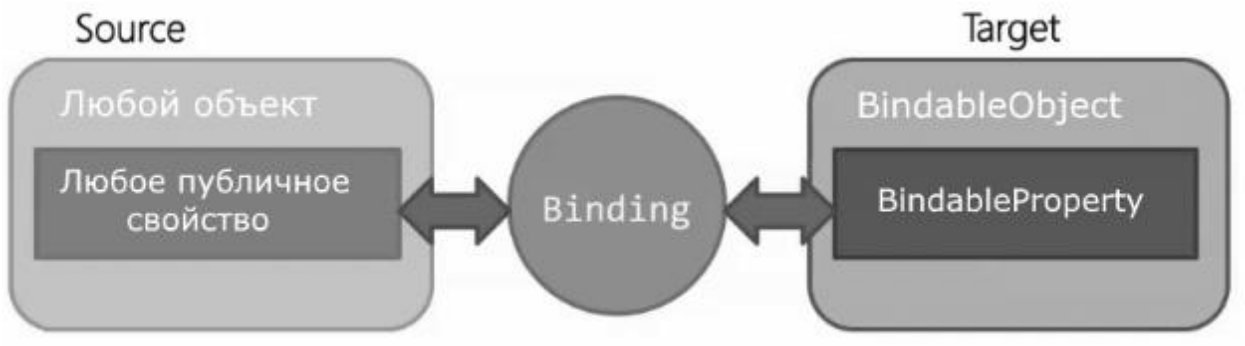
А вот источником привязки может выступать любой объект языка C#. Однако, надо понимать, что цель привязки должна автоматически изменяться при изменении источника, поэтому нам нужно извещать систему о изменении свойств источника привязки. В Xamarin, да и вообще на платформе .NET, в качестве подобного механизма извещения выступает интерфейс INotifyPropertyChanged. То есть нужно реализовать данный интерфейс в объекте-источнике.

Объект BindableObject как раз реализует INotifyPropertyChanged. Поэтому если источником привязки является стандартный визуальный элемент из Xamarin Forms, то автоматически будет изменяться и цель привязки. Ну если в качестве источника выступает не BindableObject, а какой-нибудь объект простого класса C#, то, как писалось выше, этот класс должен реализовать INotifyPropertyChanged.

Для установки привязки у объекта цели устанавливается свойство BindingContext. В качестве значения оно принимает источник привязки.

Рассмотрим на примерах, как устанавливается привязка в коде C# и в коде XAML.

Привязка в коде C#



Для привязки в коде после установки у цели привязки свойства BindingContext необходимо также у цели привязки вызвать метод SetBinding(), который свяжет свойство цели привязки со свойством источника.

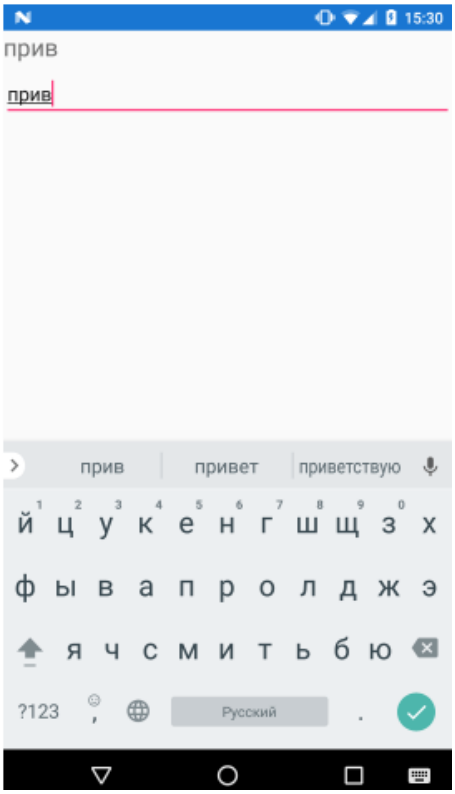
```
public partial class MainPage : ContentPage
{
   public MainPage()
```

```
    {
        Label label = new Label
        {
            FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label))
        };
        Entry entry = new Entry();

        // Устанавливаем привязку
        // источник привязки - entry, цель привязки - label
        label.BindingContext = entry;
        // Связываем свойства источника и цели
        label.SetBinding(Label.TextProperty, "Text");

        StackLayout stackLayout = new StackLayout()
        {
            Children = { label, entry}
        };
        Content = stackLayout;
    }
}
```

Выражение label.SetBinding(Label.TextProperty, "Text") устанавливает привязку свойства TextProperty у цели - элемента label к свойству Text.

Причем, что важно у объекта цели label привязка устанавливается именно у свойства BindableProperty, которым является TextProperty, а не просто для свойства Text.

В итоге при вводе данных в текстовое поле будет изменяться значение свойства Text у объекта entry. А это изменение автоматически скажется на объекте label и изменит его свойство TextProperty.



Привязка в XAML

Аналогично можно установить привязку и в xaml-коде.

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
  <StackLayout>
    <Label x:Name="label"
          BindingContext="{x:Reference Name=entryBox}"
          Text="{Binding Path=Text}"
          FontSize="Large" />
    <Entry x:Name="entryBox" />
  </StackLayout>
</ContentPage>
```

В XAML действует тот же принцип. Для установки привязки для объекта цели задается свойство BindingContext. Но теперь его значение имеет другую форму: {x:Reference Name=entryBox}. После расширения разметки x:Reference идет свойство Name и его значение - имя элемента источника. То есть в данном случае источником выступает объект с именем entryBox.

Далее устанавливается сама привязка к свойству:

```
1       Text="{Binding Path=Text}"
```

Здесь также устанавливается привязка к свойству Text. Выражение привязки заключается в фигурные скобки и состоит из слова Binding, после которого идет свойство Path и его значение - свойство объекта источника. То есть свойство Text объекта label привязано к свойству Text объекта entryBox.

Тема 3.7 ListView и работа с данными

ListView и работа с данными

https://www.youtube.com/watch?v=Et6KCB1zeX8

ListView представляет очень мощный элемет управления Xamarin Forms, который позволяет отображать список объектов и при этом кастомизировать их отображение.

ListView связывается с набором данных через свойство ItemsSource, которое принимает объект IEnumerable<T>.

Создание ListView в коде C#

Определим в коде C# объект ListView, который выводит массив строк:

```
using Xamarin.Forms;

namespace HelloApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            Label header = new Label
            {
                Text = "Список моделей",
                FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Label)),
                HorizontalOptions = LayoutOptions.Center
            };

            string[] phones = new string[] {"iPhone 7", "Samsung Galaxy S8", "Huawei P10", "LG G6"};

            ListView listView = new ListView();
            // определяем источник данных
            listView.ItemsSource = phones;
            this.Content = new StackLayout { Children = { header, listView } };
        }
    }
}
```

При выводе каждого объекта ListView по умолчанию вызывает для него метод ToString(), поэтому мы увидим строковое отображение объекта.

ListView в XAML

Аналогичный массив в XAML мы могли бы вывести следующим образом:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
```

```
            xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
            x:Class="HelloApp.MainPage">
    <ContentPage.Resources>
        <ResourceDictionary>
            <x:Array x:Key="phones" Type="{x:Type x:String}">
                <x:String>iPhone 7</x:String>
                <x:String>Google Pixel 5</x:String>
                <x:String>Huawei P10</x:String>
                <x:String>Xiaomi Mi6</x:String>
            </x:Array>
        </ResourceDictionary>
    </ContentPage.Resources>
    <StackLayout>
        <Label Text="Список моделей" HorizontalOptions="Center"
FontSize="Large" />
        <ListView x:Name="phonesList" ItemsSource="{StaticResource
Key=phones}" />
    </StackLayout>
</ContentPage>
```

Сочетание XAML и C#. Привязка к списку

Нередко источник данных для ListView определяется в коде C#. И в этом случае мы можем использовать привязку к источнику данных. Например, в коде C# определим массив, к которому будет производиться привязка:

```
using Xamarin.Forms;

namespace HelloApp
{
    public partial class MainPage : ContentPage
    {
        public string[] Phones { get; set; }

        public MainPage()
        {
            InitializeComponent();
            Phones = new string[] { "iPhone 8", "Samsung Galaxy S9", "Huawei
P10", "LG G6" };
            this.BindingContext = this;
        }
    }
}
```

Стоит отметить, что источник данных определен как публичное свойство Phones. Другой важный момент - установка контекста привязки страницы: this.BindingContext = this;. Таким образом, в XAML мы можем обратиться ко всем публичным свойствам страницы.

В коде XAML пропишем привязку к этому свойству:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
  <StackLayout>
    <Label Text="Список моделей" HorizontalOptions="Center" FontSize="Large" />
    <ListView x:Name="phonesList" ItemsSource="{Binding Phones}" />
  </StackLayout>
</ContentPage>
```

Привязка к выбранному элементу

В примере выше мы вполне могли бы обойтись без обработки события и просто установить привязку элемента Label к выбранному объекту списка:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
  <ContentPage.Resources>
    <ResourceDictionary>
      <x:Array x:Key="phones" Type="{x:Type x:String}">
        <x:String>iPhone 7</x:String>
        <x:String>Google Pixel 5</x:String>
        <x:String>Huawei P10</x:String>
        <x:String>Xiaomi Mi6</x:String>
      </x:Array>
    </ResourceDictionary>
  </ContentPage.Resources>
  <StackLayout>
    <Label Text="{Binding Source={x:Reference Name=phonesList}, Path=SelectedItem}"
           HorizontalOptions="Center" FontSize="Large" />
    <ListView x:Name="phonesList" ItemsSource="{StaticResource phones}" />
  </StackLayout>
</ContentPage>
```

Выбор элемента

Когда пользователь нажимает на элемент списка, он выделяется, а у ListView срабатывают два события: ItemTapped и ItemSelected. Между ними есть различия. Так, повторное нажатие на один и тот же элемент не вызовет повторного события ItemSelected, так как элемент остается выбанным. А вот событие ItemTapped будет срабатывать именно столько раз сколько пользователь нажал на него, даже повторно. Также событие ItemSelected будет вызвано, если с элемента будет снято выделение.

ItemSelected

Обработаем выделение элемента. Для этого определим следующий код в XAML:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
    <StackLayout>
        <Label x:Name="selected" HorizontalOptions="Center" FontSize="Large" />
        <ListView x:Name="phonesList" ItemSelected="phonesList_ItemSelected" />
    </StackLayout>
</ContentPage>
```

А в файле кода MainPage.xaml.cs определим источник данных и обработчик события ItemSelected:

```
using Xamarin.Forms;

namespace HelloApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
            string[] phones = new string[] {"iPhone 7", "Samsung Galaxy S8", "Huawei P10", "LG G6"};
            phonesList.ItemsSource = phones;
        }
        private void phonesList_ItemSelected(object sender, SelectedItemChangedEventArgs e)
        {
            if(e.SelectedItem!=null)
                selected.Text = e.SelectedItem.ToString();
        }
    }
}
```

При выборе элемента мы можем получить выбранный элемент через свойство SelectedItem объекта SelectedItemChangedEventArgs, который передается в качестве параметра. Так как событие ItemSelected может срабатывать и при снятии выделения элемента, то в обработчике необходимо вначале проверить значение e.SelectedItem на null.

Поскольку список содержит массив строк, то каждый его элемент представляет строку, которую мы можем получить с помощью метода e.SelectedItem.ToString().

Как можно увидеть на скриншоте, выбранный элемент выделяется цветом. Однако выделение на Android оранжевым цветом элементов списка, которые имеют по умолчанию розовые буквы, наверное, не самое удачное дизайнерское решение. Кроме того, возможно выделение в прицнипе неприемлимо, и в этом случае мы его можем сразу же снять после выбора элемента:

```
private void phonesList_ItemSelected(object sender, SelectedItemChangedEventArgs e)
{
   if(e.SelectedItem!=null)
      selected.Text = e.SelectedItem.ToString();
   ((ListView)sender).SelectedItem = null;
}
```

ItemTapped

Похожим образом можно обработать событие ItemTapped. Определим в XAML следующий код:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage">
   <StackLayout>
      <Label x:Name="selected" HorizontalOptions="Center" FontSize="Large" />
      <ListView x:Name="phonesList" ItemTapped="PhonesList_ItemTapped" />
   </StackLayout>
</ContentPage>
```

А в файле кода MainPage.xaml.cs определим источник данных и обработчик события ItemTapped:

```
using Xamarin.Forms;

namespace HelloApp
{
   public partial class MainPage : ContentPage
   {
      public string[] Phones { get; set; }

      public MainPage()
      {
         InitializeComponent();
         string[] phones = new string[] { "iPhone 7", "Samsung Galaxy S8", "Huawei P10", "LG G6" };
         phonesList.ItemsSource = phones;
```

```
        }

        private void PhonesList_ItemTapped(object sender,
ItemTappedEventArgs e)
        {
            if (e.Item != null)
                selected.Text = e.Item.ToString();
            ((ListView)sender).SelectedItem = null;
        }
    }
}
```

Тема 3.8 Навигация. Типы страниц

Навигация. Типы страниц

Для поддержки навигации на каждой странице Page в Xamarin Forms определено свойство Navigation. Это свойство предтавляет интерфейс INavigation, в котором есть следующие методы:

Task PushAsync(Page page)

Task PushModalAsync(Page page)

В качестве параметра здесь передается объект Page - страница, на которую надо осуществить переход.

Второй метод имеет в своем названии слово "Modal" и осуществляет переход на модальную страницу.Обычно модальные страницы используются, когда приложению нужно получить некоторую информацию от пользователя. При этом до получения информации нельзя возвращаться на предыдущую страницу.

Также имеются методы для возврата на предыдущую страницу:

Task<Page> PopAsync()

Task<Page> PopModalAsync()

Итак, рассмотрим простейший пример навигации, используя все эти методы. Создадим новый проект по типу Xamarin Forms Portable.

Так как наше приложение будет производить навигацию, то есть переходы между страницами, то вначале добавим эти вспомогательные страницы.

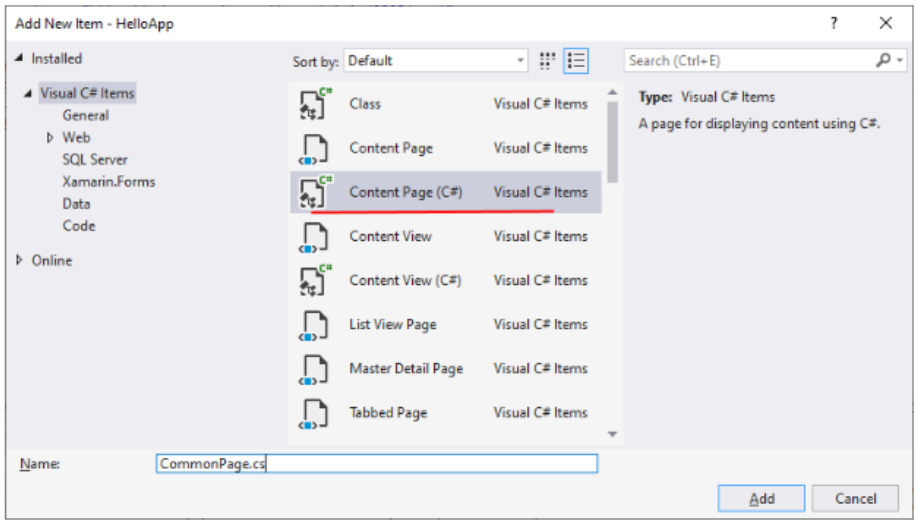
Итак, добавим в проект новый элемент ContentPage (C#), который назовем CommonPage.cs:

Определим в этом классе следующий код:

```
using System;
using Xamarin.Forms;

namespace HelloApp
{
```

```
public class CommonPage : ContentPage
{
    public CommonPage()
    {
        Title = "Common Page";
        Button backButton = new Button
        {
            Text = "Назад",
            HorizontalOptions = LayoutOptions.Center,
            VerticalOptions = LayoutOptions.Center
        };
        backButton.Clicked += BackButton_Click;
        Content = backButton;
    }
    private async void BackButton_Click(object sender, EventArgs
e)
    {
        await Navigation.PopAsync();
    }
}
}
```

На этой странице определяется единственная кнопка, по нажатию на которую осуществляется переход назад с помощью метода Navigation.PopAsync(). Поскольку метод является асинхронным, то перед ним ставится ключевое слово await, а сам обработчик кнопки помечается с помощью ключевого слова async.



Также стоит отметить, что у страницы устанавливается заголовок через свойство Title.

Подобным образом добавим вторую страницу, которую назовем ModalPage. Определим в ней следующий код:

```
using System;
```

```
using Xamarin.Forms;

namespace HelloApp
{
    public class ModalPage : ContentPage
    {
        public ModalPage()
        {
            Title = "Modal Page";
            Button backButton = new Button
            {
                Text = "Назад",
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.Center
            };
            backButton.Clicked += BackButton_Click;

            Content = backButton;
        }
        private async void BackButton_Click(object sender, EventArgs
e)
        {
            await Navigation.PopModalAsync();
        }
    }
}
```

Данная страница выглядит во многом аналогично предыдущий за тем исключением, что переход назад здесь производится с помощью метода Navigation.PopModalAsync(). То есть эта страница будет модальной.

И изменим код главной страницы MainPage:

```
using System;
using Xamarin.Forms;

namespace HelloApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            Title = "Main Page";
            Button toCommonPageBtn = new Button
            {
                Text = "На обычную страницу",
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.CenterAndExpand
```

```
            };
            toCommonPageBtn.Clicked += ToCommonPage;

            Button toModalPageBtn = new Button
            {
                Text = "На модальную страницу",
                HorizontalOptions = LayoutOptions.Center,
                VerticalOptions = LayoutOptions.CenterAndExpand
            };
            toModalPageBtn.Clicked += ToModalPage;

            Content = new StackLayout { Children = { toCommonPageBtn, toModalPageBtn } };
        }

        private async void ToModalPage(object sender, EventArgs e)
        {
            await Navigation.PushModalAsync(new ModalPage());
        }
        private async void ToCommonPage(object sender, EventArgs e)
        {
            await Navigation.PushAsync(new CommonPage());
        }
    }
}
```

Итак, здесь две кнопки. Одна производит переход на CommonPage с помощью метода Navigation.PushAsync(), а другая осуществляет переход на ModalPage посредством вызова Navigation.PushModalAsync()

Но чтобы с MainPage можно было бы переходить к CommonPage или ModalPage, нам надо в качестве главной страницы использовать объект NavigationPage, а не ContentPage, как установлено по умолчанию. Поэтому перейдем к классу App и изменим код установки главной страницы:

```
using Xamarin.Forms;

namespace HelloApp
{
    public partial class App : Application
    {
        public App()
        {
            InitializeComponent();

            MainPage = new NavigationPage(new MainPage());
        }
```

```
        protected override void OnStart() { }

        protected override void OnSleep() { }

        protected override void OnResume() { }
    }
}
```

И чтобы использовать механизм навигации в приложении, необходимо обернуть объект MainPage в NavigationPage:

```
1                MainPage = new NavigationPage(new MainPage());
```

Иначе мы не сможем использовать навигацию в приложении. Сам же класс NavigationPage визуально только добавляет заголовок страницы, который устанавливается с помощью свойства Title и который отображается рядом с иконкой приложения. Хотя в зависимости от операционной системы отображение заголовка может отличаться.

Итак, запустим приложение:

Попробуем перейти на обычную страницу, нажав на первую кнопку:

Как из скриншота можно увидеть, что в данном случае даже не нужна кнопка для возвращаения назад, так как мы можем вернуться через стрелку рядом с заголовком.

Вернемся назад и нажмем на вторую кнопку для перехода к ModalPage:

Так как для перехода к этой странице использовался метод Navigation.PushModalAsync(), то данная страница будет модальной. Основное отличие модальной страницы в визуальном плане состоит в отстутсвии заголовка, несмотря на то, что в коде мы устанавливаем свойство Title у ModalPage.

Ну и для возврата с этой страницы назад нам надо использовать метод Navigation.PopModalAsync()

При в реальности дл создания модальной страницы необязательно создавать специальный класс страницы, как здесь класс ModalPage. Так, мы могли бы использовать в качестве модальной и CommonPage.

Анимация в постраничных переходах

Все ранее расмотренные методы навигации в качестве второго параметра принимают логическое значение:

```
Task PushAsync(Page page, bool animated)
Task PushModalAsync(Page page, bool animated)
Task<Page> PopAsync(bool animated)
Task<Page> PopModalAsync(bool animated)
```

И если оно равно true, то при переходе на другую страницу осуществляется анимация, если, конечно, целевая платформа поддерживает подобную анимацию.

По умолчанию анимация включена, поэтому, чтобы ее отключить, нам надо использовать одну из этих версий, передавая значение false.

Тема 3.9. MVVM. Базы данных SQLite

MVVM. Базы данных SQLite

Кроме обычных файлов для хранения данных мы можем использовать локальные базы данных. Наиболее популярной системой баз данных для локальных приложений является SQLite, поэтому рассмотрим общие принципы работы с этой СУБД.

Для работы с SQLite можно использовать разные подходы и библиотеки. Рекомендуемой библиотекой является SQLite.NET, которая представляет простое ORM-решение (Object Relational Mapping) для Xamarin. Она позволяет работать с базой данных как с хранилищем объектов и манипулировать данными как объектами стандартных классов C# без использования выражений на языке SQL. И вначале добавим в главный проект в узел References с помощью менеджера NuGet эту библиотеку sqlite-net-pcl:

Следует отметить, что в Nuget можно найти много похожих библиотек с подобным названием. У нужной библиотеки будут следующие данные:

Автор: SQLite-net

Id: sqlite-net-pcl

Вначале определим класс, объекты которого будут храниться в базе данных. Добавим в главный проект следующий класс Friend:

```
using SQLite;

namespace HelloApp
{
    [Table("Friends")]
    public class Friend
    {
        [PrimaryKey, AutoIncrement, Column("_id")]
        public int Id { get; set; }

        public string Name { get; set; }
        public string Email { get; set; }
        public string Phone { get; set; }
    }
}
```

Класс Friend выступает в качестве модели приложения. Этот класс использует атрибуты, которые позволяют настроить его отображение на таблицу в бд. Для настройки мы можем использовать следующие атрибуты:

[PrimaryKey]: применяется к свойству типа int и указывает, что столбец в таблице, который соответствует этому свойству, будет выполнять роль первичного ключа. Составные ключи не поддерживаются

[AutoIncrement]: применяется к свойству типа int и указывает, что столбец в таблице, который соответствует этому свойству, будет инкрементировать значение на единицу при добавлении нового элемента

[Column(name)]: задает сопоставление свойства со столбцом в таблице, который имеет имя name

[Table(name)]: устанавливает название таблицы, которая будет соотвествовать данному классу

[MaxLength(value)]: устанавливает максимальную длину для строковых свойств

[Ignore]: указывает, что свойство будет игнорироваться. Это может быть полезно, если значение данного свойства не надо хранить в базе данных, и данное свойство не должно сопоставляться со столбцами из таблицы в бд

[Unique]: гарантирует, что столбец, который соответствует свойству с этим атрибутом, будет иметь уникальные неповторяющиеся значения

При запросах к базе данных будет происходить автоматическое сопоставление типов данных из SQLite с типами данных из C#. Сопоставление типов можно описать следующей таблицей:

C#	SQLite
int, long	integer, bigint
bool	integer (1 = true)
enum	integer
float	real
double	real
decimal	real
string	varchar, text
DateTime	numeric, text
byte[]	blob

Класс репозитория

Также создадим класс репозитория, через который будут идти все операции с данными:

```
using System.Collections.Generic;
using SQLite;

namespace HelloApp
{
    public class FriendRepository
    {
        SQLiteConnection database;
        public FriendRepository(string databasePath)
        {
```

```
            database = new SQLiteConnection(databasePath);
            database.CreateTable<Friend>();
        }
        public IEnumerable<Friend> GetItems()
        {
            return database.Table<Friend>().ToList();
        }
        public Friend GetItem(int id)
        {
            return database.Get<Friend>(id);
        }
        public int DeleteItem(int id)
        {
            return database.Delete<Friend>(id);
        }
        public int SaveItem(Friend item)
        {
            if (item.Id != 0)
            {
                database.Update(item);
                return item.Id;
            }
            else
            {
                return database.Insert(item);
            }
        }
    }
}
```

В конструкторе класса происходит создание подключения и базы данных (если она отсутствует). В качестве параметра извне будет передаваться путь к базе данных. При желании путь к бд можно определить и в самом конструкторе.

Для всех операций с данными используются методы, определенные в классе SQLiteConnection:

Insert: добавляет объект в таблицу

Get<T>: позволяет получить элемент типа T по id

Table<T>: возвращает все объекты из таблицы

Delete<T>: удаляет объект по id

Update<T>: обновляет объект

Query<T>: выполняет SQL-выражение и возвращает строки из таблицы в виде объектов типа T (относится к выражениям SELECT)

Execute: выполняет SQL-выражение, но ничего не возвращает (относится к операциям. где не надо возвращать результат - UPDATE, INSERT, DELETE)

Если предполагается, что к базе данных может обращаться сразу несколько потоков, то для блокирования одновременных операций с бд в классе репозитории можно использовать блок lock с заглушкой, например:

```
SQLiteConnection database;
static object locker = new object();
//.....................

public int DeleteItem(int id)
{
   lock(locker)
   {
      return database.Delete<Friend>(id);
   }
}
```

Создаваемое подключение будет общим для всего приложения, поэтому изменим файл App.xaml.cs следующим образом:

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

namespace HelloApp
{
   public partial class App : Application
   {
      public const string DATABASE_NAME = "friends.db";
      public static FriendRepository database;
      public static FriendRepository Database
      {
         get
         {
            if (database == null)
            {
               database = new FriendRepository(
                  Path.Combine(

Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData),
DATABASE_NAME));
            }
            return database;
         }
      }
      public App()
      {
         InitializeComponent();
```

```
            MainPage = new NavigationPage(new MainPage());
        }
        protected override void OnStart()
        {
        }
        protected override void OnSleep()
        {
        }
        protected override void OnResume()
        {
        }
    }
}
```

Статический объект репозитория, создаваемый при создании главной страницы приложения, будет доступен из любого места приложения.

Поскольку в нашем приложении мы будем переходить по страницам - к странице добавления или просмотра, то в качестве главной страницы устанавливается объект NavigationPage.

Добавление страниц приложения

Теперь на главной странице MainPage.xaml следующий код:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.MainPage" Title="Список друзей">
  <StackLayout>
    <ListView x:Name="friendsList" ItemsSource="{Binding}"
ItemSelected="OnItemSelected">
      <ListView.ItemTemplate>
        <DataTemplate>
          <ViewCell>
            <ViewCell.View>
              <StackLayout Orientation="Horizontal">
                <Label Text="{Binding Name}" FontSize="Medium" />
              </StackLayout>
            </ViewCell.View>
          </ViewCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
    <Button Text="Добавить" Clicked="CreateFriend" />
  </StackLayout>
</ContentPage>
```

Элемент ListView будет выводить список объектов, а при нажатии на элемент списка, будет срабатывать обработчик OnItemSelected. И также для добавления нового объекта определена кнопка.

И изменим файл кода MainPage.xaml.cs:

```
using System;
using Xamarin.Forms;

namespace HelloApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        protected override void OnAppearing()
        {
            friendsList.ItemsSource = App.Database.GetItems();
            base.OnAppearing();
        }
        // обработка нажатия элемента в списке
        private async void OnItemSelected(object sender,
SelectedItemChangedEventArgs e)
        {
            Friend selectedFriend = (Friend)e.SelectedItem;
            FriendPage friendPage = new FriendPage();
            friendPage.BindingContext = selectedFriend;
            await Navigation.PushAsync(friendPage);
        }
        // обработка нажатия кнопки добавления
        private async void CreateFriend(object sender, EventArgs e)
        {
            Friend friend = new Friend();
            FriendPage friendPage = new FriendPage();
            friendPage.BindingContext = friend;
            await Navigation.PushAsync(friendPage);
        }
    }
}
```

При переходе на любую страницу у нее вызывается метод OnAppearing(), поэтому тут мы можем установить привязку и настроить другие начальные данные.

Остальные оба обработчика - OnItemSelected и CreateFriend предусматривают переход на страницу FriendPage, которая будет отвечать за работу с одним объектом из бд. Через свойство BindingContext мы можем установить полученный объект в качестве контекста страницы, а на самой странице использовать выражения привязки для изменения значений свойств объекта.

Теперь добавим эту страницу FriendPage. В ее коде xaml пропишем следующий интерфейс:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="HelloApp.FriendPage" Title="Информация о друге">
  <StackLayout>
    <Label Text="Имя" />
    <Entry Text="{Binding Name}" />
    <Label Text="Email" />
    <Entry Text="{Binding Email}" />
    <Label Text="Телефон" />
    <Entry Text="{Binding Phone}" />
    <StackLayout Orientation="Horizontal">
      <Button Text="Сохранить" Clicked="SaveFriend" />
      <Button Text="Удалить" Clicked="DeleteFriend" />
      <Button Text="Отмена" Clicked="Cancel" />
    </StackLayout>
  </StackLayout>
</ContentPage>
```

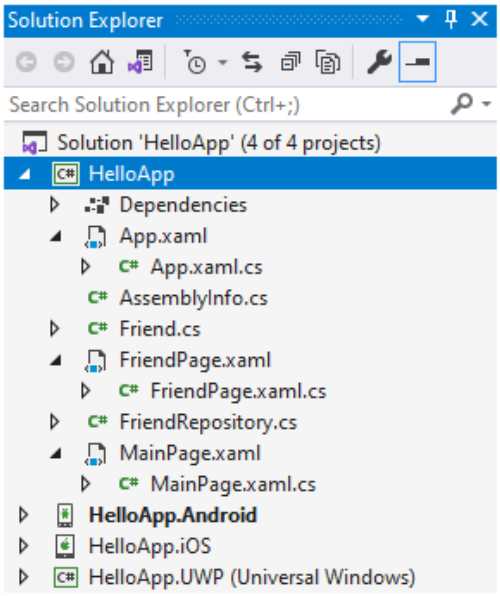
А в файле связанного кода FriendPage.xaml.cs добавим обработчики нажатия кнопок:

```
using System;
using Xamarin.Forms;

namespace HelloApp
{
    public partial class FriendPage : ContentPage
    {
        public FriendPage()
        {
            InitializeComponent();
        }

        private void SaveFriend(object sender, EventArgs e)
        {
            var friend = (Friend)BindingContext;
            if (!String.IsNullOrEmpty(friend.Name))
            {
                App.Database.SaveItem(friend);
            }
            this.Navigation.PopAsync();
        }
        private void DeleteFriend(object sender, EventArgs e)
        {
```

```
            var friend = (Friend)BindingContext;
            App.Database.DeleteItem(friend.Id);
            this.Navigation.PopAsync();
        }
        private void Cancel(object sender, EventArgs e)
        {
            this.Navigation.PopAsync();
        }
    }
}
```

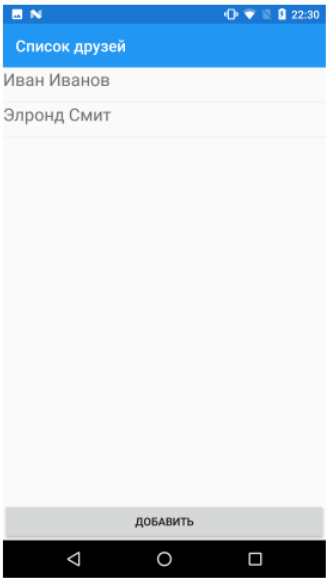
Обработчики используют методы репозитория для сохранения и удаления объекта и после этого осуществляют переход назад на главную страницу.

В итоге главный проект будет выглядеть следующим образом:



Теперь с главной страницы мы можем попасть на страницу добавления и создать там новые объекты:

После добавления все объекты будут отображаться в списке на главной странице:


22:30
Список друзей
Иван Иванов
Элронд Смит
ДОБАВИТЬ

Тестовые задания

1. Что такое Java?

Java — это компьютерный язык программирования, который является параллельным, основанным на классах и объектно-ориентированным. Преимущества объектно-ориентированной разработки программного обеспечения показаны ниже:

- Модульная разработка кода, что приводит к простоте обслуживания и модификации.
- Возможность повторного использования кода.
- Повышена надежность и гибкость кода.
- Улучшенное понимание кода.

2. Каковы концепции ООП?

Объектно-ориентированное программирование (ООП) включает в себя:

- абстракция
- Инкапсуляция
- Полиморфизм
- наследование
- Предопределенные типы должны быть объектами
- Пользовательские типы должны быть объектами
- Операции должны выполняться путем отправки сообщений объектам

3. Упомянуть некоторые особенности Java

Вот некоторые из особенностей, которые играют важную роль в популярности Java:

- Объектно-ориентированный
- Независимая платформа
- Высокая производительность
- Многопоточный
- портативный
- Безопасный

Пример кода для Helloworld в Java показан ниже:

Привет мир

```
public class Helloworld{

public static void main(String args[])
{
   System.out.println("Hello World");
}

}
```

4. Является ли Java на 100% объектно-ориентированным?

Не 100%. Java не удовлетворяет всем условиям ООП (предопределенные типы должны быть объектами), потому что она

использует восемь примитивных типов данных (Boolean, byte, char, int, float, double, long, short), которые не являются объектами.

5. Что такое абстракция?

Абстракция — это процесс отделения идей от конкретных экземпляров и, следовательно, разработки классов с точки зрения их собственной функциональности, а не деталей их реализации. Java поддерживает создание и существование абстрактных классов, которые предоставляют интерфейсы, без фактической реализации всех методов. Техника абстракции направлена на отделение деталей реализации класса от его поведения.

Абстрактный класс Person представлен ниже. У него есть абстрактный метод getName.

Человек абстрактного класса

```
public abstract class Person
{
   public abstract String getName();
}
```

Класс Employee расширяет абстрактный класс Person. Метод getName возвращает атрибут имени сотрудника.

Класс сотрудника

```
public class Employee extends Person
{
   private String name;

   public Employee(String name)
   {
    this.name = name;
   }
   public String getName()
   {
     return this.name;
   }
   public static void main (String args[])
   {
      Employee employee = new Employee("John Wilson");

      System.out.println("Employee's Name "+ employee.getName());

      Person person = new Employee("Thomas Smith");

      System.out.println("Employee-Person's Name "+ person.getName());


   }
}
```

6. Что такое инкапсуляция?

Инкапсуляция предоставляет объектам возможность скрывать свои внутренние характеристики и поведение. Каждый объект предоставляет ряд методов, к которым могут обращаться другие объекты и изменять свои внутренние данные. В Java есть три модификатора доступа: открытый, закрытый и защищенный. Каждый модификатор налагает разные права доступа на другие классы, либо в том же, либо во внешних пакетах. Некоторые из преимуществ использования инкапсуляции перечислены ниже:

- Внутреннее состояние каждого объекта защищено скрытием его атрибутов.
- Это повышает удобство использования и обслуживания кода, поскольку поведение объекта может быть независимо изменено или расширено.
- Это улучшает модульность, предотвращая нежелательное взаимодействие объектов друг с другом.

Вы можете обратиться к нашему руководству здесь для более подробной информации и примеров инкапсуляции.

Пример класса Student, который имеет атрибуты Id и Name, показан в качестве примера для инкапсуляции.

Студенческий класс

```
public class Student{
 private int id;
 private String name;

 public void setId(int id)
 {
  this.id = id;
 }

 public void setName(String name)
 {
  this.name = name;
 }

 public int getId()
 {
  return this.id;
 }

 public String getName()
 {
  return this.name;
 }

public static void main(String args[])
```

```
{
  Student student=new Student();
  student.setId(1034);
  student.setName("David Smith");

  System.out.println("Student id "+ student.getId());
  System.out.println("Student name "+ student.getName());

}

}
```

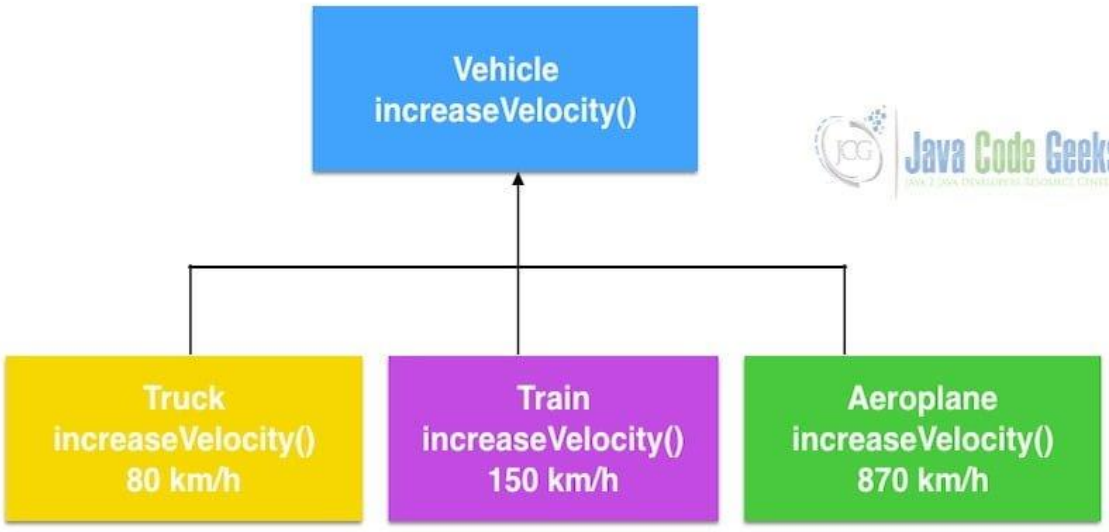
7. В чем различия между абстракцией и инкапсуляцией?

Абстракция и инкапсуляция являются дополнительными понятиями. С одной стороны, абстракция фокусируется на поведении объекта. С другой стороны, инкапсуляция фокусируется на реализации поведения объекта. Инкапсуляция обычно достигается путем сокрытия информации о внутреннем состоянии объекта и, таким образом, может рассматриваться как стратегия, используемая для обеспечения абстракции.

8. Что такое полиморфизм?

Полиморфизм — это способность языков программирования предоставлять один и тот же интерфейс для разных базовых типов данных. Полиморфный тип — это тип, операции которого также могут применяться к значениям некоторого другого типа.

Вы можете увидеть пример ниже, где интерфейс Vehicle имеет метод увеличенияVelocity. Грузовик, поезд и самолет реализуют интерфейс транспортного средства, и метод увеличивает скорость до соответствующей скорости, связанной с типом транспортного средства.



Полиморфизм

9. Какие бывают виды полиморфизма?

В Java существует два типа полиморфизма:

- Полиморфизм времени компиляции (статическое связывание). Перегрузка метода
- Полиморфизм времени исполнения (динамическое связывание) — переопределение метода

Мы можем выполнить полиморфизм перегрузкой метода и переопределением метода.

Время компиляции	**время выполнения**
Методы класса имеют одно и то же имя. Каждый метод имеет разное количество параметров. Может иметь параметры с разными типами и порядком.	у подкласса есть метод с именем как у метода суперкласса. Он имеет количество параметров, тип параметров и тип возвращаемого значения для метода суперкласса.
Перегрузка метода заключается в добавлении к поведению метода. Это может распространяться на поведение метода.	Переопределение метода заключается в изменении поведения метода.
Перегруженные методы не будут иметь одинаковую подпись.	Переопределенные методы будут иметь точно такую же сигнатуру.
Наследование не нужно в этом случае.	Наследование необходимо.

Пример кода для перегрузки метода вычитания класса Calculator показан ниже:

Калькулятор Класс

```
public class Calculator {

public int subtract(int a, int b)
{
   return a-b;
}
public double subtract( double a, double b)
{
 return a-b;
}

public static void main(String args[])
{
```

```
  Calculator calculator = new Calculator();
  System.out.println("Difference of 150 and 12 is " +calculator.subtract(150,12));
  System.out.println("Difference of 15.5 and 15.4 is " +calculator.subtract(15.50,15.4
}}
```

Переопределение метода показано ниже в классе Shape. Shape имеет метод getArea.

Класс формы

```
public class Shape
{
  public void getArea(){System.out.println("Shape Area");}
}
```

Класс Rectangle переопределяет метод getArea, и реализация метода специфична для Rectangle. Аннотация переопределения используется для указания компилятору, что метод переопределен. Читаемость кода улучшена с помощью аннотации.

Класс прямоугольника

```
public class Rectangle extends Shape{

  @Override
  public void getArea()
  {
    System.out.println("Rectangle Area");

  }



  public static void main(String args[])
  {
    Shape shape = new Shape();

    shape.getArea();

    Rectangle rectangle = new Rectangle();

    rectangle.getArea();
  }
}
```

10. Что такое наследование?

Наследование предоставляет объекту возможность получать поля и методы другого класса, называемого базовым классом. Наследование обеспечивает возможность повторного использования кода и может использоваться для добавления дополнительных функций в существующий класс без его изменения.

Пример класса Mammal показан ниже, у которого есть конструктор.

Класс млекопитающих

```
public class Mammal{

 public Mammal()
 {
  System.out.println("Mammal created");
 }

}
```

Класс Man расширяет Mammal, у которого есть конструктор по умолчанию. Пример кода показан ниже.

Мужской класс

```
public class Man extends Mammal{

 public Man()
 {
  System.out.println("Man is created");
 }
}
```

Наследование проверяется путем создания экземпляра Man с использованием конструктора по умолчанию. Пример кода показан для демонстрации наследования.

TestInheritance Class

```
public class TestInheritance{

public static void main(String args[])
{
  Man man = new Man();
 }
}
```

11. Что такое композиция?

Композиция точно такая же, как Агрегация, за исключением того, что время жизни «части» контролируется «целым». Этот контроль может быть прямым или переходным. То есть «целое» может взять на себя прямую ответственность за создание или уничтожение «части», или оно может принять уже созданную часть, а затем передать ее какому-то другому целому, которое берет на себя ответственность за нее.

Пример класса автомобиля показан ниже для демонстрации состава шин, дверей, окон и рулевого управления.

Класс автомобиля

```
public class Car
{
 private Tire[] tires;

 private Door[] doors;
```

```
  private Steering steering;

  private Window[] windows;
}

class Tire
{

}

class Door
{

}

class Steering
{

}

class Window
{

}
```

12. Что такое ассоциация?

Ассоциация представляет способность одного экземпляра отправить сообщение другому экземпляру. Это обычно реализуется с помощью указателя или ссылочной переменной экземпляра, хотя это также может быть реализовано как аргумент метода или создание локальной переменной.

13. Что такое агрегация?

Агрегация — это типичное отношение «целое / часть». Это то же самое, что и ассоциация, за исключением того, что экземпляры не могут иметь отношения циклического агрегирования.

Пример класса Person показан ниже, чтобы продемонстрировать связь агрегации с адресом.

Персональный класс

```
public class Person
{
  private Address address;

}

class Address
{
```

```
    private String city;

    private String state;

    private String country;

    private String line1;

    private String line2;

}
```

Б. Общие вопросы о Java

14. Что такое JVM?

Виртуальная машина Java (JVM) — это виртуальная машина процесса, которая может выполнять байт-код Java. Каждый исходный файл Java компилируется в файл байт-кода, который выполняется JVM.

15. Почему Java называется независимым от платформы языком программирования?

Java была разработана для того, чтобы позволить создавать прикладные программы, которые можно запускать на любой платформе, без необходимости переписывать или перекомпилировать программист для каждой отдельной платформы. Виртуальная машина Java делает это возможным, потому что она знает конкретные длины команд и другие особенности базовой аппаратной платформы.

16. В чем разница между JDK и JRE?

Java Runtime Environment (JRE) — это, по сути, виртуальная машина Java (JVM), на которой исполняются ваши Java-программы. Он также включает в себя плагины браузера для запуска апплета. Java Development Kit (JDK) — это полнофункциональный комплект разработки программного обеспечения для Java, включающий в себя JRE, компиляторы и инструменты (такие как JavaDoc и Java Debugger), для того чтобы пользователь мог разрабатывать, компилировать и выполнять приложения Java.

JDK	**JRE**
JDK расшифровывается как Java Development Kit.	JRE означает «среда выполнения Java».
JDK — это инструмент для компиляции, документирования и упаковки программного обеспечения Java.	JRE — это среда выполнения. JavaByte-код выполняется в среде.
У JDK есть JRE и инструменты разработки.	JRE — это реализация JVM

17. Что означает статическое ключевое слово?

Ключевое слово static обозначает, что к переменной или методу-члену можно получить доступ, не требуя создания экземпляра класса, к которому он принадлежит.

Пример статического метода, который показан ниже:

Статический метод

```
static void printGreeting()
   {


   }
```

18. Можете ли вы переопределить приватный или статический метод в Java?

Пользователь не может переопределить статические методы в Java , потому что переопределение методов основано на динамическом связывании во время выполнения, а статические методы статически связаны во время компиляции. Статический метод не связан ни с одним экземпляром класса, поэтому концепция не применима.

19. Можете ли вы получить доступ к нестатической переменной в статическом контексте?

Статическая переменная в Java принадлежит ее классу, и ее значение остается одинаковым для всех ее экземпляров. Статическая переменная инициализируется, когда класс загружается JVM. Если ваш код пытается получить доступ к нестатической переменной без какого-либо экземпляра, компилятор будет жаловаться, потому что эти переменные еще не созданы и они не связаны ни с одним экземпляром.

20. Какие типы данных поддерживаются Java?

Восемь примитивных типов данных, поддерживаемых языком программирования Java:

- байт
- короткая
- ИНТ
- долго
- поплавок
- двойной
- логический
- голец

21. Что такое автобокс и распаковка?

Автобокс — это автоматическое преобразование, выполняемое компилятором Java между примитивными типами и их соответствующими классами обертки объектов. Например, компилятор преобразует целое число в целое , двойное в двойное и т. Д. Если преобразование идет другим путем, эта операция называется распаковкой.

22. Что такое переопределение и перегрузка функций в Java?

Перегрузка методов в Java происходит, когда два или более методов в одном классе имеют одинаковое имя, но разные параметры. С другой стороны, переопределение метода определяется как случай, когда дочерний класс переопределяет тот же метод, что и родительский класс. Переопределенные методы должны иметь одинаковые имя, список аргументов и тип возвращаемого значения. Переопределяющий метод может не ограничивать доступ к методу, который он переопределяет.

23. Что такое конструктор?

Конструктор вызывается при создании нового объекта. У каждого класса есть конструктор . Если программист не предоставляет конструктор для класса, компилятор Java (Javac) создает конструктор по умолчанию для этого класса.

Конструктор по умолчанию в Java показан в следующем примере:

Конструктор по умолчанию

```
public Man()
{
  System.out.println("Man is created");
}
```

Конструктор, который принимает параметр, показан в примере ниже:

Конструктор

```
private String name;

   public Employee(String name)
   {
     this.name = name;
   }
```

24. Что такое перегрузка конструктора?

Перегрузка конструктора аналогична перегрузке методов в Java. Различные конструкторы могут быть созданы для одного класса. Каждый конструктор должен иметь свой собственный уникальный список параметров.

25. Что такое Copy-Constructor?

Наконец, Java поддерживает конструкторы копирования, такие как C ++, но разница заключается в том, что Java не создает конструктор копирования по умолчанию, если вы не пишете свой собственный.

Конструктор копирования для класса Employee показан ниже:

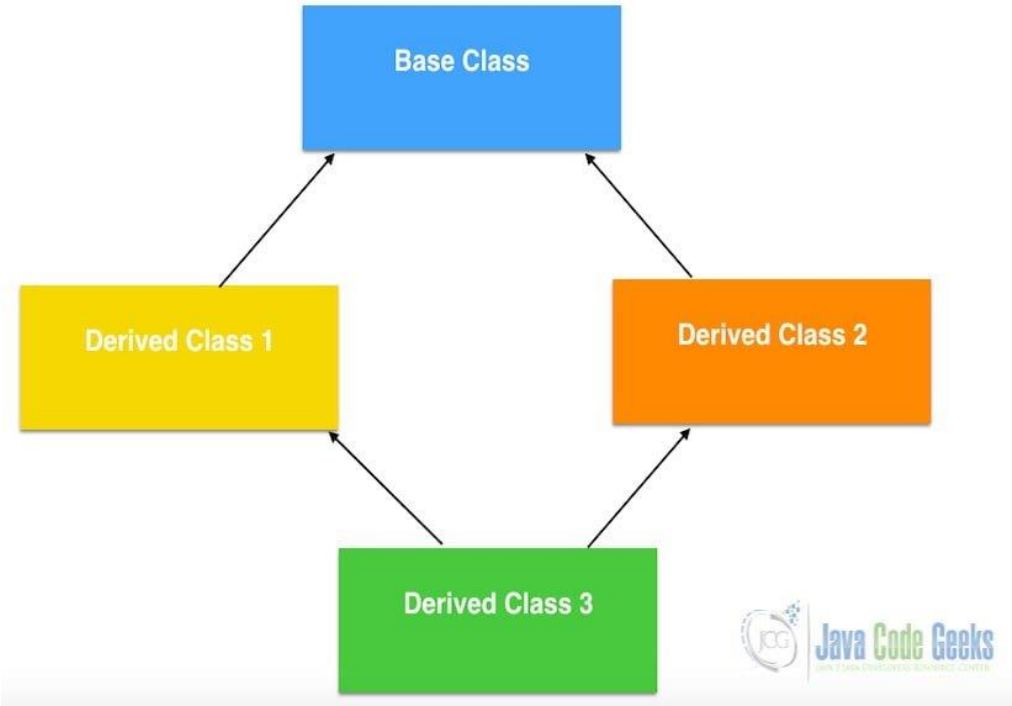
Копировать конструктор

```
public class Employee extends Person
{
   private String name;

   public Employee(String name)
   {
     this.name = name;
```

```
    }

    public Employee(Employee emp)
    {
      this.name = emp.name;
    }

}
```

26. Поддерживает ли Java множественное наследование?

Нет, Java не поддерживает множественное наследование. Каждый класс может расширяться только на один класс, но может реализовывать более одного интерфейса.



Множественное наследование

27. В чем разница между интерфейсом и абстрактным классом?

Java обеспечивает и поддерживает создание как абстрактных классов, так и интерфейсов. Обе реализации имеют некоторые общие характеристики, но отличаются следующими характеристиками:

- Все методы в интерфейсе неявно абстрактны. С другой стороны, абстрактный класс может содержать как абстрактные, так и неабстрактные методы.
- Класс может реализовывать несколько интерфейсов, но может расширять только один абстрактный класс.
- Чтобы класс реализовал интерфейс, он должен реализовать все свои объявленные методы. Однако класс может не реализовывать все объявленные методы абстрактного класса. Хотя в этом случае подкласс также должен быть объявлен как абстрактный.
- Абстрактные классы могут реализовывать интерфейсы, даже не обеспечивая реализацию методов интерфейса.

- Переменные, объявленные в интерфейсе Java, по умолчанию являются окончательными. Абстрактный класс может содержать неконечные переменные.
- Члены интерфейса Java являются общедоступными по умолчанию. Член абстрактного класса может быть приватным, защищенным или общедоступным.
- Интерфейс абсолютно абстрактный и не может быть создан. Абстрактный класс также не может быть создан, но может быть вызван, если он содержит метод main.

Также проверьте класс Abstract и различия интерфейса для JDK 8 .

Интерфейс	**Абстрактный класс**
Интерфейс имеет сигнатуры методов. У него нет никакой реализации.	Абстрактный класс имеет абстрактные методы и детали, которые должны быть переопределены.
Класс может реализовывать несколько интерфейсов	В этом случае класс может расширять только один абстрактный класс
Интерфейс имеет все абстрактные методы.	Не абстрактные методы могут быть там в абстрактном классе.
Свойства экземпляра не могут быть там в интерфейсе.	Свойства экземпляра могут быть там в абстрактном классе.
Интерфейс является публично видимым или невидимым.	Абстрактный класс может быть публичным, приватным и защищенным.
Любое изменение в интерфейсе повлияет на классы, реализующие интерфейс.	Добавление метода к абстрактному классу и его реализация не требуют изменений в коде для производных классов.
Интерфейс не может иметь конструкторов	Абстрактный класс может иметь конструкторы
Интерфейсы медленны с точки зрения производительности	Абстрактные классы быстры в выполнении методов в производных классах.

28. Что такое передача по ссылке и передача по значению?

Когда объект передается по значению, это означает, что копия объекта передается. Таким образом, даже если в этот объект вносятся изменения, это не влияет на исходное значение. Когда объект передается по ссылке, это означает, что фактический объект не передается, а передается ссылка на объект. Таким образом, любые изменения, сделанные внешним методом, также отражаются во всех местах.

Пример кода представлен ниже, который показывает передачу по значению.

Передать по значению

```
public class ComputingEngine
{
   public static void main(String[] args)
   {
      int x = 15;
      ComputingEngine engine = new ComputingEngine();
      engine.modify(x);
      System.out.println("The value of x after passing by value "+x);
   }
   public  void modify(int x)
   {
      x = 12;
   }
}
```

Ниже приведен пример передачи по ссылке в коде.

Передать по ссылке

```
public class ComputingEngine
{
   public static void main(String[] args)
   {

      ComputingEngine engine = new ComputingEngine();


      Computation computation = new Computation(65);
      engine.changeComputedValue(computation);

      System.out.println("The value of x after passing by reference "+ computation.x)

   }



   public void changeComputedValue(Computation computation)
   {
```

```
        computation = new Computation();
        computation.x = 40;
    }
}


class Computation
{
    int x;
    Computation(int i) { x = i; }
    Computation()      { x = 1; }
}
```

29. Какова цель изменчивой переменной?

Изменчивые значения переменных могут быть изменены различными потоками. У них никогда не будет возможности заблокировать и удержать замок. Синхронизация будет происходить всякий раз, когда к переменным обращаются. Использование volatile может быть быстрее блокировки, но в некоторых ситуациях оно не будет работать. Диапазон ситуаций, в которых работает volatile, был расширен в Java 5; в частности, двойная проверка блокировки теперь работает правильно.

Пример кода для изменчивой переменной показан ниже:

Изменчивая переменная

```
public class DistributedObject {

    public volatile int count = 0;

}
```

30. Какова цель временной переменной?

Временная переменная не будет сериализована, даже если класс, к которому она принадлежит, сериализован.

Пример класса, который имеет временную переменную, показан ниже:

Транзиторная переменная

```
public class Paper implements Serializable
{
    private int id;
    private String title;
    private String author;
    private transient int version = 1;

}
```

31. Что такое локальная переменная и переменная экземпляра?

Локальная переменная	Переменная экземпляра
Локальная переменная объявляется внутри метода или конструктора. Это может быть объявлено в блоке	Переменная экземпляра объявлена внутри класса.
Локальная переменная должна быть инициализирована перед использованием. Код не скомпилируется.	Инициализация переменной экземпляра не требуется. Если не инициализировано, используется значение по умолчанию.

32. Какие существуют различные модификаторы доступа в Java?

Существует четыре типа модификаторов доступа:

- Общедоступный — доступен из любого места приложения
- Защищено — доступно в пакете и подклассах в любом пакете
- Пакет Private (по умолчанию) — доступен строго внутри пакета
- Частный — доступен только в том же классе, где он объявлен

33. Разница между статическим и динамическим связыванием

Статическое связывание	Динамическое связывание
Определение процедуры связано со статическим связыванием	Примером динамического связывания является активация процедуры
Объявление имени для переменной делается для статической привязки переменной.	Привязка имени может быть динамической привязкой.
Сфера действия декларации является статически связанной.	Время жизни привязки динамически связано.

Пример кода для статического связывания показан ниже:

Статическое связывание

```
public class Shape
{
  public void getArea()
   {
     System.out.println("Shape Area");
    }


public static void main(String args[])
  {
```

```
    Shape shape = new Shape();

    shape.getArea();
  }
}
```

Пример кода для динамического связывания показан ниже:

Динамическое связывание

```
public class Rectangle extends Shape{


  public void getArea()
  {
    System.out.println("Rectangle Area");

  }



  public static void main(String args[])
  {

    Shape shape = new Rectangle();

    shape.getArea();
  }
}
```

34. Что такое классы-обертки?

Класс-оболочка преобразует Java-примитивы в объекты. Таким образом, примитивный класс-обертка — это класс-обертка, который инкапсулирует, скрывает или упаковывает типы данных из восьми примитивных типов данных, чтобы их можно было использовать для создания экземпляров объектов с методами в другом классе или в других классах. Примитивные классы-обертки находятся в Java API.

35. Что такое синглтон-класс и как мы можем создать класс синглтон?

В одноместном классе мы:

- убедитесь, что существует только один экземпляр класса singleton
- обеспечить глобальный доступ к этому экземпляру

Для создания одноэлементного класса мы:

- объявить все конструкторы класса как частные
- предоставить статический метод, который возвращает ссылку на экземпляр

Пример кода ниже показывает дважды проверенную реализацию класса Singleton.

Синглтон Класс

```
public class DoubleCheckedSingleton {
    private static volatile DoubleCheckedSingleton instance;
    public static DoubleCheckedSingleton getInstance() {
        if (instance == null) {
            synchronized (DoubleCheckedSingleton .class) {
                if (instance == null) {
                    instance = new DoubleCheckedSingleton();
                }
            }
        }
        return instance;
    }

}
```

C.Java темы

36. В чем разница между процессами и потоками?

Процесс — это выполнение программы, а поток — это отдельная последовательность выполнения в процессе. Процесс может содержать несколько потоков. Поток иногда называют облегченным процессом.

Процессы	**Потоки**
Процесс связан с выполнением программы.	Процесс состоит из нескольких шагов.
Процессы взаимодействуют друг с другом, используя межпроцессное взаимодействие.	Потоки процесса могут общаться друг с другом.
Процессы имеют контроль над дочерними процессами.	Потоки процесса могут иметь контроль над другими потоками.
Любая модификация в родительском процессе не изменяет дочерние процессы	Любая модификация в главном потоке может повлиять на поведение других потоков процесса.
Процессы выполняются в отдельных пространствах памяти.	Потоки выполняются в общих пространствах памяти.
Операционная система контролирует процесс.	Разработчик программного обеспечения имеет контроль над использованием потоков.
Процессы не зависят друг от друга.	Потоки зависят друг от друга.

37. Объясните разные способы создания темы. Какой из них вы бы предпочли и почему?

Для создания потока можно использовать три способа:

- Класс может расширять класс Thread .
- Класс может реализовывать интерфейс Runnable .
- Приложение может использовать платформу Executor для создания пула потоков.

Предпочтительным является интерфейс Runnable, так как он не требует, чтобы объект наследовал класс Thread. Если ваш дизайн приложения требует множественного наследования, вам могут помочь только интерфейсы. Кроме того, пул потоков очень эффективен и может быть легко реализован и использован.

Список использованных источников

1. Филлипс Б., Стюарт К., Марсикано К. Android. Программирование для профессионалов.
2. Черников, Разработка мобильных приложений на C# для iOS и Android
3. Яшин А.С., Сеттер Р.В., Java на примерах. Практика, практика и только практика
4. А.Жакупов, SMART 2.0. Как ставить цели, которые работают