

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**Федеральное государственное бюджетное
образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

И. Н. Дорошенко, А. Н. Токарев, Р. Н. Федюнин

ОСНОВЫ КРОССПЛАТФОРМЕННОЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

Учебно-методическое пособие

ПЕНЗА 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

И. Н. Дорошенко, А. Н. Токарев, Р. Н. Федюнин

ОСНОВЫ КРОССПЛАТФОРМЕННОЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

Учебно-методическое пособие

Пенза
Издательство ПГУ
2021

УДК 004.42
Д69

Р е ц е н з е н т

кандидат технических наук,
доцент кафедры «Математическое обеспечение и применение ЭВМ»
Пензенского государственного университета
С. В. Самуйлов

Дорошенко, Иван Николаевич.
Д69 Основы кроссплатформенной разработки приложений для
мобильных устройств : учеб.-метод. пособие / И. Н. Дорошенко,
А. Н. Токарев, Р. Н. Федюнин. – Пенза : Изд-во ПГУ, 2021. – 28 с.

Издание предназначено для лабораторных и практических занятий по курсу «Основы кроссплатформенной разработки приложений для мобильных устройств». Содержатся описания четырех лабораторных работ, которые предназначены для изучения принципов разработки кроссплатформенных мобильных приложений. Каждая лабораторная работа содержит цель выполнения, общие сведения и лабораторные задания.

Издание подготовлено на кафедре «Вычислительная техника» ПГУ и предназначено для обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника».

УДК 004.42

© Пензенский государственный
университет, 2021

СОДЕРЖАНИЕ

1. СОЗДАНИЕ ПРОСТОГО ПРИЛОЖЕНИЯ	4
2. АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЯ И РАБОТА С БАЗОЙ ДАННЫХ	11
3. УВЕДОМЛЕНИЯ (NOTIFICATIONS)	16
4. МЕНЮ И ЗАПУСК ПРИЛОЖЕНИЙ.....	21

1. СОЗДАНИЕ ПРОСТОГО ПРИЛОЖЕНИЯ

Цель: изучение работы с базовыми элементами Android-приложения.

Теоретический материал

Язык программирования Dart

Dart является языком программирования общего назначения от компании Google, который предназначен прежде всего для разработки веб-приложений (как на стороне клиента, так и на стороне сервера) и мобильных приложений.

Dart – это объектно-ориентированный язык с классами и наследованием, похожим на наследование в таких языках, как C# и Java, но также имеющим и собственные уникальные механизмы.

Класс может состоять из полей и методов, при этом методы экземпляра могут обращаться к полям экземпляра. Пример:

```
class Point {  
    num x, y;  
  
    num distanceTo(Point other) {  
        var dx = x - other.x;  
        var dy = y - other.y;  
        return sqrt(dx * dx + dy * dy);  
    }  
}
```

При описании класса возможно использование конструкторов. Конструктор – это метод с тем же именем, что и его класс. Наиболее распространенная форма конструктора, порождающий конструктор, создает новый экземпляр класса. Пример:

```
class Point {  
    num x, y;  
  
    Point(num x, num y) {  
        this.x = x;  
        this.y = y;  
    }  
}
```

Ключевое слово `this` ссылается на текущий экземпляр.

В Dart this также применяется для упрощения синтаксиса конструктора путем применения его в списках параметров. Пример:

```
class Point {  
  num x, y;  
  Point(this.x, this.y);  
}
```

Dart обладает обширным перечнем стандартных объектов, одним из которых является Map. Map – это объект, состоящий из ключей и значений. И ключи, и значения могут быть объектами любого типа. Каждый ключ встречается только один раз, а значение может использоваться неоднократно. Пример:

```
var gifts = {  
  // Key: Value  
  'first': 'partridge',  
  'second': 'turtledoves',  
  'fifth': 'golden rings'  
};
```

По ключу вы можете создавать новые дочерние элементы виджета Map или обращаться к уже существующим. Пример:

```
var gifts = Map();  
gifts['first'] = 'partridge';  
print(gifts['first']);
```

Фреймворк Flutter. Структура проектов на Flutter

Одним из наиболее известных фреймворков Dart является Flutter.

Flutter – это SDK с открытым исходным кодом для создания мобильных приложений. Он используется для разработки приложений под Android и iOS.

Приложения на Flutter имеют следующую структуру проекта:

- lib/ – папка, содержащая в себе код приложения;
- pubspec.yml – файл, в который записываются зависимости приложения, необходимые для запуска. Установка зависимостей осуществляется через команду flutter pub get;
- test/ – папка, предназначенная для хранения тестовых файлов;
- ios/ & android/ – папки, содержащие параметры настройки для каждой из платформ, в которой указываются необходимые для запуска приложения права, и отличительные особенности приложения под каждую из платформ.

В папке lib/ мы можем найти файл main.dart, являющийся главным файлом проекта, из него должен осуществляться запуск приложения при помощи функции main() (подобно языку C).

Widget

Код приложения на Flutter состоит из Widget, которые отображают элементы интерфейса, хранят стили и состояния объектов.

Выделяют два основных типа виджетов StatelessWidget и StatefulWidget.

StatelessWidget – это виджеты, предназначенные для неизменяемых элементов, то есть не имеющих внутреннего состояния и зависящих только от конфигурационных параметров. Пример:

```
class MyWidget extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return new Text('my text');  
  }  
}
```

StatefulWidget – это виджеты, рекомендованные для изменяемых элементов, с изменяемым внутренним состоянием. Под изменяемым состоянием понимается изменение внутреннего состояния экземпляра класса в зависимости от какого-либо события. Пример:

```
class MyWidget extends StatefulWidget {  
  @override  
  createState() => new MyWidgetState();  
}  
class MyWidgetState extends State<MyWidget> {  
  @override  
  Widget build(BuildContext context) {  
    return new Text('my text');  
  }  
}
```

State – это внутреннее состояние виджета типа StatefulWidget.

createState() – метод, генерирующий State.

В состав Flutter входит большое количество базовых виджетов, позволяющих упростить процесс разработки.

Все базовые виджеты делятся на следующие категории:

- виджеты доступа (ExcludeSemantics, MergeSemantics, MergeSemantics);
- виджеты анимации (AnimatedBuilder, AnimatedContainer и т.д.);
- виджеты отображения изображений и иконок (AssetBundle, Icon, Image и RawImage);
- асинхронные виджеты (FutureBuilder и StreamBuilder);
- виджеты ввода данных (Form, FormField и RawKeyListener);

- виджеты, отвечающие за сенсорные события (AbsorbPointer, Dismissible и т.д.);
- визуально-поведенческие виджеты (AppBar, BottomNavigationBar и т.д.);
- виджеты разметки и стиля приложения (MediaQuery, Padding и Theme);
- виджеты вывода информации (Text и RichText);
- виджеты с большим количеством элементов и внутренним скроллингом (ListView, CustomScrollView и т.д.).

В качестве примера в данной работе будет рассмотрен ListView. Данный виджет применяется для создания конечных и бесконечных списков дочерних виджетов.

Для создания конечного списка используется стандартный конструктор виджета ListView. При этом все содержимое списка передается через параметр children. Пример:

```
ListView(
  children: [
    new Text('1'),
    new Text('2'),
    new Text('3'),
  ]
)
```

Для создания бесконечного списка применяется конструктор ListView.builder(). Список элементов в данном случае генерируется с применением функции в параметре itemBuilder, а количество элементов (если необходимо) можно передать через параметр itemCount. Пример:

```
ListView.builder(
  itemBuilder: (context, index) {
    return new Text((index+1).toString());
  },
  itemCount: 3
)
```

При разработке приложений со столь объемными элементами интерфейса, как списки, необходимо учитывать ориентацию экрана, так как схема расположения элементов интерфейса, адаптированная под вертикальную ориентацию экрана, зачастую не подходит горизонтальной.

Существует три основных подхода к решению данной проблемы:

1. Поддержка только одной ориентации экрана путем отключения другой. Данный метод прост в реализации, но он искусственно ограничивает возможности пользователя.

2. Разработка уникальной структуры интерфейса, подходящей как под горизонтальную, так и вертикальную ориентацию экрана. Подобный механизм является трудоемким и не всегда реализуемым.

3. Разработка отдельной структуры интерфейса под каждую из ориентаций экрана. Данный метод позволяет учитывать особенности каждой из ориентаций и является менее трудоемким, чем предыдущий.

В Flutter предусмотрена возможность реализации механизма разработки отдельной структуры интерфейса под каждую из ориентаций экрана путем применения виджета MediaQuery, позволяющего отследить его состояние. Для этого, из виджета MediaQuery необходимо вызвать метод of() и передать в него контекст данного состояния. В качестве результата данная процедура возвращает экземпляр виджета MediaQueryData, хранящий информацию о текущем состоянии экрана, в том числе и его ориентацию (параметр orientation). Пример:

```
if (MediaQuery.of(context).orientation == Orientation.portrait){  
  // is portrait  
}  
else{  
  // is landscape  
}
```

Лабораторное задание № 1

Создать приложение типа Empty Activity.

Приложение должно иметь одну Activity, поддерживающую смену ориентации экрана и смену языка. На Activity необходимо разместить:

- список данных типа ListView;
- кнопку «Добавить»;
- кнопку «Удалить»;
- кнопку «Язык».

Элементы должны размещаться так, как показано на рис. 1.

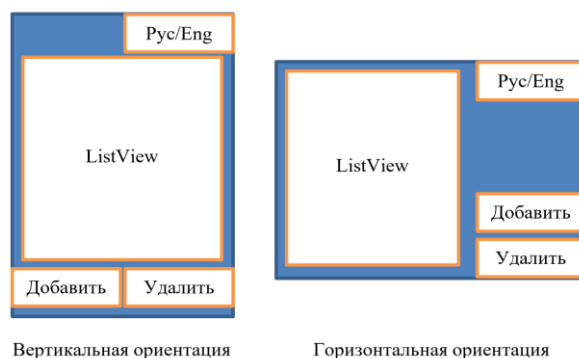


Рис. 1. Размещение элементов

По нажатию на кнопку «Добавить» необходимо добавить в список запись в соответствии с вариантом задания. Для получения различия добавляемых записей в одном из полей списка необходимо использовать автоматически инкрементируемую переменную или текущее время.

По нажатию на кнопку «Удалить» необходимо удалить из списка выделенную запись.

По нажатию на кнопку «Язык» должна происходить смена языка интерфейса.

Список данных должен поддерживать скроллинг элементов.

При смене ориентации экрана должно происходить перерасположение элементов в соответствии с приведенным рисунком.

Варианты лабораторных заданий

1. Класс «Автомобиль» (легковой/грузовой, мощность, объем двигателя, цена и т.д.).
2. Класс «Телефон» (производитель, модель, размер экрана и т.д.).
3. Класс «Книга» (тема, автор, издательство, год выпуска и т.д.).
4. Класс «Журнал» (название, тема, тираж и т.д.).
5. Класс «Студент» (ФИО, вуз, факультет, кафедра, форма обучения и т.д.).
6. Класс «Сотрудник» (ФИО, предприятие, должность, зарплата, стаж и т.д.).
7. Класс «Музыкальное произведение» (название, автор, исполнитель, жанр и т.д.).
8. Класс «Фильм» (название, режиссер, студия, жанр и т.д.).
9. Класс «Кинотеатр» (название, адрес, количество залов, репертуар и т.д.).
10. Класс «Компьютерная игра» (название, жанр, производитель, цена и т.д.).

Вопросы для самопроверки

1. Что такое Dart?
2. Что такое Flutter?
3. Чем отличается StatelessWidget от StatefulWidget?
4. В чем отличие конструктора ListView и ListView.builder()?
5. Что такое Map?
6. Сколько раз может встречаться ключ в одном Map?
7. Что такое конструктор?

Список использованных источников

1. Install. URL: <https://flutter.dev/docs/getstarted/install> (дата обращения: 20.03.2020).
2. StatelessWidget и StatefulWidget. URL: <https://flutter.su/tutorial/2-StatelessWidget-StatefulWidget> (дата обращения: 20.03.2020).
3. Flutter Documentation. URL: <https://flutter.dev/docs> (дата обращения: 20.03.2020).
4. Бакетт К. Dart в действии. М. : ДМК Пресс, 2013.
5. Пейн Р. Начало разработки приложений с Flutter: создание кросс-платформенных мобильных приложений. 2019.
6. ListView class. URL: <https://api.flutter.dev/flutter/widgets/ListView-class.html> (дата обращения: 26.03.2020).
7. ListView class. URL: <https://api.flutter.dev/flutter/widgets/MediaQuery-class.html> (дата обращения: 25.03.2020).

2 АУТЕНТИФИКАЦИЯ ПОЛЬЗОВАТЕЛЯ И РАБОТА С БАЗОЙ ДАННЫХ

Цель: работа с хранением данных (класс Preferences и база данных SQLite).

Теоретический материал

Механизмы языка Dart

Поскольку в данной лабораторной работе будет использовано сразу несколько новых операторов языка Dart, мы начнем с их описания, а затем перенесем это в контекст SDK Flutter.

Язык программирования Dart предоставляет механизмы асинхронного программирования, похожие на Task в C# и Promise в JavaScript. Асинхронные функции и методы зачастую имеют следующий вид

```
Future<T> method(...) async {}
```

Ключевое слово `async` говорит интерпретатору о том, что метод в определенный момент вычислений может отдать управление обратно в вызывающий его метод и продолжить выполнение, когда асинхронное вычисление будет выполнено. Для этого существует ключевое слово `await`. В прототипе `Future<T>` – это объект типа `T`, который будет вычислен. При вызове из синхронного метода мы можем задать функцию обратного вызова, которая будет выполнена по завершении асинхронного метода

```
future.then((T result){...});
```

Dart также предоставляет механизмы переопределения некоторых операторов для объектов классов, например

```
bool operator==(dynamic other){...}
```

Полный список переопределяемых операторов можно найти в официальной документации на сайте <https://dart.dev>

Функции и методы в языке Dart – объекты первого класса, поэтому с ними можно работать так же, как и с обычными объектами. Интересы вызывают анонимные методы. Анонимные методы – это методы, у которых нет имени, поэтому они напрямую выступают в качестве аргумента. Объявляются они следующим образом

```
(T t,...) {...}
```

Анонимные методы также могут быть асинхронными

```
(T t,...) async {...}
```

Если тело метода состоит из одного выражения, при условии, что метод имеет либо возвращаемое значение того же типа, что и внутреннее

выражение, либо тип `void`, то появляется возможность использовать стрелочную нотацию, превратив таким образом метод в лямбда-функцию (примечание: стрелочную нотацию могут иметь как именованные, так и анонимные методы).

```
int sum(int a, int b) => a+b;
```

Таким образом, оператор `=>` заменяет собой ключевое слово `return`.

В языке Dart отсутствуют ключевые слова `private`, `public` и `protected`. Механизмы инкапсулирования здесь выглядят несколько иначе: если поле или метод должны иметь область видимости лишь внутри класса, то название должно начинаться с нижнего подчеркивания (`_`). Во всех остальных случаях идентификатор является публичным. Мы также можем определить идентификатор, доступный дочерним классам, унаследовавшим родительский класс, но недоступный извне. Для этого существует специальная аннотация

```
@protected
```

Ей следует пометить поле (или метод) до определения.

Для классов существует ключевое слово `static`, им помечаются общие для всех объектов класса поля или методы (примечание: `static`-поля не наследуются).

Наряду с `Map`, язык Dart имеет такой тип коллекции, как список (`List`). В отличие от других языков с объектно-ориентированным программированием Dart не имеет типа массив, но есть возможность задать параметр `growable`, благодаря чему список будет вести себя как массив, и при попытке выполнить метод `add` произойдет ошибка времени выполнения.

Общие сведения

Зачастую в мобильных приложениях используют 2 типа долговременных хранилищ данных. `SharedPreferences` – это файл, содержащий в себе пары типа «ключ – значение». Данное хранилище поддерживает следующие типы:

1. Целочисленный.
2. Логический (булевый).
3. С плавающей запятой.
4. Строковый тип.

Благодаря сериализации мы можем сохранять и другие, более сложные объекты данных.

Большинство операций, связанных с вводом/выводом, реализованы как асинхронные методы, вследствие чего пользовательский интерфейс не блокируется на время чтения или записи. В контексте Flutter

SharedPreferences предоставляет собой класс паттерна singleton, получить экземпляр которого можно следующим образом:

```
SharedPreferences storage = await SharedPreferences.getInstance()
```

Далее мы можем работать с этим объектом для записи и чтения

```
storage.setString(key, value),
```

где key – ключ, по которому мы сохраняем значение, value – значение. Заметим, что данный метод асинхронен, то есть значение запишется не сразу, а по истечении некоторого времени (зависит от количества асинхронных задач и быстродействия устройства).

Извлечь значение можно следующим образом

```
String value = storage.getString(key)
```

В отличие от записи данный метод синхронен. Отметим, что данное хранилище применяют для небольшого количества данных, например, пользовательские настройки, данные сессии (токены).

В ситуациях, когда требуется сохранить большой объем данных, используют системы управления базами данных. В отличие от серверного программного обеспечения, данные системы являются встраиваемыми. Это означает, что теперь взаимодействие приложения и системы управления базами данных (СУБД) осуществляется в одном процессе, не по клиент-серверной архитектуре: фактически встраиваемые СУБД представляют собой API.

Для Flutter существует библиотека sqflite. Sqflite – портированная на Android СУБД SQLite. Доступ к базе данных (БД) осуществляется похожим на предыдущий способом.

```
Database database = await openDatabase(
  path,
  version: 1,
  onCreate: (Database db, int version) async {
    await db.execute('CREATE TABLE student (id INTEGER PRIMARY KEY, name TEXT, age INTEGER, height REAL)');
  }
);
```

В данном выражении мы открываем БД (в более узком смысле получаем дескриптор, поскольку данные хранятся в имеющем расширение .db файле), в качестве параметров указывая:

1) path – путь до файла (обычно это папка приложения);

2) версию БД;

3) функцию, вызываемую по созданию БД. Данный механизм удобен тем, что отпадает необходимость проверять, создана ли БД впервые или же она уже существовала.

В анонимном методе `onCreate` мы выполняем SQL-команду для создания таблицы студентов, имеющих `id` как первичный ключ, имя, возраст и рост.

С помощью объекта `database` мы можем запрашивать данные
`List<Map<String, dynamic>> rows = await database.rawQuery('select * where age > 20');`

Добавлять данные
`database.rawQuery('insert into student(name, age, height) values ('Ivan', 19, 1.79)');`

Удалять
`database.execute('delete from student where age < 17');`
`Sqflite` поддерживает и другие методы, например, методы запроса, которые автоматически подставляют данные на основе схемы таблицы и данных для записи, закрытия БД, изменения версии и др.

`Sqflite` следует использовать в тех случаях, когда объем данных велик, а также необходимо быстрое действие поиска записей.

Чтобы добавить данные пакеты в проект, в файл `pubspec.yaml` добавьте следующее:

```
shared_preferences: ^0.5.4+6
```

```
sqflite: ^1.2.0
```

Затем в командном интерпретаторе, в папке проекта выполните следующие команды:

```
flutter packages get
```

```
flutter pub get
```

После этого `flutter` разрешит зависимости.

Лабораторное задание № 2

Лабораторная работа выполняется на базе лабораторной работы № 1.

В исходное приложение необходимо добавить `Activity` входа пользователя (`Login`). Данная `Activity` должна стать стартовым при запуске приложения.

`Login Activity` должен предлагать пользователю авторизоваться в системе или зарегистрироваться в качестве нового пользователя. Данные по пользователю (`Login` и пароль) должны сохраняться с использованием класса `SharedPreferences`.

После успешного входа пользователя должно открыться окно `Activity`, реализованное в предыдущей работе.

Данные, добавляемые и удаляемые в списке `ListView`, должны сохраняться в базе `SQLite`. Алгоритм работы со списком данных следующий:

1. При регистрации нового пользователя база данных полностью очищается, и пользователь видит пустой `ListView`.

2. Добавленные данные отображаются в ListView и сохраняются в SQLite.

3. Удаленные данные стираются из ListView и удаляются из SQLite.

4. При повторном входе пользователя (повторном запуске приложения) ListView заполняется данными из SQLite (рис. 2).

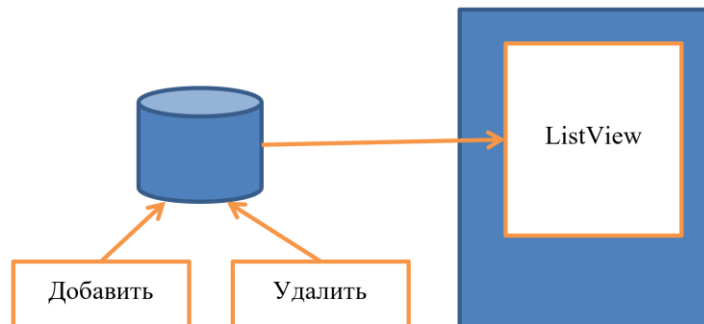


Рис. 2. Схема данных при работе с SQLite

Варианты лабораторных заданий

Вариант задания для лабораторной работы № 1.

Вопросы для самопроверки

1. Каким образом можно получить результат выполнения асинхронного метода?

2. Возможно ли переопределить операцию индексации для экземпляра класса?

3. Можно ли передать метод в качестве параметра?

4. Чем отличаются объекты типов Object от dynamic?

Список использованных источников

1. Бакетт К. Dart в действии. М. : ДМК Пресс, 2013.

2. Пейн Р. Начало разработки приложений с Flutter: создание кроссплатформенных мобильных приложений. 2019.

3. Asynchrony support. Declaring async functions. URL: <https://dart.dev/guides/language/language-tour> (дата обращения: 26.03.2020).

3. УВЕДОМЛЕНИЯ (NOTIFICATIONS)

Цель: работа с механизмами уведомлений.

Теоретический материал

Общие сведения

Уведомление выводится вне интерфейса приложения и оповещает пользователя о каком-либо событии в нем (напоминание, сообщение и другая срочная информация). Нажав на уведомление, можно перейти в приложение или выполнить действие.

Уведомления выводятся на экран по-разному, например, в виде значка в строке состояния, развернутого сообщения в панели уведомлений или пометки на значке приложения. Уведомления также могут автоматически появляться на подключенных носимых устройствах.

Для выполнения данного лабораторного занятия потребуется 2 библиотеки: `sms` и `flutter_local_notification`.

Начнем с `sms`. Для работы с входящими сообщениями используется класс `SmsReceiver`. Данный класс имеет следующее свойство

`Stream<SmsMessage> get onSmsReceived`

Это дает возможность запустить асинхронную функцию, просматривающую входящие сообщения.

Далее нам необходимо проверять состояние приложения. Это можно сделать с помощью примеси `WidgetsBindingObserver` (mixin: более подробное описание последует далее). Переопределив метод

`void didChangeAppLifecycleState(AppLifecycleState state),`

мы можем получать информацию о текущем состоянии приложения. Flutter-приложение имеет 4 состояния, описанных в табл. 1. Состояниями, которые мы будем использовать, будут `AppLifecycleState.inactive` и `AppLifecycleState.paused`.

Таблица 1

Возможные значения перечисления `AppLifecycleState`

Состояние	Комментарий
1	2
<code>AppLifecycleState.resumed</code>	Приложение работает и отвечает на ввод пользователя
<code>AppLifecycleState.inactive</code>	Приложение работает, но не отвечает на ввод пользователя (возможно перекрыто более приоритетным процессом, например, звонок)

1	2
AppLifecycleState.paused	Приложение работает, причем пользователь не видит окна приложения, не может с ним взаимодействовать (свернуто или выключен экран)
AppLifecycleState.detached	Приложение работает на Flutter Engine, но не имеет окна (view)

Для создания уведомлений используется класс `FlutterLocalNotificationsPlugin`. Его необходимо проинициализировать с помощью объектов типов `AndroidInitializationSettings` и `IOSInitializationSettings`. Затем с помощью асинхронного метода `show` мы можем инициировать создание уведомления.

Более подробное описание используемых библиотек вы можете найти в комментариях к примеру выполнения лабораторной работы.

В некоторых версиях Android и графических оболочках для стабильной работы в файл `\android\app\src\debug\AndroidManifest.xml` необходимо добавить следующую строку

```
<uses-permission android:name="android.permission.BIND_NOTIFICATION_LISTENER_SERVICE"/>
```

Данная строка разрешает прослушивать входящие сообщения.

Механизмы языка Dart

В данной лабораторной работе мы применим механизм наследования, называемый примесями (`mixin`). Примеси – это классы, которые добавляют некоторое поведение для объектов. Данный тип наследования похож на интерфейсы, но в то время как интерфейсы лишь декларируют определенное поведение, примеси добавляют конкретную реализацию как методов, так и полей.

Помимо объектов `Future<T>`, язык Dart имеет механизм асинхронных коллекций, который можно представить как поток событий. Данный механизм называется потоком, `Stream` (не следует путать с потоками в понятии операционной системы). Объекты данного класса имеют метод `listen(void callback(T){})`, который инициирует вызов функции `callback`, когда в `Stream` появляется новый объект типа `T`. Возможно также запустить асинхронную функцию-обработчик и обрабатывать появление события с помощью конструкции

```
await for (var e in stream){}
```

Тело цикла будет выполняться, когда будут возникать новые события.

Stream имеет множество методов манипулирования событиями, трансформации в стандартные коллекции. Более подробное описание вы можете найти в официальной документации на сайте dart.dev.

Язык Dart обладает таким механизмом, как свойства. Схожий механизм есть в C#. Свойства позволяют взаимодействовать с приватными переменными, что выглядит более коротко и наглядно, чем методы `getValue` и `setValue`

```
String _name;  
String get name => _name;  
Function _handler;  
set handler(Function h) => _handler = h;
```

Фактически вы можете выполнить лабораторную работу, переписав методы `_notify` под собственное задание. В качестве примера приведем приложение, генерирующее уведомление при получении сообщения.

Использование примесей

```
abstract class Car  
{  
  num speed;  
  
  Car(this.speed);  
  
  void drive()  
  {  
    print('I dirve: $speed km/h');  
  }  
}  
  
class SlowCar extends Car with Passengers{  
  SlowCar():super(50){  
    volume = 100;  
  }  
}  
  
class FastCar extends Car{  
  FastCar():super(100);  
}
```

```

mixin Passengers on Car{
  int volume;
  void getVolumeInfo(){
    print('My volume: $volume');
  }
}

void main(){
  Car fast = FastCar();
  Passengers slow = SlowCar();
  slow.getVolumeInfo();
}

```

Лабораторное задание № 3

Лабораторная работа выполняется на базе лабораторной работы № 2.

Реализовать в приложении уведомление, которое должно появляться на экране (в выпадающей шторке), только когда приложение свернуто или экран выключен. Уведомление должно генерироваться при получении SMS-сообщения.

При нажатии на уведомление оно должно удалиться с экрана (из выпадающей шторки) и должно запуститься приложение.

Варианты лабораторных заданий

1. Уведомление генерируется при получении сообщения с текстом длиной более 6 символов. Текст сообщения: «SMS-сообщение: <сообщение>».

2. Уведомление генерируется при получении сообщения с текстом, в котором указано имя текущего пользователя. Текст сообщения: «Получено SMS-сообщение для пользователя <Имя пользователя>».

3. Уведомление генерируется при получении сообщения с текстом «Hello». Текст сообщения: «SMS-сообщение с номера <номер телефона>».

4. Уведомление генерируется при получении сообщения с заданного номера телефона. Текст сообщения: «SMS-сообщение с номера <номер телефона>».

5. Уведомление генерируется при получении сообщения с текстом, в котором указано имя текущего пользователя. Текст сообщения: «SMS-сообщение: <сообщение>».

6. Уведомление генерируется при получении сообщения с заданного номера телефона. Текст сообщения: «SMS-сообщение: <сообщение>».

7. Уведомление генерируется при получении сообщения с текстом «Computer». Текст сообщения: «SMS-сообщение: <сообщение>».

8. Уведомление генерируется при получении сообщения с текстом длиной короче 5 символов. Текст сообщения: «SMS-сообщение с номера <номер телефона>».

9. Уведомление генерируется при получении сообщения с текстом, в котором 2 или более раз встречается слово «Msg». Текст сообщения: «Получено SMS-сообщение с <количество слов Msg в сообщении> слов».

10. Уведомление генерируется при получении сообщения с текстом «Note». Текст сообщения: «SMS-сообщение с номера <номер телефона>: <сообщение>».

Вопросы для самопроверки

1. Каким образом можно обработать данные, генерируемые посредством Stream?

2. Возможно ли создать ссылку на объект типа mixin?

3. Каким образом в метод можно передать объект, если он гарантированно имеет тип либо double, либо int.?

Список использованных источников

1. Бакетт К. Dart в действии. М. : ДМК Пресс, 2013.

2. Пейн Р. Начало разработки приложений с Flutter: создание кроссплатформенных мобильных приложений. 2019.

3. Adding features to a class: mixins. URL: <https://dart.dev/guides/language/language-tour> (дата обращения: 26.03.2020).

4. МЕНЮ И ЗАПУСК ПРИЛОЖЕНИЙ

Цель: работа с механизмами запуска функционала по требованию пользователя.

Теоретический материал

Общие сведения

Библиотека представляет собой набор подпрограмм, используемых при разработке программного обеспечения.

Библиотеки содержат вспомогательный код и данные, которые предоставляют автономные программные сервисы, позволяющие осуществлять модульный обмен и модификацию кода и данных.

Flutter включает в себя большое количество различных библиотек, упрощающих процесс разработки программного обеспечения.

`flutter_email_sender` – это библиотека, позволяющая отправлять электронные письма с флаттера, используя встроенный функционал платформы.

В Android данная библиотека открывает почтовое приложение по умолчанию через `Intent`. `Intent` – это объект обмена сообщениями, с помощью которого можно запросить выполнение действия у компонента другого приложения.

В iOS библиотека использует `MFMailComposeViewController` для составления электронного письма.

Для открытия почтового клиента в разрабатываемом приложении необходимо вызвать метод `send` из встроенного в библиотеку класса `FlutterEmailSender`. В качестве параметра в данную функцию необходимо передать экземпляр класса `Email`. Пример:

```
final Email email = Email(  
  body: 'Email body',  
  subject: 'Email subject',  
  recipients: ['example@example.com'],  
  cc: ['cc@example.com'],  
  bcc: ['bcc@example.com'],  
  attachmentPaths: ['/path/to/attachment.zip'],  
  isHTML: false,  
);  
await FlutterEmailSender.send(email)
```

Для применения библиотеки `flutter_email_sender` в разрабатываемом приложении в параметрах платформы Android необходимо прописать `compileSdkVersion 28`, так как он применяется в самом плагине.

`url_launcher` – это библиотека, позволяющая открыть URL-ссылку на мобильной платформе.

В состав данной библиотеки входит процедура `launch`, позволяющая открыть URL-ссылку. В качестве параметра данный метод принимает строковый аргумент.

Данная библиотека также содержит процедуру `canLaunch`, позволяющую проверить возможность открытия URL-ссылки, и, если это невозможно, возвращает значение `false`. Пример:

```
const url = 'https://flutter.dev';
if (await canLaunch(url)) {
  await launch(url);
} else {
  throw 'Could not launch $url';
}
```

Плагин `flutter_sms` позволяет отправлять SMS- и MMS-сообщения, а также выбрать получателя из списка контактов.

Данная библиотека содержит метод для отправки сообщений `sendSMS`, параметрами которого являются текст сообщения и номер получателя. Пример:

```
await sendSMS(message: message, recipients: recipients)
```

`camera` – это библиотека, позволяющая взаимодействовать с доступными устройству камерами и выводить полученное изображение в виджет.

Для использования данной библиотеки необходимо учесть особенности каждой из платформ.

Для iOS в текстовой форме добавить в файл `ios/Runner/Info.plist` следующий ключ:

```
<key>NSCameraUsageDescription</key>
<string>Can I use the camera please?</string>
<key>NSMicrophoneUsageDescription</key>
<string>Can I use the mic please?</string>
```

Для Android необходимо в файле `android/app/build.gradle` изменить значение параметра `minSdkVersion` на 21.

На начальном этапе работы с библиотекой следует воспользоваться функцией `availableCameras`, позволяющей получить доступ ко всем камерам мобильного устройства. Пример:

```
final cameras = await availableCameras()
```

Данная функция возвращает массив описаний всех камер, доступных данному устройству `List<CameraDescription>`.

Для дальнейшего использования камеры необходимо объявить контейнер типа `CameraController`, в конструкторе которого нужно

прописать одно из полученных ранее описаний камеры и разрешение устройства. Пример:

```
_controller=CameraController(cameraDescription,  
ResolutionPreset.medium)
```

После объявления контроллера требуется его инициализировать.

Пример:

```
await _controller.initialize()
```

Далее передаем инициализированный контроллер в виджет `CameraPreview`, предназначенный для передачи на экран изображения с выбранной камеры. Для регуляции размера виджета `CameraPreview` рекомендуется сделать его дочерним элементом виджета `AspectRatio`.

Пример:

```
AspectRatio(  
  child: CameraPreview(_controller)  
)
```

Затем для непосредственной работы с камерой необходимо вызвать из представленного ранее контроллера `CameraController` метод `takePicture`. Пример:

```
await _controller.takePicture(filePath)
```

`image_picker` – это библиотека, позволяющая взаимодействовать с галереями и камерой устройства.

Для работы с данной библиотекой на iOS-устройствах необходимо в текстовой форме добавить в файл `ios/Runner/Info.plist` следующий ключ:

```
<key>NSCameraUsageDescription</key>  
<string>Access to Camera</string>  
<key>NSMicrophoneUsageDescription</key>  
<string>Access to Microphone</string>  
<key>NSPhotoLibraryUsageDescription</key>  
<string>Access to PhotoLibrary</string>
```

В данной библиотеке предусмотрен метод для работы с видеофайлами `ImagePicker.pickVideo()` и для работы с изображениями `ImagePicker.pickImage()`.

Для работы с галереей в указанные ранее функции передается параметр `ImageSource.gallery`, а для взаимодействия с камерой передается значение `ImageSource.camera`. Пример:

```
_onImageButtonPressed(ImageSource.gallery)
```

`cached_network_image` – это библиотека, предназначенная для кэширования изображений, загружаемых из интернета, а также их отображения в автономном режиме.

Загрузка изображений в данном плагине осуществляется через виджет `CachedNetworkImage`, в параметре `imageUrl` которого необходимо указать URL-адрес изображения. Пример:

```
CachedNetworkImage(  
  imageUrl: "http://via.placeholder.com/350x150",  
)
```

`map_launcher` – это библиотека, позволяющая искать доступные мобильному устройству карты и запускать их с маркером для указанного местоположения.

Для работы с данной библиотекой на iOS устройствах необходимо в текстовой форме добавить в файл `ios/Runner/Info.plist` следующий ключ:

```
<key>LSApplicationQueriesSchemes</key>  
<array>  
  <string>comgooglemaps</string>  
  <string>baidumap</string>  
  <string>iosamap</string>  
  <string>waze</string>  
  <string>yandexmaps</string>  
  <string>yandexnavi</string>  
</array>
```

В библиотеке предусмотрен функционал по получению списка установленных карт, для этого из виджета `MapLauncher` необходимо вызвать метод `installedMaps`. Пример:

```
await MapLauncher.installedMaps()
```

В данном виджете `MapLauncher` есть метод `isMapAvailable`, позволяющий проверить, установлена ли карта на мобильное устройство, и метод `launchMap`, позволяющий запустить приложение «Карта». Пример:

```
if (await MapLauncher.isMapAvailable(MapType.google)) {  
  await MapLauncher.launchMap(  
    mapType: MapType.google,  
    coords: coords,  
    title: title,  
    description: description,  
  );  
}
```

`app_settings` – это библиотека, предназначенная для открытия native приложения с настройками мобильного устройства.

В данном плагине описан виджет `AppSettings`, в состав которого входят методы по открытию окна настройки Wi-Fi, конфиденциальности и т.д. Пример:

AppSettings.openWiFiSettings()

date_range_picker – это библиотека, предназначенная для создания диалогового окна календаря.

Для создания окна календаря в плагине представлен виджет DateRangePicker, в состав которого входит метод showDatePicker, инициализирующий диалоговое окно. Пример:

```
await DateRangePicker.showDatePicker(  
  context: context,  
  
  initialFirstDate: new DateTime.now(),  
  
  initialLastDate: (new DateTime.now()).add(new Duration(days: 7)),  
  firstDate: new DateTime(2015),  
  lastDate: new DateTime(2020)  
);
```

Плагин contact_picker позволяет открыть список контактов данного устройства, а также поддерживает возможность выбора одного из контактов и передачи информации о нем в приложение.

Для вызова пользовательского интерфейса со списком контактов необходимо из виджета ContactPicker вызвать метод selectContact, который возвращает информацию о выбранном контакте. Пример:

```
Contact contact = await ContactPicker().selectContact()
```

Лабораторное задание № 4

Лабораторная работа выполняется на базе лабораторной работы № 3.

Добавить в приложение выпадающее меню. Меню должно содержать информацию о пользователе, прошедшем аутентификацию. Должны быть добавлены кнопки «Добавить», «Удалить» и «Язык», которые должны повторять функционал кнопок, размещенных на основной Activity.

В меню должна быть добавлена кнопка «Выход», по нажатию на которую приложение должно попадать в окно Login Activity и пользователь должен повторно пройти аутентификацию.

В меню должна быть добавлена кнопка «Выполнить», по нажатию на которую необходимо открыть новую Activity, на которой должны располагаться элементы, необходимые для выполнения действия в соответствии с лабораторным заданием. Одним из элементов должна быть кнопка, активизирующая запуск приложения.

Варианты лабораторных заданий

1. Открытие почтового клиента и заполнение в нем: email получателя, email отправителя и заголовок сообщения.
2. Открытие URL ссылки в браузере.
3. Открытие клиента по отправке SMS-сообщений и заполнение номера получателя и текста сообщения.
4. Запуск приложения камеры.
5. Запуск приложения просмотра галереи.
6. Открытие графического изображения.
7. Открытие приложения «Карта».
8. Открытие окна настроек приложения.
9. Открытие диалогового окна календаря.
10. Открытие списка контактов.

Вопросы для самопроверки

1. Как проверить список доступных устройству камер?
2. Какой виджет предназначен для передачи на экран изображения с выбранной камеры?
3. Какой метод позволяет проверить, установлена ли карта на мобильное устройство?

Список использованных источников

1. Марко Л. Неаполь Начало флаттера: практическое руководство по разработке приложений. 2019.
2. Как спроектировать интерфейс мобильного приложения. URL: https://geekbrains.ru/posts/mob_interface (дата обращения: 16.03.2020).
3. User Interface Development Flow. 8-step Process. URL: <https://medium.com/swlh/user-interface-development-flow-537f82f00247> (дата обращения: 26.03.2020).

Учебное издание

Дорошенко Иван Николаевич,
Токарев Андрей Николаевич,
Федюнин Роман Николаевич

ОСНОВЫ КРОССПЛАТФОРМЕННОЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ ДЛЯ МОБИЛЬНЫХ УСТРОЙСТВ

Редактор *В. В. Устинская*
Технический редактор *Ю. В. Анурова*
Компьютерная верстка *Ю. В. Ануровой*

Подписано в печать 05.10.2021.
Формат 60×84¹/₁₆. Усл. печ. л. 1,63.
Заказ № 525. Тираж 28.

Издательство ПГУ
440026, Пенза, Красная, 40
Тел.: (8412) 66-60-49, 66-67-77; e-mail: iic@pnzgu.ru

Вниманию авторов!

Издательство ПГУ выпускает учебную, научную и художественную литературу, презентационную и акцидентную продукцию, а также полноцветные юбилейные и мемориальные издания в соответствии с ГОСТ 7.60–2003.

Издательство ПГУ принимает к изданию рукописи, подготовленные с использованием текстового редактора Microsoft Word for Windows версий **2003 и выше**. Формат – А4, основной шрифт – Times New Roman, 14–16 pt через одинарный интервал (минимальный размер шрифта в таблицах и сносках – 12,5 pt). Тип файла в электронном виде – doc, docx.

Работа должна содержать индекс УДК, аннотацию.

Аннотация (ГОСТ 7.86–2003, ГОСТ 7.9–1995) включает характеристику основной темы, проблемы объекта, цели работы и ее результаты. В аннотации указывают, что нового несет в себе данный документ в сравнении с другими, родственными по тематике и целевому назначению. Аннотация может включать сведения о достоинствах произведения. Текст аннотации начинают фразой, в которой сформулирована главная тема документа. Заканчивается аннотация читательским адресом.

Рисунки и таблицы должны быть размещены в тексте после ссылки на них (растровые рисунки представляются в виде отдельных файлов в формате jpg, BMP с разрешением 300 dpi, векторные рисунки в формате Corel Draw с минимальной толщиной линии 0,75 pt. Рисунки должны быть доступны для правки!). Они должны сопровождаться подрисовочными подписями, на все рисунки и таблицы в тексте должны быть ссылки.

Формулы в тексте выполняются только в редакторе формул **MathType версий 5.0** и выше. Символы греческого и русского алфавита должны быть набраны прямо, нежирно; латинского – курсивом, нежирно; обозначения векторов и матриц – прямо, жирно; цифры – прямо, нежирно. Наименования химических элементов набираются прямо, нежирно. Эти же требования необходимо соблюдать и в рисунках.

В списке литературы **нумерация источников** должна соответствовать очередности ссылок на них в тексте ([1], [2], ...). Номер источника указывается в квадратных скобках. Требования к оформлению списка литературы, русских и иностранных источников (ГОСТ 7.0.5–2008): для книг – фамилия и инициалы автора, название, город, издательство, год издания, том, количество страниц; для журнальных статей, сборников трудов – фамилия и инициалы автора, название статьи, полное название журнала или сборника, серия, год, том, номер, страницы; для материалов конференций – фамилия и инициалы автора, название статьи, название конференции, город, издательство, год, страницы.

К материалам **должна** прилагаться следующая информация: фамилия, имя, отчество, контактные телефоны.

Контакты Издательства ПГУ: (8412) 66-60-49, 66-67-77. E-mail: iic@pnzgu.ru



