

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерного проектирования
Кафедра Инженерной психологии и эргономики
Дисциплина Технологии программирования приложений

Лабораторная работа №2
«Разработка приложений с несколькими Activity. Передача данных между
Activity»

Студент группы 310901

(подпись)

Усов А.М.

Руководитель

(подпись)

Василькова А.Н.

Минск 2025

В рамках данной лабораторной работы разрабатывается приложение Taxi для платформы Android. Приложение представляет собой систему заказа такси с тремя основными Activity: регистрация пользователя, главный экран приложения и ввод маршрута движения.

Цель работы — формирование у студентов знаний о жизненном цикле Activity, навыков создания и вызова нового Activity, передачи данных между Activity, хранения данных с помощью класса SharedPreferences, создания всплывающих сообщений и логирования.

Скриншоты графических представлений первого, второго и третьего Activity

Первое Activity — RegistrationActivity

Ниже представлен внешний вид первого Activity приложения Taxi в Android Studio, демонстрирующий экран регистрации/входа пользователя.

На рисунке 1 показан интерфейс экрана регистрации пользователя. При первом запуске приложения отображается форма регистрации с полями для ввода телефона, имени и фамилии. При повторном запуске форма преобразуется в форму входа с предзаполненными данными из SharedPreferences.



Рисунок 1 – Activity (RegistrationActivity)

Второе Activity — TaxiActivity

Ниже представлен внешний вид второго Activity приложения Taxi в Android Studio, демонстрирующий главный экран приложения.

На рисунке 2 показан интерфейс главного экрана приложения. На экране отображается информация о пользователе (имя, фамилия, телефон), полученная из Intent, а также информация о маршруте движения,

восстановленная из SharedPreferences. Кнопка «Call Taxi» активируется только после установки маршрута.

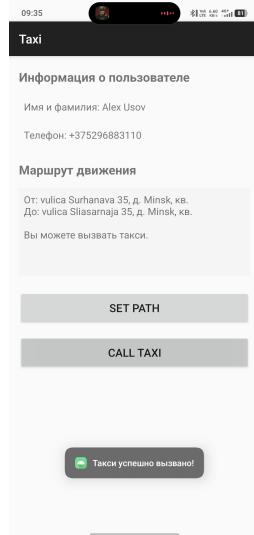


Рисунок 2 – Activity (TaxiActivity)

Третье Activity — RouteActivity

Ниже представлен внешний вид третьего Activity приложения Taxi в Android Studio, демонстрирующий экран ввода маршрута движения.

На рисунке 3 показан интерфейс экрана ввода маршрута движения. Пользователь может указать точку отправления (вручную или автоматически через GPS) и точку назначения (вручную или выбрав на карте). После заполнения всех полей и нажатия кнопки «OK» данные передаются обратно во второе Activity через механизм startActivityForResult.

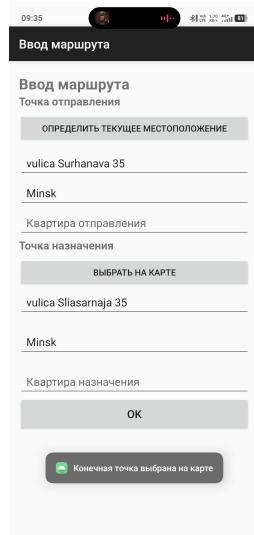


Рисунок 3 – Activity (RouteActivity)

Код XML-файлов графических представлений первого, второго и третьего Activity

XML-файл первого Activity — activity_registration.xml

Ниже представлен код XML-файла графического представления первого Activity приложения Taxi.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/textViewTitle"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/registration_title"
        android:textSize="24sp"
        android:textStyle="bold"
        android:layout_marginBottom="32dp" />

    <EditText
        android:id="@+id/editTextPhone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/phone_hint"
        android:inputType="phone"
        android:layout_marginBottom="16dp"
        android:padding="12dp" />

    <EditText
        android:id="@+id/editTextFirstName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/first_name_hint"
        android:inputType="textPersonName"
        android:layout_marginBottom="16dp"
        android:padding="12dp" />

    <EditText
        android:id="@+id/editTextLastName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/last_name_hint"
        android:inputType="textPersonName"
        android:layout_marginBottom="32dp"
        android:padding="12dp" />

    <Button
        android:id="@+id/buttonRegistration"
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="@string/button_registration"
        android:textSize="18sp"
        android:padding="16dp" />

    </LinearLayout>
```

В данном XML-файле используется `LinearLayout` с вертикальной ориентацией для размещения элементов интерфейса. Элементы включают заголовок (`TextView`), три поля ввода (`EditText`) для телефона, имени и фамилии, а также кнопку регистрации (`Button`).

XML-файл второго Activity — `activity_taxi.xml`

Ниже представлен код XML-файла графического представления второго Activity приложения Taxi.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="100sp"
        android:layout_marginBottom="16dp"
        android:text="@string/user_info_title"
        android:textSize="20sp"
        android:textStyle="bold" />

    <TextView
        android:id="@+id/textViewUserName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/user_name_default"
        android:textSize="16sp"
        android:layout_marginBottom="8dp"
        android:padding="8dp" />

    <TextView
        android:id="@+id/textViewPhone"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/phone_default"
        android:textSize="16sp"
        android:layout_marginBottom="24dp"
        android:padding="8dp" />

    <TextView
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text="@string/route_title"
        android:textSize="20sp"
        android:textStyle="bold"
        android:layout_marginBottom="16dp" />

    <TextView
        android:id="@+id/textViewRoute"
        android:layout_width="match_parent"
        android:layout_height="142dp"
        android:layout_marginBottom="24dp"
        android:background="#F5F5F5"
        android:minHeight="100dp"
        android:padding="8dp"
        android:text=""
        android:textSize="16sp" />

    <Button
        android:id="@+id/buttonSetPath"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_set_path"
        android:textSize="18sp"
        android:layout_marginBottom="16dp"
        android:padding="16dp" />

    <Button
        android:id="@+id/buttonCallTaxi"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:enabled="false"
        android:padding="16dp"
        android:text="@string/button_call_taxi"
        android:textSize="18sp" />

</LinearLayout>

```

В данном XML-файле используется `LinearLayout` с вертикальной ориентацией. Интерфейс включает текстовые поля для отображения информации о пользователе и маршруте, а также две кнопки: для установки маршрута и вызова такси. Кнопка вызова такси изначально отключена (`android:enabled="false"`) и активируется только после установки маршрута.

XML-файл третьего Activity — `activity_route.xml`

Ниже представлен код XML-файла графического представления третьего Activity приложения Taxi.

```

<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"

```

```
    android:layout_height="match_parent">



    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:padding="16dp">



        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="100sp"
            android:text="@string/route_input_title"
            android:textSize="24sp"
            android:textStyle="bold" />


        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Точка отправления"
            android:textSize="18sp"
            android:textStyle="bold"
            android:layout_marginBottom="8dp" />


        <Button
            android:id="@+id/buttonGetLocation"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/button_get_location"
            android:textSize="14sp"
            android:layout_marginBottom="8dp"
            android:padding="12dp" />


        <EditText
            android:id="@+id/editTextStartStreet"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/start_street_hint"
            android:inputType="text"
            android:padding="12dp" />


        <EditText
            android:id="@+id/editTextStartHouse"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/start_house_hint"
            android:inputType="text"
            android:padding="12dp" />


        <EditText
            android:id="@+id/editTextStartApartment"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/start_apartment_hint"
```

```
        android:inputType="text"
        android:padding="12dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Точка назначения"
        android:textSize="18sp"
        android:textStyle="bold"
        android:layout_marginBottom="8dp" />

    <Button
        android:id="@+id/buttonSelectOnMap"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="12dp"
        android:text="@string/button_select_on_map"
        android:textSize="14sp" />

    <EditText
        android:id="@+id/editTextEndStreet"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/end_street_hint"
        android:inputType="text"
        android:layout_marginBottom="16dp"
        android:padding="12dp" />

    <EditText
        android:id="@+id/editTextEndHouse"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/end_house_hint"
        android:inputType="text"
        android:layout_marginBottom="16dp"
        android:padding="12dp" />

    <EditText
        android:id="@+id/editTextEndApartment"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/end_apartment_hint"
        android:inputType="text"
        android:padding="12dp" />

    <Button
        android:id="@+id/buttonOk"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/button_ok"
        android:textSize="18sp"
        android:padding="16dp" />

</LinearLayout>
```

```
</ScrollView>
```

В данном XML-файле используется ScrollView для обеспечения возможности прокрутки содержимого при большом количестве элементов. Внутри размещен LinearLayout с полями для ввода адреса точки отправления и точки назначения, а также кнопками для автоматического определения местоположения, выбора точки на карте и подтверждения ввода.

Код Kotlin-файлов первого, второго и третьего Activity

Kotlin-файл первого Activity — RegistrationActivity.kt

Ниже представлен код файла RegistrationActivity.kt, содержащий логику первого Activity приложения Taxi.

```
package com.example.lt2

import android.content.Context
import android.content.Intent
import android.content.SharedPreferences
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class RegistrationActivity : AppCompatActivity() {

    private lateinit var textViewTitle: TextView
    private lateinit var editTextPhone: EditText
    private lateinit var editTextFirstName: EditText
    private lateinit var editTextLastName: EditText
    private lateinit var buttonRegistration: Button
    private lateinit var sharedpreferences: SharedPreferences

    companion object {
        private const val TAG = "RegistrationActivity"
        private const val PREFS_NAME = "TaxiPrefs"
        private const val KEY_PHONE = "phone"
        private const val KEY_FIRST_NAME = "first_name"
        private const val KEY_LAST_NAME = "last_name"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate")
        setContentView(R.layout.activity_registration)

        sharedpreferences = getSharedPreferences(PREFS_NAME,
            Context.MODE_PRIVATE)

        textViewTitle = findViewById(R.id.textViewTitle)
```

```

editTextPhone = findViewById(R.id.editTextPhone)
editTextFirstName = findViewById(R.id.editTextFirstName)
editTextLastName = findViewById(R.id.editTextLastName)
buttonRegistration = findViewById(R.id.buttonRegistration)

// Восстановление сохраненных данных
val savedPhone = sharedpreferences.getString(KEY_PHONE, "")
val savedFirstName = sharedpreferences.getString(KEY_FIRST_NAME, "")
val savedLastName = sharedpreferences.getString(KEY_LAST_NAME, "")

val isRegistered = savedPhone?.isNotEmpty() == true &&
                    savedFirstName?.isNotEmpty() == true &&
                    savedLastName?.isNotEmpty() == true

if (isRegistered) {
    // При повторном запуске - показываем "Вход" и "Log in"
    textViewTitle.text = getString(R.string.login_title)
    editTextPhone.setText(savedPhone)
    editTextFirstName.setText(savedFirstName)
    editTextLastName.setText(savedLastName)
    buttonRegistration.text = getString(R.string.button_log_in)
} else {
    // При первом запуске - показываем "Регистрация" и "Register"
    textViewTitle.text = getString(R.string.registration_title)
    buttonRegistration.text = getString(R.string.button_registration)
}

buttonRegistration.setOnClickListener {
    val phone = editTextPhone.text.toString()
    val firstName = editTextFirstName.text.toString()
    val lastName = editTextLastName.text.toString()

    if (phone.isNotEmpty() && firstName.isNotEmpty() &&
        lastName.isNotEmpty()) {
        // Сохранение данных в SharedPreferences
        sharedpreferences.edit().apply {
            putString(KEY_PHONE, phone)
            putString(KEY_FIRST_NAME, firstName)
            putString(KEY_LAST_NAME, lastName)
            apply()
        }

        // Запуск второго Activity с передачей данных
        val intent = Intent(this, TaxiActivity::class.java).apply {
            putExtra("phone", phone)
            putExtra("firstName", firstName)
            putExtra("lastName", lastName)
        }
        startActivity(intent)
    }
}
}

override fun onStart() {

```

```

        super.onStart()
        Log.d(TAG, "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume")
    }

    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }

    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }

    override fun onRestart() {
        super.onRestart()
        Log.d(TAG, "onRestart")
    }
}

```

В классе `RegistrationActivity` реализована логика регистрации и входа пользователя: использование `SharedPreferences` для сохранения и восстановления данных пользователя между сессиями; проверка наличия сохраненных данных для определения режима работы (регистрация или вход); явный вызов второго Activity (`TaxiActivity`) с передачей данных через `Intent` и методы `putExtra()`; логирование всех методов жизненного цикла Activity с использованием класса `Log` и тега `TAG`.

Kotlin-файл второго Activity — `TaxiActivity.kt`

Ниже представлен код файла `TaxiActivity.kt`, содержащий логику второго Activity приложения `Taxi`.

```

package com.example.lt2

class TaxiActivity : AppCompatActivity() {

    private lateinit var textViewUserName: TextView
    private lateinit var textViewPhone: TextView
    private lateinit var textViewRoute: TextView
    private lateinit var buttonSetPath: Button
    private lateinit var buttonCallTaxi: Button

```

```

private lateinit var sharedPreferences: SharedPreferences

companion object {
    private const val TAG = "TaxiActivity"
    private const val REQUEST_CODE_ROUTE = 1
    private const val PREFS_NAME = "TaxiPrefs"
    private const val KEY_ROUTE_START_STREET = "route_start_street"
    private const val KEY_ROUTE_START_HOUSE = "route_start_house"
    private const val KEY_ROUTE_START_APARTMENT = "route_start_apartment"
    private const val KEY_ROUTE_END_STREET = "route_end_street"
    private const val KEY_ROUTE_END_HOUSE = "route_end_house"
    private const val KEY_ROUTE_END_APARTMENT = "route_end_apartment"
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    Log.d(TAG, "onCreate")
    setContentView(R.layout.activity_taxi)

    sharedPreferences = getSharedPreferences(PREFS_NAME,
        Context.MODE_PRIVATE)

    textViewUserName = findViewById(R.id.textViewUserName)
    textViewPhone = findViewById(R.id.textViewPhone)
    textViewRoute = findViewById(R.id.textViewRoute)
    buttonSetPath = findViewById(R.id.buttonSetPath)
    buttonCallTaxi = findViewById(R.id.buttonCallTaxi)

    // Получение данных из Intent
    val phone = intent.getStringExtra("phone") ?: ""
    val firstName = intent.getStringExtra("firstName") ?: ""
    val lastName = intent.getStringExtra("lastName") ?: ""

    // Вывод информации о пользователе
    textViewUserName.text = getString(R.string.user_name_default) + "
        $firstName $lastName"
    textViewPhone.text = getString(R.string.phone_default) + " $phone"

    // Восстановление сохраненного маршрута
    restoreRoute()

    buttonSetPath.setOnClickListener {
        // Неявный вызов третьего Activity через startActivityForResult
        val intent = Intent().apply {
            action = "com.example.lt2.ACTION_ROUTE"
            addCategory(Intent.CATEGORY_DEFAULT)
        }
        startActivityForResult(intent, REQUEST_CODE_ROUTE)
    }

    buttonCallTaxi.setOnClickListener {
        Toast.makeText(this, getString(R.string.taxi_called_message),
            Toast.LENGTH_LONG).show()
    }
}

```

```

}

private fun restoreRoute() {
    val startStreet = sharedPreferences.getString(KEY_ROUTE_START_STREET,
        "")
    val startHouse = sharedPreferences.getString(KEY_ROUTE_START_HOUSE,
        "")
    val startApartment =
        sharedPreferences.getString(KEY_ROUTE_START_APARTMENT, "")
    val endStreet = sharedPreferences.getString(KEY_ROUTE_END_STREET, "")
    val endHouse = sharedPreferences.getString(KEY_ROUTE_END_HOUSE, "")
    val endApartment =
        sharedPreferences.getString(KEY_ROUTE_END_APARTMENT, "")

    if (startStreet?.isEmpty() == true && endStreet?.isEmpty() ==
        true) {
        val routeInfo = """
            От: $startStreet, д. $startHouse, кв. $startApartment
            До: $endStreet, д. $endHouse, кв. $endApartment

            Вы можете вызвать такси.
        """.trimIndent()

        textViewRoute.text = routeInfo
        buttonCallTaxi.isEnabled = true
    }
}

private fun saveRoute(
    startStreet: String,
    startHouse: String,
    startApartment: String,
    endStreet: String,
    endHouse: String,
    endApartment: String
) {
    sharedPreferences.edit().apply {
        putString(KEY_ROUTE_START_STREET, startStreet)
        putString(KEY_ROUTE_START_HOUSE, startHouse)
        putString(KEY_ROUTE_START_APARTMENT, startApartment)
        putString(KEY_ROUTE_END_STREET, endStreet)
        putString(KEY_ROUTE_END_HOUSE, endHouse)
        putString(KEY_ROUTE_END_APARTMENT, endApartment)
        apply()
    }
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data:
    Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    Log.d(TAG, "onActivityResult: requestCode=$requestCode,
        resultCode=$resultCode")
}

```

```

if (requestCode == REQUEST_CODE_ROUTE && resultCode ==
    → Activity.RESULT_OK) {
    data?.let {
        val startStreet = it.getStringExtra("startStreet") ?: ""
        val startHouse = it.getStringExtra("startHouse") ?: ""
        val startApartment = it.getStringExtra("startApartment") ?: ""
        val endStreet = it.getStringExtra("endStreet") ?: ""
        val endHouse = it.getStringExtra("endHouse") ?: ""
        val endApartment = it.getStringExtra("endApartment") ?: ""

        // Сохранение маршрута в SharedPreferences
        saveRoute(startStreet, startHouse, startApartment, endStreet,
            → endHouse, endApartment)

        val routeInfo = """
            От: $startStreet, д. $startHouse, кв. $startApartment
            До: $endStreet, д. $endHouse, кв. $endApartment

            Вы можете вызвать такси.
        """.trimIndent()

        textViewRoute.text = routeInfo
        buttonCallTaxi.isEnabled = true
    }
}
}

override fun onStart() {
    super.onStart()
    Log.d(TAG, "onStart")
}

override fun onResume() {
    super.onResume()
    Log.d(TAG, "onResume")
}

override fun onPause() {
    super.onPause()
    Log.d(TAG, "onPause")
}

override fun onStop() {
    super.onStop()
    Log.d(TAG, "onStop")
}

override fun onDestroy() {
    super.onDestroy()
    Log.d(TAG, "onDestroy")
}

override fun onRestart() {
    super.onRestart()
}

```

```

        Log.d(TAG, "onRestart")
    }
}

```

В классе `TaxiActivity` реализована логика главного экрана приложения: получение данных из `Intent` с помощью методов `getStringExtra()`; неявный вызов третьего `Activity` через `startActivityForResult()` с использованием `action` и `category`; обработка результата от третьего `Activity` в методе `onActivityResult()`; сохранение и восстановление маршрута с помощью `SharedPreferences`; использование класса `Toast` для отображения всплывающих сообщений; логирование всех методов жизненного цикла `Activity`.

Kotlin-файл третьего Activity — `RouteActivity.kt`

Ниже представлен код файла `RouteActivity.kt`, содержащий логику третьего `Activity` приложения `Taxi` (фрагмент с основными методами).

```

package com.example.lt2

class RouteActivity : AppCompatActivity() {

    private lateinit var editTextStartStreet: EditText
    private lateinit var editTextStartHouse: EditText
    private lateinit var editTextStartApartment: EditText
    private lateinit var editTextEndStreet: EditText
    private lateinit var editTextEndHouse: EditText
    private lateinit var editTextEndApartment: EditText
    private lateinit var buttonOk: Button
    private lateinit var buttonGetLocation: Button
    private lateinit var buttonSelectOnMap: Button

    companion object {
        private const val TAG = "RouteActivity"
        private const val MAP_ACTIVITY_REQUEST_CODE = 2
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate")
        setContentView(R.layout.activity_route)

        editTextStartStreet = findViewById(R.id.editTextStartStreet)
        editTextStartHouse = findViewById(R.id.editTextStartHouse)
        editTextStartApartment = findViewById(R.id.editTextStartApartment)
        editTextEndStreet = findViewById(R.id.editTextEndStreet)
        editTextEndHouse = findViewById(R.id.editTextEndHouse)
        editTextEndApartment = findViewById(R.id.editTextEndApartment)
        buttonOk = findViewById(R.id.buttonOk)
        buttonGetLocation = findViewById(R.id.buttonGetLocation)
        buttonSelectOnMap = findViewById(R.id.buttonSelectOnMap)
    }
}

```

```

buttonGetLocation.setOnClickListener {
    getCurrentLocation()
}

buttonSelectOnMap.setOnClickListener {
    val intent = Intent(this, MapActivity::class.java)
    startActivityForResult(intent, MAP_ACTIVITY_REQUEST_CODE)
}

buttonOk.setOnClickListener {
    val startStreet = editTextStartStreet.text.toString()
    val startHouse = editTextStartHouse.text.toString()
    val startApartment = editTextStartApartment.text.toString()
    val endStreet = editTextEndStreet.text.toString()
    val endHouse = editTextEndHouse.text.toString()
    val endApartment = editTextEndApartment.text.toString()

    // Возврат во второе Activity с результатом
    val resultIntent = Intent().apply {
        putExtra("startStreet", startStreet)
        putExtra("startHouse", startHouse)
        putExtra("startApartment", startApartment)
        putExtra("endStreet", endStreet)
        putExtra("endHouse", endHouse)
        putExtra("endApartment", endApartment)
    }
    setResult(RESULT_OK, resultIntent)
    finish()
}
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data:
→ Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == MAP_ACTIVITY_REQUEST_CODE && resultCode ==
→ Activity.RESULT_OK) {
        data?.let {
            val endStreet = it.getStringExtra("endStreet") ?: ""
            val endHouse = it.getStringExtra("endHouse") ?: ""
            val endApartment = it.getStringExtra("endApartment") ?: ""

            editTextEndStreet.setText(endStreet)
            editTextEndHouse.setText(endHouse)
            editTextEndApartment.setText(endApartment)

            Toast.makeText(
                this,
                "Конечная точка выбрана на карте",
                Toast.LENGTH_SHORT
            ).show()
        }
    }
}

```

```

    override fun onStart() {
        super.onStart()
        Log.d(TAG, "onStart")
    }

    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume")
    }

    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause")
    }

    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop")
    }

    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy")
    }

    override fun onRestart() {
        super.onRestart()
        Log.d(TAG, "onRestart")
    }
}

```

В классе `RouteActivity` реализована логика ввода маршрута движения: явный вызов `MapActivity` через `startActivityForResult()` для выбора точки на карте; обработка результата от `MapActivity` в методе `onActivityResult()`; возврат данных во второе `Activity` через `setResult()` и `finish()`; использование класса `Toast` для отображения информационных сообщений; логирование всех методов жизненного цикла `Activity`.

Выход

В ходе выполнения лабораторной работы было разработано Android-приложение `Taxi` с тремя основными `Activity`, реализующее функционал регистрации пользователя, отображения информации и ввода маршрута движения.

Были изучены и применены следующие технологии и подходы: создание нескольких `Activity` и управление их жизненным циклом; явный и неявный вызов `Activity` с помощью класса `Intent`; передача данных между

Activity через Intent и методы putExtra(); получение результата от Activity через startActivityForResult() и onActivityResult(); хранение данных с помощью класса SharedPreferences; создание всплывающих сообщений с помощью класса Toast; логирование работы приложения с помощью класса Log и анализ логов в LogCat.

Приложение успешно демонстрирует навыки работы с несколькими Activity, передачи данных между ними и управления жизненным циклом компонентов Android-приложений.

Ответы на контрольные вопросы

1. Как с помощью класса Toast создать всплывающее сообщение?

Всплывающее сообщение создают через Toast: формируют объект вызовом Toast.makeText, передавая контекст (например, текущую Activity), текст и длительность показа, после чего вызывают show(). Длительность задаётся константами LENGTH_SHORT (короткий показ) или LENGTH_LONG (более длительный показ).

2. В каких случаях необходимо логирование?

Логирование используют, когда нужно понять, как именно работает приложение: при отладке и поиске ошибок, при анализе падений и обращений пользователей, для наблюдения за жизненным циклом компонентов, а также при измерении производительности и выявлении «узких мест». Кроме того, логи помогают фиксировать важные события и состояния, чтобы позже восстановить картину происходившего.

3. Что представляет собой окно LogCat? Какие существуют уровни логирования?

LogCat — это окно в Android Studio, в котором отображаются сообщения, которые пишут само приложение и система Android. Оно удобно тем, что позволяет искать и фильтровать записи по уровню важности, тегу и другим параметрам.

Основные уровни логирования (от менее критичных к более критичным): VERBOSE (подробные технические сообщения), DEBUG (информация для отладки), INFO (обычные информационные сообщения), WARN (предупреждения), ERROR (ошибки) и ASSERT/WTF (критические ситуации, которые не должны происходить в нормальном режиме).

4. Как программно реализовать логирование?

Логирование реализуют через класс Log, выбирая метод в зависимости от уровня важности сообщения (verbose, debug, info, warn, error и т.д.). Обычно передают TAG (идентификатор, чаще всего имя класса) и текст сообщения, чтобы затем удобно фильтровать записи в LogCat. При обработке

ошибок дополнительно можно передать исключение, чтобы в логе был виден стек вызовов.

5. Как создать новое Activity (опишите работу с java-классом, layout-файлом, файлом конфигурации AndroidManifest.xml)?

Новое Activity создают так: сначала добавляют класс (на Java или Kotlin), который наследуется от Activity/AppCompatActivity, и в onCreate задают разметку через setContentView. Затем создают XML-разметку в каталоге res/layout и описывают в ней интерфейс. После этого Activity регистрируют в AndroidManifest.xml внутри секции application, указывая android:name с именем класса; если планируется неявный запуск, дополнительно добавляют intent-filter с нужным action и category.

6. Для чего используется контекст приложения Context?

Context нужен как «точка доступа» к окружению приложения: через него получают ресурсы (строки, цвета и т.п.), обращаются к системным сервисам, открывают SharedPreferences, работают с файлами и БД, а также запускают компоненты (Activity, Service, BroadcastReceiver). Activity сама является Context, поэтому часто в качестве контекста используют this.

7. Для чего используются объекты класса Intent?

Intent применяют как описание «намерения» выполнить действие: запустить конкретный экран (явно) или запросить выполнение действия по описанию (неявно), передать параметры между компонентами, стартовать сервис или отправить широковещательное сообщение. Также Intent используют, когда нужно запустить Activity и затем получить от него результат.

8. Как выполнить явный вызов Activity?

Явный запуск выполняют, когда точно известно, какой экран нужно открыть: создают Intent, указывают текущий контекст и класс целевого Activity, при необходимости добавляют данные (extras), затем вызывают startActivity.

9. Как выполнить неявный вызов Activity?

Неявный запуск делают через описание действия: в Intent задают action, при необходимости добавляют category и другие параметры (например, data). Дальше система выбирает подходящее Activity по intent-filter, поэтому у целевого Activity в AndroidManifest.xml должен быть объявлен фильтр с тем же action и подходящей категорией (обычно DEFAULT).

10. Какие состояния предусмотрены жизненным циклом Activity?

В жизненном цикле Activity выделяют состояния: created (создано), started (видимо), resumed (в фокусе и доступно для взаимодействия),

paused (теряет фокус), stopped (не видно, но ещё существует) и destroyed (уничтожено).

11. Какие методы автоматически срабатывают при смене состояния Activity? Как можно использовать эти методы?

При смене состояния Android вызывает методы жизненного цикла: onCreate, onStart, onResume, onPause, onStop, onRestart и onDestroy. Их используют, чтобы правильно инициализировать интерфейс, запускать и останавливать ресурсоёмкие операции (камера, GPS, сетевые подписки), сохранять состояние и данные, освобождать ресурсы, а также писать диагностические логи.

12. Как выполняется передача данных с помощью Intent?

Данные передают через extras: в Intent добавляют пары ключ-значение (строки, числа, логические значения и т.д.), а в целевом Activity извлекают их из intent по тем же ключам. Если нужно передать более сложный объект, его обычно делают Parcelable или, реже, Serializable.

13. В каком виде хранятся данные с помощью класса SharedPreferences?

SharedPreferences сохраняет данные как пары «ключ-значение», которые записываются в XML-файл. Обычно хранят строки, целые числа, long, float, boolean и набор строк (Set<String>). Физически файл располагается в каталоге приложения, например: /data/data/имя_пакета/shared_prefs/имя_prefs.xml.

14. В каких случаях целесообразно хранить данные с помощью класса SharedPreferences?

SharedPreferences удобно применять для небольших и простых данных, которые должны переживать перезапуск приложения: пользовательские настройки, флаги вроде «первый запуск» или «пользователь вошёл», а также небольшие значения профиля. Для больших объёмов, сложных структур и данных, которые часто меняются, лучше использовать базу данных или файлы. Для чувствительной информации предпочтительны защищённые хранилища и/или шифрование.