Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

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

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

Студент группы 310901 _____ Усов А.М.
(подпись)

Руководитель _____ Василькова А.Н.
(подпись)

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.

0.1 Скриншоты графических представлений первого, второго и третьего Activity

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

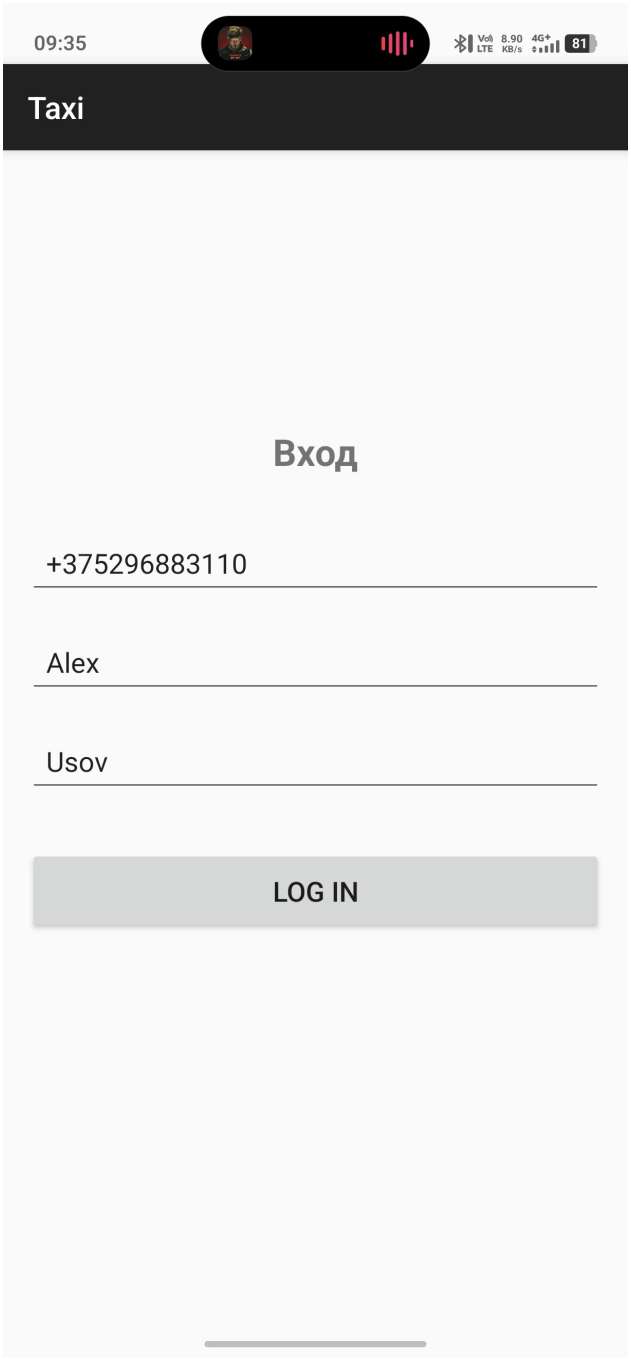

Рисунок 1 – Графическое представление первого Activity (RegistrationActivity)

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.

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

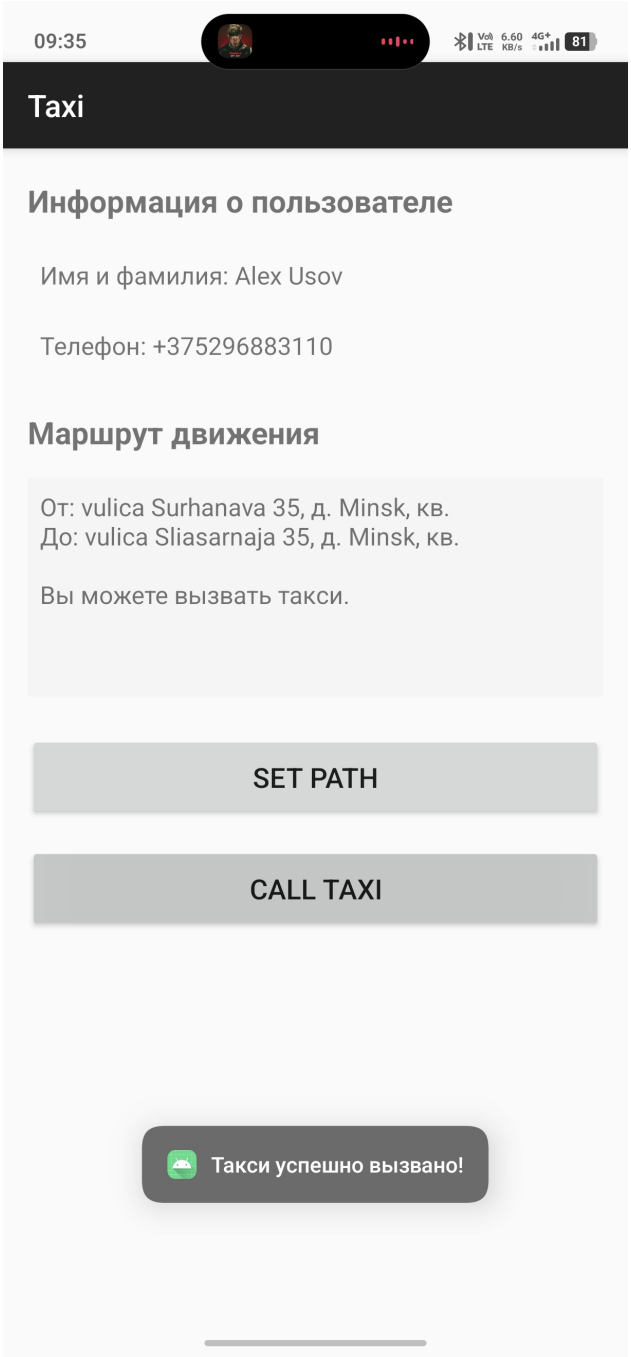

Рисунок 2 – Графическое представление второго Activity (TaxiActivity)

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.

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

Рисунок 3 – Графическое представление третьего Activity (RouteActivity)

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.

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

0.2.1 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`).

0.2.2 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.

0.2.3 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.

0.3 Код Kotlin-файлов первого, второго и третьего Activity

0.3.1 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 сеансами;
- 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`.

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

```
package com.example.lt2

import android.app.Activity
```

```
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.TextView
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

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?.isNotEmpty() == true && endStreet?.isNotEmpty() ==
    ↪  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.

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

```
package com.example.lt2

import android.app.Activity
import android.content.Intent
import android.os.Bundle
import android.util.Log
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
```

```
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.

0.4 Ответы на контрольные вопросы

1. Как с помощью класса Toast создать всплывающее сообщение?

Для создания всплывающего сообщения с помощью класса `Toast` используется статический метод `makeText()`, который принимает контекст, текст сообщения и длительность отображения:

```
Toast.makeText(context, "Текст сообщения", Toast.LENGTH_SHORT).show()
Toast.makeText(context, "Текст сообщения", Toast.LENGTH_LONG).show()
```

`LENGTH_SHORT` — короткое отображение (около 2 секунд), `LENGTH_LONG` — длительное отображение (около 3.5 секунд). Метод `show()` отображает сообщение на экране.

2. В каких случаях необходимо логирование?

Логирование необходимо в следующих случаях:

- отладка приложения для отслеживания выполнения кода и выявления ошибок;
- мониторинг работы приложения в реальных условиях эксплуатации;
- анализ производительности и оптимизация приложения;

– отслеживание жизненного цикла компонентов (Activity, Service и др.);
– диагностика проблем пользователей и анализ крашей приложения;
– документирование важных событий и состояний приложения.

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

LogCat — это инструмент в Android Studio, который отображает логи, генерируемые приложением и системой Android. Окно LogCat позволяет фильтровать логи по тегам, уровням важности и другим критериям.

Уровни логирования в Android (от наименьшего к наибольшему):

– `Log.v()` — VERBOSE — самый подробный уровень, для детальной диагностики;
– `Log.d()` — DEBUG — отладочная информация;
– `Log.i()` — INFO — информационные сообщения;
– `Log.w()` — WARN — предупреждения о потенциальных проблемах;
– `Log.e()` — ERROR — ошибки, требующие внимания;
– `Log.wtf()` — ASSERT — критические ошибки (What a Terrible Failure).

4. Как программно реализовать логирование?

Логирование реализуется с помощью класса `Log` и его статических методов:

```
Log.v(TAG, "Verbose message")
Log.d(TAG, "Debug message")
Log.i(TAG, "Info message")
Log.w(TAG, "Warning message")
Log.e(TAG, "Error message", exception)
```

`TAG` — строка-идентификатор, обычно имя класса или компонента, для удобной фильтрации в LogCat. В качестве третьего параметра можно передать объект исключения (`Exception`) для логирования стека вызовов.

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

Создание нового Activity включает три шага:

1. Создание Java/Kotlin класса:

– создать класс, наследующийся от `AppCompatActivity` (или `Activity`);
– переопределить метод `onCreate()`, в котором вызвать `setContentView()` с ID layout-файла.

2. Создание layout-файла:

– создать XML-файл в папке `res/layout/` (например, `activity_example.xml`);
– определить структуру пользовательского интерфейса с помощью View-элементов.

3. Регистрация в AndroidManifest.xml:

– добавить элемент `<activity>` внутри тега `<application>`;
– указать атрибут `android:name` с полным именем класса (например, `.ExampleActivity`);
– при необходимости добавить `<intent-filter>` для неявного вызова Activity.

6. Для чего используется контекст приложения Context?

Контекст приложения (`Context`) предоставляет доступ к ресурсам и системным сервисам Android:

– доступ к ресурсам приложения (строки, цвета, drawable и др.) через методы `getString()`, `getColor()` и др.;
– доступ к системным сервисам (LocationManager, SharedPreferences и др.);
– запуск Activity, Service, BroadcastReceiver;
– работа с файловой системой и базами данных;
– получение информации о приложении и его компонентах.

`Activity` является контекстом, поэтому методы, требующие контекста, могут использовать `this` или `this@ActivityName`.

7. Для чего используются объекты класса Intent?

Объекты класса `Intent` используются для:

– **явного вызова Activity** — указание конкретного класса Activity для запуска;
– **неявного вызова Activity** — указание действия (`action`) и категории (`category`), система сама определяет подходящее Activity;
– **передачи данных** между Activity через методы `putExtra()`;
– **запуска Service** и отправки BroadcastReceiver;
– **получения результата** от Activity через `startActivityForResult()`.

8. Как выполнить явный вызов Activity?

Явный вызов Activity выполняется с указанием конкретного класса:

```
val intent = Intent(this, TargetActivity::class.java)
intent.putExtra("key", "value")
startActivity(intent)
```

Или с передачей данных:

```
val intent = Intent(this, TargetActivity::class.java).apply {
    putExtra("phone", phone)
    putExtra("name", name)
}
startActivity(intent)
```

9. Как выполнить неявный вызов Activity?

Неявный вызов Activity выполняется с указанием действия и категории:

```
val intent = Intent().apply {
    action = "com.example.app.ACTION_NAME"
    addCategory(Intent.CATEGORY_DEFAULT)
}
startActivity(intent)
```

В `AndroidManifest.xml` Activity должно иметь соответствующий `<intent-filter>`:

```
<activity android:name=".TargetActivity">
    <intent-filter>
        <action android:name="com.example.app.ACTION_NAME" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
```

10. Какие состояния предусмотрены жизненным циклом Activity?

Жизненный цикл Activity включает следующие состояния:

- **Created** — Activity создано, но еще не запущено;
- **Started** — Activity видимо пользователю, но не в фокусе;
- **Resumed** — Activity активно и находится в фокусе (пользователь может с ним взаимодействовать);
- **Paused** — Activity частично видимо, но не в фокусе (например, когда появляется диалог);
- **Stopped** — Activity полностью скрыто, но еще существует в памяти;
- **Destroyed** — Activity уничтожено и удалено из памяти.

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

При смене состояния Activity автоматически вызываются следующие методы:

- `onCreate()` — вызывается при создании Activity, используется для инициализации компонентов и установки layout;

– `onStart()` — вызывается когда Activity становится видимым, используется для запуска анимаций или обновления UI;
– `onResume()` — вызывается когда Activity становится активным, используется для возобновления операций (GPS, камера и др.);
– `onPause()` — вызывается когда Activity теряет фокус, используется для приостановки операций и сохранения данных;
– `onStop()` — вызывается когда Activity становится невидимым, используется для остановки операций и освобождения ресурсов;
– `onRestart()` — вызывается перед `onStart()` при перезапуске остановленного Activity;
– `onDestroy()` — вызывается перед уничтожением Activity, используется для финальной очистки ресурсов.

Эти методы можно использовать для управления ресурсами, сохранения и восстановления состояния, логирования и оптимизации производительности.

12. Как выполняется передача данных с помощью Intent?

Передача данных с помощью `Intent` выполняется через методы `putExtra()`:

```
val intent = Intent(this, TargetActivity::class.java).apply {
    putExtra("stringKey", "значение")
    putExtra("intKey", 42)
    putExtra("booleanKey", true)
    putExtra("parcelableKey", parcelableObject)
}
startActivity(intent)
```

Получение данных в целевом Activity:

```
val stringValue = intent.getStringExtra("stringKey") ?: ""
val intValue = intent.getIntExtra("intKey", 0)
val booleanValue = intent.getBooleanExtra("booleanKey", false)
```

Для передачи сложных объектов используется интерфейс `Parcelable` или `Serializable`.

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

Данные в `SharedPreferences` хранятся в виде пар «ключ-значение» в XML-файле. Поддерживаются следующие типы данных:

– `String` — строковые значения;
– `Int` — целочисленные значения;
– `Long` — длинные целочисленные значения;

– `Float` — числа с плавающей точкой;
– `Boolean` — логические значения;
– `Set<String>` — множества строк (начиная с API 11).

Данные сохраняются в файле `/data/data/<package_name>/shared_pr`

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

`SharedPreferences` целесообразно использовать для хранения:
– настроек пользователя (язык интерфейса, тема оформления, параметры отображения);
– простых данных пользователя (имя, email, телефон);
– флагов состояния приложения (первый запуск, статус авторизации);
– небольших конфигурационных данных;
– данных, которые должны сохраняться между сеансами работы приложения.

`SharedPreferences` не рекомендуется использовать для:
– больших объемов данных (предпочтительнее база данных SQLite);
– сложных структур данных (предпочтительнее база данных или файлы);
– чувствительных данных (предпочтительнее шифрование);
– данных, требующих частых обновлений в многопоточной среде.

0.5 Заключение

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.

0.6 Результаты вывода в лог очередности вызовов методов жизненного цикла первого, второго и третьего Activity

При запуске приложения и переходе между Activity в LogCat отображается следующая последовательность вызовов методов жизненного цикла:

0.6.1

```
D/RegistrationActivity: onCreate
D/RegistrationActivity: onStart
D/RegistrationActivity: onResume
D/RegistrationActivity: onPause
D/TaxiActivity: onCreate
D/TaxiActivity: onStart
D/TaxiActivity: onResume
D/RegistrationActivity: onStop
```

При нажатии кнопки регистрации первое Activity переходит в состояние `onPause`, затем создается второе Activity (`onCreate`), оно запускается (`onStart`) и становится активным (`onResume`), после чего первое Activity останавливается (`onStop`).

0.6.2

```
D/TaxiActivity: onPause
D/RouteActivity: onCreate
D/RouteActivity: onStart
D/RouteActivity: onResume
D/TaxiActivity: onStop
```

При нажатии кнопки «Set path» второе Activity переходит в состояние `onPause`, создается третье Activity (`onCreate`), оно запускается и становится активным, после чего второе Activity останавливается.

0.6.3

```
D/RouteActivity: onPause
D/TaxiActivity: onRestart
D/TaxiActivity: onStart
D/TaxiActivity: onResume
D/RouteActivity: onStop
D/RouteActivity: onDestroy
```

При нажатии кнопки «OK» в третьем Activity оно переходит в состояние `onPause`, второе Activity перезапускается (`onRestart`), запускается (`onStart`) и становится активным (`onResume`), после чего третье Activity останавливается и уничтожается (`onDestroy`).

0.6.4 При возврате пользователя на главный экран через кнопку «Назад»:

```
D/TaxiActivity: onPause
D/RegistrationActivity: onRestart
D/RegistrationActivity: onStart
D/RegistrationActivity: onResume
D/TaxiActivity: onStop
D/TaxiActivity: onDestroy
```

Второе Activity переходит в состояние `onPause`, первое Activity перезапускается, запускается и становится активным, после чего второе Activity останавливается и уничтожается.