

Лабораторная работа № 1

Создание макетов для Android-приложения.

1. Теоретический материал.

1.1. Язык Kotlin

В Kotlin необходимо явно обрабатывать пустые ссылки.

Kotlin различает типы с поддержкой и без поддержки значения null.

Kotlin использует такое разделение для всех типов: на поддерживающие null (с окончанием ?) и не поддерживающие null: например, String и String?, Person и Person?.

Any — это родительский класс для всех классов, не поддерживающих значения null, а Any? — это родительский класс для всех классов с поддержкой этого значения любой тип, не поддерживающий null, является дочерним типом соответствующего типа, поддерживающего null

Оператор точки (.), который также называют оператором разыменования, нельзя использовать с поддерживающими null-типами, потому что он может вызвать NPE (NullPointerException):

```
val city: City? = map[companyName]?.manager?.address?.city
```

Kotlin позволяет также принять на себя всю ответственность за происходящее, т. е. за возможные исключения NPE:

```
val city: City? = map[companyName]!! .manager!! .address!! .city // В этом фрагменте, если какой-то элемент будет иметь значение null (кроме city), будет возбуждено исключение NPE.
```

Kotlin поддерживает переопределение методов и свойств в классах потомках при наследовании.

В производном классе для переопределения свойства перед ним указывается аннотация **override**. Если свойство определяется через первичный конструктор, то также перед его определением ставится аннотация open

При переопределении в производном классе к этим функциям применяется аннотация **override**. Чтобы функции базового класса можно было переопределить, к ним применяется аннотация open.

Запретить дальнейшее переопределение функции в классах-наследниках можно с применением ключевого слова **final**.

1.2. Жизненный цикл Activity

Компонентом для создания визуального интерфейса в приложении Android является activity (активность, действие). Все объекты activity представляют собой объекты класса **android.app.Activity**.

После запуска приложения его activity могут находиться в одном из состояний Активное (Resumed), Приостановленное (Paused), Остановленное (Stopped). При переходах из состояния в состояние вызываются методы жизненного цикла: **onCreate**, **onStart**, **onResume**, **onPause**, **onStop**, **onDestroy**. Отношения между методами жизненного цикла приведены на рисунках №№ 1-2.

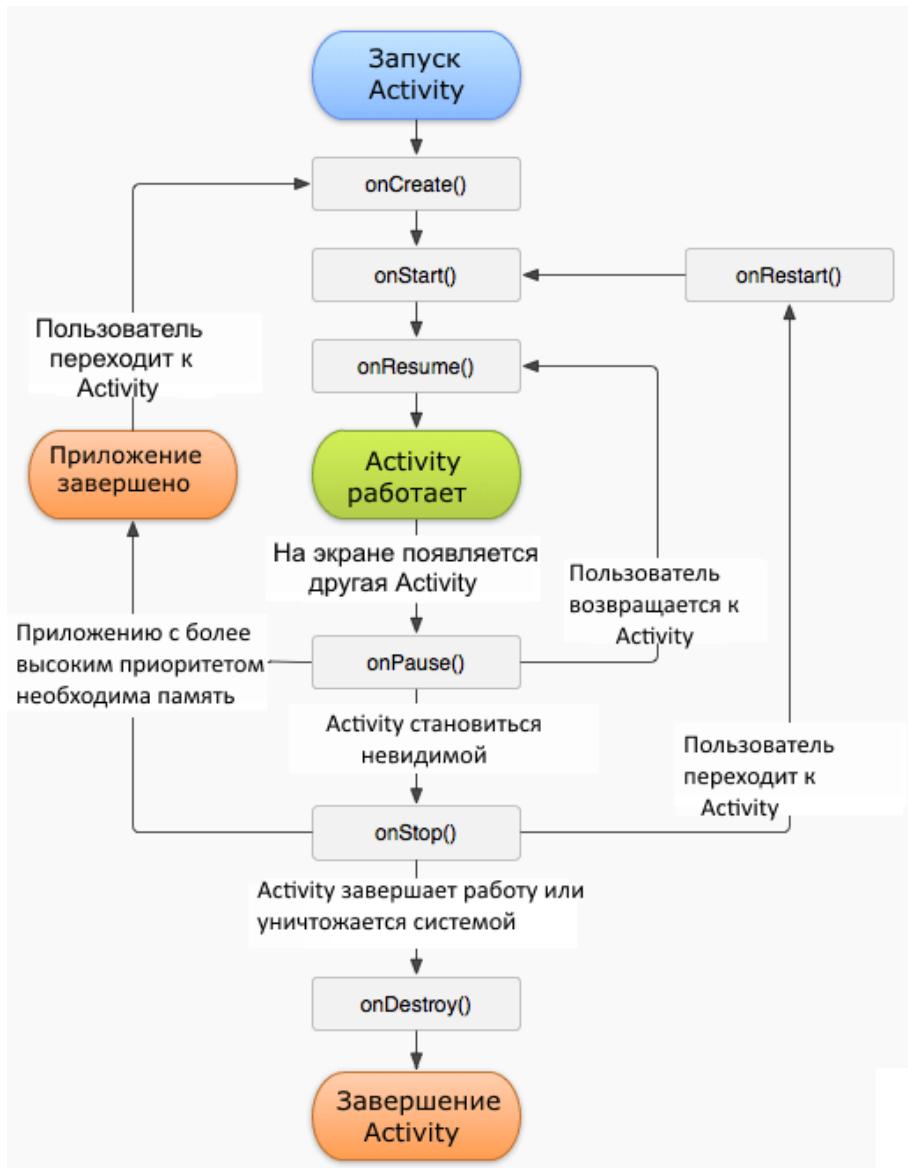


Рисунок № 1.

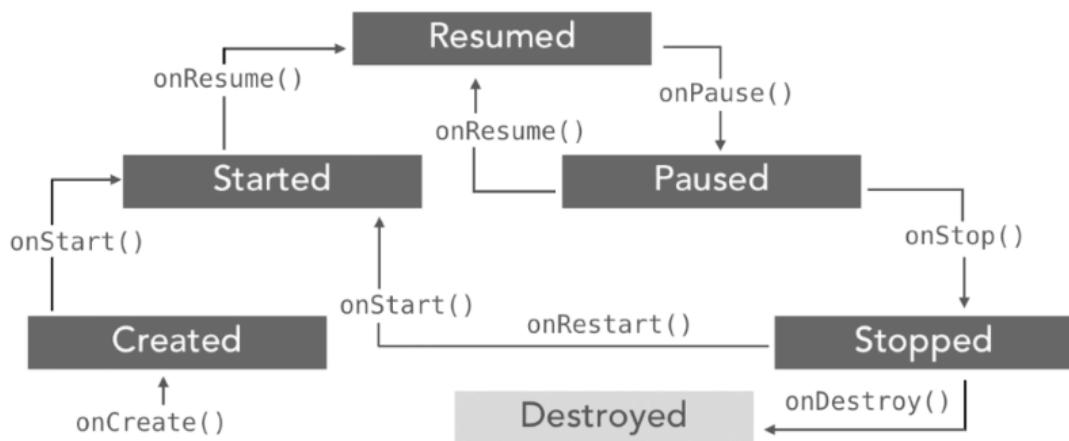


Рисунок № 2.

Смена состояния activity может происходить как из-за действий пользователя, например поворот экрана, так и из-за действий самого приложения, например, из-за переключения пользователя с одного приложения на другое, или если внутри приложения открывается новая activity.

1.3. Запуск Activity

Наиболее распространенный способ наладить взаимодействие между Android приложениями или между activity в рамках одного приложения – использование класса **android.content.Intent**.

Объект класса **Intent** необходимо в классе **Activity** из которого будет осуществляться вызов другой **Activity**. Запуск **Activity** осуществляется путем передачи созданного **intent** в метод **startActivity** если не требуется возвращаемого значения. При создании явного **intent** в конструктор класса передается текущий контекст и название класса, в который будет передан **intent**. Пример приведен в листинге № 1.

Листинг 1

```
val openIntent = Intent(this, SecondActivity::class.java)
startActivity(openIntent)
```

Открываемая таким образом **activity** должна быть прописана в манифесте, иначе ее запуск вызовет сбой работы приложения (приведен на рисунке № 3).

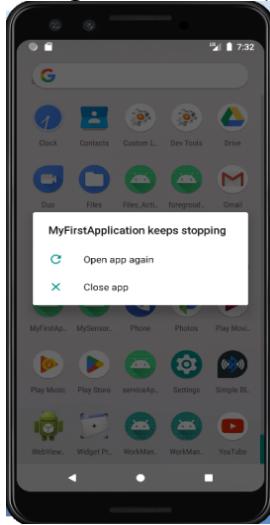


Рисунок 3

После создания **intent** в нем может быть указана информация, которую необходимо передать в запускаемую **activity**, как показано в листинге № 2.

Листинг 2

```
val openIntent = Intent(this, SecondActivity::class.java)
openIntent.putExtra("message", "Hello from FirstActivity")
startActivity(openIntent)
```

В запускаемой **activity** передаваемая через **intent** информация доступна для извлечения и дальнейшего использования, как продемонстрировано в листинге № 3.

Листинг 3

```
override fun onCreate(savedInstanceState: Bundle?) {
...
val arguments: Bundle? = intent.extras
val str: String = arguments!!["message"].toString()
```

Intent имеет внутри объект **Bundle** который используется для хранения и передачи данных. В объект **Bundle** сохраняются переданные в **Intent** пары. **Intent** предоставляет интерфейс для сохранения, но фактически хранит данные

Bundle.

В Android объект Bundle используется для передачи данных между компонентами приложения, чаще всего между различными Activity. Он работает по принципу «ключ-значение», позволяя упаковывать и распаковывать различные типы данных, такие как строки, числа и другие примитивные типы, а также объекты, реализующие интерфейс Parcelable, для их последующего получения по ключу.

Можно напрямую использовать Bundle, для сохранения передаваемых значений. Пример приведен в листинге № 4.

Листинг 4

```
val bundle = Bundle()
bundle.putString("message", "Hello from FirstActivity")
bundle.putInt("user_id", 42)
val intent = Intent(this, SecondActivity::class.java)
intent.putExtra("my_bundle", bundle) // Передаем Bundle как дополнительный
// параметр
startActivity(intent)
```

Неявный Intent создается похожим образом. Пример создания неявного Intent для вызова приложения для совершения звонка приведен в листинге № 5.

Листинг 5

```
val intent = Intent(Intent.ACTION_DIAL)
number = Uri.parse("tel:+70000000000")
intent.setData(number);

if (intent.resolveActivity(packageManager) != null)
    startActivity(intent)
} else {
    // Если нет приложения для совершения звонка
    Log.e("MainActivity", "Нет приложения для совершения звонка")
}
```

Пример из листинга № 5 можно немного сократить используя возможности языка kotlin. Пример приведен в листинге № 6.

Листинг 6

```
val intent = Intent(Intent.ACTION_DIAL).apply {
    data = Uri.parse("tel:+70000000000")
}

if (intent.resolveActivity(packageManager) != null)
    startActivity(intent)
} else {
    // Если нет приложения для совершения звонка
    Log.e("MainActivity", "Нет приложения для совершения звонка")
}
```

1.4. Сохранение состояния Activity

При повороте экрана данные которые были введены в поля ввода, а также состояния кнопок и другие данные будут потеряны. Чтобы избежать подобных ситуаций и следует сохранять и восстанавливать состояние Activity. Для этого

следует использовать методы **onRestoreInstanceState** и **onSaveInstanceState**. Метод **onRestoreInstanceState** восстанавливает состояние Activity, метод **onSaveInstanceState** восстанавливает сохраненное состояние. Оба этих метода в качестве параметра принимают объект Bundle, который хранит состояние activity. Пример использования приведен в листинге № 7.

Листинг 7

```
override fun onSaveInstanceState(outState: Bundle) {  
    super.onSaveInstanceState(outState)  
    outState.putString("my_text", editText.text.toString())  
}  
  
override fun onRestoreInstanceState(savedInstanceState: Bundle) {  
    super.onRestoreInstanceState(savedInstanceState)  
    val savedText = savedInstanceState.getString("my_text")  
    editText.setText(savedText)  
}  
  
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main) }
```

Можно не использовать **onRestoreInstanceState**, а использовать параметр метода **onCreate** в который в качестве аргумента передается сохраненный с помощью **onSaveInstanceState** объект Bundle. Пример использования приведен в листинге № 8.

Листинг 8

```
override fun onSaveInstanceState(outState: Bundle) {  
    super.onSaveInstanceState(outState)  
    outState.putString("my_text", editText.text.toString()) // Сохраняем текст из  
EditText  
}  
  
override fun onCreate(savedInstanceState: Bundle?) {  
    super.onCreate(savedInstanceState)  
    setContentView(R.layout.activity_main)  
  
    if (savedInstanceState != null) {  
        val savedText = savedInstanceState.getString("my_text")  
        editText.setText(savedText) // Восстанавливаем текст  
    }  
}
```

1.5. Работа с логированием

Для логирования событий используется класс **android.util.Log**.

Логирование производится вызовом методов **Log.*()**, в которые передается TAG - случайное строковое значение и строка, которая выводится в консоли Logcat в нижней части Android Studio. Если эта консоль по умолчанию скрыта, то мы можем перейти к ней через пункт меню View -> Tool Windows -> Logcat.

Сообщения системного журнала выводятся в консоль AndroidStudio, как на рисунке №4.

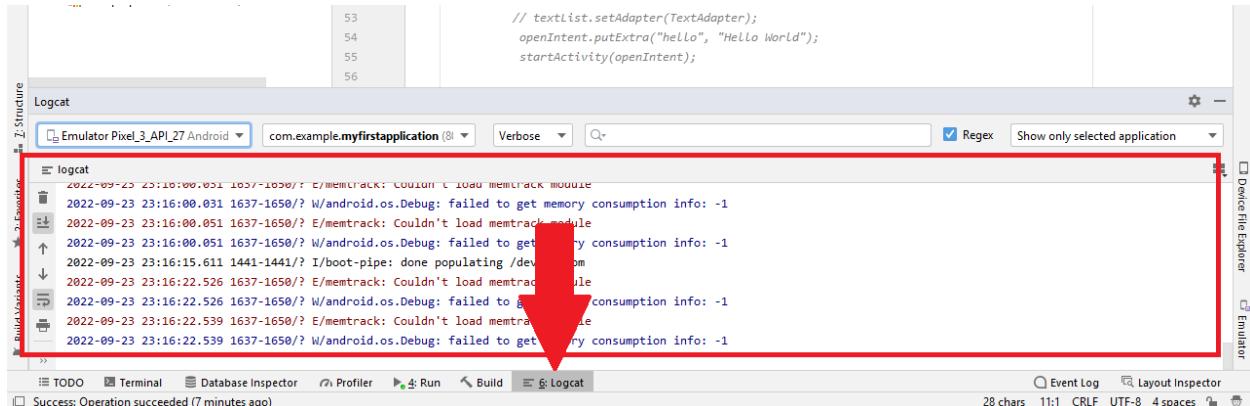


Рисунок 4

Сообщения разбиваются по категориям и для отображения каждой категории существуют собственные методы:

- **Log.e()** - ошибки (error)
- **Log.w()** - предупреждения (warning)
- **Log.i()** - информация (info)
- **Log.d()** - отладка (degub)
- **Log.v()** - подробности (verbose).

Пример вызова метода:

```
Log.i("AppLogger", "Сообщение")
```

Результат приведен на рисунке № 5.

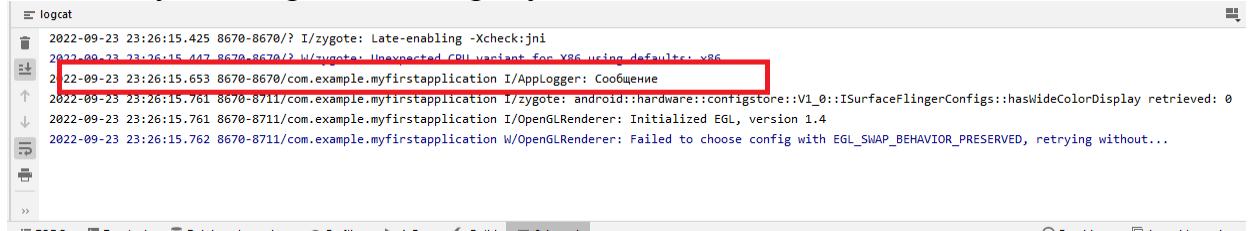


Рисунок № 5

2. Задание для самостоятельной реализации

1) Создать следующие окна приложения:

- Окно «Приветствие» в приложении содержащее логотип, поле ввода логина и пароля, кнопки «Вход», «Регистрация» и «Выход».

- Окно «Меню», содержащее кнопки: «Старт», «Профиль», «Настройки», «Сообщить о проблеме», «Выход».

- Окно «Настройки» имеющее минимум одну настройку - смена стиля День/Ночь.

- Окно «Профиль»: вывести основные поля из формы регистрации из лабораторной № 2.

2) Реализовать переключение Activity:

- При нажатии кнопки «Вход», должно отображаться окно «Меню».

- При нажатии кнопки «Регистрация», должно отображаться окно «Регистрация» реализованное в лабораторной работе № 2.

- При нажатии на кнопку «Профиль», должно отображаться окно

«Профиль».

- При нажатии на кнопку «Настройки», должно отображаться окно настройки.

- При нажатии кнопки «Сообщить о проблеме», должно быть открыто либо почтовое приложение в системе, либо приложение для совершения звонков.

3) В окне «Вход» и «Регистрация» предусмотреть сохранение заполненных полей при повороте экрана.

4) Для activity представляющих окна «Вход» и «Регистрация» реализовать все методы жизненного цикла. В каждом методе реализовать логирование выводящее сообщение с названием activity и сработавшего метода жизненного цикла.

5) Вынести все строковые ресурсы в файл ресурсов. (При желании и другие ресурсы)

3. Вопросы для самопроверки

- Что такое Activity?

- Как создать Activity?

- Как использовать Activity в приложении?

- Как сделать Activity главной?

- Какие методы жизненного цикла Activity вы знаете?

- Какие методы жизненного цикла Activity выполняются в противоположных случаях?

- Для чего используется метод onCreate?

- Для чего используется метод onStart?

- Для чего используется метод onStop?

- Для чего используется метод onPause?

- Для чего используется метод onResume?

- Для чего используется метод onDestroy?

- Для чего используется метод onRestart?

- Как переопределить метод жизненного цикла Activity?

- В каком методе жизненного цикла Activity нужно производить связывание макета и Activity?

- Какие методы жизненного цикла Activity вызываются при повороте смартфона с открытым приложением?

- Какие методы жизненного цикла Activity вызываются при сворачивании приложения?

- Какие методы жизненного цикла Activity вызываются при возвращении свернутого приложения?

- Какие методы жизненного цикла вызываются при закрытии приложения?

- Какие методы жизненного цикла вызываются при сворачивании приложения, а затем закрытия его из диспетчера?

- Как открыть из одного Activity другое?

- Что такое Bundle?

- Для чего используется Bundle?

- Что такое Intent?

- Какие виды Intent бывают?

- Как передать данные через Intent?

- Что используется внутри Intent для передачи данных?
- Что такое явный Intent?
- Что такое неявный Intent?
- Как создать Intent?
- Как с помощью Intent вызвать другое Activity?
- Как сохранить данные используемых в Activity?
- Как реализовать логирование в Android?