Лабораторная работа №4,5

«Обработка событий: Взаимодействие с различными элементами на экране. Взаимодействие Activity с элементами экрана»

Цели:

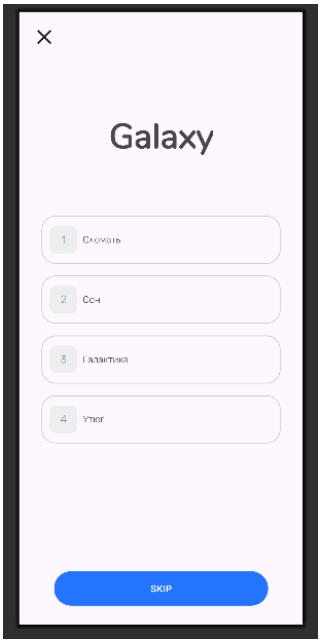
- научиться обращаться к элементам разметки в активити при помощи идентификаторов и ViewBinding;
- научиться изменять атрибуты в файле-разметки.

ПРАКТИЧЕСКАЯ ЧАСТЬ

Мы научились обращаться к элементам экрана, теперь начинаем взаимодействовать с ними. И сейчас перед нами стоит задача научиться реагировать на нажатия кнопок и изменять контент на экране в зависимости от действия. Мы учимся работать с инструментами, поэтому, чтобы не занимать много времени допускаем некоторые упрощения.

Дизайн-макет мобильного приложения можно посмотреть здесь https://clck.ru/34ow4g.

Экран изучения слова открывается перед нами в виде, который продемонстрирован в макете «01.1 – learning». Все слова помечены серым, отображается кнопка SKIP. Приведем верстку в это исходное положение. Для этого скройте атрибутом visibility инфо блок и вернем кнопку пропуска.



Теперь нужно подготовить сценарий, по которому будем действовать. При разработке программы лучше всего сначала проговорить (или написать

на листочке) что нужно сделать.

- При клике на правильное слово:
  - окрашивается фон цифры варианта и контур контейнера в зеленый цвет, цифру в белый;
  - скрывается кнопка пропуска;
  - отображается блок с успешным блоком и кнопкой продолжения.
- При клике на неправильное слово мы
  - окрашивается контейнер и его элементы в красный;
  - подсвечивается зеленым правильное слово;
  - показывается красный блок, информирующий что вариант неверный и кнопку продолжить.
- По кнопке продолжить сбрасывается состояние до исходного. Так как больше пока что показывать нечего.

Из описания выше можно выделить 3 состояния контейнера:

1) Нейтральный ответ (когда ничего не выбрано);
2) Правильный ответ;
3) Неправильный ответ.

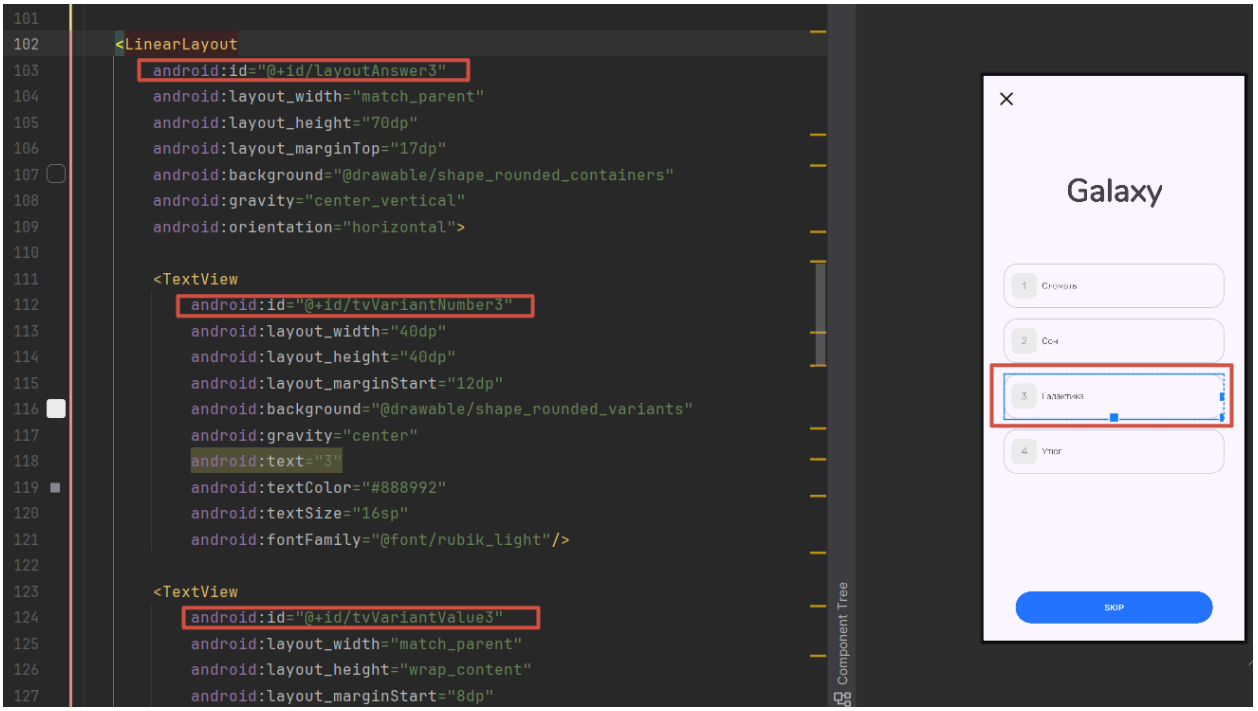
Обработка правильного ответа

Допустим 3 элемент - это правильный ответ (так и есть на самом деле). Чтобы применить к нему особые атрибуты, сперва необходимо отловить нажатие по этому контейнеру.

В Android все действия, связанные с пользовательским вводом, обрабатываются через обработчики событий. Когда пользователь производит какое-то действие на экране, например, нажимает на кнопку, система генерирует событие. Обработчик событий реагирует на это событие и мы можем задать любое действие для выполнения.

Обработчик событий можно привязать к любому view на экране. В нашем случае это контейнер, поэтому сперва задаем id элементу лейаута - layoutAnswer3. Сразу же зададим id для внутренних элементов которым надо

будет передавать окрашивание по клику - tvVariantNumber3 и tvVariantValue3:



Чтобы работать с идентификаторами воспользуемся ViewBinding. Для этого:

– откройте файл gradle «build.gradle.kts (Module :app)»

– в блоке android перед его закрывающей фигурной скобкой вставьте код:

– синхронизируйте проект: кнопка Sync now в правом верхнем углу. Дождитесь завершения синхронизации

– откройте активити MainActivity и измените код:

```
10  class MainActivity : AppCompatActivity() {
11
12      private lateinit var binding: ActivityLearnWordBinding
13
14      override fun onCreate(savedInstanceState: Bundle?) {
15          super.onCreate(savedInstanceState)
16          binding = ActivityLearnWordBinding.inflate(layoutInflater)
17          setContentView(binding.root)
18      }
19  }
```

Не забудьте импортировать ActivityLearnWordBinding (красная лампочка слева или сочетание клавиш Alt+Enter).

Итак, в этой активити будем отслеживать клик, поэтому для его отлова используется метод setOnClickListener(). Выбираем тот, что с лямбдой:

Сигнатура лямбды - it с типом View. Эта view и есть тот объект, к которому мы обращаемся, то есть контейнер слова. И все что происходит внутри фигурных скобок будет выполнено при клике по этому контейнеру.

Создадим метод **markAnswerCorrect()** для обработчика нажатия. Этот метод будем вызывать при клике на третий элемент:

```
14    override fun onCreate(savedInstanceState: Bundle?) {
15        super.onCreate(savedInstanceState)
16        binding = ActivityLearnWordBinding.inflate(layoutInflater)
17        setContentView(binding.root)
18
19        binding.layoutAnswer3.setOnClickListener {
20            markAnswerCorrect()
21        }
22    }
23
24    private fun markAnswerCorrect() {
25        TODO( reason: "Not yet implemented")
26    }
27 }
```

Пока этот метод ничего не делает. Будет следовать по сценарию:

- окрашивается фон цифры варианта и контур контейнера в зеленый цвет, цифру в белый:

Для обводки контура правильно выбранного ответа будем задавать новый ShapeDrawable с зеленым контуром. Создайте его, взяв цвет из макета (можно скопировать готовый shape для обводки **shape_rounded_containers** и назвать копию **shape_rounded_containers_green** и изменить только цвет).

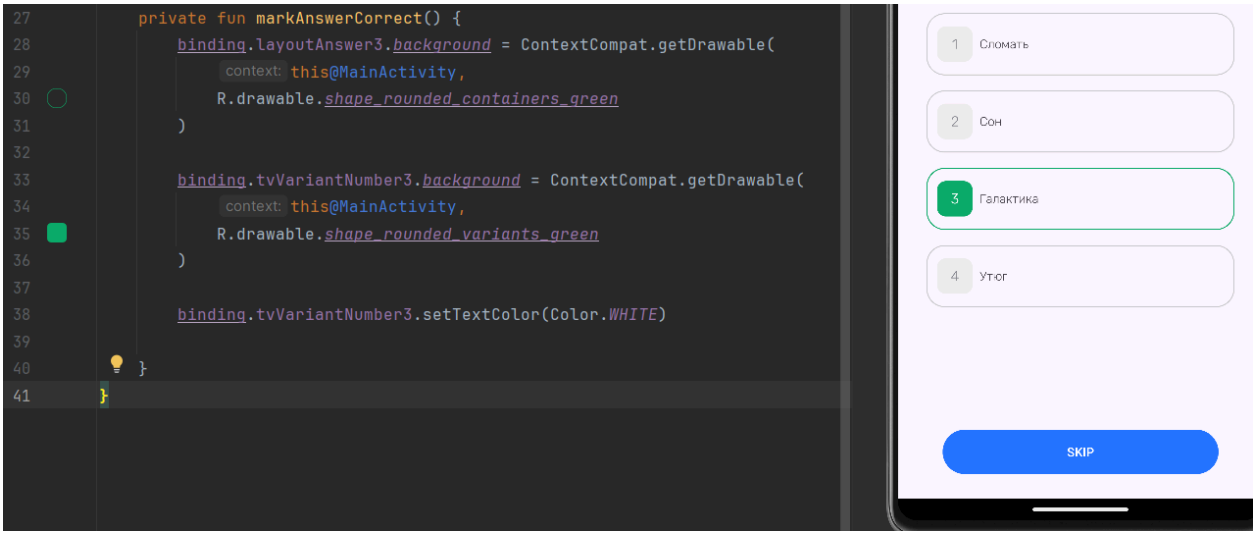
Процедура обращения к ресурсам аналогичная к цвету из прошлого урока. За исключением того, что поменялся тип. Мы задаем ShapeDrawable, поэтому изменился метод и указание типа drawable поле класса R.

```
private fun markAnswerCorrect() {
    binding.layoutAnswer3.background = ContextCompat.getDrawable(
        context: this@MainActivity,
        R.drawable.shape_rounded_containers_green
    )
}
```

По аналогии задаем новый фон для контейнера с цифрой. Используем **shape_rounded_variants_green**.

Процедура обращения к ресурсам аналогичная, за исключением цвета. Можно конечно сделать как при обращении к атрибуту drawable, но текст при

изменении будет являться стандартным – белым, поэтому воспользуемся обширной библиотекой Color и выберем белый цвет (WHITE):



Обязательно проверьте работоспособность при запуске.

А еще задать цвет для текста «Галактика» можно третьим способом:

```
binding.tvVariantValue3.setTextColor(Color.parseColor( colorString: "#0EAD69"))
```

- скрывается кнопка пропуска:

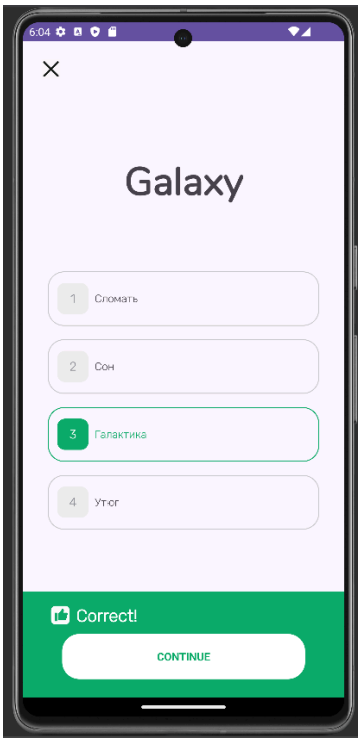
При правильном ответе нужно скрыть кнопку пропуска слова - добавляем id кнопки **btnSkip** и id для лайаута **layoutResult**, который скрыли в начале лабораторной.

Обращаемся к ним и устанавливаем значения:

```
            binding.tvVariantValue3.setTextColor(Color.parseColor( colorString: "#0EAD69"))

            binding.btnSkip.visibility = View.GONE
            binding.layoutResult.visibility = View.VISIBLE
        }
    }
```

- отображается блок с успешным блоком и кнопкой продолжения:



Обработка неправильного ответа

Создадим метод **markAnswerWrong()** для обработчика нажатия. Этот метод будем вызывать при клике на остальные элементы, которые будут являться неправильными. Пока создадим для обработчика для первого варианта ответа:

```
    binding.layoutAnswer1.setOnClickListener {
        markAnswerWrong()
    }
}

private fun markAnswerWrong() {
    TODO( reason: "Not yet implemented")
}
```

Задание 1: Проделайте ту же работу что для варианта с правильным ответом. Не забудьте про цвет из макета.

После выполнения задания при тестировании можно заметить, что выводится контейнер с правильным ответом.

Задание 2: Для неправильного ответа нужно изменить:

1) цвет контейнера;

2) иконку с пальцем вниз;

3) текст «Wrong!»;

4) цвет текста внутри кнопки.

Обработка нейтрального состояния

В целом тут все должно быть понятно. Обращаемся к “айдишникам”, меняем цвета и ShapeDrawable на исходные. Но хотелось бы продемонстрировать реализацию поизящнее, показав красоту Kotlin.

Сначала объявим блок with(binding), чтобы сократить код за счет отсутствия повторений вызова binding:

```
private fun markAnswerNeutral() {
    with(binding){

}
```

Далее мы имеем два лейаута layoutAnswer1 и layoutAnswer3 к которым нужно применить одну и ту же стилизацию. А когда действия необходимо повторить нужен цикл. Объявим цикл for и в нем же создаем список из требуемых для обработки лейаутов:

```
private fun markAnswerNeutral() {
    with(binding){
        for (layout in listOf(layoutAnswer1, layoutAnswer3)) {

        }
```

Теперь просто применим свойство background к переменной layout:

```
with(binding){
    for (layout in listOf(layoutAnswer1, layoutAnswer3)) {
        layout.background = ContextCompat.getDrawable(
            context: this@MainActivity,
            R.drawable.shape_rounded_containers,
        )
    }
}
```

То же самое провернем и для значений слов, окрашивая их обратно в серый цвет:

```
for (textView in listOf(tvVariantValue1, tvVariantValue3)) {
    textView.setTextColor(
        Color.parseColor( colorString: "#888992")
    )
}
```

Наконец обработаем TextView с цифрой. Здесь понадобится две модификации на одно вью. Можно вызывать их по отдельности, а можно применить специальную конструкцию apply{}. Это extension функция, которая применяется к объекту и вызывает для него методы внутри фигурных скобок. Таким образом один раз обратившись к переменной textView, устанавливаем и фон, и цвет текста:

```
for (textView in listOf(tvVariantNumber1, tvVariantNumber3)) {
    textView.apply {
        background = ContextCompat.getDrawable(
            context: this@MainActivity,
            R.drawable.shape_rounded_variants,
        )
        setTextColor(Color.parseColor( colorString: "#888992"))
    }
}
```

Наконец, убираем отображение инфоблока и возвращаем кнопку Skip:

```
layoutResult.isVisible = false
btnSkip.isVisible = true
```

Протестируйте приложение.