

ЗАНЯТИЕ 2.10

ПРАКТИЧЕСКОЕ ЗАНЯТИЕ

Тема: Создание меню в Android-приложениях. Отображение списков с использованием адаптеров.

Для решения упражнений данного практического занятия необходимо выполнить задания занятий №2.6 и №2.8.

Упражнение №1

Создайте новое Android-приложение для телефона с использованием шаблона Empty Activity (или Empty View Activity, в зависимости от версии Android Studio).

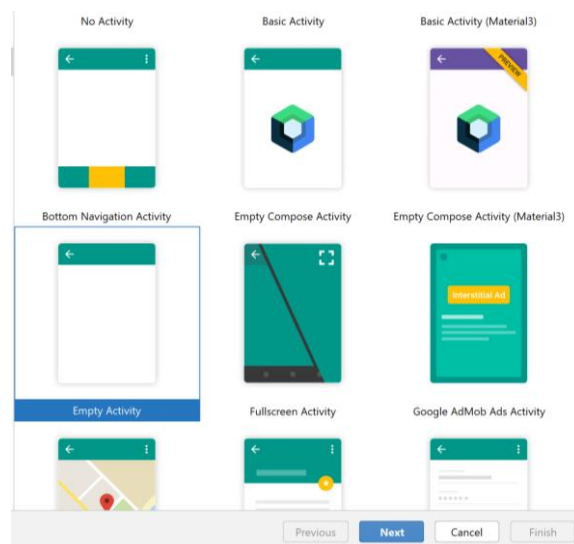


Рисунок 1

Установите минимальную версию API 21. Язык разработки – Java (не Kotlin). Имя приложения – ActionSimpleApp.

Empty Activity

Creates a new empty activity

Name:

Package name:

Save location:

Language:

Minimum SDK:

i Your app will run on approximately **99,3%** of devices.
[Help me choose](#)

☐ Use legacy android.support libraries ?

Рисунок 2

Дождитесь создания приложения и корректного обновления плагина Gradle. Перейдите в файл макета (в режим Code) **activity_main.xml**. Замените контейнер **ConstraintLayout** на **RelativeLayout** и поместите в центр экрана кнопку **Button** (существующий виджет **TextView** можно удалить):

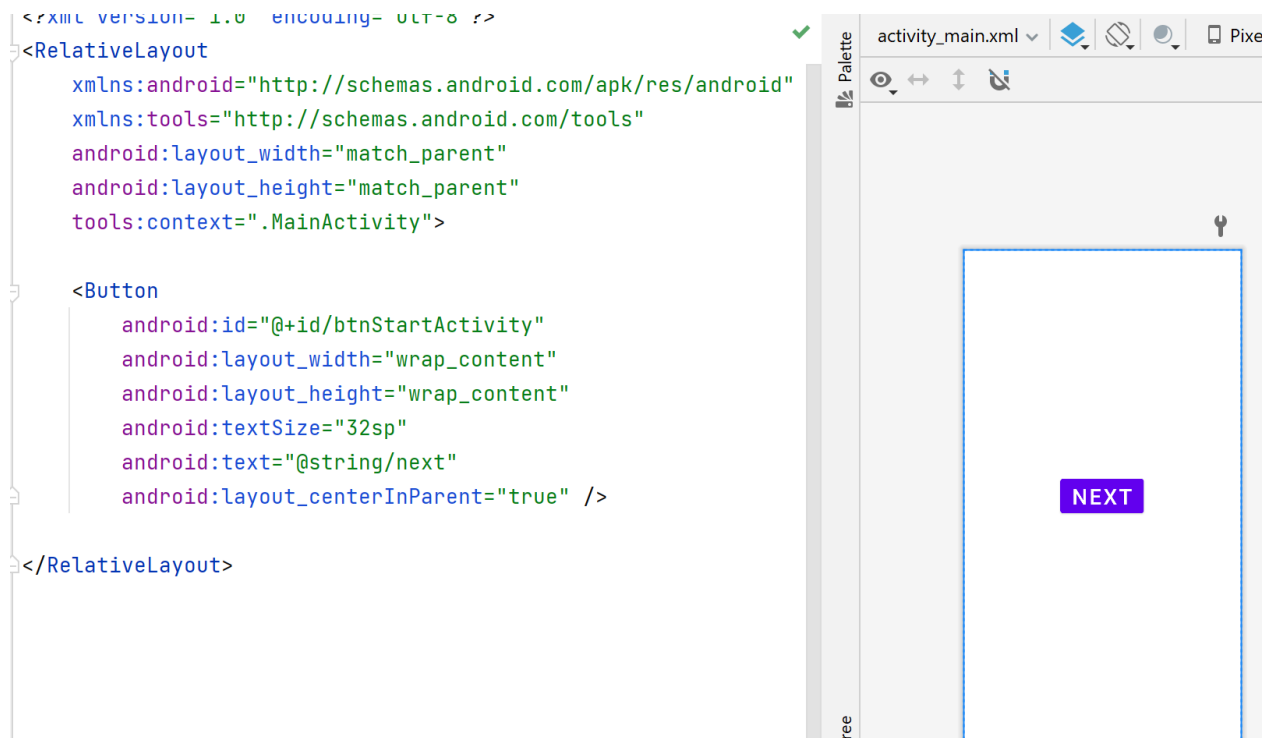


Рисунок 1

Обратите внимание, что ссылки на строки используются из файла ресурсов **strings.xml**:

```
<string name="app_name">ActionSimpleApp</string>
<string name="next">Next</string>
```

Рисунок 2

Создайте еще одну **activity**. Быстро это можно сделать через контекстное меню проекта. Выберите шаблон **Empty Activity** (или **Empty Views Activity** в зависимости от версии IDE).

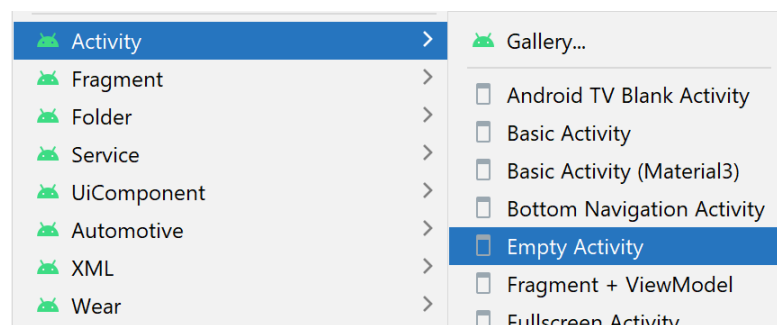
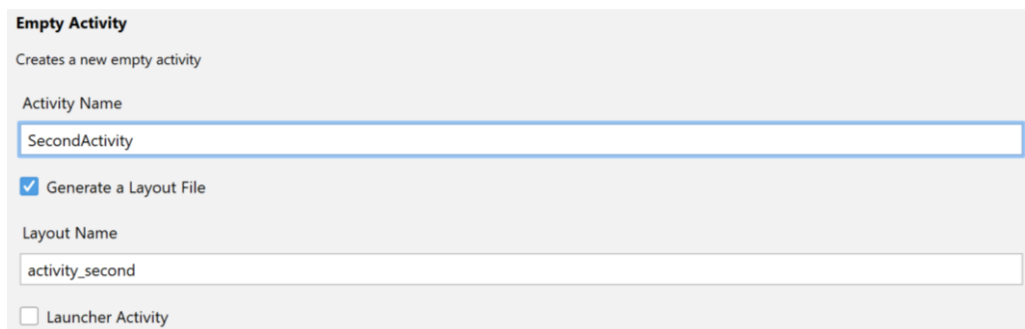


Рисунок 3

Назовите новую activity понятным для вас названием и завершите создание:



Empty Activity
Creates a new empty activity

Activity Name
SecondActivity

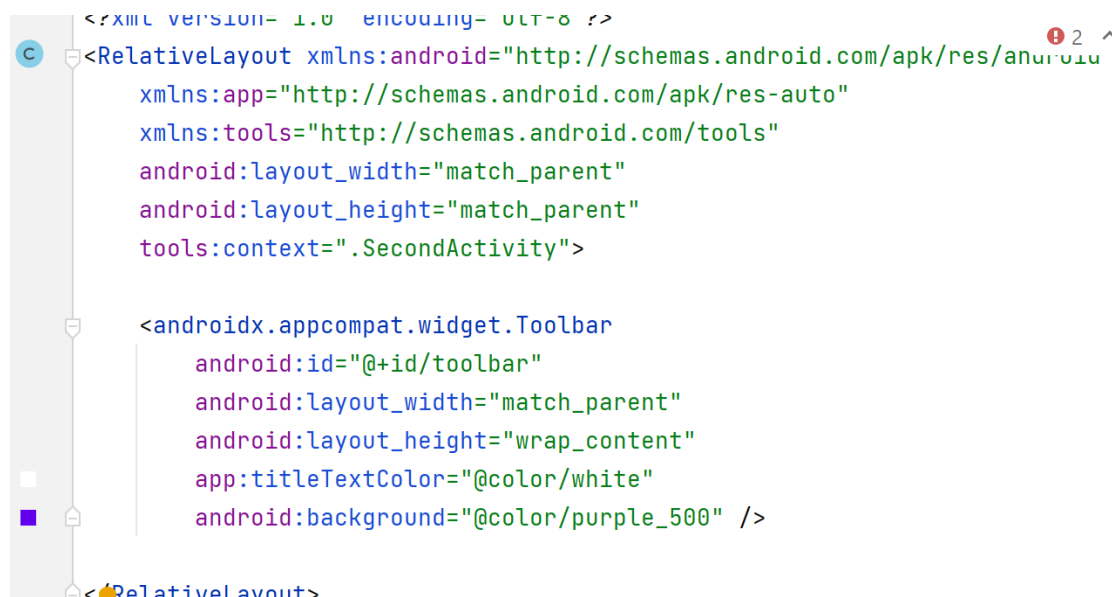
☒ Generate a Layout File

Layout Name
activity_second

☐ Launcher Activity

Рисунок 4

Для новой activity автоматически создался layout-файл макета. Откройте его. Он пустой (только контейнер `ConstraintLayout`). Замените контейнер `ConstraintLayout` на `RelativeLayout` и поместите в него новый виджет – `Toolbar`:



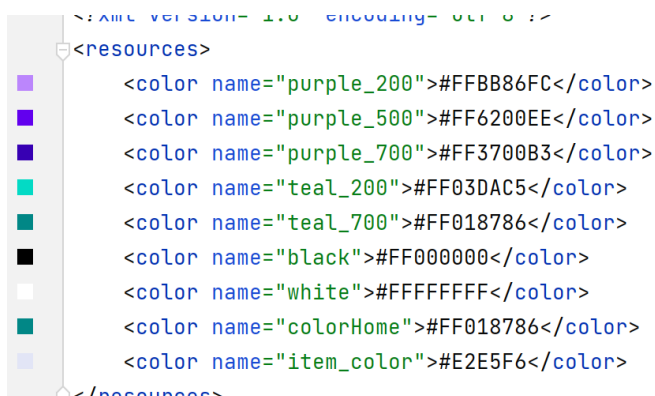
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:titleTextColor="@color/white"
        android:background="@color/purple_500" />

</RelativeLayout>
```

Рисунок 5

Ссылки на использованные цвета содержатся в файле ресурсов `colors.xml`:



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFB886FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="colorHome">#FF018786</color>
    <color name="item_color">#E2E5F6</color>
</resources>
```

Рисунок 6

Так как в activity в качестве ActionBar предполагается использовать Toolbar необходимо отключить первый (они не могут существовать вместе). Сделать это можно через настройку темы (theme) для конкретной activity. Перейдите в файл ресурсов res/themes/themes.xml (там хранятся стили). Добавьте новую тему (стиль), в которой не будет ActionBar:

```
<style name="Theme.ActionSimpleApp.NoActionBar">
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
    <item name="colorControlNormal">@color/white</item>
</style>
```

Рисунок 7

Необходимо созданную тему связать с activity. Сделать это можно через в файле манифеста AndroidManifest.xml:

```
<activity
    android:name=".SecondActivity"
    android:exported="false"
    android:theme="@style/Theme.ActionSimpleApp.NoActionBar" />
```

Рисунок 8

Далее необходимо создать макет меню, которое будет содержаться в Toolbar. Создайте каталог, в котором будут храниться все макеты меню (если в вашем проекте его нет).

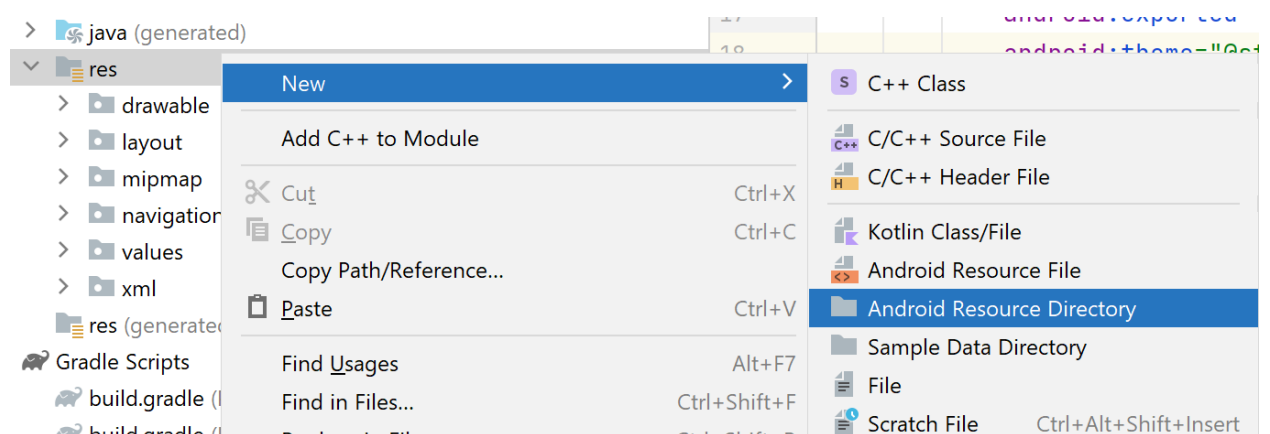


Рисунок 9

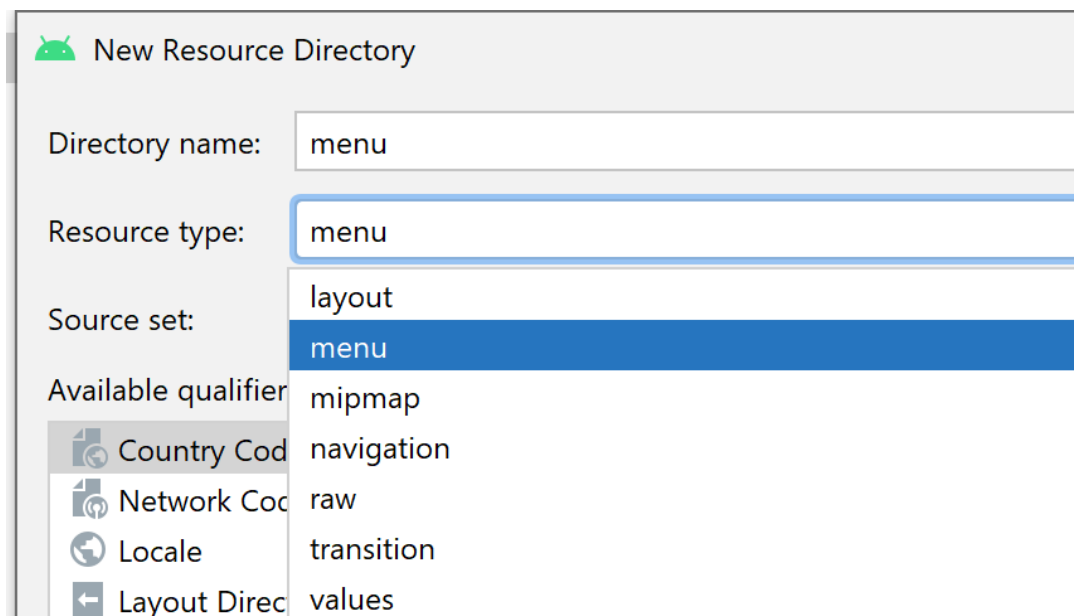


Рисунок 10

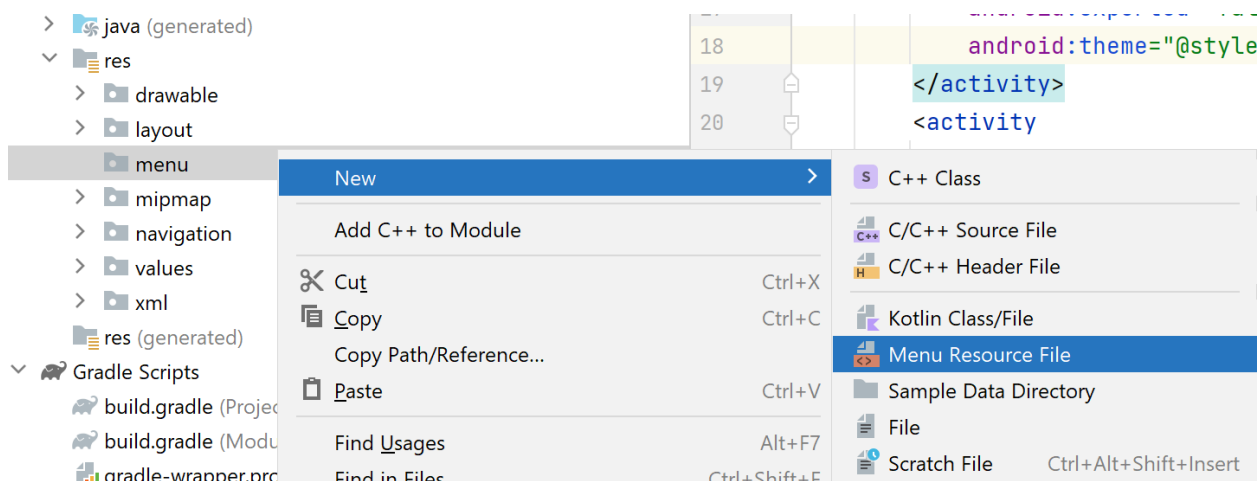


Рисунок 11

Назовите новый файл макета понятным для вас названием (в данном примере используется название `simple_menu`) и завершите создание.

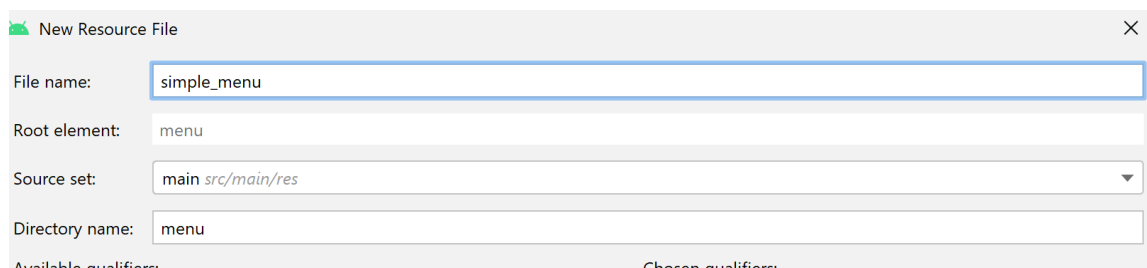
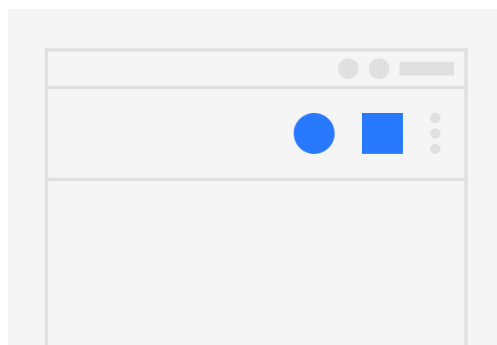


Рисунок 12

Для использования иконок в меню необходимо их поместить в каталоги `drawable`. Ресурс, где можно скачать популярные иконки <https://romannurik.github.io/AndroidAssetStudio/index.html>. На данном ресурсе можно подобрать и создать главную иконку для приложения, иконки для уведомлений, для `ActionBar` и пр.

Выберете иконки для ActionBar:



Action bar/tab icon generator

Generate icons for the app bar or tab bars.

Рисунок 13

В создаваемом меню будет пункт «избранное». Подберите иконку соответствующего дизайна и скачайте ее.

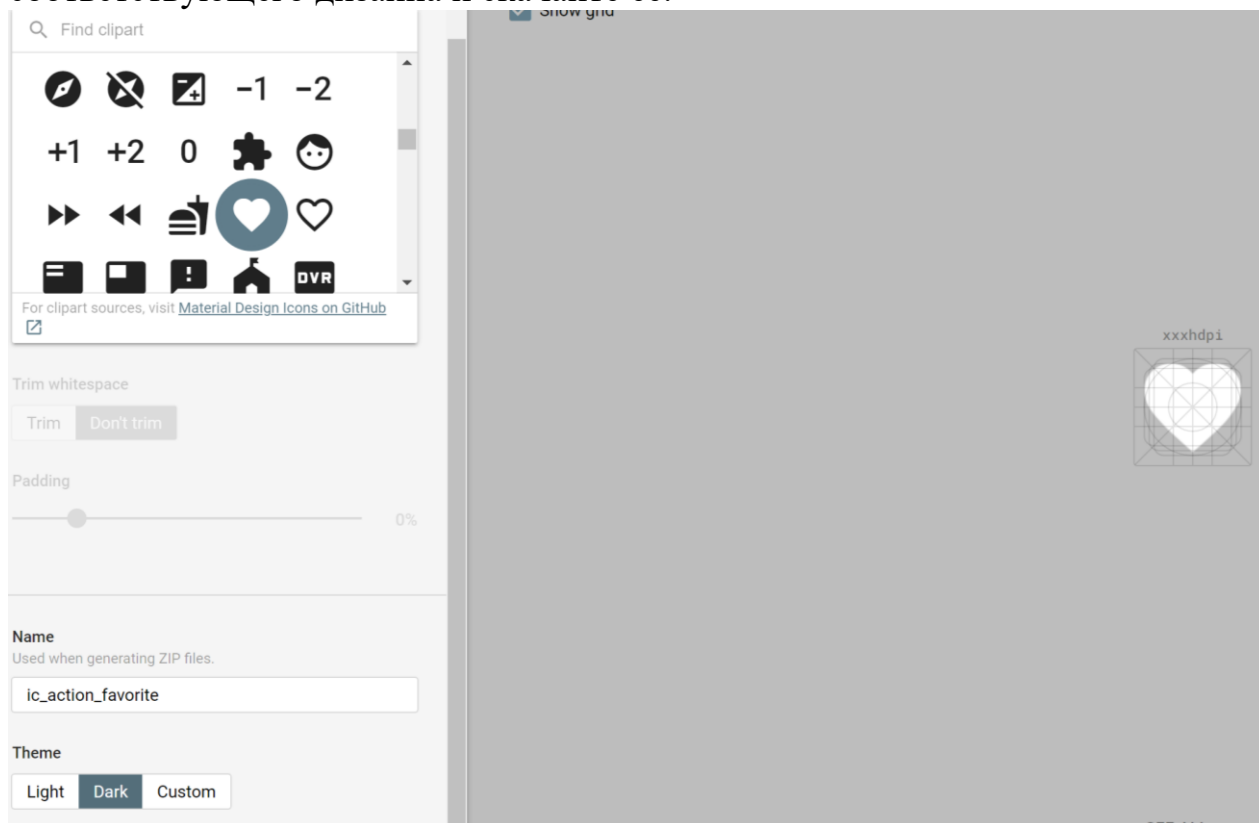


Рисунок 14

Будет загружен архив. В данном архиве содержится каталог res с подкаталогами drawable (иконки для разных размеров диагоналей устройств). Для различных разрешений экрана используются папки drawable-hdpi, drawable-mdpi, drawable-ldpi и пр.. Существует аналогичное деление для размеров экрана: drawable-normal, drawable-large и т.д.. Можно совмещать эти способы и создавать папки типа drawable-normal-hdpi. Для изображений, которые должны оставаться неизменными вне зависимости от разрешения

экрана, следует создать папку `drawable-nodpi`. Для памятки приведены некоторые используемые размеры изображений для значков и фона экрана:

- `res/drawable-ldpi` - (120 DPI) (QVGA):
`ic_launcher.png` (значок), 36×36 `background.png` (фон), 320×240 pixels
- `res/drawable-mdpi` - (160 DPI) (HVGA):
`background.png`, 320×480
- `res/drawable-hdpi` - (240 DPI) (WVGA):
`ic_launcher.png`, 72×72
`background.png`, 800×480
- `res/drawable-xhdpi` (320 DPI) (WSVGA or HDTV):
`ic_launcher.png`, 96×96
`background.png`, 1024×600 или 1280×720 pixels

Скопируйте каталог `res` из загруженного архива в каталог с проектом: `~PROJECT_NAME\app\src\main\`.

Теперь к иконке можно обращаться по ее имени (в данном случае `ic_action_favorite`).

Откройте созданный файл макета меню (в данном примере он называется `simple_menu.xml`) и создайте три пункта (у каждого должен быть `id` и режим отображения `showAsAction`):

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/
      xmlns:app="http://schemas.android.com/apk/res-auto">
  <item
      android:id="@+id/action_favorite"
      android:icon="@drawable/ic_action_favorite"
      android:title="Избранное"
      app:showAsAction="ifRoom"/>
  <item
      android:id="@+id/action_update"
      android:title="Обновить"
      app:showAsAction="never"/>
  <item
      android:id="@+id/action_exit"
      android:title="Выйти"
      app:showAsAction="never"/>
</menu>
```

Рисунок 15

В режиме дизайна:

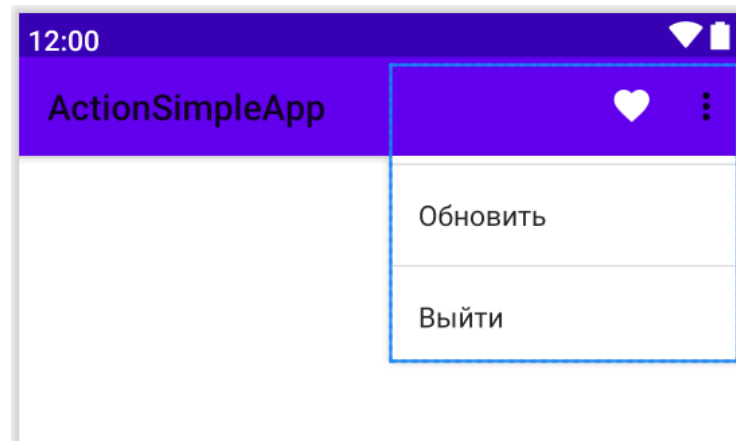


Рисунок 16

Атрибут **app:showAsAction** по сути определяет поведение меню в ActionBar. Значение **never** означает, что элемент меню не должен выводиться в заголовке, а только в всплывающем меню, т.е. находиться за тремя точками. Если вы установите значение **always**, то пункт Settings сразу появится в заголовке вашего приложения. Также доступны значения **ifRooms**, **withText** и **collapseActionView**. Попробуйте самостоятельно. Например, **ifRoom** выводит пункт меню, если позволяет место. Если пунктов будет много, то они будут только мешаться. Как правило, в таком варианте выводят очень короткое слово или значок для частых операций, чтобы избежать лишнего щелчка на три точки.

Вернитесь в класс второй созданной activity и в методе onCreate реализуйте код для работы с ActionBar/Toolbar:

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_second);  
    Toolbar tb = findViewById(R.id.toolbar);  
    setSupportActionBar(tb);  
    ActionBar ab = getSupportActionBar();  
    if (ab != null) {  
        ab.setTitle("Second");  
        ab.setHomeButtonEnabled(true); // включение поддержки кнопки назад в ActionBar  
        ab.setDisplayHomeAsUpEnabled(true); // отображение кнопки назад  
    }  
}
```

Рисунок 17

Необходимо связать созданный макет меню (XML-файл) с activity. Переопределите в классе activity метод **onCreateOptionsMenu**:


```

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.simple_menu, menu);
    return true;
}

```

Рисунок 18

Именно данный метод отвечает за появление меню у activity. В методе вызывается `MenuInflater.inflate` и передается идентификатор файла макета меню. Вызов заполняет экземпляр меню командами, определенными в файле. Метод должен возвращать значение **true**, чтобы меню было видимым на экране.

Вернитесь в класс главной activity (в данном примере это `MainActivity`) и реализуйте запуск второй activity по нажатию на кнопку с надписью NEXT. Слушатель в данном случае реализован через анонимный класс, а он в свою очередь использует «лямбду»:

```

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btnStart = findViewById(R.id.btnStartActivity);
        btnStart.setOnClickListener(view -> {
            startActivity(new Intent(getApplicationContext(), SecondActivity.class));
        });
    }
}

```

Рисунок 19

Запустите приложение, нажмите на кнопку и убедитесь, что меню второй activity корректно отображается:

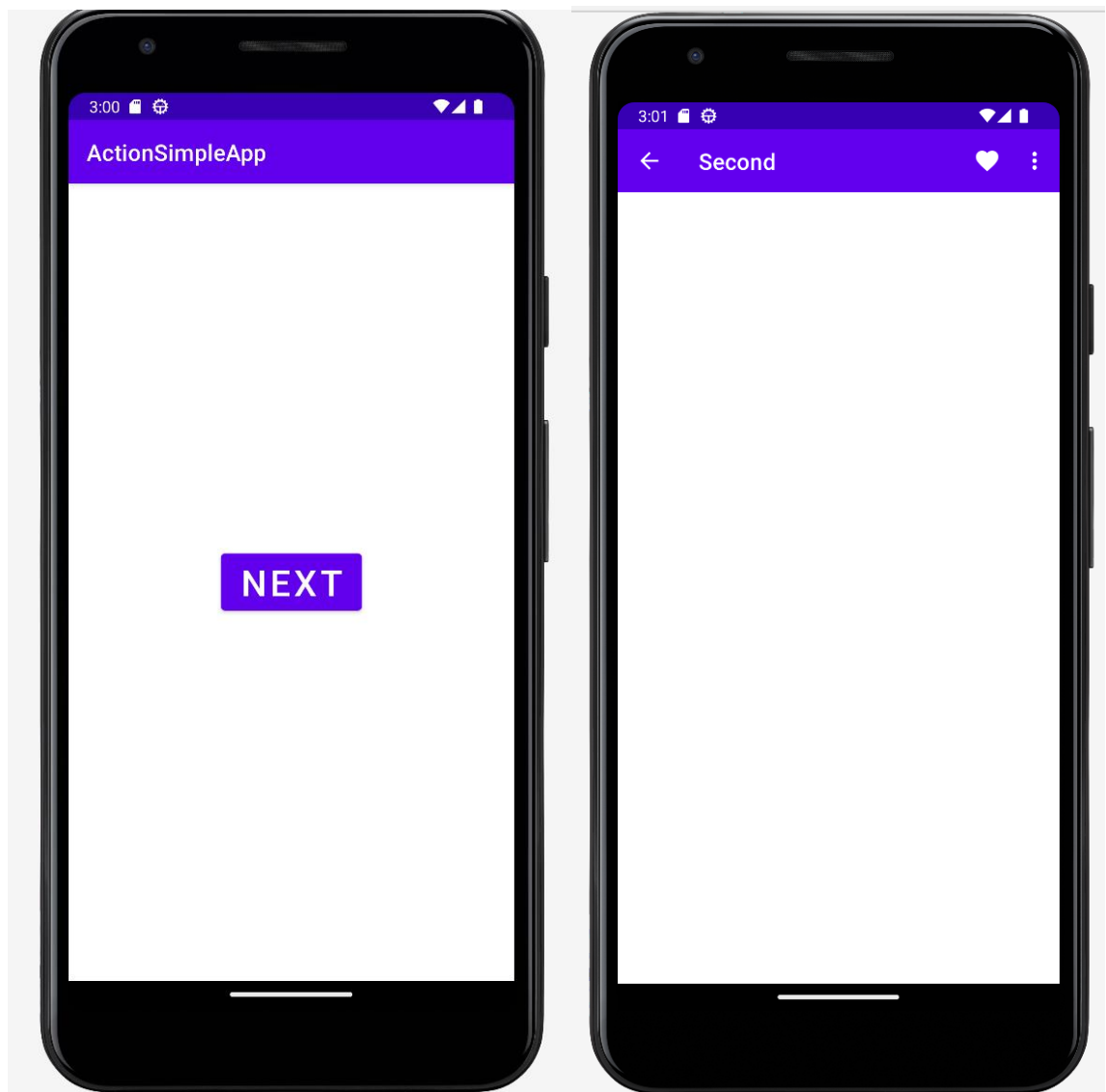


Рисунок 20

Но при нажатии на элементы меню ничего не происходит.

Для реализации обработки событий кликов на пункты меню необходимо реализовать метод **onOptionsItemSelected** (его также надо переопределить в классе activity, где располагается созданное меню). Обратите внимание, что разделение в операторе switch происходит по значению id элемента меню. `Android.R.id.home` – это стандартный идентификатор кнопки назад в ActionBar.

```

@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_favorite:
            Toast.makeText(context: this, text: "Click favorite icon on Toolbar",
                           Toast.LENGTH_LONG).show();
            return true;
        case R.id.action_update:
            Toast.makeText(context: this, text: "Click update on Toolbar",
                           Toast.LENGTH_LONG).show();
            return true;
        case R.id.action_exit:
            finish();
            return true;
        case android.R.id.home:
            onBackPressed();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

```

Рисунок 21

Запустите приложение и убедитесь, что обработка нажатий на пункты меню работает корректно (в том числе нажатие на стрелку «назад» в ActionBar).

Упражнение №2

Продолжайте работу в текущем приложении. В данном упражнении будет изучены особенности отображения списка в **RecyclerView** с помощью адаптера.

Откройте файл макета созданной второй activity (там, где было реализовано меню) и добавьте новый виджет – **RecyclerView**. Он будет занимать весь экран и располагаться под Toolbar. Отступы со всех сторон будут равны 4dp.

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/and ✓
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondActivity">

    <androidx.appcompat.widget.Toolbar
        android:id="@+id/toolbar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:titleTextColor="@color/white"
        android:background="@color/purple_500" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recyclerView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/toolbar"
        android:layout_marginStart="4dp"
        android:layout_marginTop="4dp"
        android:layout_marginEnd="4dp"
        android:layout_marginBottom="4dp" />
</RelativeLayout>

```

Рисунок 22

В данном упражнении будет отображен список людей (объекты класса Person). Необходимо создать класс Person и описать для него свойства, конструктор, сеттеры и геттеры. Создать класс можно через контекстное меню пакета (правой кнопкой мыши по имени пакета), New -> Java Class:

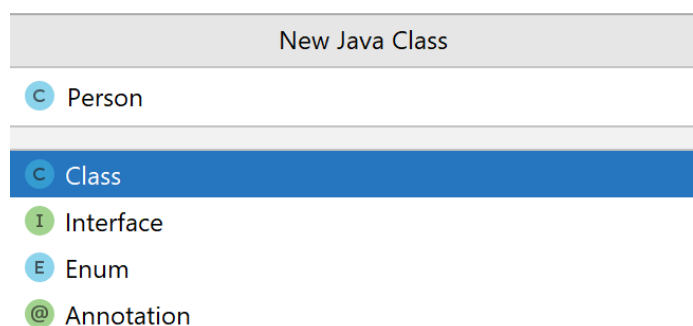


Рисунок 23

Укажите переменные для класса Person, а также **сгенерируйте** (не нужно писать вручную!) конструктор, сеттеры и геттеры (правой кнопкой мыши внутри класса, далее Generate):

```

public class Person {

    private String firstName;
    private String lastName;
    private int age;
    private String avatar;

    public Person(String firstName, String lastName, int age, String avatar) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.avatar = avatar;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public int getAge() {
        return age;
    }

    public String getAvatar() {
        return avatar;
    }

}

```

Рисунок 24

В классе activity, в которой будет отображаться список, подготовьте данные для отображения. В примере данные будут искусственно созданы программно (коллекция). Но обычно данные такого характера загружаются с сервера, базы данных или создаются в ходе работы приложения.

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);
    Toolbar tb = findViewById(R.id.toolbar);
    setSupportActionBar(tb);
    ActionBar ab = getSupportActionBar();
    if (ab != null) {
        ab.setTitle("Second");
        ab.setHomeButtonEnabled(true); // включение поддержки кнопки назад в ActionBar
        ab.setDisplayHomeAsUpEnabled(true); // отображение кнопки назад
    }
    // имитация получения списка
    ArrayList<Person> persons = new ArrayList<>();
    persons.add(new Person( firstName: "Ivan",   lastName: "Ivanov",  age: 19,  avatar: null));
    persons.add(new Person( firstName: "Petya",  lastName: "Petrov",  age: 18,  avatar: null));
    persons.add(new Person( firstName: "Masha",  lastName: "Mashina", age: 28,  avatar: null));
    persons.add(new Person( firstName: "John",   lastName: "Wick",    age: 45,  avatar: null));
}

```

Рисунок 25

Далее необходимо в res/layout создать макет для элемента списка:

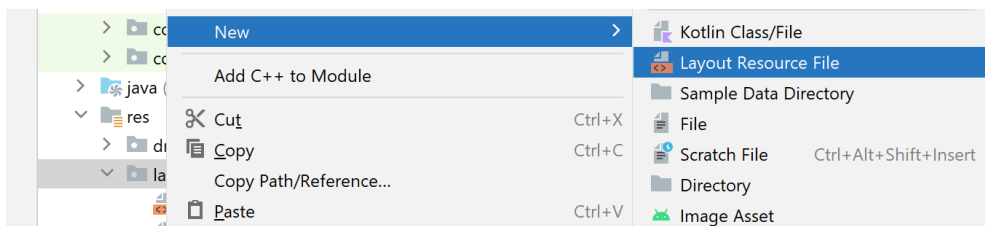


Рисунок 26

Имя для файла можно указать любое (в примере – **item_person**).

В данном случае для каждого элемента нужно отобразить: имя, фамилию, возраст и аватар. Добавьте соответствующие виджеты в созданный файл макета, чтобы все эти данные могли корректно отображаться:

```
<RelativeLayout
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="100dp"
        app:cardUseCompatPadding="true"
        card_view:cardCornerRadius="3dp"
        card_view:cardElevation="3dp">
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:background="@color/item_color"
            android:orientation="horizontal">
            <ImageView
                android:layout_width="100dp"
                android:layout_height="100dp"
                android:src="@drawable/ic_empty_ava"/>
            <RelativeLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:layout_margin="8dp">
                <TextView
                    android:id="@+id/tvPersonName"
                    android:textSize="24sp"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content" />
                <TextView
                    android:id="@+id/tvPersonAge"
                    android:layout_below="@id/tvPersonName"
                    android:textSize="24sp"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"/>
            </RelativeLayout>
        </LinearLayout>
    </androidx.cardview.widget.CardView>
</RelativeLayout>
```

Рисунок 27

Компонент **CardView** появился в Android Lollipop (API 21), но благодаря библиотеке совместимости доступен и для старых устройств. По сути является дальнейшим развитием **FrameLayout** и позволяет создавать красивую карточку с тенью (**elevation**) и закруглёнными углами (**cornerRadius**), который служит контейнером для других компонентов.

Картинку для пустой аватарки (в виджете **ImageView**) можно загрузить здесь <https://goo.su/M4cz>. Назовите файл латиницей любым удобным названием (в примере **ic_empty_ava.png**) и скопируйте его в **~PROJECT_NAME\app\src\main\res\drawable**.

Убедитесь, что в режиме дизайна макет отображается корректно:

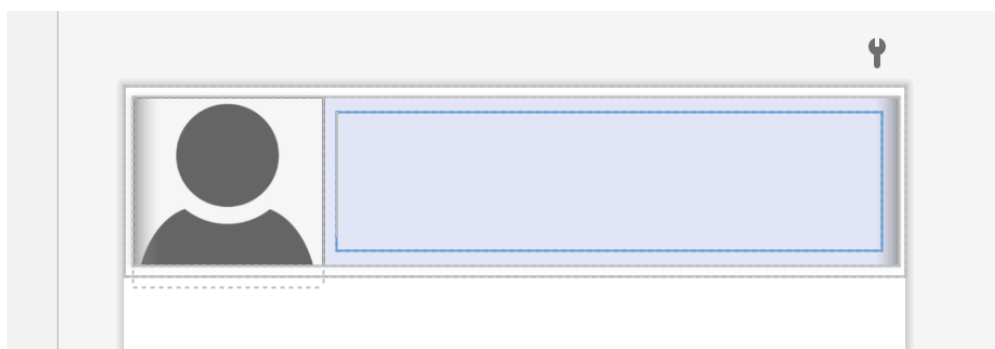


Рисунок 28

Для связки данных и визуальных компонентов необходимо создать специальный класс-наследник **RecyclerView.Adapter** (адаптер).

Создайте класс через контекстное меню, укажите для него имя (в данном примере **PersonAdapter**). В созданном классе добавьте конструктор, который будет принимать в качестве аргумента коллекцию данных для отображения:

```
public class PersonAdapter extends RecyclerView.Adapter<PersonAdapter.ViewHolder> {  
  
    private ArrayList<Person> persons;  
  
    public PersonAdapter(ArrayList<Person> persons) {  
        this.persons = persons;  
    }  
}
```

Рисунок 29

Среда разработки подчеркивает ошибку, так как класс **RecyclerView.Adapter** – абстрактный, поэтому необходимо переопределить все его абстрактные методы. Воспользуйтесь подсказкой IDE:

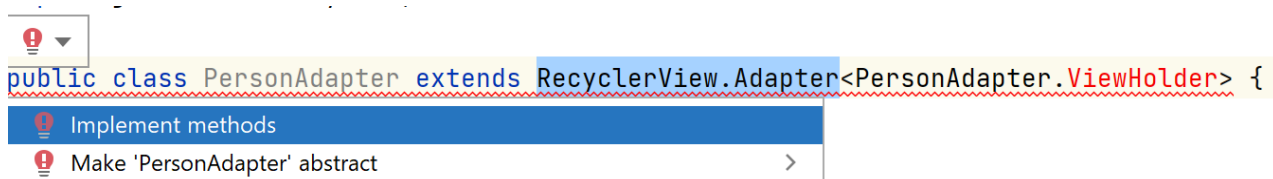


Рисунок 30

Были добавлены три метода:

onCreateViewHolder: возвращает объект ViewHolder, который служит для оптимизации ресурсов и является своеобразным контейнером для всех элементов, входящих в список.

onBindViewHolder: выполняет привязку объекта ViewHolder к объекту Person по определенной позиции.

getItemCount: возвращает количество объектов в списке.

Обязанности Adapter'a:

- создание ViewHolder'ов
- заполнение ViewHolder'ов информацией
- уведомление RecyclerView о том какие элементы изменились.
- обработка касаний
- частичное обновление данных
- информация о переиспользовании ViewHolder'a

Для создания ViewHolder нужно создать соответствующий класс. В данном классе будет реализовано получение ссылок на виджеты элементов списка.

Добавьте класс внутри класса PersonAdapter класс наследник RecyclerView.ViewHolder:

```
public class PersonAdapter extends RecyclerView.Adapter<PersonAdapter.ViewHolder> {  
  
    private ArrayList<Person> persons;  
  
    public PersonAdapter(ArrayList<Person> persons) {  
        this.persons = persons;  
    }  
  
    public static class ViewHolder extends RecyclerView.ViewHolder {  
  
        private final TextView nameView;  
        private final TextView ageView;  
  
        ViewHolder(View view){  
            super(view);  
            nameView = view.findViewById(R.id.tvPersonName);  
            ageView = view.findViewById(R.id.tvPersonAge);  
        }  
    }  
  
    @NonNull  
    @Override
```

Рисунок 31

В конструкторе созданного класса инициализируются ссылки на виджеты, которые участвуют в отображении данных элемента списка. После этого заполните код трех (ранее сгенерированных) методов:


```

@NonNull
@Override
public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.item_person, parent,
        attachToRoot: false);
    return new ViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull PersonAdapter.ViewHolder holder, int position) {
    Person person = persons.get(position);
    holder.nameView.setText(String.format("%s %s", person.getFirstName(), person.getLastName()));
    holder.ageView.setText(String.format("%d лет", person.getAge()));
}

@Override
public int getItemCount() {
    return persons.size();
}

```

Рисунок 32

Класс адаптера готов! Вернитесь в класс activity, где будет отображаться список. Добавьте код, который свяжет RecyclerView и список с данными через адаптер:

```

// имитация получения списка
ArrayList<Person> persons = new ArrayList<>();
persons.add(new Person( firstName: "Ivan",   lastName: "Ivanov",   age: 19,   avatar: null));
persons.add(new Person( firstName: "Petya",  lastName: "Petrov",  age: 18,   avatar: null));
persons.add(new Person( firstName: "Masha",  lastName: "Mashina", age: 28,   avatar: null));
persons.add(new Person( firstName: "John",   lastName: "Wick",    age: 45,   avatar: null));
RecyclerView rcView = findViewById(R.id.recyclerView);
PersonAdapter adapter = new PersonAdapter(persons);
rcView.setLayoutManager(new LinearLayoutManager( context: this,
    LinearLayoutManager.VERTICAL, reverseLayout: false));
rcView.setAdapter(adapter);

```

Рисунок 33

LayoutManager отвечает за то, как именно элементы будут располагаться на экране.

LayoutManager бывает:

- LinearLayoutManager (линейное размещение элементов)
- GridLayoutManager (табличное)
- StaggeredGridLayoutManager (сложное)

Стандартные LayoutManager



Linear



Grid



Grid + span



StaggeredGrid

Рисунок 34

В данном случае используется **LinearLayoutManager** (вертикальный). Запустите приложение, убедитесь, что список корректно отображается:

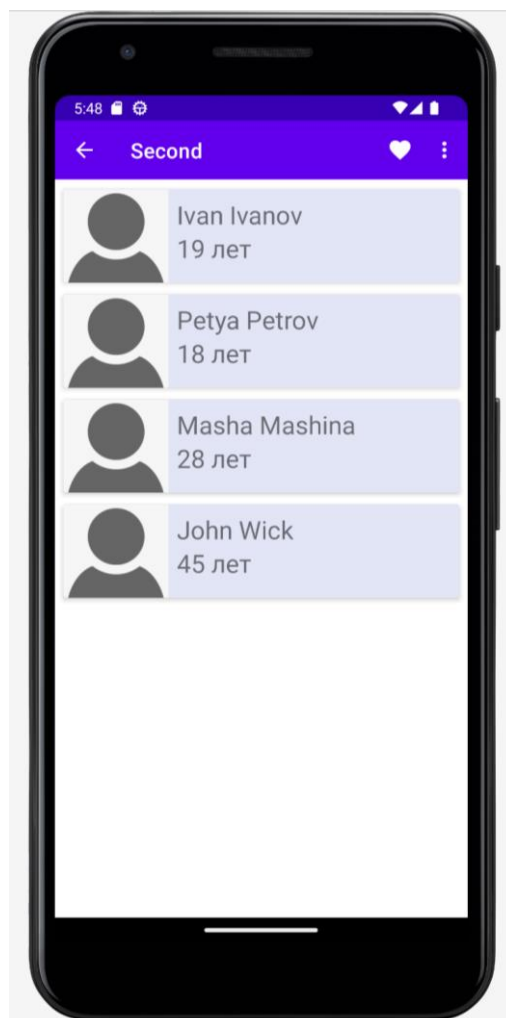


Рисунок 34

Попробуйте поменять тип расположения элементов списка на горизонтальное (`LinearLayoutManager.HORIZONTAL`). Посмотрите как изменилось отображение списка.

Дополнительное задание (для самостоятельного ознакомления и выполнения): реализовать обработку нажатия на элемент списка.

Измените код класса адаптера. Создайте внутри класса интерфейс, который будет отвечать за событие клика по элементу списка (имя может быть любым, но лучше использовать логичное название):

```
public class PersonAdapter extends RecyclerView.Adapter<PersonAdapter.ViewHolder> {  
    interface OnPersonClickListener{  
        void onPersonClick(Person person, int position);  
    }  
}
```

Рисунок 35

Следующий шаг – определение в классе адаптера переменной для хранения объекта этого интерфейса и получение для нее значения в конструкторе:

```
public class PersonAdapter extends RecyclerView.Adapter<PersonAdapter.ViewHolder> {  
    interface OnPersonClickListener{  
        void onPersonClick(Person person, int position);  
    }  
  
    private ArrayList<Person> persons;  
    private OnPersonClickListener onPersonClickListener;  
  
    public PersonAdapter(ArrayList<Person> persons) {  
        this.persons = persons;  
    }  
  
    public PersonAdapter(ArrayList<Person> persons, OnPersonClickListener onPersonClickListener) {  
        this.persons = persons;  
        this.onPersonClickListener = onPersonClickListener;  
    }  
}
```

Рисунок 36

Таким образом, вне кода адаптера мы можем установить любой объект слушателя и передать его в адаптер. И третий шаг – вызов метода слушателя при нажатии на элемент в списке в методе `onBindViewHolder`:

```

@Override
public void onBindViewHolder(@NonNull PersonAdapter.ViewHolder holder, int position) {
    Person person = persons.get(position);
    holder.nameView.setText(String.format("%s %s", person.getFirstName(), person.getLastName()));
    holder.ageView.setText(String.format("%d лет", person.getAge()));
    holder.itemView.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            onPersonClickListener.onPersonClick(person, holder.getAdapterPosition());
        }
    });
}
}

```

Рисунок 37

Класс ViewHolder имеет поле **itemView**, которое представляет интерфейс для одного объекта в списке и фактически объект View. А у этого объекта есть метод `setOnClickListener()`, через который можно подключить стандартный слушатель нажатия `OnClickListener` и в его методе `onClick()` вызвать метод созданного интерфейса, передав ему необходимые данные - выбранный объект `Person` и его позицию в списке.

Может возникнуть вопрос, а почему бы сразу тут и не обработать нажатие на элемент? К чему создавать дополнительный интерфейс, его переменную и вызывать его метод? Конечно, можно попытаться прямо тут обработать нажатия, но это не является хорошей или распространенной практикой, поскольку, чаще необходимо определить обработку нажатия в классе `activity`, потому что сами данные идут через нее.

Реализуйте созданный интерфейс в классе `activity`, где происходит создание адаптера и передача в него данных. Переопределите метод `onPersonClick` этого интерфейса.

```

public class SecondActivity extends AppCompatActivity implements PersonAdapter.OnPersonClickListener {

```

Рисунок 38

```

@Override
public void onPersonClick(Person person, int position) {
    Toast.makeText(context: this, text: "Click on " +
        person.getFirstName() +
        " " +
        person.getLastName(), Toast.LENGTH_LONG).show();
}
}

```

Рисунок 39

Используйте новый конструктор для создания `PersonAdapter`:

```

PersonAdapter adapter = new PersonAdapter(persons, onPersonClickListener: this);

```

Рисунок 40

Запустите приложение и убедитесь, что обработка кликов на пункты списка работает.

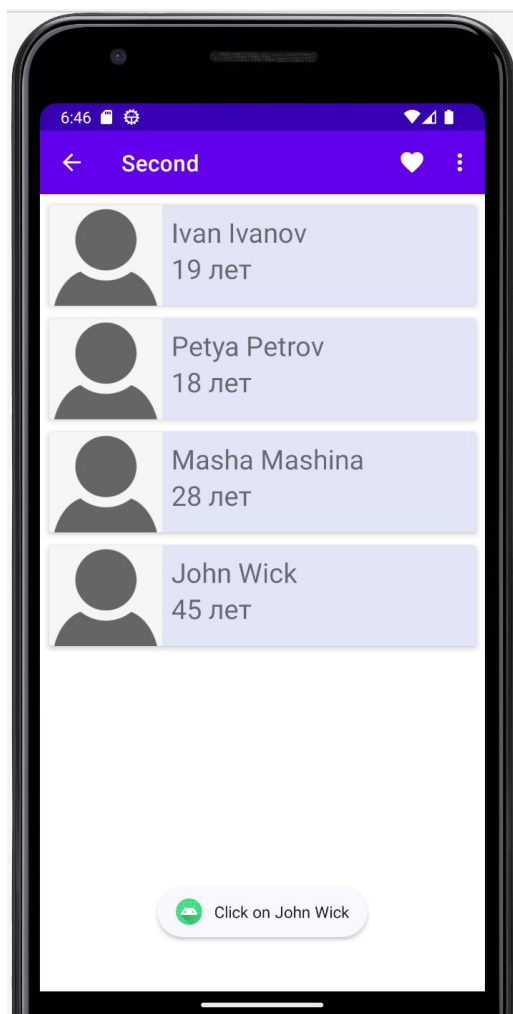


Рисунок 41

Упражнение №3

Продолжайте работу в текущем приложении. Создайте еще одну activity. Выбирайте шаблон Empty Activity (или Empty Views Activity в зависимости от версии IDE).

Назовите новую activity понятным для вас названием (в данном примере название будет BottomActivity) и завершите создание.

В данном упражнении необходимо создать навигацию в приложении с использованием **BottomNavigationView**.

Создайте новый файл макета для меню (в каталоге res/menu). Этот файл будет описывать содержимое **BottomNavigationView**. В данном примере название файла – bottom_nav_menu.xml

Содержимое созданного файла:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <menu xmlns:android="http://schemas.android.com/apk/res/android">
3     <item
4         android:id="@+id/navigation_home"
5         android:icon="@drawable/ic_home_black_24dp"
6         android:title="@string/title_home" />
7
8     <item
9         android:id="@+id/navigation_dashboard"
10        android:icon="@drawable/ic_dashboard_black_24dp"
11        android:title="@string/title_dashboard" />
12
13    <item
14        android:id="@+id/navigation_notifications"
15        android:icon="@drawable/ic_notifications_black_24dp"
16        android:title="@string/title_notifications" />
17 </menu>

```

Рисунок 42

Иконки для элементов меню (home, dashboard, notification) можно загрузить с ресурса <https://romannurik.github.io/AndroidAssetStudio/index.html>. Скопируйте загруженные иконки в каталоги drawable приложения. В атрибуте **android:icon** указывается ссылка на иконку по ее имени.

Строковые константы в файле strings.xml:

```

<string name="title_activity_bottom">BottomActivity</string>
<string name="title_home">Home</string>
<string name="title_dashboard">Dashboard</string>
<string name="title_notifications">Notifications</string>
<string name="title_activity_drawer">DrawerActivity</string>

```

Рисунок 43

Откройте файл макета последней созданной activity (BottomActivity) и добавьте в нем **BottomNavigationView**:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout
3     xmlns:android="http://schemas.android.com/apk/res/android"
4     xmlns:app="http://schemas.android.com/apk/res-auto"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:id="@+id/container">
8
9     <com.google.android.material.bottomnavigation.BottomNavigationView
10        android:id="@+id/bottomNavigationView"
11        android:layout_width="match_parent"
12        android:layout_height="wrap_content"
13        app:menu="@menu/bottom_nav_menu"
14        app:layout_constraintBottom_toBottomOf="parent"
15        app:itemIconTint="@color/black"
16        app:itemTextColor="@color/black"
17        app:labelVisibilityMode="labeled" />
18 </androidx.constraintlayout.widget.ConstraintLayout>

```

Рисунок 44

В атрибуте `app:menu` указывается ссылка на файл макета меню, отображаемого в **BottomNavigationView**.

Переключитесь в режим дизайна и убедитесь, что меню корректно отображается:

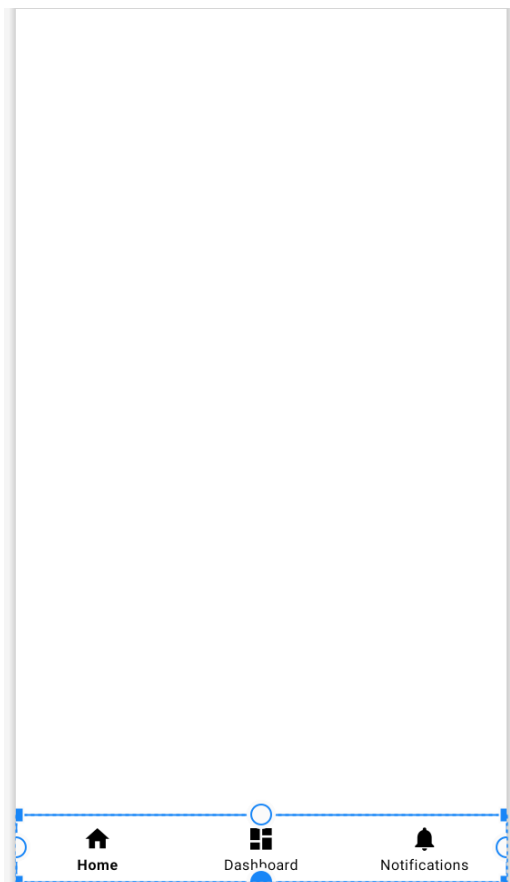


Рисунок 45

По нажатию на пункты меню в контейнере будут меняться фрагменты. **Создайте три фрагмента (см. упражнение №3 предыдущего практического занятия) и макеты (layout-XML) для них.** В данном примере были созданы `HomeFragment`, `DashboardrFragment`, `NotificationFragment`.

Перейдите в класс `BottomActivity` и создайте метод для оптимизации замены фрагментов:

```
private void replaceFragment(Fragment fragment) {  
    FragmentTransaction transaction = getSupportFragmentManager().beginTransaction();  
    transaction.replace(R.id.container, fragment);  
    transaction.addToBackStack(null);  
    transaction.commit();  
}
```

Рисунок 46

Код метода `onCreate` включает в себя регистрацию и реализацию слушателя обработки кликов на элементы меню. По нажатию на элемент меню в контейнер помещается соответствующий фрагмент:

```

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_bottom);
    getSupportActionBar().setTitle("Example bottomNavigationView");
    replaceFragment(new HomeFragment());
    BottomNavigationView bottomNavigationView = findViewById(R.id.bottomNavigationView);
    bottomNavigationView.setOnItemSelectedListener(new NavigationBarView.OnItemSelectedListener() {
        @Override
        public boolean onNavigationItemSelected(@NonNull MenuItem item) {
            switch (item.getItemId()) {
                case R.id.navigation_home:
                    replaceFragment(new HomeFragment());
                    return true;
                case R.id.navigation_dashboard:
                    replaceFragment(new DashboardFragment());
                    return true;
                case R.id.navigation_notifications:
                    replaceFragment(new NotificationFragment());
                    return true;
            }
            return false;
        }
    });
}

```

Рисунок 47

Код, выделенный серым цветом, можно заменить на лямбду:

```

bottomNavigationView.setOnItemSelectedListener(item -> {
    switch (item.getItemId()) {
        case R.id.navigation_home:
            replaceFragment(new HomeFragment());
            return true;
        case R.id.navigation_dashboard:
            replaceFragment(new DashboardFragment());
            return true;
        case R.id.navigation_notifications:
            replaceFragment(new NotificationFragment());
            return true;
    }
    return false;
});

```

Рисунок 48

Перейдите в класс MainActivity и измените код так, чтобы по нажатию на кнопку NEXT запускалась BottomActivity.

Запустите приложение, убедитесь, что фрагменты в BottomActivity меняются при выборе соответствующего элемента BottomNavigationView.