Модуль 3. Основы программирования Android приложений

Тема 3.3. Внутренние и анонимные классы

2 часа

Оглавление

Тема 3.3. Внутренние и анонимные классы

3.3.1. Понятие внутреннего класса

Зачастую при разработке программы нам необходимо создать какой-то небольшой вспомогательный класс, который нецелесообразно выделять в какую-то отдельную сущность, создавать для него отдельный файл и т. д. Например класс Назначение внутри класса Посылка. В Java есть соответствующая синтаксическая возможность. Можно объявлять классы внутри другого - вложенные классы (nested classes). Например так:

```
public class Parcel { // посылка
	...
	class Destination { // место назначения
			public String street; // улица
			public int homeNumber; // номер дома
			public int roomNumber; // номер квартиры
	}
	...
}
```

В примере выше мы создали класс Destination в классе Parcel. Вложенные классы могут оказаться полезными в следующих ситуациях:

- когда вложенный класс слишком незначителен, чтобы выделять его как отдельный полноценный класс - улучшение читаемости кода;
- когда вложенный класс используется только в одном другом классе и больше нигде в программе - группировка связанных классов;
- когда вложенному классу необходим доступ к полям и методам другого класса - инкапсуляция.

Как вы можете видеть, синтаксис объявления совершенно идентичен синтаксису объявления обычного класса, только он пишется внутри класса - внешнего класса.

Вложенные классы могут быть объявлены с ключевым словом static, в таком случае это статические вложенные классы, а могут быть объявлены без ключевого слова static, такие вложенные классы называется внутренними. Таким образом в примере выше мы имеем частный случай вложенного класса - внутренний класс. Именно о таких классах речь пойдет далее.

Важной особенностью именно внутренних классов (и то, что отличает их от статических), что они имеют доступ ко всем полям и методам внешнего класса. Внутренние классы в Java делятся на такие 3 типа:

1. Внутренние классы-члены (member inner classes) - в примере выше как раз такой класс.
2. Локальные классы (local inner classes).
3. Анонимные классы (anonymous inner classes).

Обсудим все 3 типа по порядку, уделив особое внимание анонимным классам, как самым часто используемым на практике в Android-программировании.

3.3.2. Внутренние классы-члены

Внутренние классы-члены ассоциируются не самим внешним классом, а с конкретными его экземплярами. Это проявляется, в частности в том, как будет выглядеть вызов конструктора такого класса:

```
Parcel parcel = new Parcel(); // создаем посылку - внешний класс
Parcel.Destination destination = parcel.new Destination(); // для данного
экземпляра класса Посылка
                                                    // создаем экземпляр класса
Назначение
```

На внутренние классы-члены в Java действуют некоторые ограничения:

- они не могут содержать статических объявлений, за исключением static final полей - констант;
- внутри таких классов нельзя объявлять перечисления.

3.3.3. Локальные внутренние классы

Этот тип внутреннего класса отличается тем, что он объявляется внутри методов класса. Пример:

```
public class Handler {
      public void handle(String requestPath) { // метод обработки некоего запроса
            class Path { // класс для работы с частями URL
                  List<String> parts = new ArrayList<String>(); // внутреннее
представление
                  Path(String path) { … } // конструктор из строки
                  int size() { ... } // количество компонентов URL
                  String get(int i) { … } // нужный компонент
                  boolean startsWith(String s) { ... } // начинается ли ссылка с
данной строки
            }

            Path path = new Path(requestPath); // создаем наш вспомогательный
класс
            if (path.startsWith("/page")) {
                  pageId = path.get(1);
            }
            if (path.startsWith("/post")) {
                  categoryId = path.get(1);
                  postId = path.get(2);
            }
                        ...
      }
```

```
}
```

Здесь в методе handle() объявляется вспомогательный класс Path, который помогает разобрать гиперссылку на компоненты. Это достаточно типичное применение внутреннего локального класса. Локальные классы имеют доступ ко всем полям и методам обрамляющего класса, но, кроме того, они еще имеют доступ к переменным метода, в котором объявлены, если те являются final.

У локальных классов свои ограничения:

- они видны только в пределах блока, в котором объявлены (внутри фигурных скобок);
- они не могут содержать в себе статических объявлений за исключением констант - static final.

3.3.4. Анонимные классы

Самый часто используемый на практике вид внутреннего класса. Вот пример использования при программировании для Android:

```
public class EditFragment extends Fragment {
	private boolean edited = false; // редактировали ли мы информацию на форме

	@Override
	public void onStart() { // при показе формы на экране
		super.onStart();
		// создаем анонимный класс - наследник класса TextWatcher
		TextWatcher textWatcher = new TextWatcher() {
			@Override // переопределяем один из методов TextWatcher,
				// предназначенный для реакции на изменение текста в
поле ввода
			public void afterTextChanged(Editable s) {
				edited = true; // фиксируем, что информация была
отредактирована
			}
		};
		// устанавливаем полю ввода обработчиком событий наш анонимный класс
		EditText edit = (EditText) getView().findViewById(R.id.nameEditText);
		edit.addTextChangedListener(textWatcher);
	}

	@Override
	public void onPause() {  // при скрытии формы с экрана
		super.onPause();
		if (edited) // если на форме что-то редактировалось, сохраняем
информацию в БД
			updateNoteByEdits();
	}
}
```

Как видите собственно создание анонимного класса представляет из себя вызов конструктора родительского класса с последующим написанием за ним в фигурных скобках тела анонимного класса - наследника. Отсюда следует важное ограничение анонимных классов - они не могут

содержать конструкторы. Аргументы, указанные в скобках, автоматически используются для вызова конструктора базового класса с указанными параметрами. В остальном анонимные классы имеют те же ограничения, что и локальные внутренние классы.

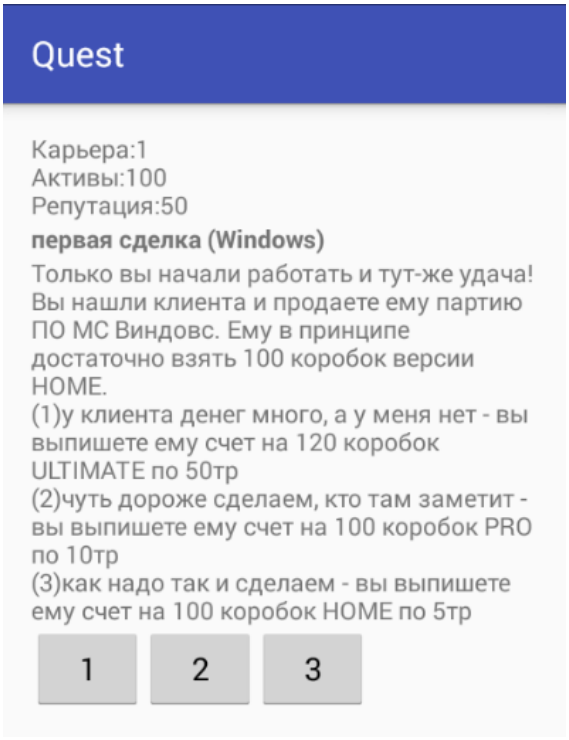
Минипроект 3.1

Теперь перепишем мини-проект нашего карьерного квеста под Android с использованием новоприобретенных знаний об анонимных классах. Для начала создадим разметку:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity">

   <LinearLayout
       android:orientation="vertical"
       android:layout_width="match_parent"
       android:layout_height="match_parent">

       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Статус"
           android:id="@+id/status"
       />

       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Заголовок ситуации"
           android:id="@+id/title"
           android:textStyle="bold"
       />

       <TextView
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Описание ситуации"
           android:id="@+id/desc"
       />

       <LinearLayout
           android:orientation="horizontal"
           android:layout_width="match_parent"
           android:layout_height="match_parent"
           android:id="@+id/layout"></LinearLayout>
   </LinearLayout>

</RelativeLayout>
```

Как видите, здесь просто 3 текстовых поля для отображения соответствующих компонентов ситуации (текущий статус персонажа, название ситуации и ее детальное описание). LinearLayout внизу нужен для того, чтобы в ходе выполнения программы на нем размещать кнопки, служащие для выбора желаемого варианта развития событий.

Вот как эта разметка будет выглядеть уже непосредственно в ходе игры:



Ну что же. Пришло время написать код приложения, которое нарисует нам эту картинку. Обратите внимание, что здесь используются те же классы, что в текстовой версии игры, в них ничего переделывать не надо. Вот мы и пожинаем плоды объектного подхода к программированию. Ядро игры получилось независимым от платформы. Переписав лишь десяток-другой строчек кода мы получаем версию игры для платформы (Android). В первоначальном подходе, без использования ООП, код игры пришлось бы переписывать полностью, потому-что логика игрового процесса была бы перемешана с кодом, специфичным именно для данной платформы.

```
public class MainActivity extends AppCompatActivity {
    Character manager; // наш персонаж
    Story story; // наша история

    // в этом методе мы размещаем всю информацию, специфичную для текущей
    // ситуации на форме приложения, а также размещаем кнопки, которые
    // позволят пользователю выбрать дальнейший ход событий
    private void updateStatus(){
        // не забываем обновить репутацию в соответствии с новым
        // состоянием дел
        manager.A+=story.current_situation.dA;
        manager.K+=story.current_situation.dK;
        manager.R+=story.current_situation.dR;
        // выводим статус на форму
        ((TextView) findViewById(R.id.status)).
            setText("Карьера:"+manager.K+
```

```
            "\nАктивы:"+manager.A+"\nРепутация:"+manager.R);
        // аналогично для заголовка и описания ситуации
        ((TextView) findViewById(R.id.title)).
            setText(story.current_situation.subject);
        ((TextView) findViewById(R.id.desc)).
            setText(story.current_situation.text);
        ((LinearLayout) findViewById(R.id.layout)).removeAllViews();
        // размещаем кнопку для каждого варианта, который пользователь
        // может выбрать
        for (int i = 0; i < story.current_situation.direction.length; i++){
            Button b = new Button(this);
            b.setText(Integer.toString(i+1));
            final int buttonId = i; // напоминаю, что в анонимных классах
                    // можно использовать только те переменные метода,
                    // которые объявлены как final
            // создаем новый анонимный класс и устанавливаем его
            // обработчиком нажатия на кнопку
            b.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    go(buttonId); // поскольку анонимный класс имеет полный
                            // доступ к методам и переменным родительского,
                            // то мы просто вызываем нужный нам метод.
                }
            });
            // добавляем готовую кнопку на разметку
            ((LinearLayout) findViewById(R.id.layout)).addView(b);
        }
    }

    // метод для перехода на нужную ветку развития
    private void go(int i){
        story.go(i+1);
        updateStatus();
        // если история закончилась, выводим на экран поздравление
        if(story.isEnd())
            Toast.makeText(this, "Игра закончена!", Toast.LENGTH_LONG).show();
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // создаем нового персонажа и историю
        manager = new Character("Вася");
        story = new Story();
        // в первый раз выводим на форму весь необходимый текст и элементы
        // управления
        updateStatus();
    }
}
```

Как видим, здесь анонимный класс используется для обработки события нажатия на кнопку, а информация в него передается при помощи константы, созданной в методе перед созданием анонимного класса. Это достаточно типичное для Android-программирования применение анонимных классов.

Список источников

1. http://www.quizful.net/post/inner-classes-java
2. http://easy-code.ru/lesson/java-nested-classes

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю IT ШКОЛЫ SAMSUNG Варюхину Сергею Валерьевичу.