Навчальна дисципліна

Розробка мобільних застосувань

Лектор - к.т.н., доцент

Баклан Ігор Всеволодович

Site: baklaniv.at.ua

E-mail: iaa@ukr.net

2016-2017

Лекція №2. Розробка інтерактивних застосувань

мы построим приложение для выбора пива

В лекції

В главе 1 вы узнали, как создать простейшее приложение при помощи мастера New Project в Android Studio и как изменить текст, отображаемый в макете. Но когда вы создаете Android-приложение обычно это приложение должно что-то *делать*.

В этой главе мы покажем, как создать приложение, взаимодействующее с пользователем. В приложении Beer Adviser пользователь выбирает вид пива, который он предпочитает, щелкает на кнопке и получает список рекомендуемых сортов.

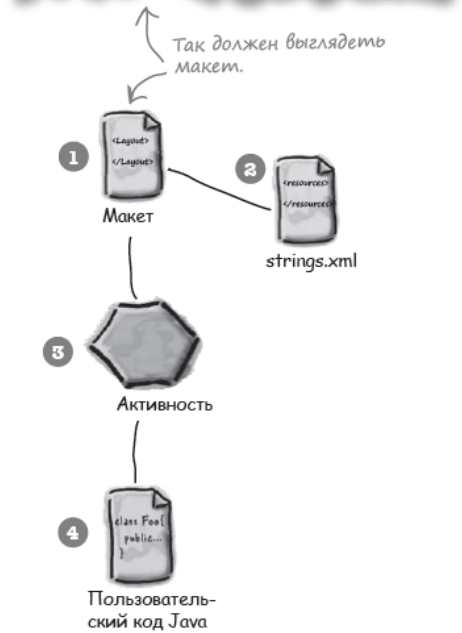
Приложение имеет следующую структуру:

1. **Макет определяет, как будет выглядеть приложение.**
   Он состоит из трех компонентов графического интерфейса:
   - раскрывающегося списка значений, в котором пользователь выбирает нужный вид пива;
   - кнопки, которая при нажатии возвращает подборку сортов пива;
   - надписи для вывода сортов пива.

2. **Файл strings.xml включает все строковые ресурсы, необходимые макету, — например, текст надписи на кнопке, входящей в макет.**

3. **Активность определяет, как приложение должно взаимодействовать с пользователем.**
   Она получает вид пива, выбранный пользователем, и использует его для вывода списка сортов, которые могут представлять интерес для пользователя. Для решения этой задачи используется вспомогательный класс Java.

4. **Класс Java, содержащий логику приложения.**
   Класс включает метод, который получает вид пива в параметре и возвращает список сортов пива указанного типа. Активность вызывает метод, передает ему вид пива и использует полученный ответ.



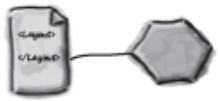
Что нужно сделать

Итак, приступим к построению приложения Beer Adviser. Работа состоит из нескольких шагов (все они будут подробно рассмотрены в этой главе):

1. **Создание проекта.**
   Мы создаем совершенно новое приложение, для которого нужно будет создать новый проект. Как и в предыдущей главе, в этот проект достаточно включить базовый макет и активность.

2. **Обновление макета.**
   Когда основная структура приложения будет готова, следует отредактировать макет и включить в него все компоненты графического интерфейса, необходимые для работы приложения.

3. **Связывание макета с активностью.**
   Макет создает только визуальное оформление. Чтобы приложение могло выполнять разумные действия, необходимо связать макет с кодом Java в активности.

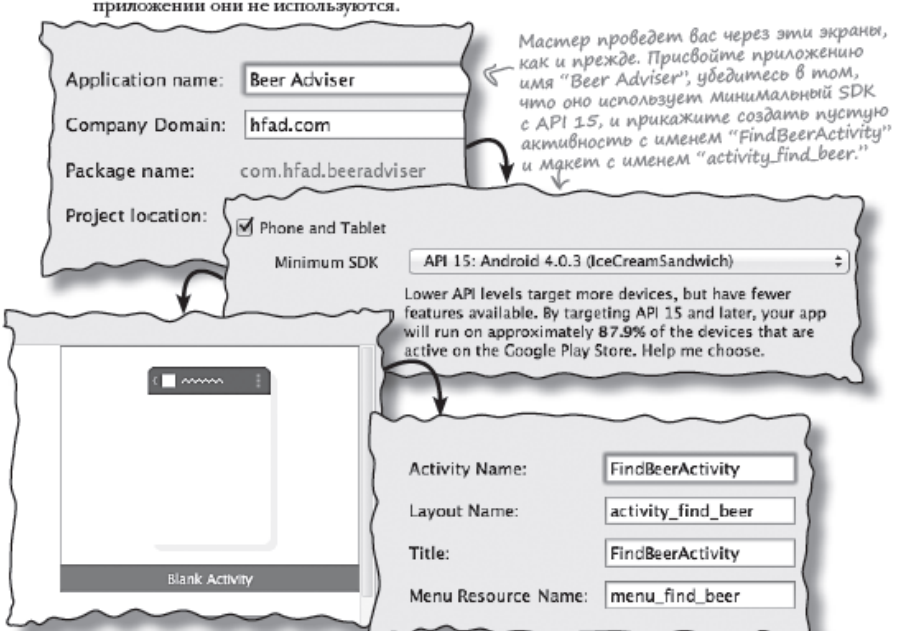


4. **Программирование логики приложения.**
   Мы добавим в приложение класс Java, который будет возвращать пользователю правильные сорта пива в зависимости от их выбора.

Создание проекта

Работа начинается с создания нового приложения (это делается почти так же, как в предыдущей главе):

- Откройте Android Studio и выберите на заставке строку "Start a new Android Studio project". Запускается мастер, уже знакомый вам по главе 1.
- По запросу мастера введите имя приложения "Beer Adviser"; убедитесь в том, что в окне сгенерировано имя пакета `com.hfad.beeradviser`.
- Чтобы приложение работало на большинстве телефонов и планшетов, выберите минимальную версию SDK с API 15 и проследите за тем, чтобы флажок "Phone and Tablet" был установлен. Это означает, что на любом телефоне или планшете, на котором выполняется приложение, должна быть установлена как минимум версия API 15. Большинство устройств на базе Android соответствует этому критерию.
- Выберите пустую активность в качестве активности по умолчанию. Присвойте ей имя "FindBeerActivity", а макету – имя "activity_find_beer". Подтвердите значения по умолчанию для текста заголовка (Title) и имени ресурса меню (Menu Resource Name), так как в этом приложении они не используются.



Мы создали активность и макет по умолчанию

Когда вы щелкнете на кнопке Finish, среда Android Studio создаст новый проект, содержащий активность *FindBeerActivity.java* и макет *activity_find_beer.xml*. Начнем с редактирования файла макета. Для этого перейдите в папку *app/src/main/res/layout* и откройте файл *activity_find_beer.xml*.

Как и в предыдущем приложении, мастер создал макет по умолчанию с элементом <TextView>, содержащим текст "Hello world!":

Разметка XML макета

```
<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="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".FindBeerActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

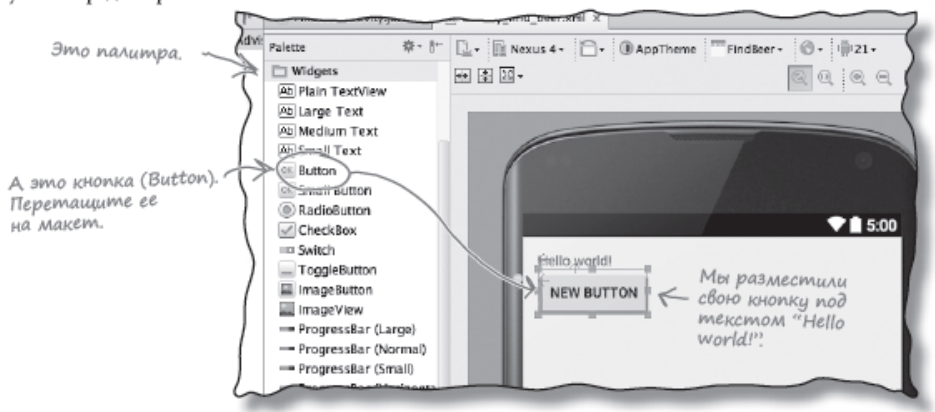
Эти элементы относятся к макету в целом. Они определяют его ширину и высоту, а также величину отступов от краев макета.

Элемент <TextView> из разметки XML отображается в визуальном редакторе.

Визуальный редактор

Добавление компонентов в визуальном редакторе

Добавить компоненты графического интерфейса в макет можно двумя способами: в разметке XML или в визуальном редакторе. Начнем с добавления кнопки в визуальном редакторе. Слева от визуального редактора располагается палитра с компонентами графического интерфейса, которые можно перетаскивать мышью на макет. Просмотрите раздел Widgets и найдите в нем компонент кнопки (Button). Щелкните на нем и перетащите на макет в визуальном редакторе.



Изменения, внесенные в визуальном редакторе, отражаются в XML

Такое перетаскивание компонентов графического интерфейса является удобным способом обновления макета. Переключившись на редактор кода, вы увидите, что в результате добавления кнопки в визуальном редакторе в файле появилось несколько строк кода:

```
...
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignLeft="@+id/textView" />
```

Код, добавленный визуальным редактором, зависит от того, где вы разместили кнопку. Возможно, ваша разметка макета будет отличаться от нашей; не беспокойтесь — скоро мы ее изменим.

Элементу TextView, который вы видели ранее, присвоен идентификатор.

Новый элемент Button описывает кнопку, которую вы перетащили на макет. Он будет более подробно рассмотрен ниже.

В activity_find_beer.xml появилась новая кнопка

Редактор добавил новый элемент <Button> в файл *activity_find_beer.xml*:

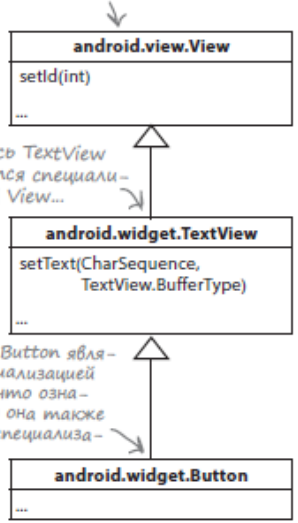
```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="New Button"
    android:id="@+id/button"
    android:layout_below="@+id/textView"
    android:layout_alignLeft="@+id/textView" />
```

В мире Android кнопка нажимается пользователем, чтобы инициировать какое-либо действие. Кнопка обладает свойствами, управляющими ее позицией, размером, внешним видом и методами активности, которые она должна вызывать. Эти свойства существуют не только у кнопок — они есть и у других компонентов графического интерфейса, включая надписи.

Кнопки и надписи — субклассы одного класса Android View

Тот факт, что кнопки и надписи имеют так много общих свойств, вполне логичен — оба компонента наследуют от одного класса Android **View**. Более подробные описания свойств будут приведены ниже, а пока рассмотрим несколько типичных примеров.

Класс View содержит множество разных методов. Они будут рассмотрены позднее в книге.



android:id

Имя, по которому идентифицируется компонент. Свойство `ID` используется для управления работой компонента из кода активности, а также для управления размещением компонентов в макете:

```
android:id="@+id/button"
```

android:text

Свойство сообщает Android, какой текст должен выводиться в компоненте. В случае <Button> это текст, выводимый на кнопке:

```
android:text="New Button"
```

android:layout_width, android:layout_height

Эти свойства задают базовую ширину и высоту компонента. Значение `"wrap_content"` означает, что размеры компонента должны подбираться по размерам содержимого:

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

- [x] Создание проекта
- [ ] **Обновление макета**
- [ ] Подключение активности
- [ ] Программирование логики

Подробнее о коде макета

Давайте внимательнее рассмотрим код макета и разобьем его так, чтобы происходящее стало более понятным (не беспокойтесь, если ваш код выглядит немного иначе — просто следите за логикой):

Элемент RelativeLayout.

```
<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="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".FindBeerActivity">
```

Надпись

```
    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView" />
```

Кнопка

```
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_below="@+id/textView"
        android:layout_alignLeft="@+id/textView" />
```

```
</RelativeLayout>
```

Закрывает элемент RelativeLayout.

Элемент RelativeLayout

Код макета начинается с элемента- `<RelativeLayout>`. Элемент `<RelativeLayout>` сообщает Android, что компоненты графического интерфейса в макете должны отображаться *относительно* друг друга. Например, вы можете приказать, чтобы один компонент отображался слева от другого, или они должны быть выровнены по некоторой общей линии.

В нашем примере кнопка располагается прямо под надписью, поэтому кнопка отображается относительно надписи.

Также существуют и другие способы размещения компонентов графического интерфейса. Вскоре вы о них узнаете.

Элемент TextView

Внутри элемента <RelativeLayout> первым идет элемент надписи <TextView>:

```
...
    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textView" />
...
```

Никакие свойства, указывающие местонахождение надписи в макете, не были заданы, поэтому по умолчанию Android выводит ее в левом верхнем углу экрана. Обратите внимание: надписи присваивается идентификатор `textView`. Вы поймете, зачем он нужен, когда мы перейдем к следующему элементу.

В режиме относительного размещения <RelativeLayout> компоненты графического интерфейса размещаются относительно друг друга.

Элемент Button

Содержимое элемента <RelativeLayout> завершает элемент кнопки <Button>:

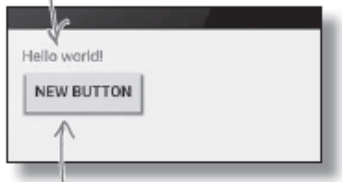
```
...
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Button"
        android:id="@+id/button"
        android:layout_below="@+id/textView"
        android:layout_alignLeft="@+id/textView" />
...
```

При добавлении кнопки в макет мы разместили ее так, что она находится под надписью, а левый край кнопки выравнивается по левому краю надписи. Кнопка размещается *относительно* надписи, и этот факт отражен в разметке XML:

```
android:layout_below="@+id/textView"
android:layout_alignLeft="@+id/textView"
```

Также возможны другие варианты разметки XML, достигающей того же визуального эффекта. Например, приведенный фрагмент XML указывает, что *кнопка размещается под надписью*. Также мы могли воспользоваться эквивалентной командой, указывающей, что *надпись размещается над кнопкой*.

По умолчанию надпись размещается в левом верхнем углу.



Свойства кнопки заданы так, что она отображается под надписью, а ее левый край выравнивается по вертикали с левым краем надписи.

Изменения в XML...

Вы уже видели, как изменения, вносимые в визуальном редакторе, отражаются в разметке XML макета. Также справедливо и обратное: все изменения, вносимые в разметке XML макета, отражаются в визуальном редакторе.

Попробуйте заменить код из файла *activity_find_beer.xml* следующим фрагментом:

```
<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:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".FindBeerActivity" >

    <Spinner
        android:id="@+id/color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="37dp" />

    <Button
        android:id="@+id/find_beer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/color"
        android:layout_below="@+id/color"
        android:text="Button" />

    <TextView
        android:id="@+id/brands"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/find_beer"
        android:layout_below="@+id/find_beer"
        android:layout_marginTop="18dp"
        android:text="TextView" />

</RelativeLayout>
```

Раскрывающийся список значений в системе Android. Компонент предназначен для выбора одного значения из представленного набора.

Элемент <Spinner> создает раскрывающийся список значений.

Кнопка размещается под раскрывающимся списком и выравнивается с ним по левому краю.

Надпись размещается под кнопкой и выравнивается по левому краю кнопки.

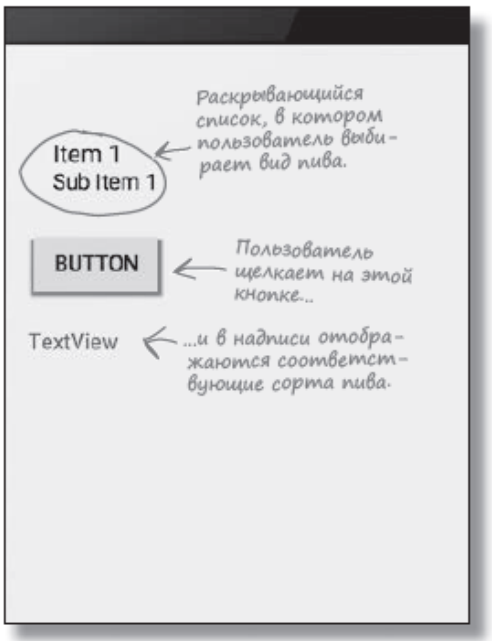
Замените содержимое *activity_find_beer.xml* показанной разметкой XML.

...отражаются в визуальном редакторе

- [x] Создание проекта
- [ ] Обновление макета
- [ ] Подключение активности
- [ ] Программирование логики

После внесения изменений в XML макета перейдите в визуальный редактор. Вместо макета с надписью и расположенной под ней кнопкой должен отображаться макет, в котором надпись отображается под кнопкой.

Над кнопкой располагается **раскрывающийся список** (spinner). Если коснуться его, на экране появляется список, из которого пользователь выбирает одно значение.



Мы показали, как добавлять компоненты графического интерфейса в макет в визуальном редакторе и как добавлять их в разметку XML. Скорее всего, для получения желаемых результатов вы будете чаще работать с XML напрямую, без использования визуального редактора. Дело в том, что прямое редактирование XML позволяет более точно управлять макетом, а также сокращает зависимость разработчика от среды разработки.

Раскрывающийся список предоставляет пользователю набор значений, из которого пользователь выбирает один вариант.

Компоненты графического интерфейса — кнопки, раскрывающиеся списки, надписи — обладают похожими атрибутами, так как все они являются специализациями View. Классы всех этих компонентов наследуют от одного класса Android View.

- [x] Создание проекта
- [ ] Обновление макета
- [ ] Подключение активности
- [ ] Программирование логики

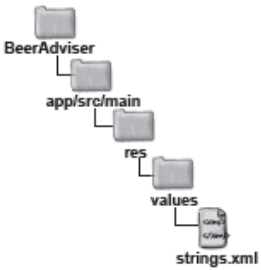
Использование строковых ресурсов вместо жестко запрограммированного текста

Прежде чем запускать приложение, необходимо внести еще одно изменение. На данный момент тексты кнопки и надписи задаются жестко запрограммированными строковыми значениями. Как упоминалось в главе 1, желательно заменить их ссылками на файл строковых ресурсов *strings.xml*. И хотя такая замена не является строго необходимой, это полезная привычка. Использование файла строковых ресурсов для статического текста упрощает создание версий приложения на других языках, а если вдруг потребуется изменить формулировки во всем приложении, достаточно будет внести изменения в одном централизованном месте.

Откройте файл *app/src/main/res/values/strings.xml*. При переходе в режим XML он должен выглядеть примерно так:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Beer Adviser</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>

</resources>
```

Эти строки созданы средой Android Studio.



Прежде всего удалите ресурс "hello_world", так как он нам больше не нужен. Добавьте новый ресурс с именем "find_beer" и значением "Find Beer!". Когда это будет сделано, добавьте новый ресурс с именем "brands", но пока не вводите значение.

Новый код должен выглядеть примерно так:

```
...
    <string name="app_name">Beer Adviser</string>
    <string name="hello_world">Hello world!</string>
    <string name="action_settings">Settings</string>
    <string name="find_beer">Find Beer!</string>
    <string name="brands"></string>
...
```

Необходимо удалить строковый ресурс hello_world и добавить два новых ресурса с именами find_beer и brands.

Внесение изменений в макет для использования строковых ресурсов

Теперь изменим элементы кнопки и надписи в разметке XML макета, чтобы они использовали два только что добавленных строковых ресурса.

Откройте файл *activity_find_beer.xml* и внесите следующие изменения:

- Замените строку `android:text="Button"` строкой `android:text="@string/find_beer"`.
- Замените строку `android:text="TextView"` строкой `android:text="@string/brands"`.


```
...
    <Spinner
        android:id="@+id/color"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="37dp" />

    <Button
        android:id="@+id/find_beer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/color"
        android:layout_below="@+id/color"
        android:text="@string/find_beer" />

    <TextView
        android:id="@+id/brands"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/find_beer"
        android:layout_below="@+id/find_beer"
        android:layout_marginTop="18dp"
        android:text="@string/brands" />
...
```

На кнопке выводится значение строкового ресурса find_beer.

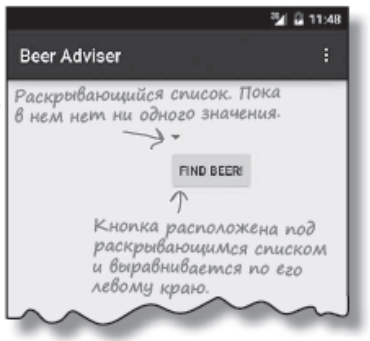
Значение строкового ресурса brands выводится в надписи. В настоящее время надпись пуста, но все будущие изменения строкового значения будут отражаться автоматически.

Посмотрим, что же получилось

- [x] Создание проекта
- [ ] Обновление макета
- [ ] Подключение активности
- [ ] Программирование логики

Над приложением еще придется поработать, но давайте посмотрим, как выглядит результат на текущий момент. Сохраните внесенные изменения и выберите команду "Run 'app'" из меню Run. Когда вам будет предложено выбрать способ запуска, выберите запуск в эмуляторе. Терпеливо подождите, пока приложение загрузится. Рано или поздно оно появится на экране.

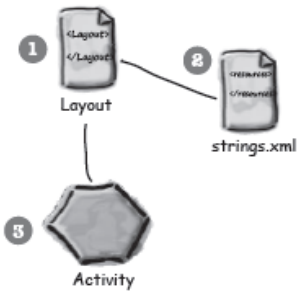
Попробуйте прикоснуться к раскрывающемуся списку. Возможно, это и не очевидно, но при прикосновении на экране должен появиться список значений — просто мы еще не добавили в него ни одного значения.



Что мы успели сделать

Ниже кратко перечислены основные действия, которые были выполнены на настоящий момент:

1. **Мы создали макет, определяющий внешний вид приложения.**
   Макет включает в себя раскрывающийся список, кнопку и надпись.
2. **Файл strings.xml включает необходимые строковые ресурсы.**
   Мы добавили текст для кнопки и пустую строку для сортов пива.
3. **Активность определяет, как приложение должно взаимодействовать с пользователем.**
   Среда Android Studio сгенерировала базовую активность, но она пока так и осталась в исходном виде.



Часто Задаваемые Вопросы

В: При запуске приложения макет несколько отличается от того, как он выглядел в визуальном редакторе. Почему?

О: Визуальный редактор очень старается правильно показать, как будет выглядеть макет, но у него есть свои ограничения. Например, в нашей разметке XML указано, что раскрывающийся список должен использовать горизонтальное выравнивание по центру, но в визуальном редакторе это может быть неочевидно.

На практике всегда лучше работать прямо с XML. Разметка дает более точное представление о том, что происходит, и предоставляет более точные средства контроля.

В: Кажется, в макете была еще и надпись?

О: Надпись есть, но пока она не содержит никакого текста, поэтому вы ее не видите. Она появится позднее в этой главе, когда мы настроим надпись для вывода текста.

Добавление значений в список

На данный момент макет включает раскрывающийся список, но в этом списке нет никаких данных. Чтобы список приносил пользу, в нем должен отображаться список значений. Пользователь выбирает в этом списке то значение, которое ему нужно. Список значений для раскрывающегося списка можно определить практически так же, как мы определим текст на кнопке и надписи: нужно создать для него **ресурс**. До сих пор в файле *strings.xml* определялись одиночные строковые значения. Все, что нам нужно, — это определить *массив* строковых значений и передать ссылку на него раскрывающемуся списку.

Ресурсы — непрограммные данные (например, графика или строки), используемые в приложении.

Добавление ресурса массива очень похоже на добавление ресурса строки

Как вы уже знаете, для добавления строкового ресурса в файл *strings.xml* необходимо выполнить следующие действия:

```
<string name="имя_строки">значение</string>
```

где `имя_строки` — идентификатор строки, а значение — собственно строковое значение.
Синтаксис добавления массива строк выглядит так:

```
<string-array name="имя_массива_строк">   ← Имя массива.
    <item>значение1</item>
    <item>значение2</item>                 } Значения в массиве. Добавьте
    <item>значение3</item>                   столько, сколько потребуется.
    ...
</string-array>
```

где `имя_массива` — имя массива, а `значение1`, `значение2`, `значение3` — отдельные строковые значения, входящие в массив.
Давайте включим ресурс `string-array` в наше приложение. Откройте файл *strings.xml* и добавьте в него следующий фрагмент:

```
...
    <string name="brands"></string>
    <string-array name="beer_colors">
        <item>light</item>
        <item>amber</item>
        <item>brown</item>
        <item>dark</item>
    </string-array>
</resources>
```

В файл strings.xml добавляется элемент string-array. Он определяет массив строк с именем beer_colors, состоящий из четырех значений: light, amber, brown и dark.

Передача массива строк раскрывающемуся списку

Для обращения к массиву строк в макете используется синтаксис, сходный с синтаксисом получения строкового значения. Вместо конструкции

```
"@string/имя_строки"
```

используется конструкция

```
"@array/имя_массива"
```

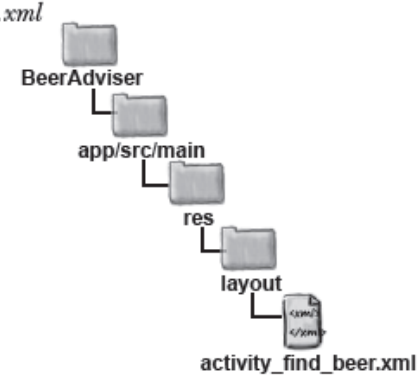
Используйте @string для обращения к строке и @array — для обращения к массиву.

где `имя_массива` — имя массива.

Используем эту ссылку в макете. Откройте файл макета *activity_find_beer.xml* и добавьте в элемент `<Spinner>` атрибут `entries`:

```
...
    <Spinner
        ...
        android:layout_marginTop="37dp"
        android:entries="@array/beer_colors" />
...
```

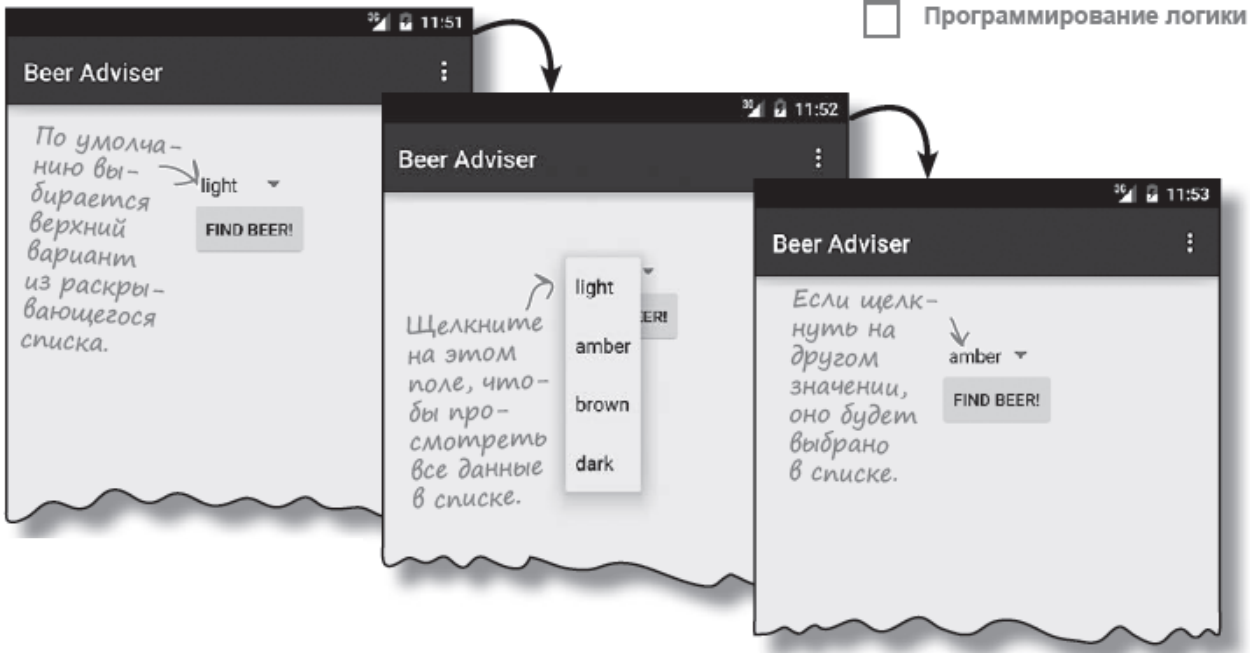
Это означает "данные раскрывающегося списка берутся из массива beer_colors".



Тест-драйв раскрывающегося списка

- [x] Создание проекта
- [x] **Обновление макета**
- [ ] Подключение активности
- [ ] Программирование логики

Посмотрим, как эти изменения отразились на приложении. Сохраните изменения и запустите приложение. Результат должен выглядеть примерно так:



Кнопка должна что-то делать

Создание проекта
Обновление макета
Подключение активности
Программирование логики

Пока что мы добавили в макет новые компоненты графического интерфейса и заполнили раскрывающийся список массивом значений. Теперь нужно добиться того, чтобы приложение реагировало на значение, выбранное в раскрывающемся списке. Оно должно работать приблизительно так:

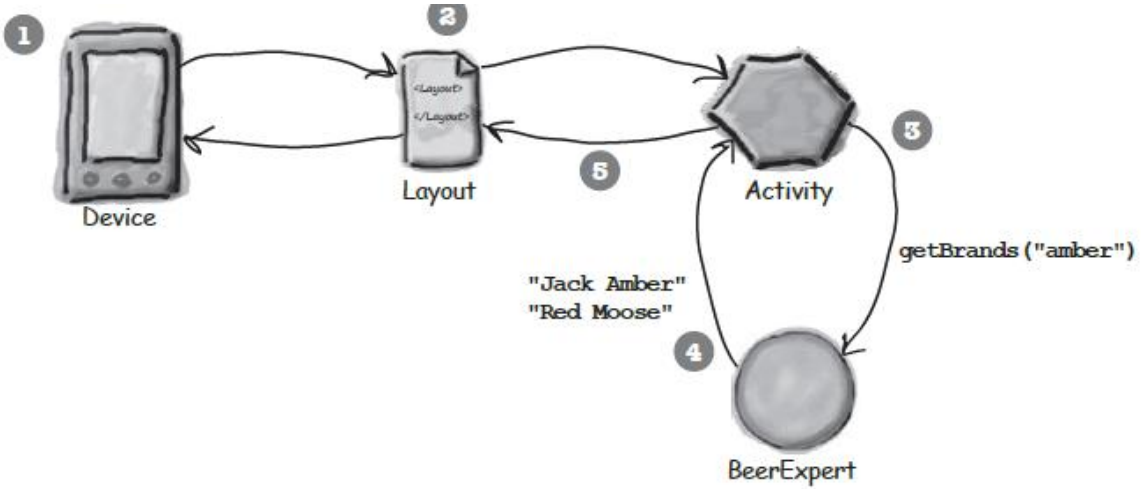
1. **Пользователь выбирает вид пива в раскрывающемся списке.**
   Пользователь щелкает на кнопке, чтобы найти сорта пива данного вида.

2. **Макет указывает, какой метод активности должен вызываться при щелчке на кнопке.**

3. **Метод активности получает от раскрывающегося списка выбранное значение (вид пива) и передает его методу getBrands() класса Java с именем BeerExpert.**

4. **Метод getBrands() класса BeerExpert находит сорта пива, соответствующие заданному виду, и возвращает их активности в виде объекта ArrayList со строковыми данными.**

5. **Активность получает ссылку на надпись из макета и присваивает ее свойству text список подходящих сортов.**
   Информация отображается на устройстве.



Для начала нужно добиться того, чтобы при нажатии кнопки вызывался метод.

Как заставить кнопку вызвать метод

Если вы добавляете в макет кнопку, то скорее всего, когда пользователь щелкает на этой кнопке, в приложении что-то должно происходить. Но для этого необходимо, чтобы при щелчке на кнопке вызывался некий метод вашей активности.

Чтобы щелчок на кнопке приводил к вызову метода активности, необходимо внести изменения в двух файлах:

- **Изменения в файле макета activity_find_beer.xml.** Необходимо указать, какой метод активности должен вызываться при щелчке на кнопке.
- **Изменения в файле активности FindBeerActivity.java.** Необходимо написать метод, который будет вызываться при щелчке.

Начнем с макета.

onClick и метод, вызываемый при щелчке

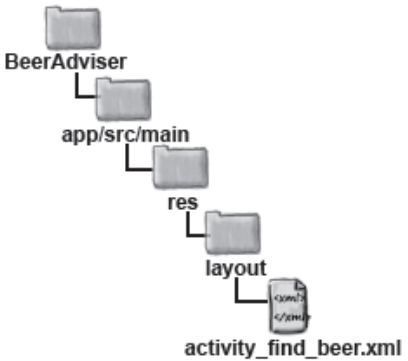
Чтобы сообщить Android, какой метод должен вызываться при щелчке на кнопке, достаточно всего одной строки разметки XML. Все, что для того нужно — добавить атрибут `android:onClick` в элемент `<button>` и указать имя вызываемого метода:

```
android:onClick="имя_метода"
```

Это означает "когда пользователь щелкает на компоненте, вызвать метод активности с именем имя_метода".

Посмотрим, как это делается. Откройте файл макета *activity_find_beer.xml* и добавьте в элемент `<button>` новую строку XML, которая сообщает, что при щелчке на кнопке должен вызываться метод `onClickFindBeer()`:

```
...
    <Button
        android:id="@+id/find_beer"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/color"
        android:layout_below="@+id/color"
        android:text="@string/find_beer"
        android:onClick="onClickFindBeer" />
...
```



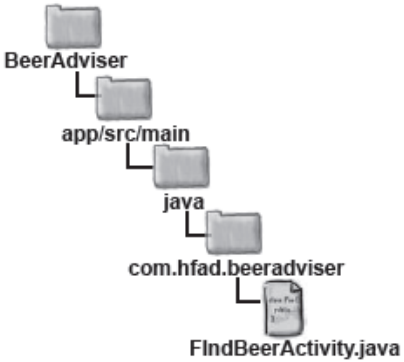
При щелчке на кнопке вызвать метод onClickFindBeer() активности. Мы создадим этот метод на нескольких ближайших страницах.

После внесения изменений сохраните файл.

Теперь макет знает, какой метод активности следует вызвать; но мы еще должны написать сам метод. Давайте посмотрим, как выглядит активность.

Как выглядит код активности

В процессе создания проекта для приложения мы приказали мастеру сгенерировать простейшую активность с именем `FindBeerActivity`. Код этой активности хранится в файле *FindBeerActivity.java*. Откройте этот файл — перейдите в папку *app/src/main/java* и сделайте на нем двойной щелчок.



Вы увидите, что среда Android Studio сгенерировала за вас довольно большой объем кода Java. Вместо того, чтобы подробно анализировать весь сгенерированный код, мы просто предложим заменить его кодом, приведенным ниже. Дело в том, что большая часть кода активности, сгенерированного Android Studio, не обязательна, а мы хотим сосредоточиться на фундаментальных аспектах Android, а не на особенностях отдельной среды разработки. Итак, удалите код, который сейчас находится в файле *FindBeerActivity.java*, и замените его следующим кодом:

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;

public class FindBeerActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }
}
```

Класс расширяет класс Android Activity.

Метод onCreate() вызывается при исходном создании активности.

Метод setContentView сообщает Android, какой макет использует активность. В данном случае это макет activity_find_beer.

Этот код — все, что необходимо для создания простейшей активности. Как видите, в нем создается класс, который расширяет класс `android.app.Activity` и реализует метод `onCreate()`.

Все активности должны расширять класс `Activity`. Класс `Activity` содержит набор методов, которые превращают обычный класс Java в полноценную активность Android.

Все активности также должны реализовать метод `onCreate()`. Метод `onCreate()` вызывается при создании объекта активности и используется для настройки основных параметров — например, выбора макета, с которым связывается активность. Это делается при помощи метода `setContentView()`. В приведенном примере вызов `setContentView(R.layout.activity_find_beer)` сообщает Android, что эта активность использует макет `activity_find_beer`.

На предыдущей странице мы добавили атрибут `onClick` к кнопке в макете и присвоили ему значение `onClickFindBeer`. Теперь нужно добавить этот метод в активность, чтобы он вызывался при нажатии кнопки. Таким образом, активность будет реагировать на нажатия пользователем кнопки в интерфейсе.

Замените код вашей версии *FindBeerActivity.java* кодом, приведенным на этой странице.

Добавление в активность метода onClickFindBeer()

Метод `onClickFindBeer()` должен иметь строго определенную сигнатуру; в противном случае он не будет вызываться при щелчке на кнопке, указанной в макете. Он имеет следующую форму:

```
public void onClickFindBeer(View view) {
}
```

Метод должен быть объявлен открытым (public).

Метод должен возвращать void.

Метод должен иметь один параметр с типом View.

Если метод имеет другую сигнатуру, он не будет реагировать на прикосновение пользователя к кнопке. Дело в том, что Android незаметно для пользователя ищет открытый метод, возвращающий void, имя которого совпадает с именем метода, указанного в разметке XML макета.

Параметр `View` на первый взгляд кажется несколько странным, но для его присутствия имеется веская причина. Он определяет компонент графического интерфейса, инициировавший вызов метода (в данном случае это кнопка). Как упоминалось ранее, компоненты графического интерфейса — такие, как кнопки и надписи, — все являются специализациями `View`.

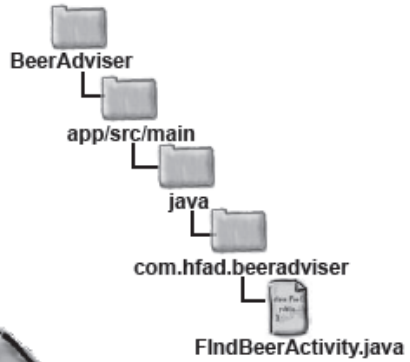
Итак, обновим наш код активности. Добавьте в код активности метод `onClickFindBeer()`, приведенный ниже:

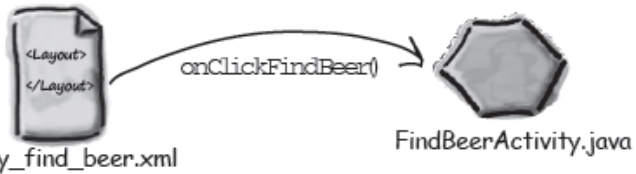
Чтобы метод мог реагировать на щелчки на кнопке, он должен быть объявлен открытым (public), возвращать void и получать один параметр типа View.

Мы используем этот класс, поэтому его необходимо импортировать.

Добавьте метод onClickFindBeer() в FindBeerActivity.java.

```
...
import android.view.View;

public class FindBeerActivity extends Activity {
...
    //Call when the user clicks the button
    public void onClickFindBeer(View view){
    }
}
```





Метод onClickFindBeer() должен что-то делать

Итак, мы создали в активности метод `onClickFindBeer()`. Далее нужно позаботиться о том, чтобы при выполнении этого метода что-то происходило. Приложение должно выводить подборку сортов пива, соответствующих виду, выбранному пользователем.

Для этого необходимо сначала получить ссылки на оба компонента графического интерфейса в макете – раскрывающийся список и надпись. С помощью этих ссылок мы сможем получить значение, выбранное в списке (вид пива), и вывести текст в надписи.

Использование findViewById() для получения ссылки на компонент

Для получения ссылки на два компонента графического интерфейса можно воспользоваться методом `findViewById()`. Метод `findViewById()` получает идентификатор компонента в виде параметра и возвращает объект `View`. Далее остается привести возвращаемое значение к правильному типу компонента (например, TextView или Button).

Посмотрим, как метод `findViewById()` используется для получения ссылки на надпись с идентификатором brands:

Нас интересует специализация View с идентификатором brands.

```
TextView brands = (TextView) findViewById(R.id.brands);
```

brands имеет тип TextView, поэтому ссылка приводится к этому типу.

Присмотритесь к тому, как задается идентификатор надписи. Вместо того, чтобы передавать имя компонента, мы передаем идентификатор вида `R.id.brands`. Что это означает? Что такое R?

R.java — специальный файл Java, который генерируется инструментарием Android при создании или построении приложения. Он находится в папке *app/build/generated/source/r/debug* вашего проекта — внутри папки, имя которой совпадает с именем пакета приложения. Android использует R для отслеживания ресурсов, используемых в приложении; среди прочего, этот класс позволяет получать ссылки на компоненты графического интерфейса из кода активности.

Открыв файл *R.java*, вы увидите, что он содержит серию внутренних классов, по одному для каждого типа ресурсов. Обращение к каждому ресурсу этого типа осуществляется через внутренний класс. Скажем, `R` включает внутренний класс с именем `id`, а в этот внутренний класс входит значение `static final brands`. Строка кода

```
(TextView) findViewById(R.id.brands);
```

использует это значение для получения ссылки на надпись `brands`.

R — специальный класс Java, который позволяет получать ссылки на ресурсы в приложении.

***R.java* генерируется за вас.** Вы не сможете изменять код, находящийся в `R`, но полезно знать, что он существует.

Получив ссылку на объект View, вы можете вызывать его методы

Метод `findViewById()` предоставляет Java-версию компонента графического интерфейса. Это означает, что вы можете читать и задавать свойства компонента при помощи методов, предоставляемых классом Java. Давайте разберемся подробнее.

Назначение текста в компоненте TextView

Как вы уже видели, для получения ссылки на компонент надписи в Java используется синтаксис

```
TextView brands = (TextView) findViewById(R.id.brands);
```

При выполнении этой строки кода создается объект класса `TextView` с именем `brands`. После этого вы можете вызывать методы этого объекта `TextView`. Допустим. вы хотите, чтобы в надписи brands отображался текст "Gottle of geer". Класс `TextView` содержит метод `setText()`, используемый для задания свойства text. Он используется следующим образом:

```
brands.setText("Gottle of geer");
```

Объекту TextView с именем brands назначается текст "Gottle of geer".

Получение выбранного значения в раскрывающемся списке

Вы также можете получить ссылку на раскрывающийся список; это делается практически так же, как для надписи. Снова используется метод `findViewById()`, но на этот раз результат приводится к типу `Spinner`:

```
Spinner color = (Spinner) findViewById(R.id.color);
```

Вы получаете объект `Spinner` и можете вызывать его методы. Например, вот как происходит получение текущего выбранного варианта в списке и преобразование его к типу `String`:

```
String.valueOf(color.getSelectedItem())
```

Получает выбранный вариант в списке и преобразует его в String.

Конструкция

```
color.getSelectedItem()
```

возвращает обобщенный объект Java. Дело в том, что значения раскрывающегося списка не обязаны быть объектами String – это могут быть, например, изображения. В нашем случае известно, что значения представляют собой объекты String, поэтому мы используем метод `String.valueOf()` для преобразования выбранного варианта из `Object` в `String`.

Обновление кода активности

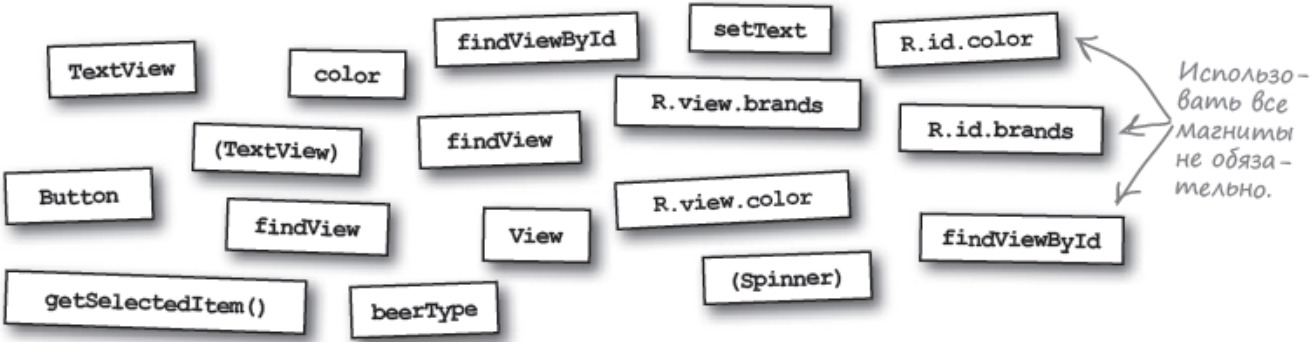
Вы уже знаете достаточно для того, чтобы написать часть кода метода `onClickFindBeer()`. Вместо того, чтобы писать весь необходимый код за один подход, начнем с получения варианта, выбранного в раскрывающемся списке, и отображения его в надписи.

Развлечения с магнитами

Кто-то написал метод **`onClickFindBeer()`** для нашей новой активности на холодильнике, заменив пустые места магнитами. К сожалению, от сквозняка магниты упали на пол. Сможете ли вы снова собрать код метода?

Метод должен получать вид пива, выбранный в раскрывающемся списке, и выводить его в надписи.

```
//Вызывается при щелчке на кнопке
public void onClickFindBeer(_____view) {

    //Получить ссылку на TextView
    _____ brands = _____ _____(_____);

    //Получить ссылку на Spinner
    Spinner_____= _____ _____(_____);

    //Получить вариант, выбранный в Spinner
    String _____ = String.valueOf(color._____);

    //Вывести выбранный вариант
    brands._____(beerType);
```

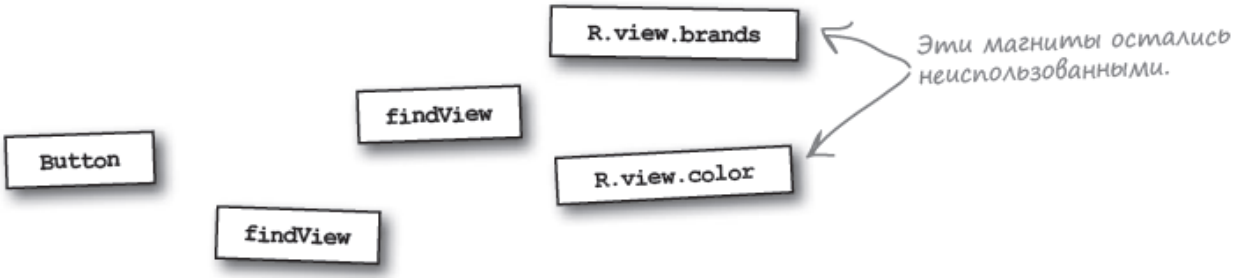

TextView
color
findViewById
setText
R.id.color
R.view.brands
R.id.brands
(TextView)
findView
Button
findView
View
R.view.color
findViewById
(Spinner)
getSelectedItem()
beerType
Использовать все магниты не обязательно.

Развлечения с магнитами. Решение

Кто-то написал метод `onClickFindBeer()` для нашей новой активности на холодильнике, заменив пустые места магнитами. К сожалению, от сквозняка магниты упали на пол. Сможете ли вы снова собрать код метода?

Метод должен получать вид пива, выбранный в раскрывающемся списке, и выводить его в надписи.

```
//Вызывается при щелчке на кнопке
public void onClickFindBeer(View view) {

    //Получить ссылку на TextView
    TextView brands = (TextView) findViewById(R.id.brands);

    //Получить ссылку на Spinner
    Spinner color = (Spinner) findViewById(R.id.color);

    //Получить вариант, выбранный в Spinner
    String beerType = String.valueOf(color.getSelectedItem());

    //Вывести выбранный вариант
    brands.setText(beerType);
}
```

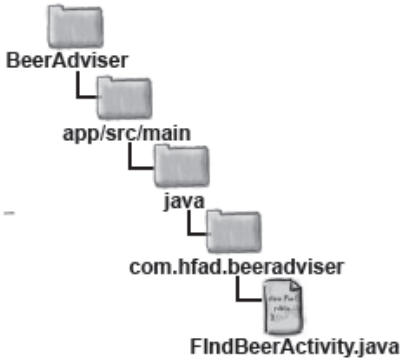

R.view.brands
findView
Button
R.view.color
findView
Эти магниты остались
неиспользованными.

Первая версия активности

Наш хитроумный план заключается в том, чтобы строить активность поэтапно и тестировать ее на каждом этапе. В итоге активность должна получить значение, выбранное в раскрывающемся списке, вызвать метод вспомогательного класса Java, а затем вывести подходящие сорта пива. От первой версии требуется совсем немного: она должна убедиться в том, что выбранный вариант правильно читается из списка. Ниже приведен код активности вместе с методом, который мы собрали воедино на предыдущей странице. Внесите эти изменения в *FindBeerActivity.java* и сохраните файл:

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;

public class FindBeerActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }

    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Вывести выбранный вариант
        brands.setText(beerType);
    }
}
```

Мы используем эти дополнительные классы.



Этот метод не изменялся.

findViewById возвращает объект View. Его необходимо преобразовать к правильному подтипу View.

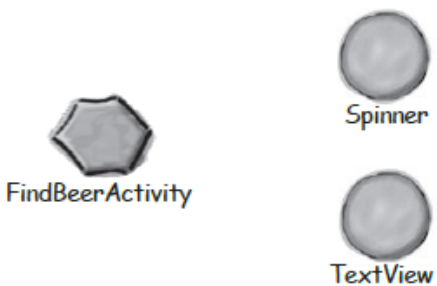
getSelectedItem возвращает Object. Этот объект необходимо преобразовать в String.

Что делает этот код

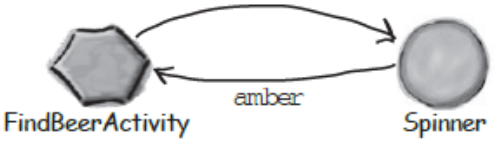
Прежде чем переходить к тестированию приложения, разберемся, что же делает этот код.

1. **Пользователь выбирает вид пива в раскрывающемся списке и щелкает на кнопке Find Beer. Это приводит к вызову метода public void onClickFindBeer(View) активности.** Макет сообщает, какой метод активности должен вызываться при щелчке на кнопке, при помощи свойства `android:onClick` кнопки.

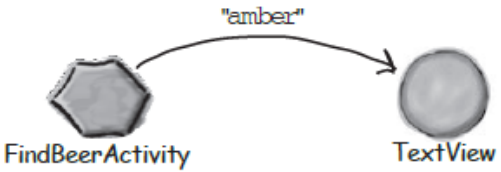
2. **Чтобы получить ссылки на компоненты графического интерфейса TextView и Spinner, активность вызывает метод findViewById().**



3 **Активность получает текущее выбранное значение в раскрывающемся списке и преобразует его в String.**



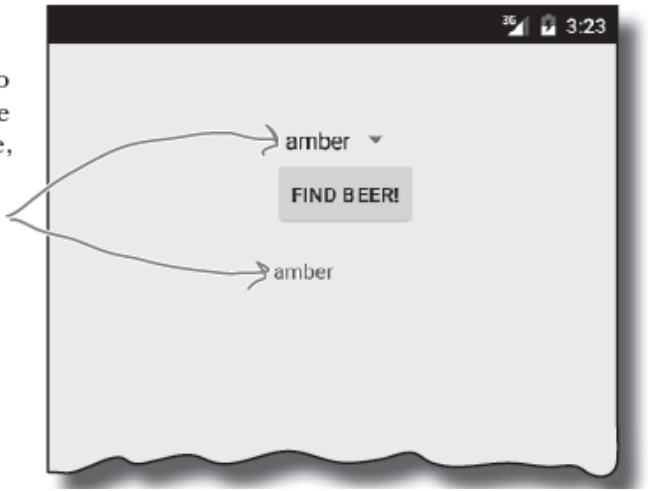
4 **Затем активность задает свойство text компонента TextView, чтобы текущий вариант из списка отображался в надписи.**



Тест-драйв

Внесите изменения в файл активности, сохраните его и запустите приложение. Теперь при щелчке на кнопке Find Beer значение варианта, выбранного в списке, выводится в надписи.

Выбранный вид пива отображается в надписи.



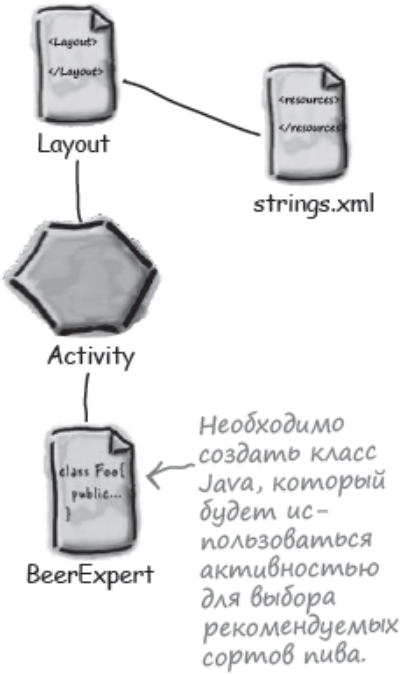
Построение вспомогательного класса Java

Как упоминалось в начале главы, приложение Beer Adviser решает, какие сорта пива рекомендовать пользователю, при помощи вспомогательного класса Java. Этот класс Java содержит самый обычный код Java, и ничто в нем не указывает на то, что этот код используется Android-приложением.

Спецификация вспомогательного класса Java

Вспомогательный класс Java должен удовлетворять следующим требованиям:

- Класс должен принадлежать пакету `com.hfad.beeradviser`.
- Классу должно быть присвоено имя `BeerExpert`.
- Класс должен предоставлять один метод `getBrands()`, который получает желательный вид пива (в виде String) и возвращает контейнер `List<String>` с рекомендуемыми сортами.

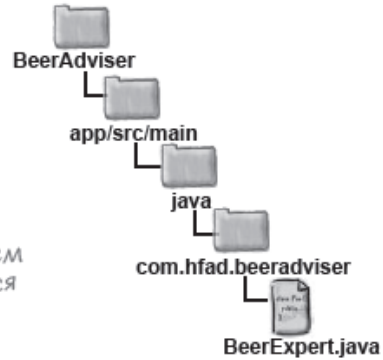


Построение и тестирование класса Java

Классы Java бывают чрезвычайно сложными, в них могут быть задействованы вызовы нетривиальной логики приложения. Либо постройте собственную версию класса, либо воспользуйтесь нашей готовой версией, приведенной ниже:

```
package com.hfad.beeradviser;
import java.util.ArrayList;
import java.util.List;

public class BeerExpert {
    List<String> getBrands(String color) {
        List<String> brands = new ArrayList<String>();
        if (color.equals("amber")) {
            brands.add("Jack Amber");
            brands.add("Red Moose");
        } else {
            brands.add("Jail Pale Ale");
            brands.add("Gout Stout");
        }
        return brands;
    }
}
```

Обычный код Java — в нем нет ничего, относящегося к специфике Android.



Добавьте класс BeerExpert в свой проект. Выделите пакет *com.hfad.beeradviser* в папке *app/src/main/java* и выполните команду File→New...→Java Class. Новый класс будет создан в этом пакете.

Активность дополняется вызовом метода вспомогательного класса Java для получения НАСТОЯЩИХ рекомендаций

Во второй версии активности мы доработаем метод `onClickFindBeer()`, чтобы он вызывал метод класса `BeerExpert` для получения рекомендаций. Все необходимые изменения содержат только традиционный код Java. Вы можете попытаться написать код и запустить приложение самостоятельно, или же переверните страницу и повторяйте за нами.

Доработайте активность, чтобы она вызывала метод `getBrands()` класса `BeerExpert` и выводила результаты в надписи.

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;
```

Мы добавили эту строку за вас.

```
public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
...
    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);

        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());

        [                                                    ]
        [                                                    ]
        [                                                    ]
        [                                                    ]

    }
}
```

Для получения рекомендаций необходимо использовать класс BeerExpert, поэтому мы добавили и эту строку.

Попробуйте дописать метод onClickFindBeer().

Решение

Доработайте активность, чтобы она вызывала метод `getBrands()` класса `BeerExpert` и выводила результаты в надписи.

```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();
...
    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());

        //Получить рекомендации от класса BeerExpert
        List<String> brandsList = expert.getBrands(beerType);
        StringBuilder brandsFormatted = new StringBuilder();
        for (String brand : brandsList) {
            brandsFormatted.append(brand).append('\n');
        }

        //Display the beers
        brands.setText(brandsFormatted);

    }
}
```

Получить контейнер List с сортами пива.

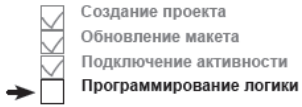
Построить String по данным из List.

Каждый сорт выводится с новой строки.

Вывести результаты в надписи.

Реализация BeerExpert содержит только традиционный код Java, поэтому не беспокойтесь, если ваш код немного отличается от нашего.

Код активности, версия 2



Ниже приведена полная версия кода активности. Внесите изменения в свою версию *FindBeerActivity.java*, убедитесь в том, что класс `BeerExpert` включен в проект, и сохраните изменения:


```
package com.hfad.beeradviser;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.view.View;
import android.widget.Spinner;
import android.widget.TextView;
import java.util.List;

public class FindBeerActivity extends Activity {
    private BeerExpert expert = new BeerExpert();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_find_beer);
    }

    //Вызывается при щелчке на кнопке
    public void onClickFindBeer(View view) {
        //Получить ссылку на TextView
        TextView brands = (TextView) findViewById(R.id.brands);
        //Получить ссылку на Spinner
        Spinner color = (Spinner) findViewById(R.id.color);
        //Получить вариант, выбранный в Spinner
        String beerType = String.valueOf(color.getSelectedItem());
        //Получить рекомендации от класса BeerExpert
        List<String> brandsList = expert.getBrands(beerType);
        StringBuilder brandsFormatted = new StringBuilder();
        for (String brand : brandsList) {
            brandsFormatted.append(brand).append('\n');
        }
        //Вывести сорта пива
        brands.setText(brandsFormatted);
    }
```

В программе используется этот класс.

Добавьте экземпляр BeerExpert как приватную переменную.

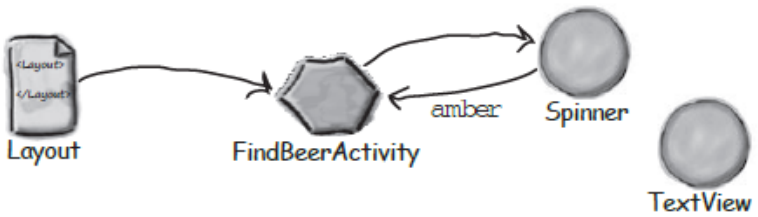
Класс BeerExpert используется для получения набора сортов.

Здесь строится объект String, в котором каждый сорт выводится с новой строки.

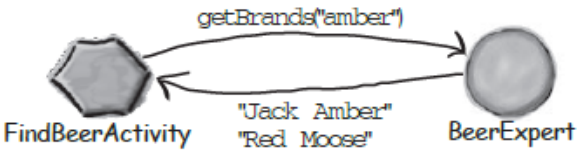
Содержимое String отображается в надписи.

Что происходит при выполнении кода

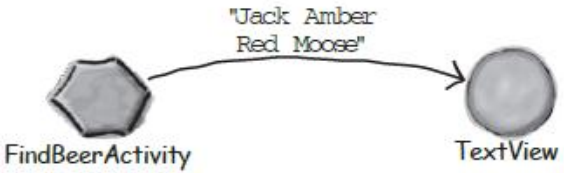
1 **Когда пользователь щелкает на кнопке Find Beer, вызывается метод onClickFindBeer() из класса активности.**
Метод создает ссылку на раскрывающийся список и надпись и получает текущее значение, выбранное в списке.



2 **Метод onClickFindBeer() вызывает метод getBrands() из класса BeerExpert, передавая ему вид пива, выбранный в раскрывающемся списке.**
Метод `getBrands()` возвращает список сортов пива.



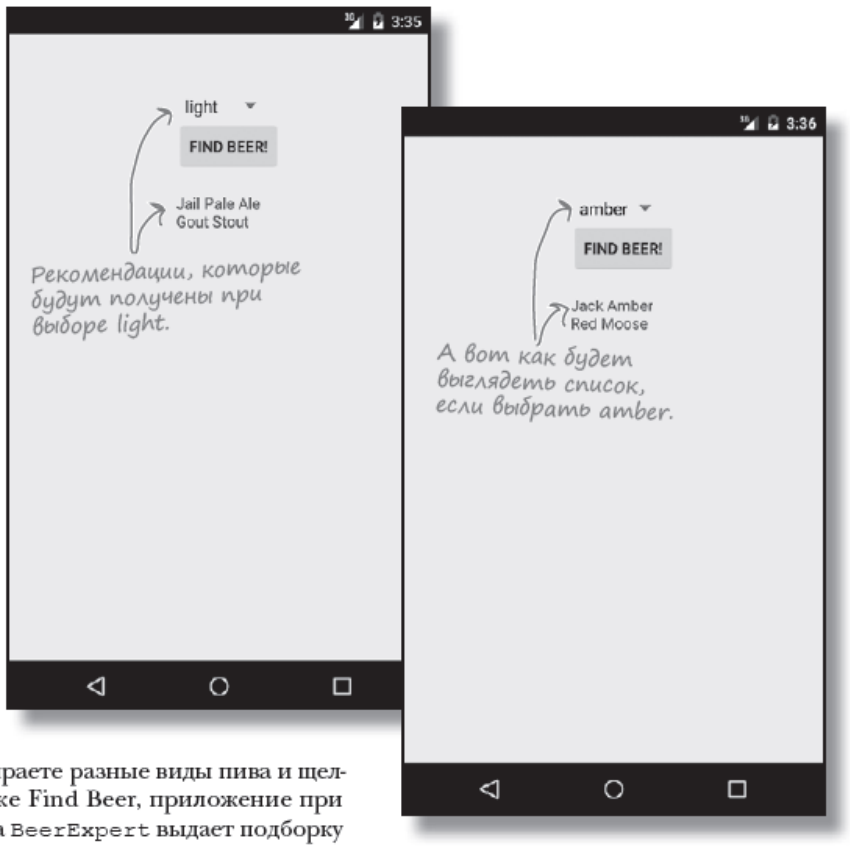
5 **Метод onClickFindBeer() форматирует список сортов и использует его для задания свойства text надписи.**



Тест-драйв

- Создание проекта
- Обновление макета
- Подключение активности
- **Программирование логики**

После того как приложение будет изменено, запустите его. Поэкспериментируйте, выбирая разные виды пива и щелкая на кнопке Find Beer.



Когда вы выбираете разные виды пива и щелкаете на кнопке Find Beer, приложение при помощи класса `BeerExpert` выдает подборку подходящих сортов.

ВИСНОВКИ

- Элемент `Button` используется для добавления кнопки.
- Элемент `Spinner` используется для добавления раскрывающегося списка.
- Все компоненты графического интерфейса наследуют от класса Android `View`.
- Массив строковых значений создается конструкцией следующего вида:

```
<string-array name="array">
    <item>string1</item>
    ...
</string-array>
```

- Для обращения к `string-array` в макете используется синтаксис:

```
"@array/array_name"
```

- Чтобы при щелчке на кнопке вызывался метод, включите в макет следующий атрибут:

```
android:onClick="clickMethod"
```

  При этом в активности должен существовать соответствующий метод:

```
public void clickMethod(View view){
}
```

- Класс *R.java* генерируется средой. Он позволяет получать ссылки на макеты, компоненты графического интерфейса, строки и другие ресурсы в коде Java.
- Метод `findViewById()` возвращает ссылку на компонент.
- Метод `setText()` задает текст компонента.
- Метод `getSelectedItem()` возвращает вариант, выбранный в раскрывающемся списке.
- Чтобы добавить вспомогательный класс в проект Android, выполните команду File→New...→Java Class.