



ТЕХНОТРЕК

Занятие №7

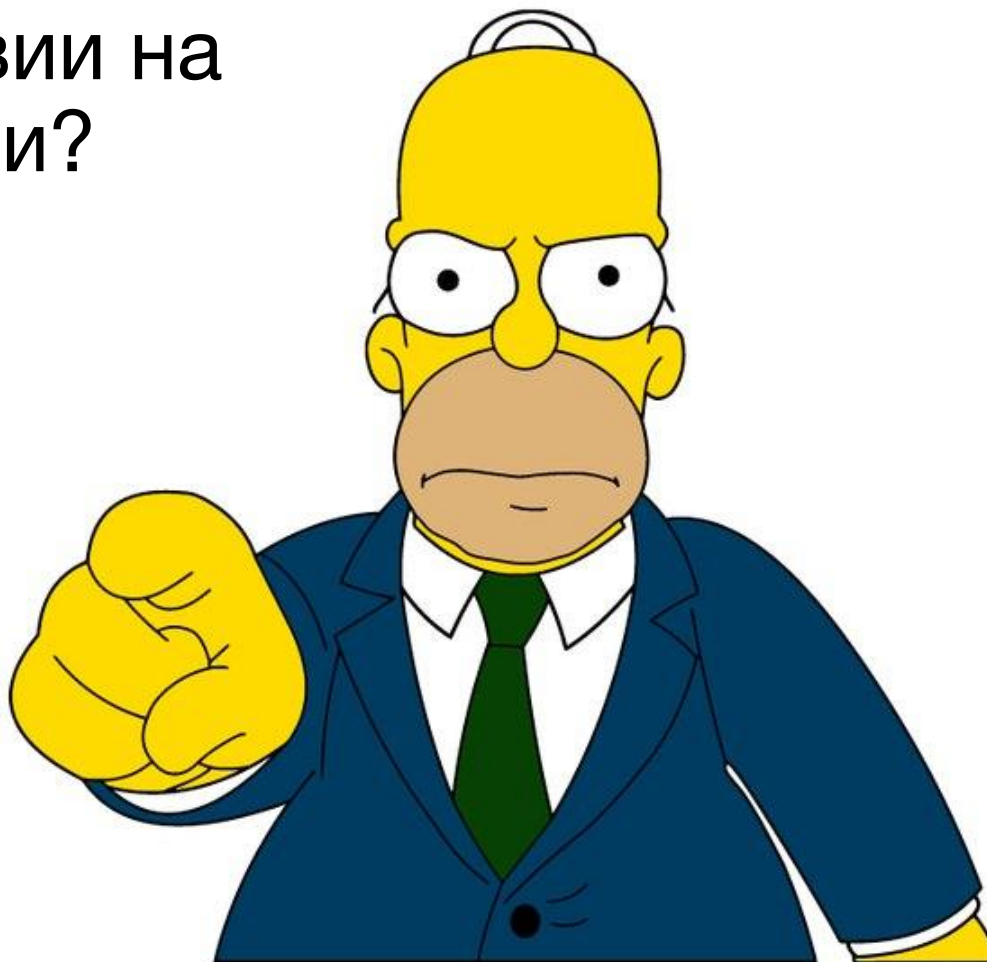
Разработка приложений на Android

Кирилл Филимонов
Юрий Береза

Напоминание



А ты отметился о
присутствии на
занятии?



Agenda



1. Работа с ресурсами
2. Размеры экранов
3. Поддержка разных экранов
4. Темы и стили
5. Автоматическое тестирование

Поддержка разных размеров



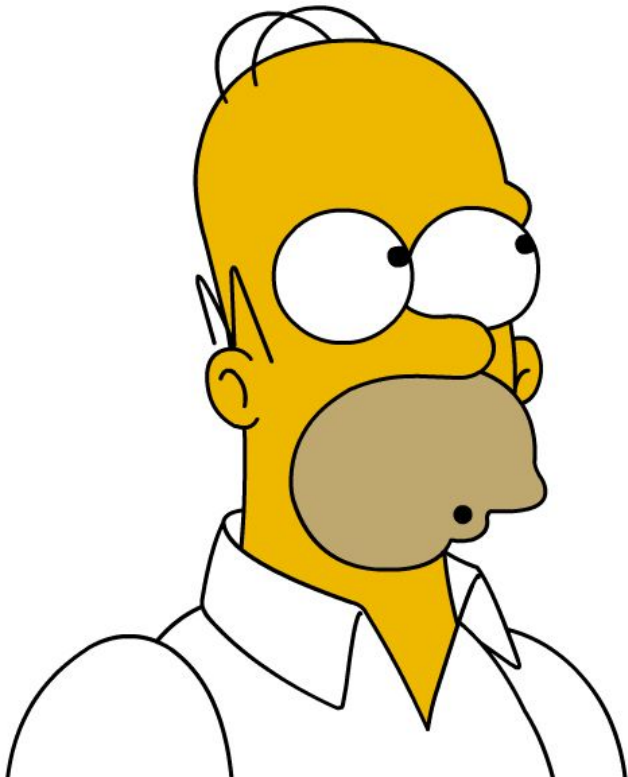
1. Почему это важно
2. Где это используется
3. Что вообще входит в понятие разных размеров



Термины



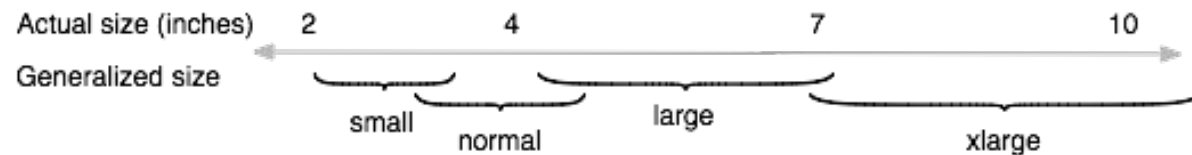
1. Размер экрана (Screen size)
2. Плотность экрана (Screen density)
3. Ориентация (Orientation)
4. Разрешение (Resolution)
5. Density-independent pixel



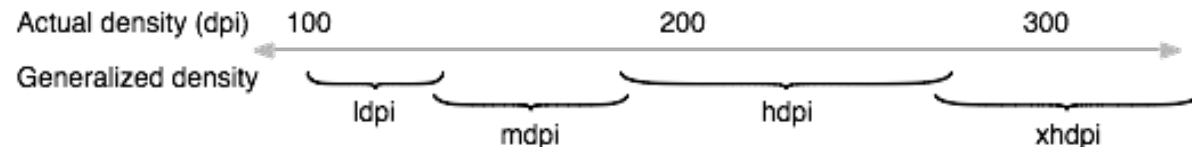
Размеры экранов



- 4 размера: small, normal, large, and xlarge
 - xlarge как минимум 960dp x 720dp
 - large как минимум 640dp x 480dp
 - normal как минимум 470dp x 320dp
 - small как минимум 426dp x 320dp



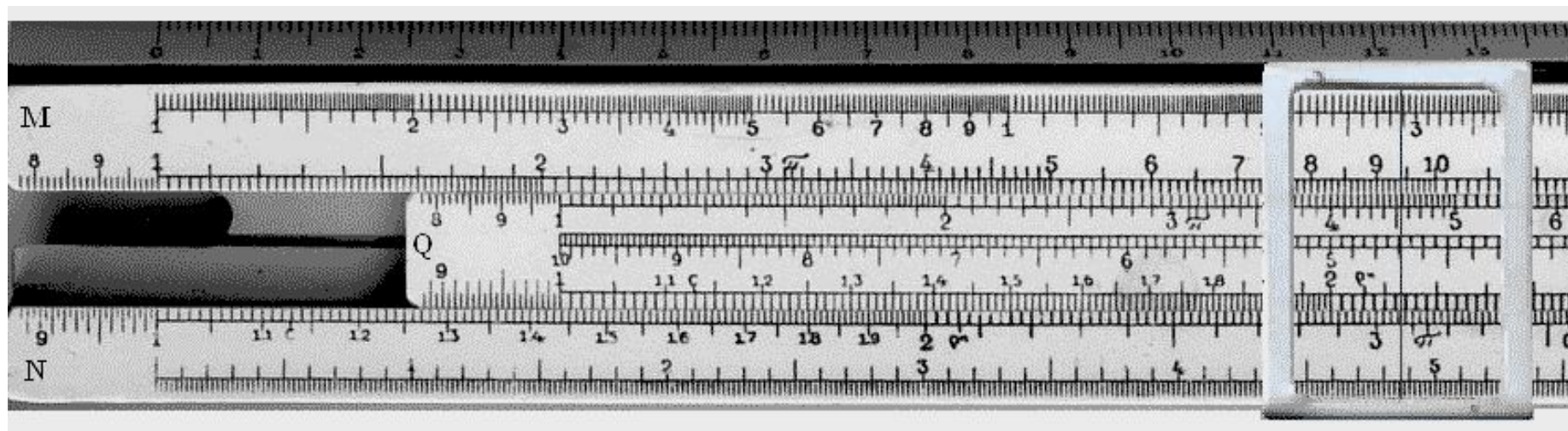
- Набор плотностей :
 - ldpi (low) ~120dpi
 - mdpi (medium) ~160dpi
 - hdpi (high) ~240dpi
 - xhdpi (extra-high) ~320dpi
 - xxhdpi (extra-extra-high) ~480dpi
 - xxxhdpi (extra-extra-extra-high) ~640dpi



Единицы измерения



- dp – density-independent pixels
- sp – scale-independent pixels
- pt – point 1/72 дюйма
- px – пиксели
- mm – миллиметры
- in – дюймы



Использование квалификаторов



Какие бывают квалификаторы

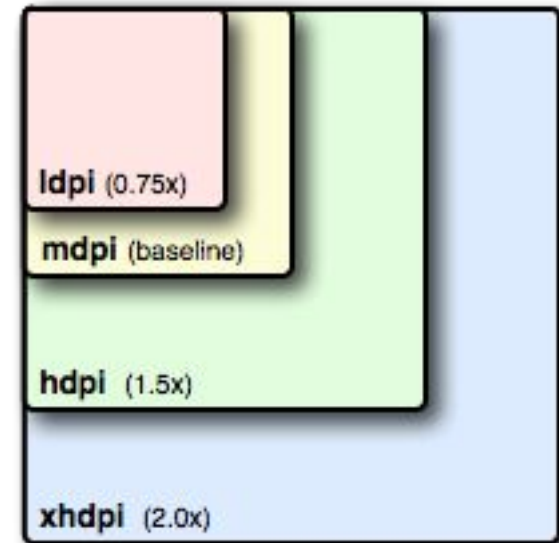


- Размеры: small, normal, large, xlarge
- Плотность: ldpi, mdpi, hdpi, xhdpi, xxhdpi, xxxhdpi, nodpi, tvdpi, anydpi
- Ориентация: land, port, square
- Соотношение сторон: long, notlong
- Заданная минимальная ширина: $sw < N > dp$
- Доступная ширина: $w < N > dp$
- Доступная высота: $h < N > dp$

Создание наборов картинок



1. 36x36 (0.75x) for low-density
2. 48x48 (1.0x baseline) for medium-density
3. 72x72 (1.5x) for high-density
4. 96x96 (2.0x) for extra-high-density
5. 180x180 (3.0x) for extra-extra-high-density
6. 192x192 (4.0x) for extra-extra-extra-high-density (launcher icon only)



VectorDrawables



- Появились API 21 (Android 5.0)
- Стали доступным в AppCompat Library 23.2

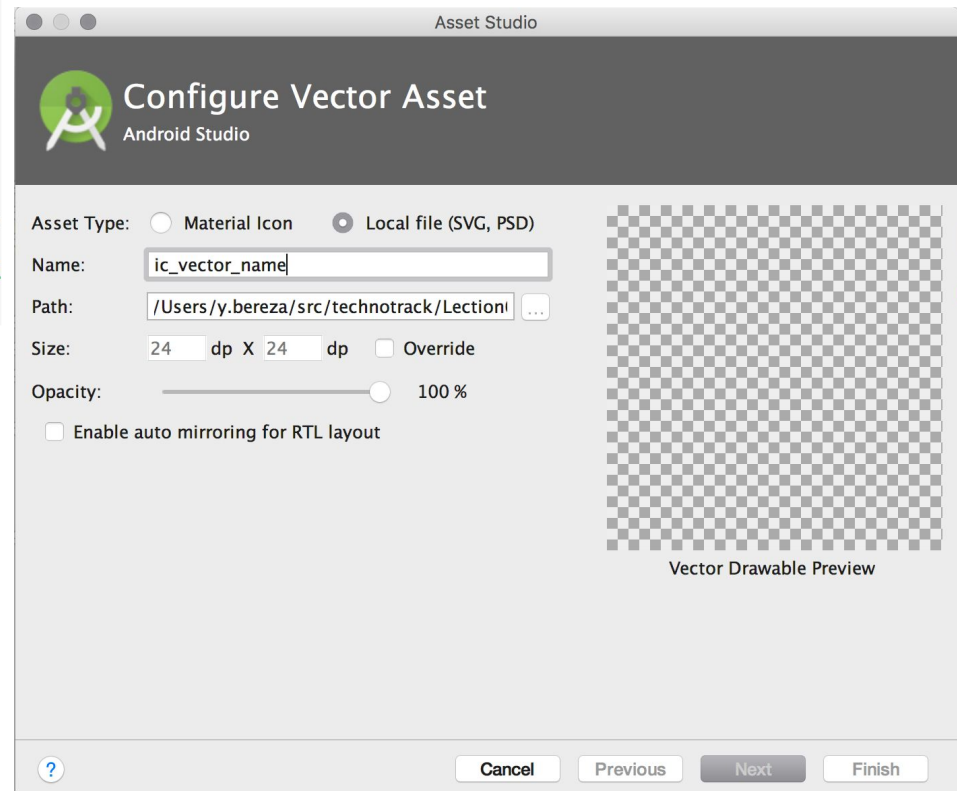
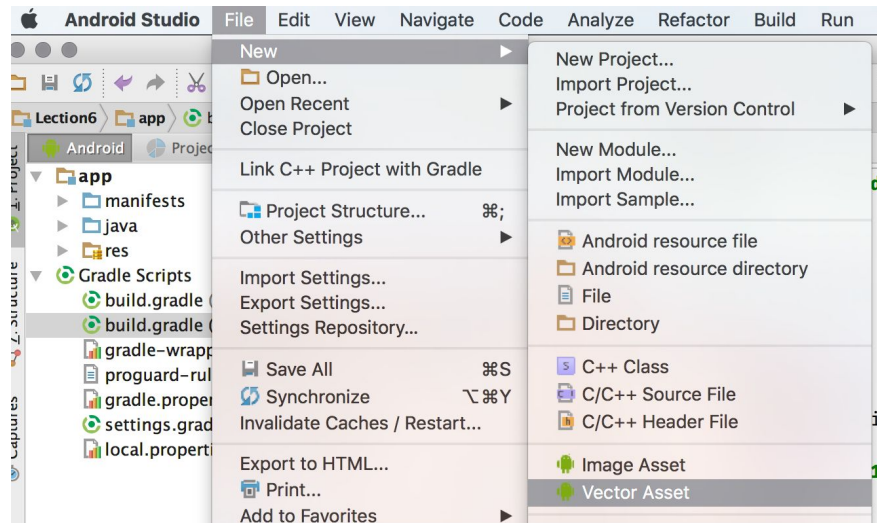
build.gradle

```
android {  
    defaultConfig {  
        vectorDrawables.useSupportLibrary = true  
    }  
}  
  
dependencies {  
    compile 'com.android.support:appcompat-v7:v23.4.0'  
}
```

MainActivity.java

```
class MainActivity extends AppCompatActivity {  
    static {  
        AppCompatActivityDelegate.setCompatVectorFromResourcesEnabled(true);  
    }  
}
```

VectorDrawables



Как поддерживать множество разрешений

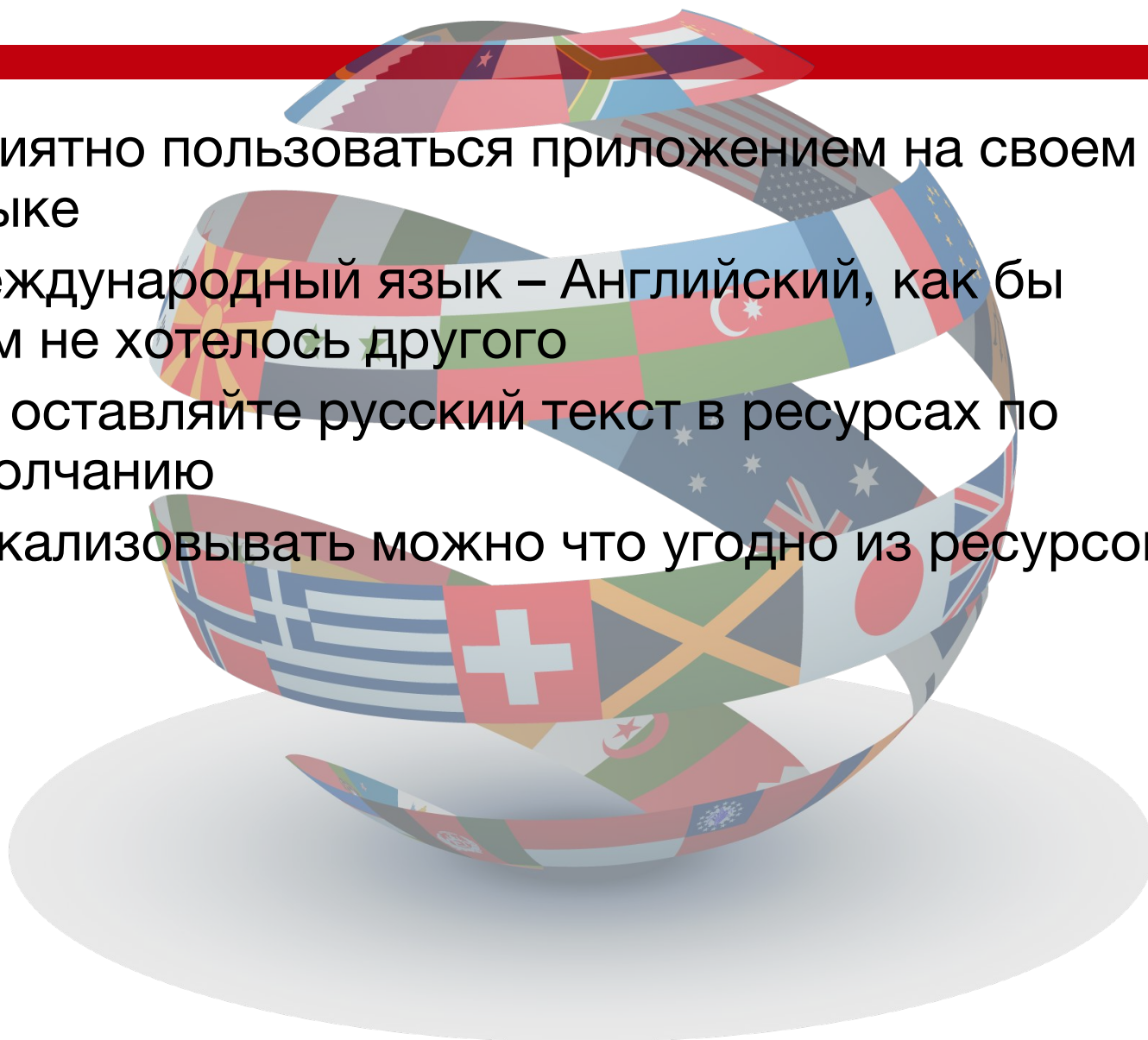


- Если это необходимо пишите разные layout для разных экранов
- Используйте разные картинки
- Использовать VectorDrawables
- Проверяйте типичные разрешения
- Используйте sp только для текста
- Используйте dp для размеров
- Не используйте AbsoluteLayout
- Не используйте px
- Используйте ресурсы под размеры экрана

Локализация приложения



- Приятно пользоваться приложением на своем языке
- Международный язык – Английский, как бы нам не хотелось другого
- Не оставляйте русский текст в ресурсах по умолчанию
- Локализовать можно что угодно из ресурсов



Поддержка RTL



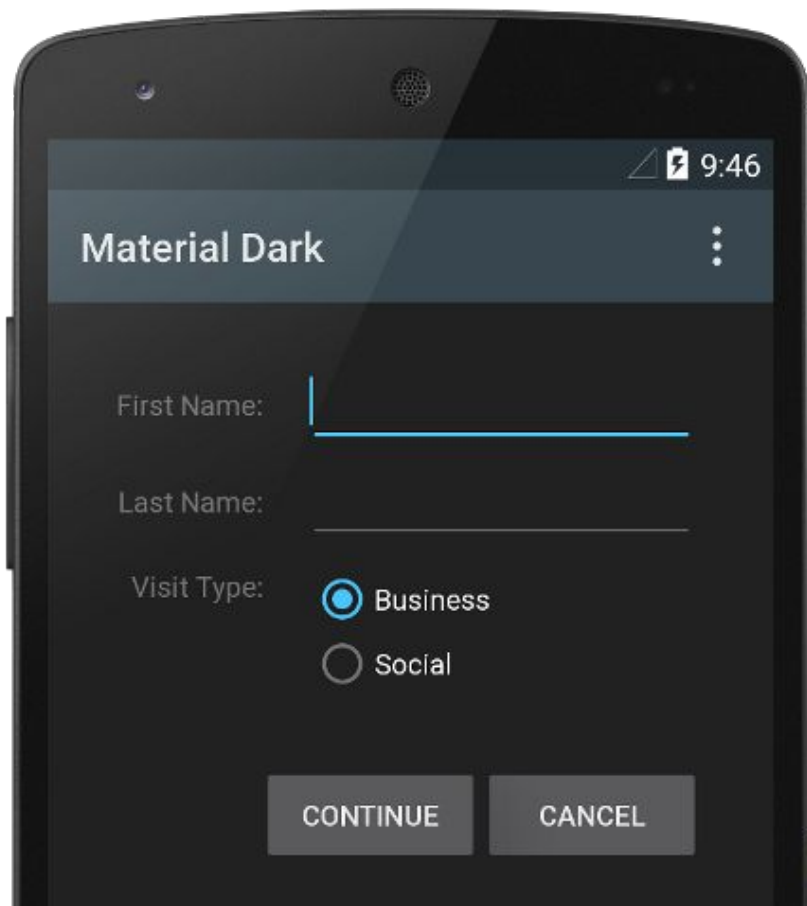
1. Все работает по умолчанию, если не выключить в манифесте (`android:supportsRtl="false"`)
2. Надо обращать внимание на `margin`, `padding`, `align` и `gravity` и использовать вместо `left` и `right`, `start` и `end`



Android Styles & Themes

- Что такое тема
- Для чего нужны
- Как обеспечивается
- Стили
- Наследование
- Темы активити
- Темы приложения
- Применение
- Редактор тем
- Как переключать темы

Темы



Material Dark

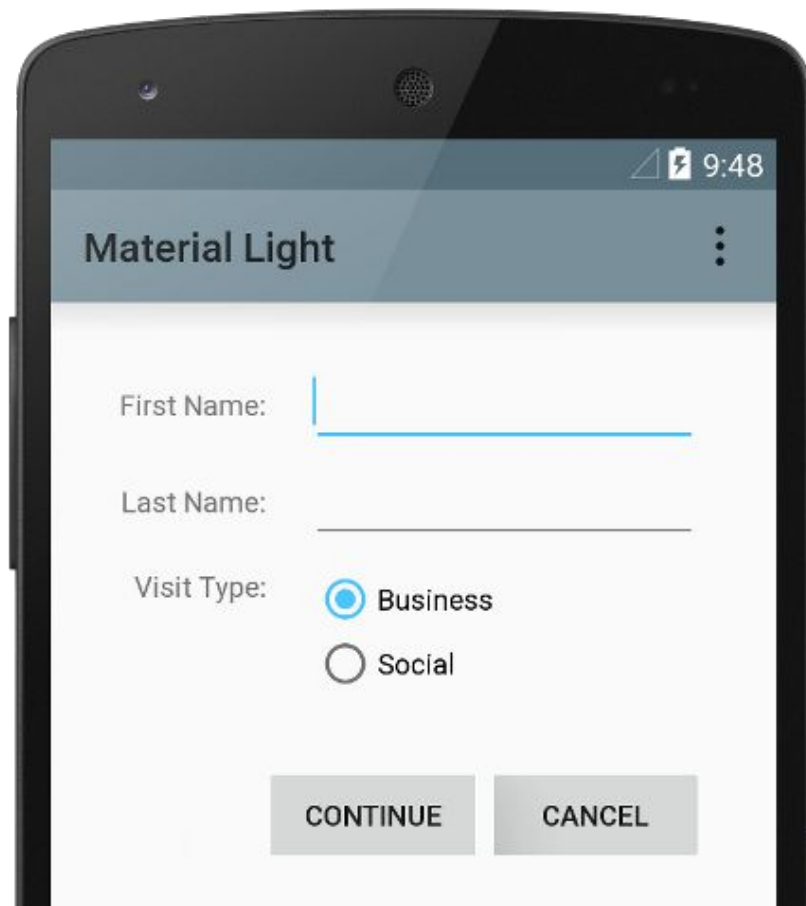
First Name:

Last Name:

Visit Type: ☒ Business ☐ Social

CONTINUE CANCEL

This mockup shows a dark-themed mobile app interface. The title bar is dark blue with the text 'Material Dark' and a three-dot menu icon. The background is dark gray. The form fields have light gray borders and labels. The 'Business' radio button is selected and highlighted with a blue circle. The 'CONTINUE' and 'CANCEL' buttons are light gray.



Material Light

First Name:

Last Name:

Visit Type: ☒ Business ☐ Social

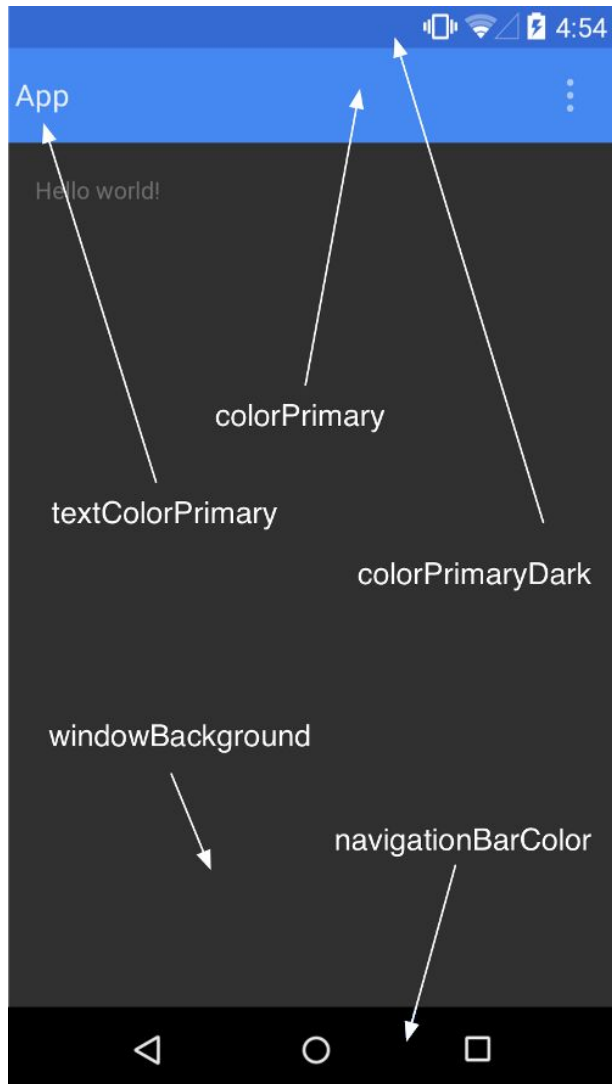
CONTINUE CANCEL

This mockup shows a light-themed mobile app interface. The title bar is light blue with the text 'Material Light' and a three-dot menu icon. The background is white. The form fields have light gray borders and labels. The 'Business' radio button is selected and highlighted with a blue circle. The 'CONTINUE' and 'CANCEL' buttons are light gray.

Темы

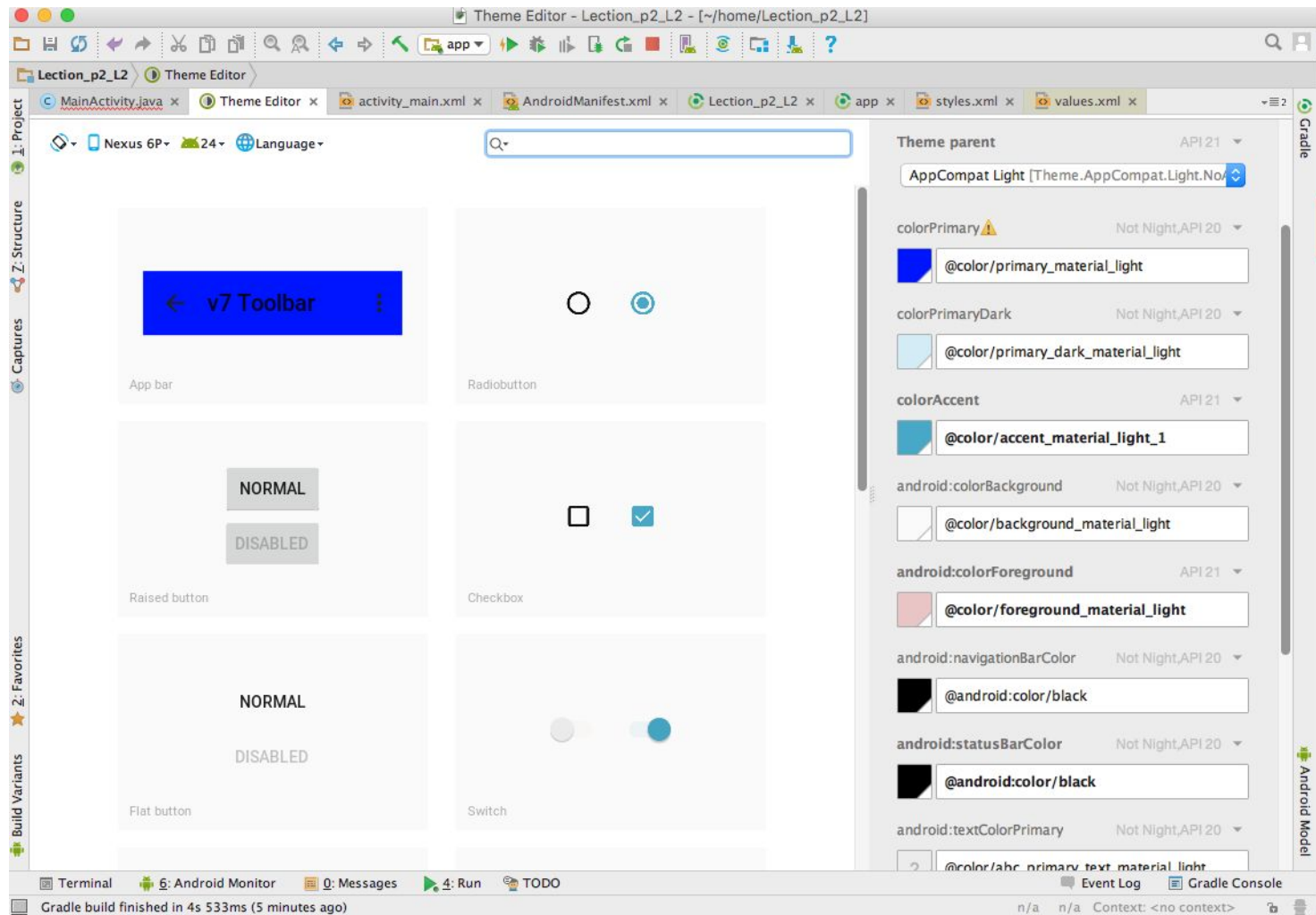
- Красота
- Единообразие
- Легкость в настройке и поддержке
- Возможность менять

Темы



- Основаны на стилях
- Задаются специальные параметры темы
- Есть редактор (не полный) в Android Studio
- Можно задавать на приложение
- Можно задавать на активности
- Внутри тем могут быть стили

Редактор тем



Пример определения темы

```
<resources xmlns:android="http://schemas.android.com/apk/res/android">

  <!-- ...generated stuff here -->

  <!-- This is the generated app theme -->
  <style name="AppTheme" parent="AppBaseTheme">
    <!-- These are your custom properties -->
    <item name="android:buttonStyle">@style/Widget.Button.Custom</item>
    <item name="android:textViewStyle">@style/Widget.TextView.Custom</item>
  </style>

  <!-- This is the custom button styles for this application -->
  <style name="Widget.Button.Custom" parent="android:Widget.Button">
    <item name="android:textColor">#0000FF</item>
  </style>

  <!-- This is the custom textview styles for this application -->
  <style name="Widget.TextView.Custom" parent="android:Widget.TextView">
    <item name="android:textColor">#00FF00</item>
  </style>
</resources>
```

Дневная/Ночная темы

- Android Support 23.2 добавилось поддержка дневной ночной темы
- Можно сделать автоматическое переключение
- Можно использовать не только цвета
 - Используйте **-night** квалификатор ресурсов

Варианты работы

- **MODE_NIGHT_NO**. Всегда используется дневная тема (светлая).
- **MODE_NIGHT_YES**. Всегда используется ночная тема (темная).
- **MODE_NIGHT_AUTO**. Автоматическое изменение между светлой/темной, в зависимости от времени суток.
- **MODE_NIGHT_FOLLOW_SYSTEM** (по умолчанию). Это системный параметр, который является по существу MODE_NIGHT_NO но предполагается что следует за изменением темы в операционке.

Стили

- Ресурсы
- Лежат в `res/values/styles.xml`
- В стилях можно задать любой атрибут
- На ходу поменять стиль элемента нельзя
- Можно задать при создании
- Можно экспортировать из элемента
- Стиль можно наследовать

Пример определения стиля

res/values/styles.xml

```
<style name="LargeFont">
```

```
    <item name="android:textSize">40sp</item>
```

```
</style>
```

```
<style name="LargeRedFont" parent="@style/LargeFont">
```

```
    <item name="android:textColor">#C80000</item>
```

```
</style>
```

res/layouts/main_activity.xml

```
<TextView
```

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

```
    style="@style/LargeRedFont"
```

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

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

```
    android:text="@string/hello_world" />
```

Базовые темы и стили

Тема:

https://github.com/aosp-mirror/platform_frameworks_support/blob/master/v7/appcompat/res/values/themes_base.xml

Стили:

https://github.com/aosp-mirror/platform_frameworks_support/blob/master/v7/appcompat/res/values/styles_base.xml



- Нужны ли они
- Для чего они используются
- Какие преимущества дают разработчику
- Можно ли обойтись без них
- Почему их надо использовать
- Что такое черный ящик
- Как прикрутить и использовать

Для чего нужны



- Для проверки ошибок
- Чтобы проверить фичи
- Чтобы избежать неявных изменений в работе приложений
- Для того чтобы сократить цикл разработки
- «Чтобы было»
 - Хвалиться покрытием
 - Говорить всем что у нас TDD
 - Прочие раздражающие окружающих вещи





Существует два типа unit тестов

- Выполняются на локальной машине во время сборки. Это обычные JUnit тесты.
- Тесты запускаемы на Android устройстве или эмуляторе.

JUnit тесты



- Для того чтобы добавить их в проект, необходимо в директории src создать директорию test, внутри которой создать директорию java. Далее структура директорий будет согласно стандартным java путям, основываясь на имени пакета и имени класса.
- В build.gradle в секции dependencies необходимо прописать зависимость от JUnit

```
dependencies {  
    testCompile 'junit:junit:4.12'  
}
```

директории src/test/java/<package>/<testclassname>.java

JUnit тесты



- Метод, который необходимо протестировать должен начинаться с аннотации **@Test**
- Имена у методов могут быть любые. Главное - аннотации.
- Если необходимо произвести какую-то инициализацию класса до и после выполнения тесте, необходимо к ф-циям инициализации/деинициализации добавить аннотации **@Before** и **@After**, методы будут вызываться **каждый раз** перед вызовом метода **@Test**.
- Объект класса также будет создаваться заново перед вызовом каждого тестового метода.
- Если необходимо иметь какую-то переменную, которая будет инициализирована один раз, необходимо добавить статическую переменную и инициализировать ее в статическом методе обозначенного аннотацией **@BeforeClass**

JUnit пример



```
public class DipnetTest {
    private File inputdata;

    @Before
    public void init() {
        inputdata = new File("inputdata.txt");
    }
    @After
    public void cleanup() {
        inputdata.delete();
    }

    @Test
    public void selfTest() throws Exception {
        Assert.assertTrue(false); // test failed
    }
}
```


JUnit пример



```
public class DipnetTest {
    private static File inputdata;

    @BeforeClass
    public static void init() {
        DipnetTest.inputdata = new File("inputdata.txt");
    }

    @AfterClass
    public static void cleanup() {
        DipnetTest.inputdata.delete();
        DipnetTest.inputdata = null;
    }
}
```

Unit тесты на устройстве



- Тестируется код зависящий от Android SDK
- Выполняются они медленнее чем JUnit
- Необходимо наличие подключенного устройства/эмулятора

Настройки тестов



- Необходимо создать правильную структуру проекта.
- В build.gradle в секции dependencies необходимо прописать зависимость от Junit
- Необходимо в секции defaultConfig указать класс, с помощью которого будут запускаться тесты
- Android приложение, внутри которого будут запускаться тесты, **это отдельное приложение**, у него будет другой Application ID, отличный от нашего разрабатываемого приложения.
- Контекст тестового приложения будет отличным от контекста основного приложения.
- Настройки и файлы хранящиеся во внутренней памяти будут свои.
- Application ID для приложения, содержащего тесты, будет сгенерировано автоматически, или его можно указать принудительно

Пример build.gradle



```
dependencies {
    // android on-device testing-only dependencies
    androidTestCompile 'com.android.support.test:runner:0.5'
    androidTestCompile 'com.android.support.test:rules:0.5'
}

defaultConfig {
    applicationId "com.my.application"
    testApplicationId "com.my.application.test"
    testInstrumentationRunner
    "android.support.test.runner.AndroidJUnitRunner"

    minSdkVersion 15
    targetSdkVersion 24
}
```

Тесты



- Теперь можно писать сами юнит-тесты. Принцип такой же, как для обычных JUnit тестов, но с небольшим отличием - класс, содержащий тестовые методы, должен быть обозначен аннотацией **@RunWith(AndroidJUnit4.class)**

```
import org.junit.runner.RunWith;
```

```
@RunWith(AndroidJUnit4.class)
public class MyTestClass {
    @Test
    public void myTestMethod() {

    }
}
```



- Самый главный объект из Android SDK, который зачастую необходим - это Context. Получить к нему доступ можно получить через класс
- Контекст тестового приложения, будет отличным от контекста основного приложения
- Получить к нему доступ можно через статический метод `InstrumentationRegistry.getContext()`.
- Для доступа к контексту нашего основного приложения есть метод `InstrumentationRegistry.getTargetContext()`. Это может потребоваться для создания интента, который будет запускать активность или сервис, которые, находятся в другом application id

```
@RunWith(AndroidJUnit4.class)
public class MyTestClass {
    private Intent getDetailsActivityLaunchIntent() {
        Intent returnIntent =
            new Intent(InstrumentationRegistry.getTargetContext(),
                      DetailsActivity.class);
        return returnIntent;
    }
}
```

Тестирование Activity и Service



- Для тестирования активностей и сервисов в системе unit тестов на андроиде есть правила - Rules.
- За тестирование активности отвечает правило ActivityTestRule,
- За тестирование сервисов отвечает ServiceTestRule.
- Активности и сервисы создаются операционной системой путем отправки запроса на их запуск через Intent.

Пример тестирования Activity



```
import org.junit.Rule;
import org.junit.runner.RunWith;

@RunWith(AndroidJUnit4.class)
public class DetailsActivityTest {
    @Rule
    public ActivityTestRule<DetailsActivity> mDetailsActivityRule = new
    ActivityTestRule<DetailsActivity>(DetailsActivity.class, false, false);
}
```

Три параметра, передаваемых в конструктор это:

1. Имя класса активности
2. Булевый параметр который позволяет активности реагировать или не реагировать на нажатии пальцем (touch mode)
3. Булевый параметр обозначающий, стоит ли каждый раз пересоздавать активность перед вызовом каждого метода обозначенного аннотацией @Test



- Тесты запускаются в потоке отличном от потока ввода.
- Как мы помним, обращаться к эл-там пользовательского интерфейса можно только из потока ввода.
- Для решения этой проблемы у класса `ActivityTestRule` есть метод `runOnUiThread`, который получает на вход `Runnable` объект.

Пример



```
@RunWith(AndroidJUnit4.class)
public class DetailsActivityTest {
    @Rule
    public ActivityTestRule<DetailsActivity> mDetailsActivityRule = new
ActivityTestRule<DetailsActivity>(DetailsActivity.class, false, false);
    private static final int TEST_APP_ID = 67;
    private Intent getDetailsActivityLaunchIntent() {
        Intent returnIntent = new Intent(InstrumentationRegistry.getTargetContext(),
DetailsActivity.class);
        return returnIntent;
    }
    @Test
    public void testActivityUnauthorized() throws Throwable {
        final DetailsActivity detailsActivity =
mDetailsActivityRule.launchActivity(getDetailsActivityLaunchIntent());
        try {
            mDetailsActivityRule.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    TextView taskId = (TextView)
detailsActivity.findViewById(R.id.tv_appId);
                    Assert.assertEquals(String.valueOf(TEST_APP_ID),
taskId.getText().toString());
                }
            });
        } catch (InterruptedException e) {
            Assert.assertTrue("Interrupted exception", false);
        }
    }
}
```

Тестирование Service



- За тестирование сервисов, отвечает правило ServiceTestRule.
- С помощью него можно запустить сервис и связаться с ним, например через интерфейс описанный в AIDL файле.
- В методе setUp, обозначенного аннотацией @Before осуществляет запуск сервиса и связывание с ним через интерфейс.

Пример



```
@RunWith(AndroidJUnit4.class)
public class InAppBillingTest {
    public static final String PACKAGE_NAME = "com.my.Advanced_Systemcare";
    private static final String INAPP = "inapp";
    private static final String SUBS = "subs";
    @Rule
    public ServiceTestRule mInAppBillingTestRule;
    private IInAppBillingService mBillingService;
    @Before
    public void setUp() throws Exception {
        mInAppBillingTestRule = new ServiceTestRule();
        Intent billingIntent = new
Intent("com.my.application.billing.InAppBillingService.BIND");
        billingIntent.setPackage("com.my.application");
        IBinder billingBinder = mInAppBillingTestRule.bindService(billingIntent);
        mBillingService = IInAppBillingService.Stub.asInterface(billingBinder);
    }
    @Test
    public void testIsBillingSupported() throws Exception {
        Assert.assertTrue(mBillingService.isBillingSupported(1, PACKAGE_NAME,
INAPP) == RESULT_OK);
        Assert.assertTrue(mBillingService.isBillingSupported(1, PACKAGE_NAME,
SUBS) == RESULT_OK);
    }
}
```

Важно!



- ВАЖНО - в тестовом классе может быть только одно тестовое правило. Это означает, что нельзя тестировать в одном классе два сервиса или две активности.
- Запомните: Один класс - одно правило, один сервис или активность.
- Если вы попытаетесь создать еще один сервис, будет возвращен объект на предыдущий сервис и вы получите `ClassCastException`.



Библиотека для написания тестов Android UI

- Быстро запускается
- Быстро разворачивается
- Быстро прикручивается
- Используется для тестирования UI как черного ящика

```
@Test
public void greeterSaysHello() {
    onView(withId(R.id.name_field))
        .perform(typeText("Steve"));
    onView(withId(R.id.greet_button))
        .perform(click());
    onView(withText("Hello Steve!"))
        .check(matches(isDisplayed()));
}
```

Установка



- Прописать зависимости в gradle

```
androidTestCompile 'com.android.support.test.espresso:espresso-core:2.2.2'  
androidTestCompile 'com.android.support.test:runner:0.5'
```

- Прописать запуск в gradle в разделе android.defaultConfig

```
testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
```

- Создать тест

```
@RunWith(AndroidJUnit4.class)  
@LargeTest  
public class HelloWorldEspressoTest {
```

- Запустить его

Что можно делать



- Искать вью
 - По ID
 - По свойству
- Эмулировать
 - Клик
 - Ввод
 - Скролл
- Проверять всякое
 - Текст, детей и пр
 - Наличие вью
 - Интенты и их содержимое
 - WebView

Espresso cheat sheet



Ссылка на Espresso cheat sheet

<https://goo.gl/3KJdAR>



- Самая свежая документация и примеры использования библиотеки поддержки тестов лежит

тут: <https://google.github.io/android-testing-support-library/>

UI Automator



- Более тяжелая, чем Espresso
- Требуется более высокий minSdk (18)
- Более “black box”
- Больше возможностей по работе с устройством
- Возможно скоро будет deprecated

```
dependencies {  
    ...  
    androidTestCompile 'com.android.support.test:uiautomator-v18:2.1.1'  
}
```

Для проверки можно использовать uiautomatorviewer из SDK



ТЕХНОТРЕК

**Спасибо за
внимание!**

Кирилл Филимонов Kirill.Filimonov@gmail.com

Юрий Береза – ybereza@gmail.com