Лекция №6
«Интерфейс пользователя»

Москва 2019

Атрибуты activity

- Можно activity заставить находиться только в портретной ориентации:

  android:screenOrientation="portrait"

- И в горизонтальной:

  android:screenOrientation="landscape"

- Предотвращение уничтожения при смене положения:

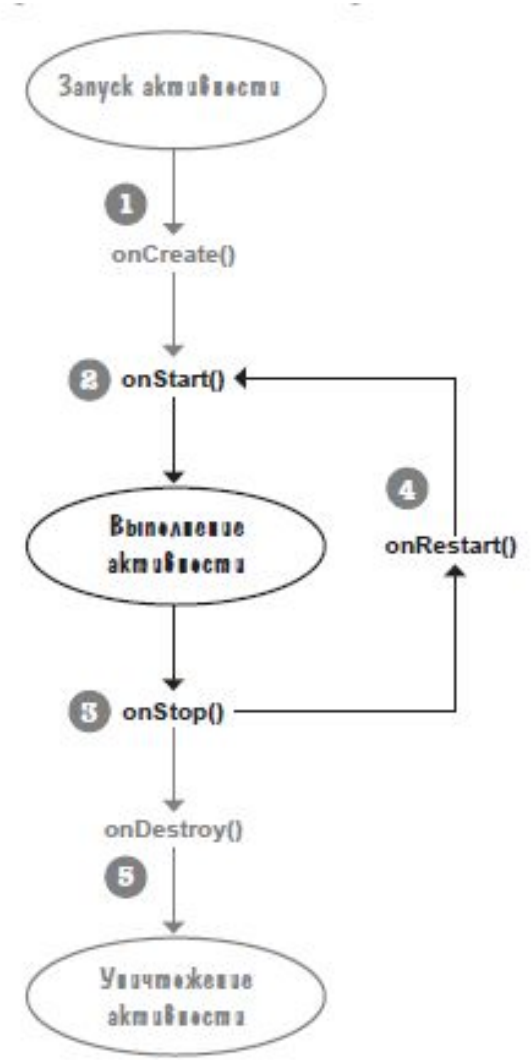
  android:configChanges="orientation"

- Список атрибутов см. здесь:
  http://developer.android.com/guide/topics/manifest/activity-element.html

Activity всегда находится в одном из трех состояний:

- Окно находится на вершине стека окон, и поэтому видно нам - в этом случае мы говорим, что acitivty запущено.
- Поверх окна появилось какое-то диалоговое окно, при этом первое все равно видно нам, но оно потеряло фокус. В этом случае мы говорим, что activity встало на паузу.
- Если activity полностью перекрыто другим activity, оно останавливается. В этом случае система также может уничтожить остановившееся activity, если ей будет не хватать памяти.

Жизненный цикл активностей



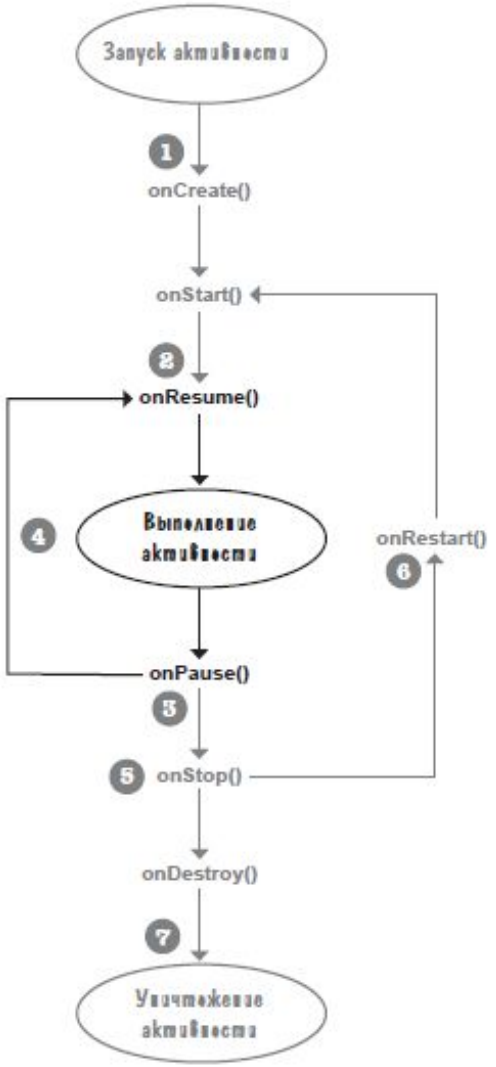
Отслеживать эти события и реагировать на них, закрывать базу данных, сохранять файлы.

Состояние меняются автоматически в зависимости от действия пользователей.

OnCreate – приложение сверстано, но не доступно для пользователя

OnStart – активность создано но пока не готово для запуска

Жизненный цикл активностей



1. **Активность запускается, выполняются ее методы onCreate() и onStart().**
   В этой точке активность видна, но еще не обладает фокусом.

2. **Метод onResume() вызывается после метода onStart(). Он выполняется тогда, когда активность собирается перейти на передний план.**
   После выполнения метода `onResume()` активность обладает фокусом, а пользователь может взаимодействовать с ней.

3. **Метод onPause() выполняется тогда, когда активность перестает находиться на переднем плане.**
   После выполнения метода `onPause()` активность остается видимой, но не обладает фокусом.

4. **Если активность снова возвращается на передний план, вызывается метод onResume().**
   Активность, которая многократно теряет и получает фокус, может проходить этот цикл несколько раз.

5. **Если активность перестает быть видимой пользователю, вызывается метод onStop().**
   После выполнения метода `onStop()` активность не видна пользователю.

6. **Если активность снова станет видимой, вызывается метод onRestart(), за которым следуют onStart() и onResume().**
   Активность может проходить через этот цикл многократно.

7. **Наконец, активность уничтожается.**
   При переходе от выполнения к уничтожению метод `onPause()` вызывается до того, как активность будет уничтожена. Также обычно при этом

Ресурсы Android приложения

- **animator/**: xml-файлы, определяющие анимацию свойств
- **anim/**: xml-файлы, определяющие tween-анимацию
- **color/**: xml-файлы, определяющие список цветов
- **drawable/**: Графические файлы (*.png*, *.jpg*, *.gif*)
- **mipmap/**: Графические файлы, используемые для иконок приложения под различные разрешения экранов
- **layout/**: xml-файлы, определяющие пользовательский интерфейс
- **menu/**: xml-файлы, определяющие меню приложения
- **raw/**: различные файлы, которые сохраняются в исходном виде
- **values/**: xml-файлы, которые содержат различные используемые в приложении значения, например, ресурсы строк
- **xml/**: Произвольные xml-файлы

Структура папки res

```
/res/values/strings.xml
           /colors.xml
          /dimens.xml
          /attrs.xml
          /styles.xml
   /drawable/*.png
              /*.jpg
              /*.gif
              /*.9.png
    /anim/*.xml
    /layout/*.xml
    /raw/*.*
    /xml/*.xml
/assets/*.*/*.*
```

Значения

Values

Values:

- strings - строковые константы
- colors - цветовые константы
- dimens - константы размеров
- attrs - атрибуты
- styles - константы стилей

Структура папки values

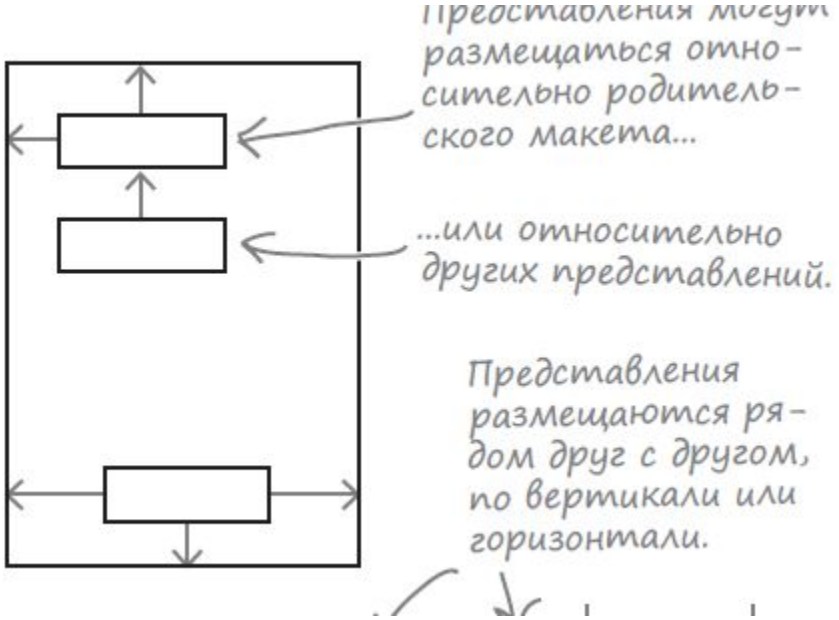
```
/res/values/strings.xml
           /colors.xml
          /dimens.xml
          /attrs.xml
          /styles.xml
```

Все что добавляется в ресурсы, получает автоматический идентификатор.
В папке layout храниться верстка активностей.
Все отображения являются потомками класса View.

Относительный макет

Относительный макет (RelativeLayout)

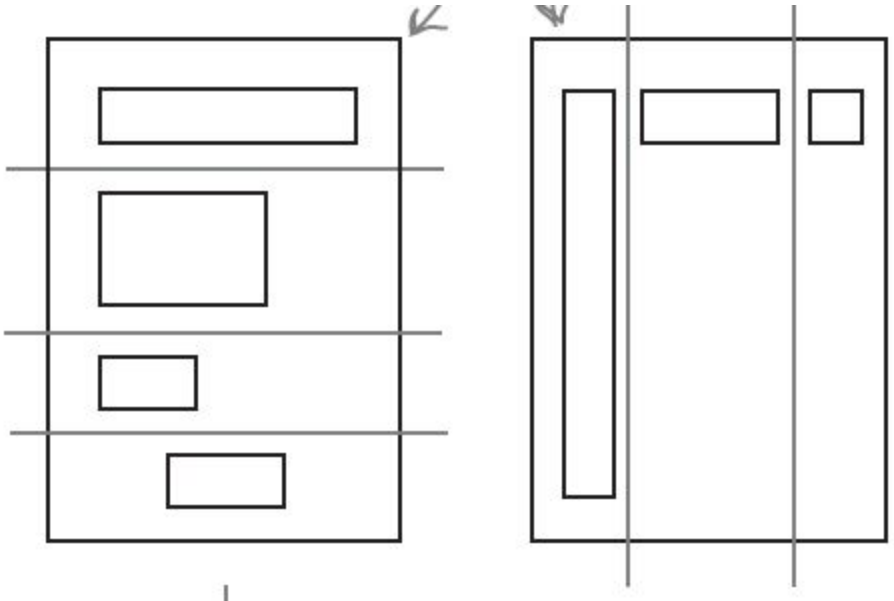
В **относительном макете** входящие в него представления размещаются в относительных позициях. Позиция каждого представления определяется относительно других представлений в макете или относительно его родительского макета. Например, надпись можно разместить относительно верхнего края родительского макета, раскрывающийся список разместить под текстовым представлением, а кнопку — относительно нижнего края родительского макета.



Парадигма MVC (Yii) , отображение дистанцировано от контроллера и данных

Линейный макет (LinearLayout)

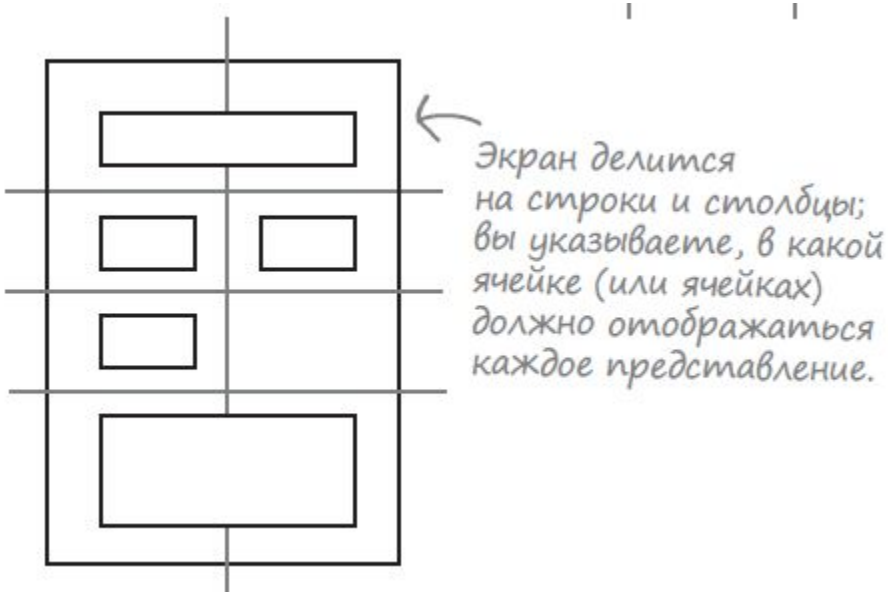
В **линейном макете** представления размещаются рядом друг с другом по вертикали или горизонтали. Если используется вертикальное размещение, представления отображаются в один столбец. В варианте с горизонтальным размещением представления выводятся в одну строку.



Табличный макет

Табличный макет (GridLayout)

В табличном макете экран делится на строки и столбцы, на пересечении которых находятся ячейки. Вы указываете, сколько столбцов должно входить в макет, где должны отображаться представления, и сколько строк или столбцов они должны занимать.



Linear Layout выравнивание по вертикали бессмысленно

Сообщает Android, что вы используете относительный макет.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    ...>
    ...
</RelativeLayout>
```

Атрибуты layout_width и layout_height задают размер макета.

Здесь также могут быть другие атрибуты.

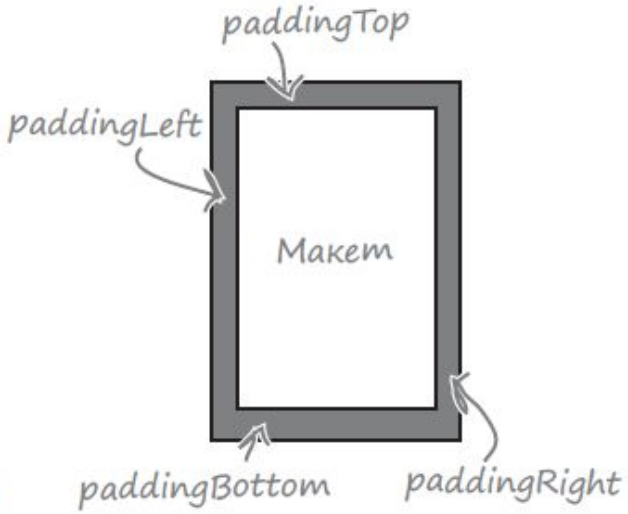
Отступы

Отступы

Если вы хотите, чтобы макет окружало некоторое пустое пространство, воспользуйтесь атрибутами `padding`. Эти атрибуты сообщают Android, какое расстояние должно отделять стороны макета от родителя. Следующий фрагмент приказывает Android добавить ко всем сторонам макета отступы величиной 16dp:

```
<RelativeLayout ...
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp">
    ...
</RelativeLayout>
```

Добавить отступы величиной 16dp.



Атрибуты `android:padding*` не являются обязательными, но они

Отступы

layout - ресурс разметки

Еще один из важных видов ресурсов - ресурсы разметки, которые отвечают за внешний вид приложения.

Данные ресурсы представлены в формате XML.

Ресурс разметки формы (layout resource) - это ключевой тип ресурсов, применяемый при программировании пользовательских интерфейсов в Android.

Стандартные типы разметки:

- FrameLayout
- LinearLayout
- TableLayout
- RelativeLayout
- GridLayout

Разметка LinearLayout

Не больше трех LinearLayout использвоать вложенные
Альтернатива табличной верстки.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="List1"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

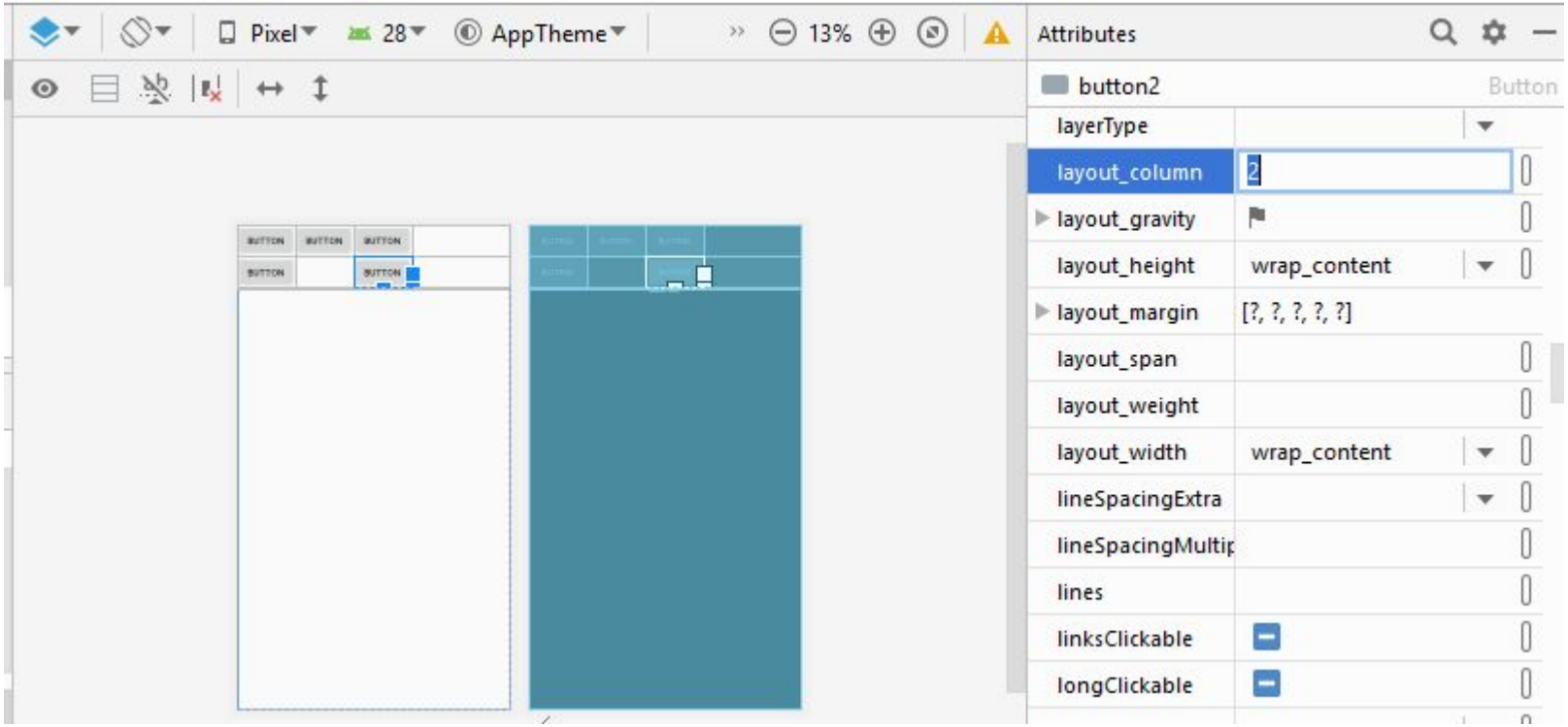
MAIN – главная активность,
LAUNCHER – активность которая запускается.

Разметка LinearLayout

Работа с Logcat

- import android.util.Log;
- Log.i(MY_TAG, myMessage);

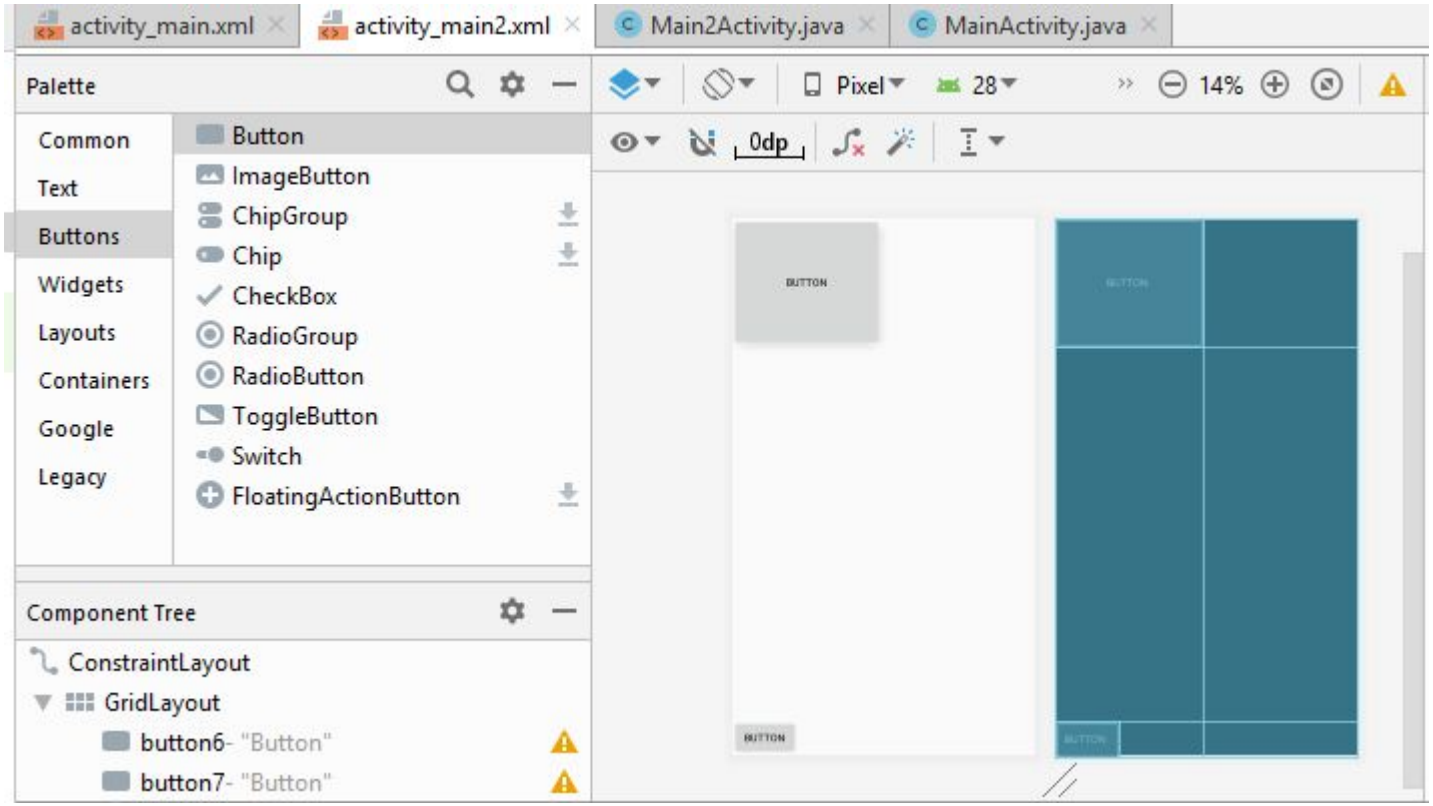
Табличный дизайн



Возможно программное добавление компонентов в табличный дизайн

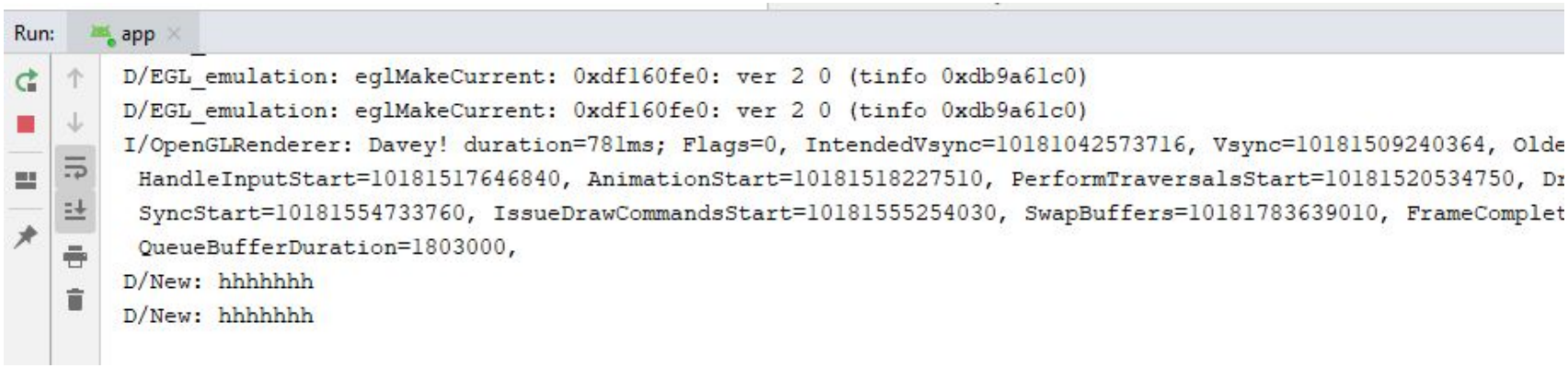
Grid Layout

Один из наиболее удобных макетов. Автоматически растягиваются компоненты

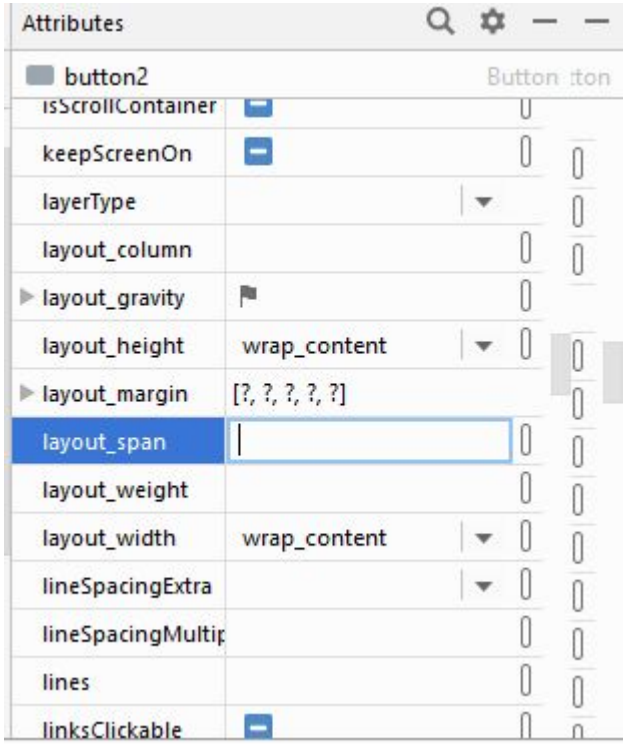


Вывод Log сообщений

```
    public void onButtonClick(View view) {
        Log.d( tag: "New", msg: "hhhhhhh");

    }
}
```



Табличный дизайн



Объединение ячеек

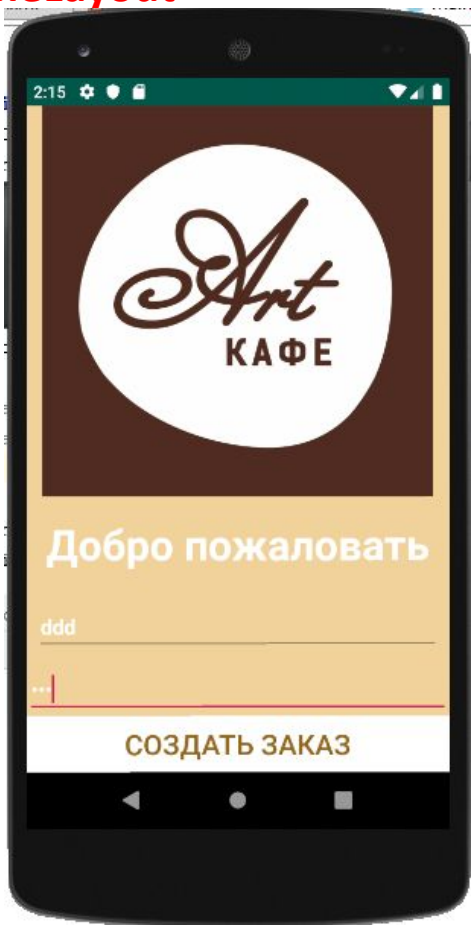
FrameLayout

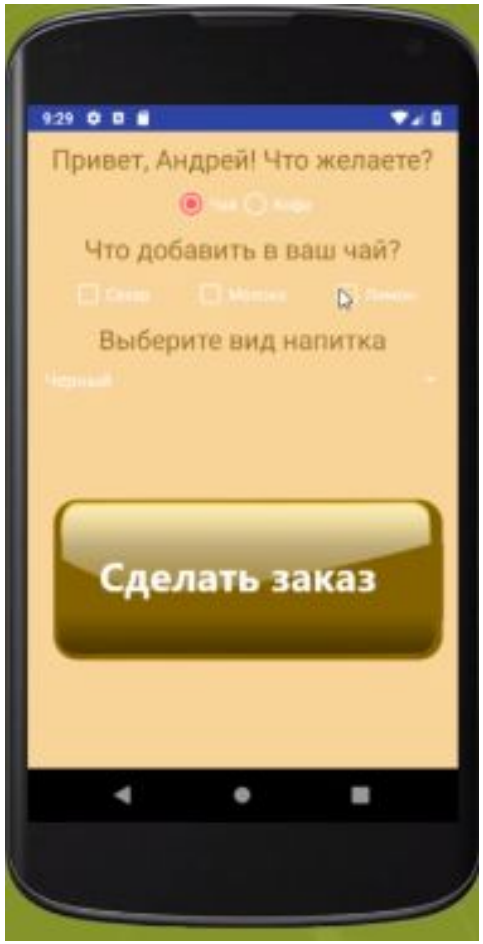
Для мультипликации, частой смены картинок используется FrameLayout

Контейнер где нет выравнивания.

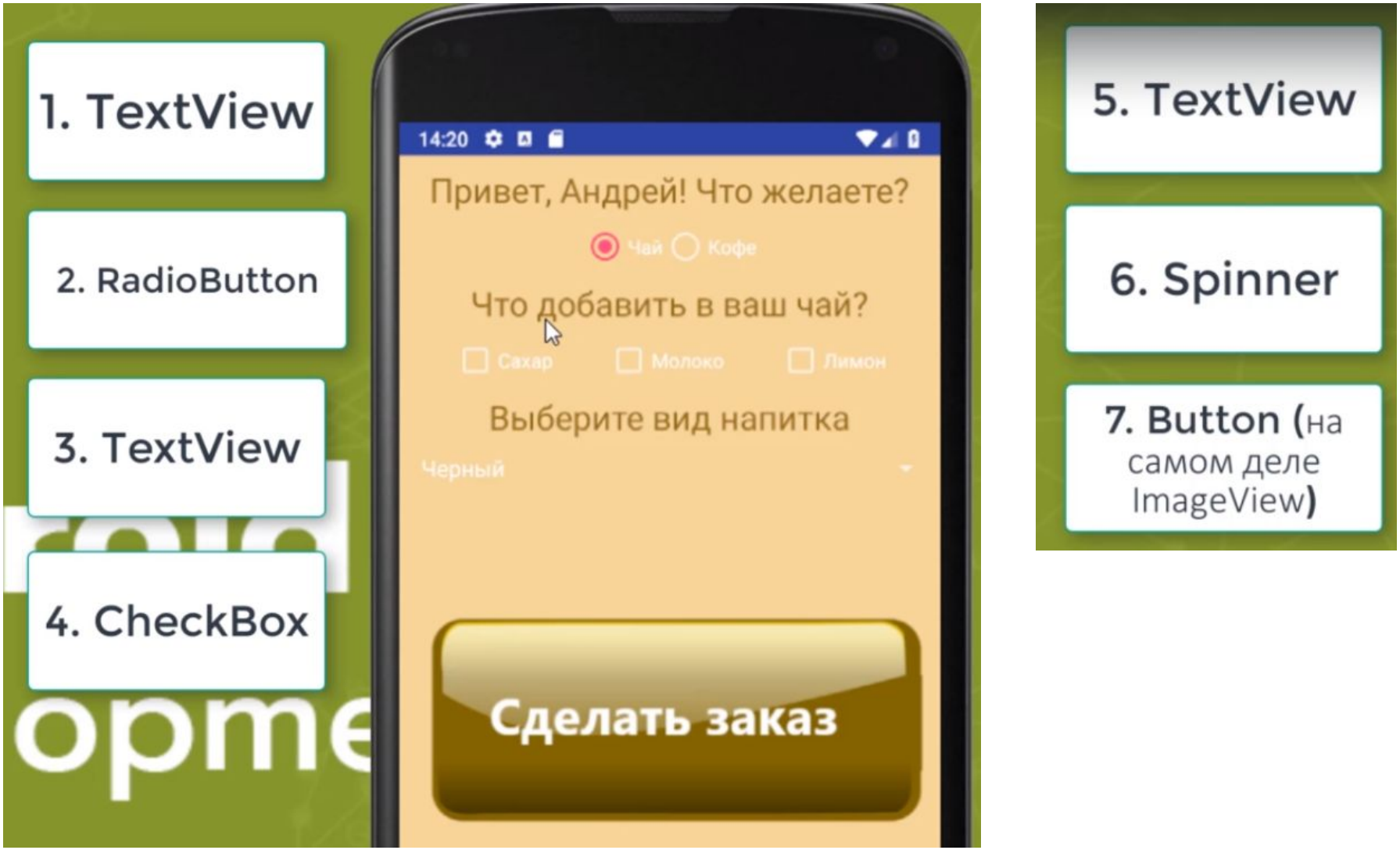
CafeOrder

Для мультипликации, частой смены картинок используется FrameLayout

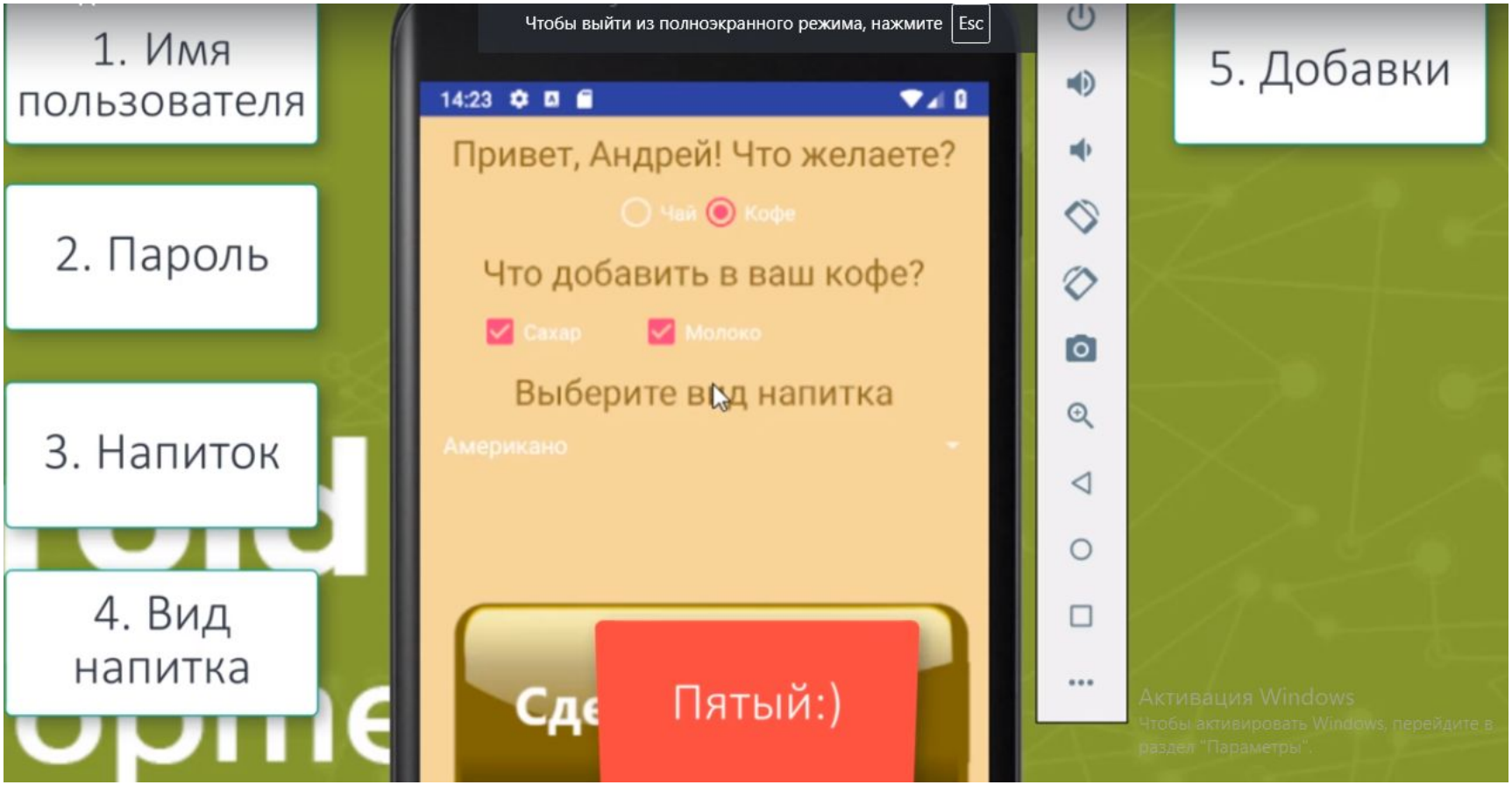




CafeOrder


1. TextView
2. RadioButton
3. TextView
4. CheckBox
14:20
Привет, Андрей! Что желаете?
Чай
Кофе
Что добавить в ваш чай?
Сахар
Молоко
Лимон
Выберите вид напитка
Черный
Сделать заказ
5. TextView
6. Spinner
7. Button (на самом деле ImageView)

CafeOrder


1. Имя пользователя
2. Пароль
3. Напиток
4. Вид напитка
Чтобы выйти из полноэкранного режима, нажмите Esc
14:23
Привет, Андрей! Что желаете?
Чай
Кофе
Что добавить в ваш кофе?
Сахар
Молоко
Выберите вид напитка
Американо
Пятый:)
5. Добавки
Активация Windows
Чтобы активировать Windows, перейдите в раздел "Параметры".

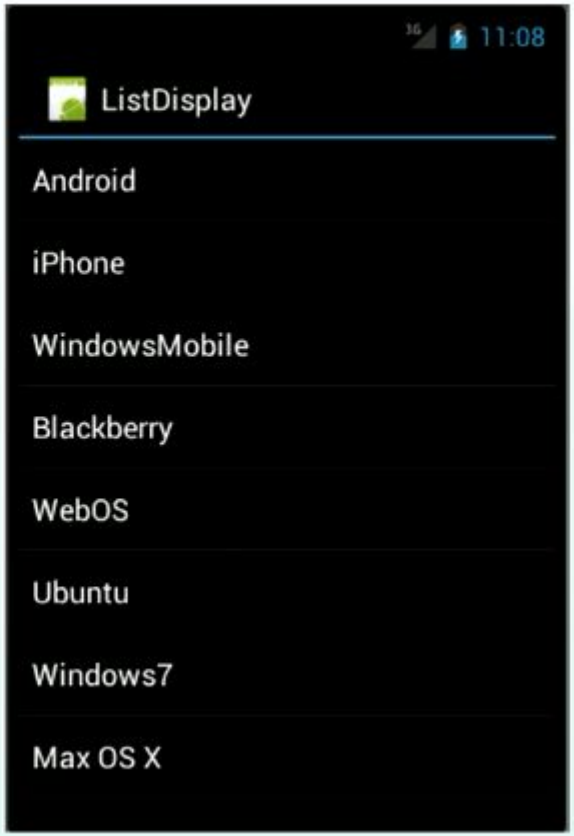
ListView - список

Списки — это вещь обычная в современных мобильных устройствах.

Мы можем перемещаться по списку и выбирать нужный нам элемент,

чтобы открыть что-то. Так и у Андроида он имеется.

Каждый список использует адаптер по умолчанию, его можно менять, сделав список кастомизированым.



BaseAdapter

- **ArrayAdapter<T>** - предназначен для работы с ListView. Данные представлены в виде массива, которые размещаются в отдельных элементах TextView
- **SpinnerAdapter** - адаптер для связки данных с элементом Spinner. Это тоже интерфейс, как **ListAdapter** и работает по схожему принципу
- **SimpleAdapter** - адаптер, позволяющий заполнить данными список более сложной структуры, например, два текста в одной строке списка.
- **CursorAdapter** - предназначен для работы с **ListView**, предоставляет данные для списка через курсор, который должен иметь колонку с именем "_id"
- **ResourceCursorAdapter** - этот адаптер дополняет **CursorAdapter** и может создавать виды из ресурсов
- **HeaderViewListAdapter** - расширенный вариант ListAdapter, когда ListView имеет заголовки.
- **WrapperListAdapter** - еще один адаптер для списков.

Признаки «хорошего» кода

Хороший код – это самодокументируемый код, плохой код к которому требуется инструкция.

Как только к коду прилагается инструкция – это плохо.

Эффект отложенной ошибки, если ошибка происходит в одном месте, но узнаем мы о ней в другом месте сильно после. С этим надо бороться, чтобы ошибки видели в момент возникновения. С этим могут помочь свойства.

Повторений кода поменьше. В том месте где есть ошибка мы сигнализируем об этом и предоставляем возможность программисту возможность понять где ош

```
class Pixel
{
    double r;
    double g;
    double b;
    ссылок: 3
    public double Check(double value)
    {
        if (value < 0 || value > 1) throw new ArgumentException();
        return value;
    }
    ссылок: 0
    public double R
    {
        get { return r; }
        set
        {
            r = Check(value);
        }
    }
```

Признаки «хорошего» кода

Можно замаскировать ошибку. Но это плохая практика. Никогда не узнаю, что у меня ошибка в алгоритме. Лишаете возможность увидеть ошибку на другом уровне на уровне алгоритма.

Хорошо защищенный класс, который страхует нас от возможных ошибок.

Массив объектов – большие накладные расходы

```
ссылок: 0
class Pixel2
{
    double r;
    double g;
    double b;
    ссылок: 3
    public double Check(double value)
    {
        if (value < 0) value = 0;
        if (value > 1) value = 1;
        return value;
    }
    ссылок: 0
```

NUnit

```
int result = SerialPortParser.ParsePort("COM1");
Assert.That(result, Is.EqualTo(1));
// older style of Assets in NUnit
// Assert.AreEqual(1, result);
```

В Юнит тестах у нас всегда есть фактически результат и ожидаемый результат ожидаемый результат обозначает как мы определяем правильное поведение системы

```
Is.EqualTo(1)
```

Ожидаемый результат

Ожидаемый результат - как мы определяем правильное поведение системы
Фактический результат обозначает как система ведёт себя в реальности
Наша задача сравнить ожидаемое и фактическое поведение

Unit Testing Considerations

1. Testers are usually responsible for writing or performing manual tests
2. Unit tests are much faster than manual tests,
   so they provide the feedback very quickly
3. Unit Tests create a safety net, enabling fearless introducing of changes
4. Don't write unit tests for pet projects which are not much bigger than "Hello World" app
5. If you don't write unit tests, you are not a professional!

Тестировщики не делают unit тестов. Благодаря unit тестам вносить изменения намного проще
Unit тест - это функция, который вызывает другую функцию и затем проверяет правильность конечного результата.
Тестируемая система - это единица тестируемая Unit тестом.

Ответственность программиста

Разработчики несут ответственность за свой код. Тестировщики не делают unit тестов. Благодаря unit тестам вносить изменения намного проще.

Клятва программиста:

1. Я не напишу вредоносный код
2. Созданный мною код всегда будет моей лучшей работой
3. Я не допущу сознательной неисправности в ходе твоего поведения или структуры
4. Я буду часто выпускать небольшие версии, чтобы не мешать прогрессу остальных.
5. Я буду неустанно совершенствовать свои творения при каждой возможности .
6. Я всегда буду следить за тем , чтобы другие могли прикрыть меня а я мог прикрыть их.
7. Я не сделаю ничего что приведет к ухудшению я буду прилагать все усилия чтобы сохранять наибольшую продуктивность как собственную так и других
8. Я не буду давать обещание без уверенности в том, что смогу их выполнять
9. Я никогда не перестану учиться и совершенствовать свое ремесло

Ограничения NUbit

Major logical constraints:

- Is
- Has
- Does

Есть три главных логических ограничения. Эти ограничения представлены соответственно названными классами в нём ограничения многослойная модель на следующем уровне у нас могут быть но необязательно другие логические ограничения

2nd level constraints:

- All
- Not
- Some

"All" constraint:

```
string[] array = new string[] { "abc", "bad", "dba" };
Assert.That(array, Is.All.Contains("b"));
```

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
Assert.That(array, Has.All.GreaterThan(0));
```

Второе утверждение здесь проверяет все ли элементы массива больше 0 в данном случае выражения ограничения будет вычислено как True поскольку все значения больше нуля

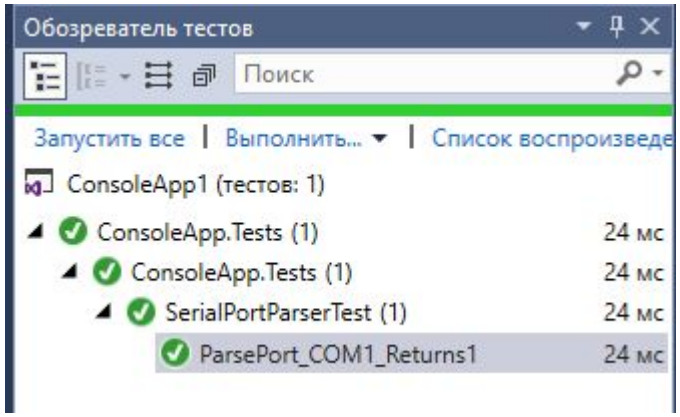
NUnit

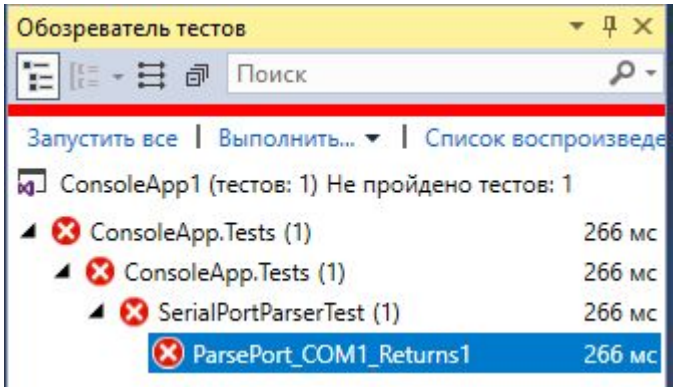
```
[TestFixture]
ссылок: 0
public class SerialPortParserTest
{
    [Test]
    ссылок: 0
    public void ParsePort_COM1_Returns1()
    {
        int result = SerialPortParser.ParsePort("COM1");
        Assert.That(result, Is.EqualTo(1));
        // older style of Assets in NUnit
       // Assert.AreEqual(1, result);

    }
}
```

Пометим классы и методы тестирования специальными атрибутами чтобы их можно было обнаружить
[Атрибут]

NUnit

```
namespace ConsoleApp.Tests
{
    [TestFixture]
    ссылок: 0
    public class SerialPortParserTest
    {
        [Test]
        ссылок: 0
        public void ParsePort_COM1_Returns1()
        {
            int result = SerialPortParser.ParsePort("COM1");
            Assert.That(result, Is.EqualTo(1));
            // older style of Assets in NUnit
            // Assert.AreEqual(1, result);

        }
    }
```





Возможные правила именования

“void ShouldAddTwoNumbers()”

“void Sum_ShouldAddTwoNumbers”

Naming pattern “UnitUnderTest_Scenario_ExpectedOutcome”.

For example, “void ParsePort_COM1_Returns1()”.

Имя функции должно состоять из трёх частей разделенных подчёркиванием.
Первая часть: просто имя тестируемой функций или что-то более абстрактное,
вторая: аргумент переданных тестируемые функцию или более абстрактной сценарий вроде invalid Stream Format, в целом отражает условия в которых будет тестироваться программная единица.
третья часть: ожидаемое поведение в зависимости от результатов тестирования функции. Например, возвращает значение или меняет состояние (SetNumberToZero) или вызывает другой метод

Отладка в Unit

```
namespace ConsoleApp.Tests
{
    [TestFixture]
    ссылок: 0
    public class SerialPortParserTest
    {
        [Test]
        ссылок: 0
        public void ParsePort_COM1_Returns1()
        {
            int result = SerialPortParser.ParsePort("COM1");
            Assert.That(result, Is.EqualTo(1));
            // older style of Assets in NUnit
            // Assert.AreEqual(1, result);
```

Код, который пишется для того, чтобы прошел некоторый тест, по определению поддается тестированию. Более того, создается сильная мотивация для разбиения программы на модули, чтобы каждый мо дуль можно было тестировать независимо.

Ограничения NOT

"Not" constraint:

```
Assert.That(array, Is.Not.Length.EqualTo(4));
Assert.That(@"C:\tmp.txt", Does.Not.Exist);
Assert.That(42, Is.Not.Null);
Assert.That(42, Is.Not.True);
Assert.That(42, Is.Not.False);
Assert.That(2.5, Is.Not.NaN);
Assert.That(2 + 2, Is.Not.EqualTo(3));
```

"Does" constraint:

```
string phrase = "Are you OK?";
Assert.That(phrase, Does.EndWith("!"));
Assert.That(phrase, Does.Not.EndWith("?"));
Assert.That(phrase, Does.Not.Contain("goodbye"));
```

Ограничение Does чаще всего используются когда вы пишете утверждение относительно строк.

Утверждения Has

"Has" constraint:

```
object[] strings = new object[] { "abc", "bad", "cab", "bad", "dad" };
Assert.That(strings, Has.Some.StartsWith("ba"));
```

Ограничение Has хотя бы одна строка содержит возвращает True

```
object[] doubles = new object[] { 0.99, 2.1, 3.0, 4.05 };
Assert.That(doubles, Has.Some.EqualTo(1.0).Within(.05));
```

Содержит ли значение 1,0 плюс, минус 0,05. Выражение будет вычислено как True

"Or & And" compound constraints:

```
Assert.That(5, Is.LessThan(1).Or.GreaterThan(10));
Assert.That(5, Is.LessThan(10).And.GreaterThan(1));
```

Утверждения Has

"Has" constraint:

```
object[] strings = new object[] { "abc", "bad", "cab", "bad", "dad" };
Assert.That(strings, Has.Some.StartsWith("ba"));
```

Ограничение Has хотя бы одна строка содержит возвращает True

```
object[] doubles = new object[] { 0.99, 2.1, 3.0, 4.05 };
Assert.That(doubles, Has.Some.EqualTo(1.0).Within(.05));
```

Содержит ли значение 1,0 плюс, минус 0,05. Выражение будет вычислено как True

"Or & And" compound constraints:

```
Assert.That(5, Is.LessThan(1).Or.GreaterThan(10));
Assert.That(5, Is.LessThan(10).And.GreaterThan(1));
```

Полиморфизм

полиморфизм – это различная реализация однотипных действий. Классическая фраза, которая коротко объясняет полиморфизм – «Один интерфейс, множество реализаций»

Приведу примеры из жизни. В автомобилях есть рулевое колесо. Это колесо является интерфейсом между водителем и автомобилем, который позволяет поворачивать автомобиль.

Механическая реализация руля у автомобилей может быть разная, но при этом результат получается одинаковым – колесо вправо – автомобиль вправо, и наоборот.

Полиморфизм

Полиморфизм позволяет :

-писать более абстрактные, расширяемые программы,

-- один и тот же код используется для объектов разных классов,

- улучшается читабельность кода.

- избавляет разработчика от написания, чтения и отладки множества if-else/switch-case конструкций.

Пример программы Copy

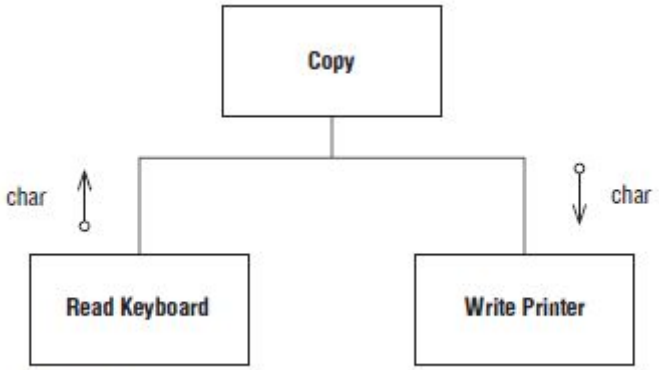


Рис. 7.1. Структурная диаграмма программы Copy

Листинг 7.1. Программа Copy

```
public class Copier
{
  public static void Copy()
  {
    int c;
    while((c=Keyboard.Read()) != -1)
      Printer.Write(c);
  }
}
```

Доработка №1 программы Copy

```
public class Copier
{
//не забудьте сбросить этот флаг
public static bool ptFlag = false;
public static void Copy()
{
int c;
while((c=(ptFlag ? Flash() .Read(): Keyboard.Read())) != -1)
Printer.Write(c);
}
}
```

Если программа хочет вызвать Copy для чтения с флешки, то долж-
на сначала установить для переменной ptFlag значение true.
А после завершения программы Copy нужно сбросить этот флаг, иначе следующий обратившийся будет читать с флешки, а не с клавиатуры.
Чтобы напомнить программистам о необходимости сбрасывать флаг, вы включили комментарий.

```
<усл. Выр> ? <блок№1>:<блок №2>
If (усл. Выр)
   {блок№1}
else
   {блок №2}
```

Доработка №1 программы Copy

```
public class Copier
{
//не забудьте сбросить этот флаг
public static bool ptFlag = false;
public static void Copy()
{
int c;
while((c=(ptFlag ? Flash() .Read(): Keyboard.Read())) != -1)
Printer.Write(c);
}
}
```

Если программа хочет вызвать Copy для чтения с флешки, то долж-
на сначала установить для переменной ptFlag значение true.
А после завершения программы Copy нужно сбросить этот флаг, иначе следующий обратившийся будет читать с флешки, а не с клавиатуры.
Чтобы напомнить программистам о необходимости сбрасывать флаг, вы включили комментарий.

```
<усл. Выр> ? <блок№1>:<блок №2>
If (усл. Выр)
   {блок№1}
else
   {блок №2}
```

Модификация программы Copy

```
public class Copier
{
//не забудьте сбросить эти флаги
public static bool ptFlag = false;
public static bool ptFlag2 = false;

public static void Copy()
{
…
}
}
```

В данном случае команда следовала принципу открытости/закрыто-сти. Он требует проектировать модули так, чтобы их можно было расширять без модификации. Столкнувшись с таким отсутствием эластичности, гибкие разработчики сразу поняли, что направление зависимости от модуля Copy к устройству ввода следует инвертировать, воспользовавшись принципом инверсии зависимости

Модификация программы Copy

Листинг 7.4. Вторая модификация программы Copy

```
public interface Reader
{
  int Read();
}

public class KeyboardReader : Reader
{
  public int Read() {return Keyboard.Read();}
}

public class Copier
{
  public static Reader reader = new KeyboardReader();
  public static void Copy()
  {
    int c;
    while((c=(reader.Read())) != -1)
      Printer.Write(c);
  }
}
```