ТЕХНОТРЕК

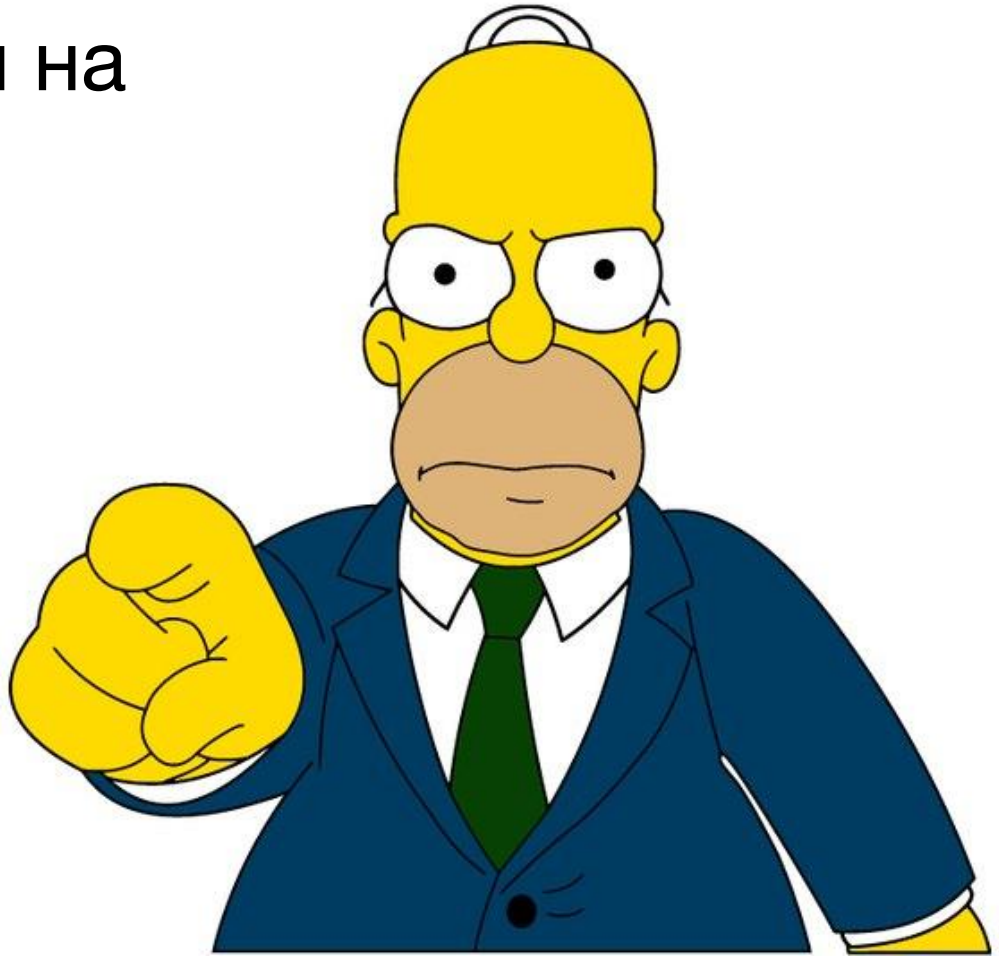
Занятие №5

Разработка приложений на Android

Кирилл Филимонов
Юрий Береза

Напоминание

А ты отметился о присутствии на занятии?

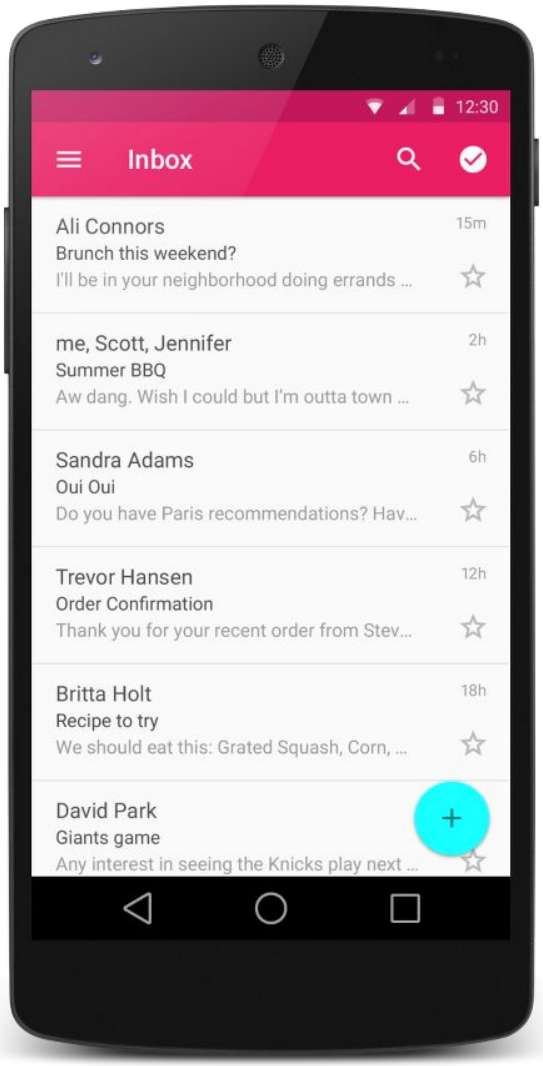


Agenda

- Adapter Views (Контейнеры)
- RecyclerView
- Сервисы
- Job Scheduler

Контейнеры

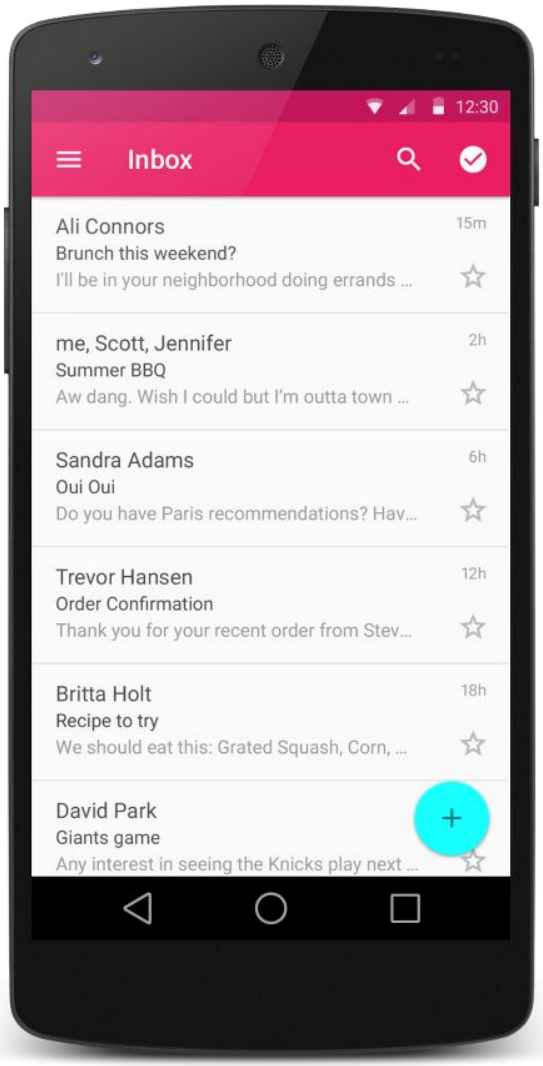
- GridView
- ListView
- Spinners



Контейнеры

View на примере ListView

- Список элементов
- Может быть очень большим



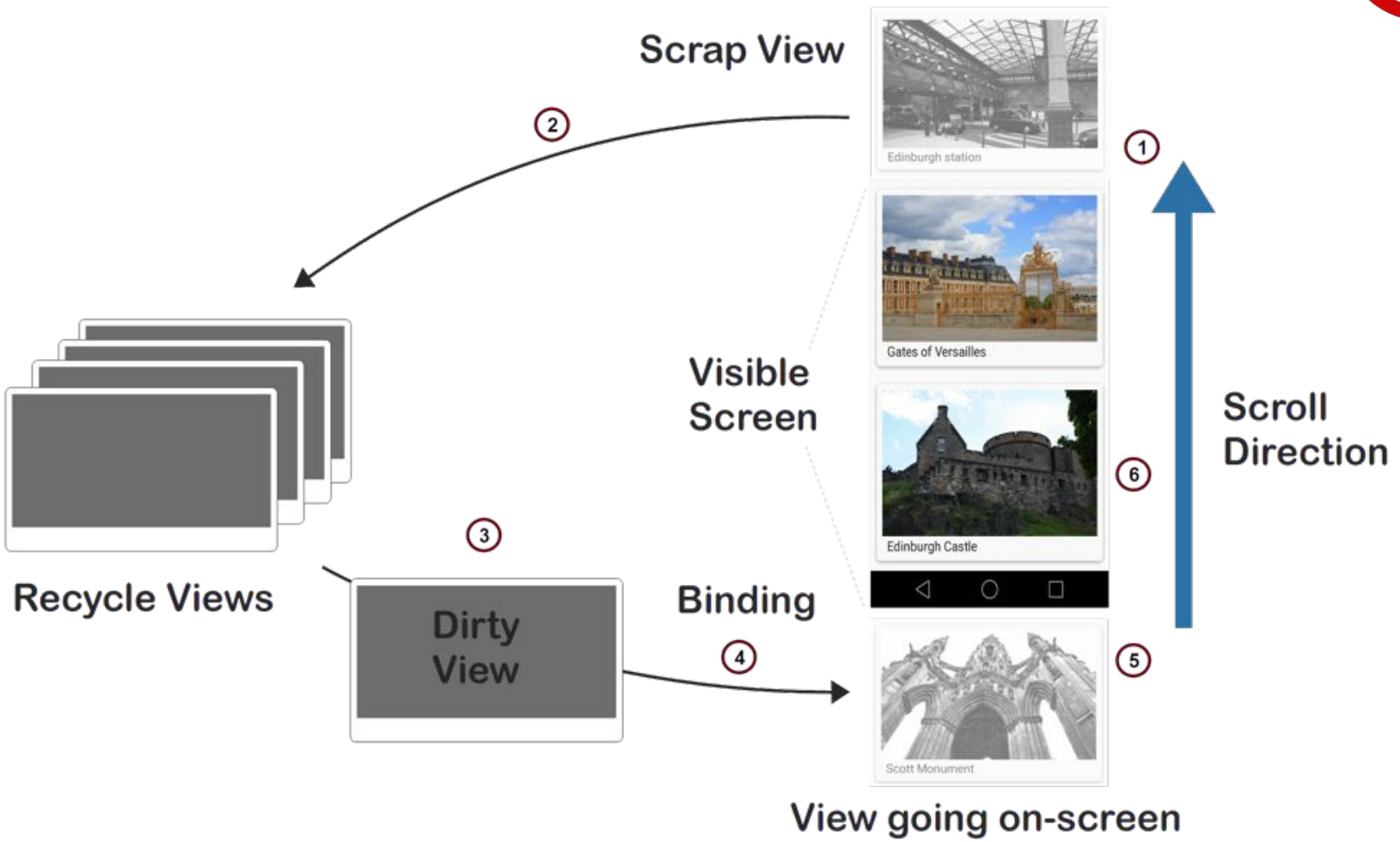
Контейнеры

Адаптеры

Адаптеры контейнеров

- Контейнеры заполняются используя специальный класс - Адаптер. Адаптер позволяет узнать сколько элементов в контейнере и ответственен за их предоставление контейнеру.
  - getView()
  - getCount()
  - getItem()
  - getItemId()
- Все адаптеры, содержащиеся в Android, дополняют базовый адаптер **BaseAdapter**. Вот список готовых адаптеров про которые мы поговорим сегодня:
  - **Base Adapter** - базовый адаптер
  - **ArrayAdapter**<T> - предназначен для работы с ListView. Данные представлены в виде массива, которые размещаются в отдельных элементах TextView.

ArrayAdapter



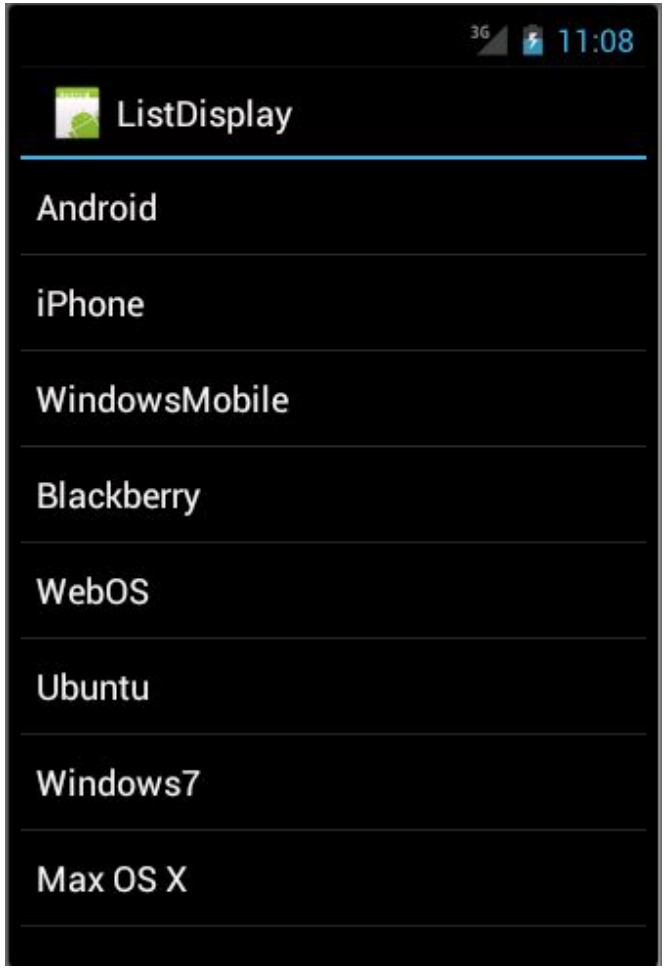
ArrayAdapter

```
public class MyListActivity extends ListActivity {
  public void onCreate(Bundle icicle) {
    super.onCreate(icicle);
    String[] values = new String[] { "Android", "iPhone", "WindowsMobile",
                     "Blackberry", "WebOS", "Ubuntu",
                                   "Windows7", "Max OS X",
                "Linux", "OS/2" };
    ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
              android.R.layout.simple_list_item_1, values);
    setListAdapter(adapter);
  }

  @Override
  protected void onListItemClick(ListView l, View v, int position, long id)
{
    String item = (String) getListAdapter().getItem(position);
    Toast.makeText(this, item + " выбран", Toast.LENGTH_LONG).show();
  }
}
```

ArrayAdapter

- Адаптер все будет делать за вас
- Отдавать количество
- Отдавать элементы



BaseAdapter

```
public class MultipleItemsList extends ListActivity {
private MyCustomAdapter mAdapter;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mAdapter = new MyCustomAdapter();
    for (int i = 0; i < 50; i++) {
      mAdapter.addItem("item " + i);
    }
    setListAdapter(mAdapter);
  }

  private class MyCustomAdapter extends BaseAdapter {
    private ArrayList mData = new ArrayList();
    …
    public int getCount() {return mData.size();}
    public String getItem(int position) { return mData.get(position);}
    public long getItemId(int position) { return position;}
    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
    …
    }
  }

  public static class ViewHolder {
    public TextView textView;
  }
}
```

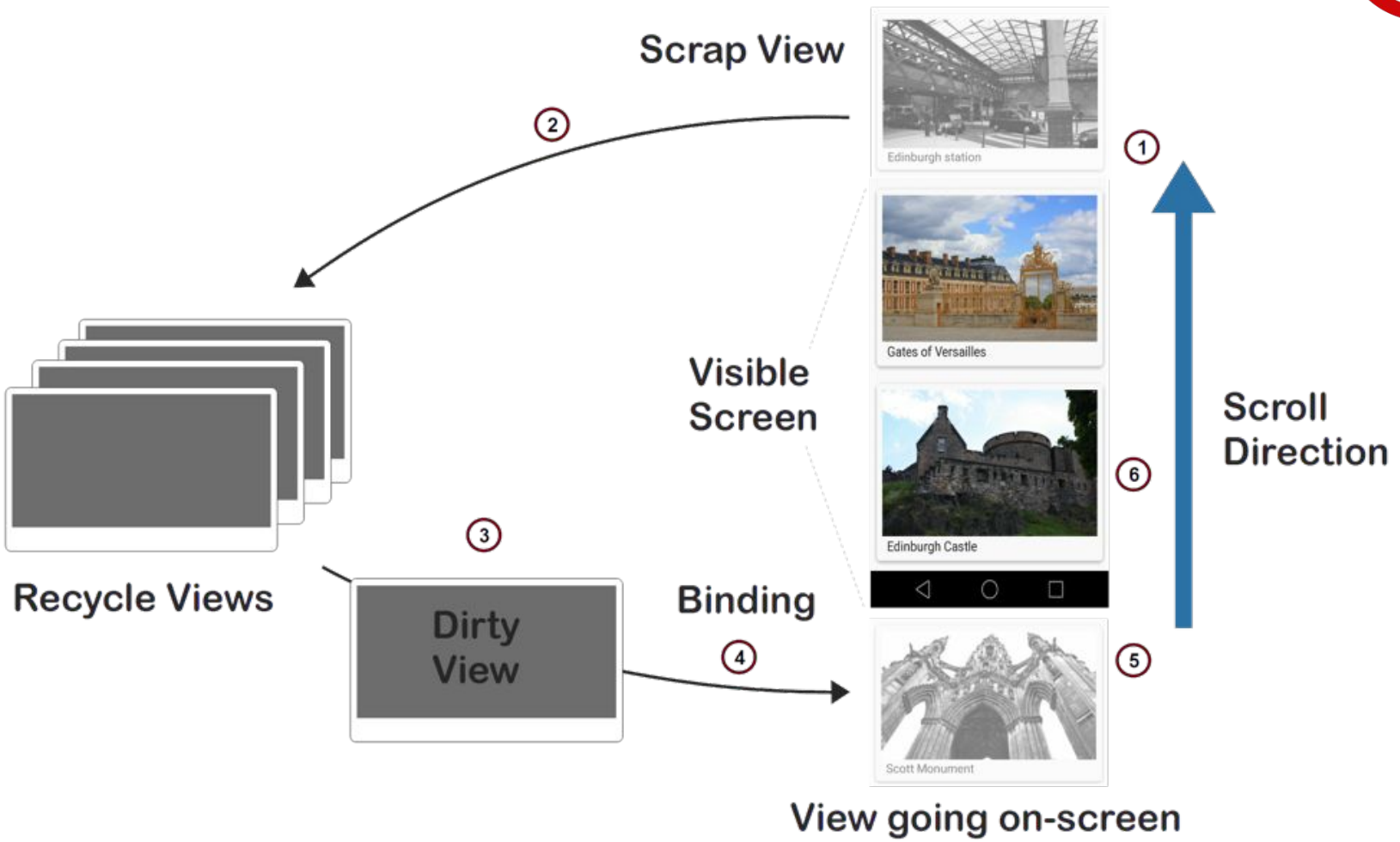
ViewHolder Pattern

```
@Override
public View getView(int position, View convertView, ViewGroup
parent) {
  ViewHolder holder = null;
  if (convertView == null) {
    convertView = mInflater.inflate(R.layout.item1, null);
    holder = new ViewHolder();
    holder.textView =
        (TextView)convertView.findViewById(R.id.text);
    convertView.setTag(holder);
  } else {
    holder = (ViewHolder)convertView.getTag();
  }
  holder.textView.setText(mData.get(position));
  return convertView;
}
```

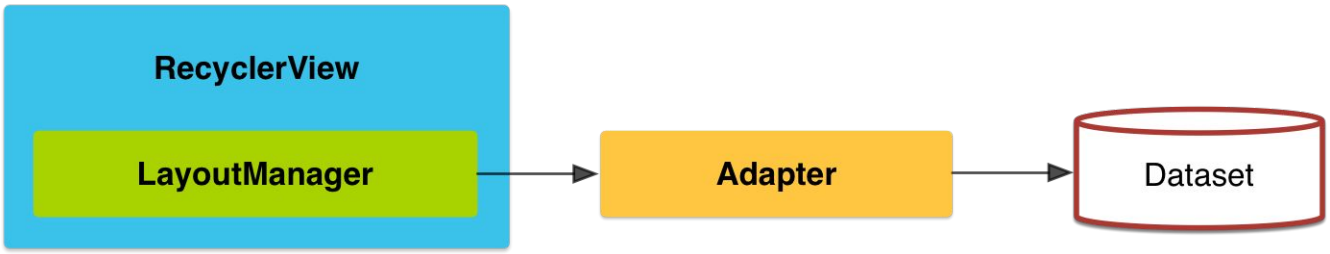
Адаптеры контейнеров

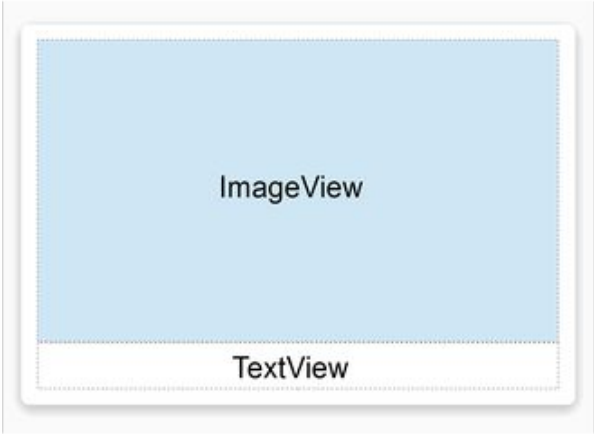
- Контейнеры заполняются используя специальный класс - Адаптер. Адаптер позволяет узнать сколько элементов в контейнере и ответственен за их предоставление контейнеру.
  - •getView()
  - •getCount()
  - •getItem()
  - •getItemId()
- Все адаптеры, содержащиеся в Android, дополняют базовый адаптер **BaseAdapter**. Вот список готовых адаптеров про которые мы поговорим сегодня:
  - •**Base Adapter** - базовый адаптер
  - •**ArrayAdapter**<T> - предназначен для работы с ListView. Данные представлены в виде массива, которые размещаются в отдельных элементах TextView.

Цикл жизни элемента списка

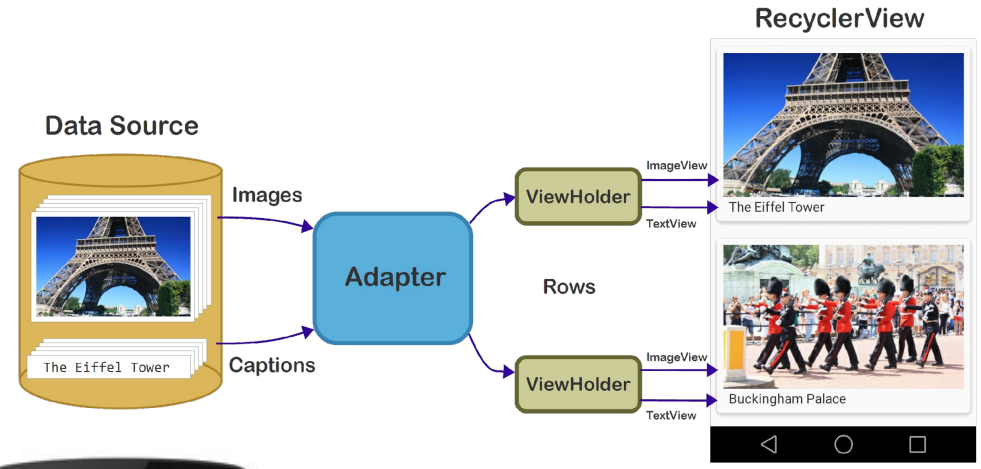


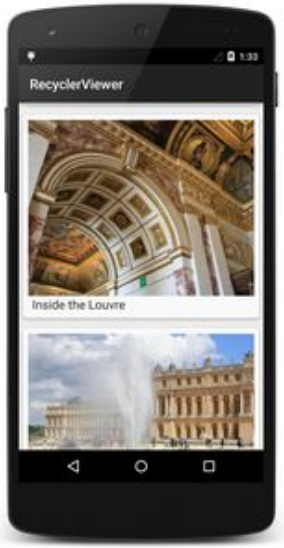
RecyclerView



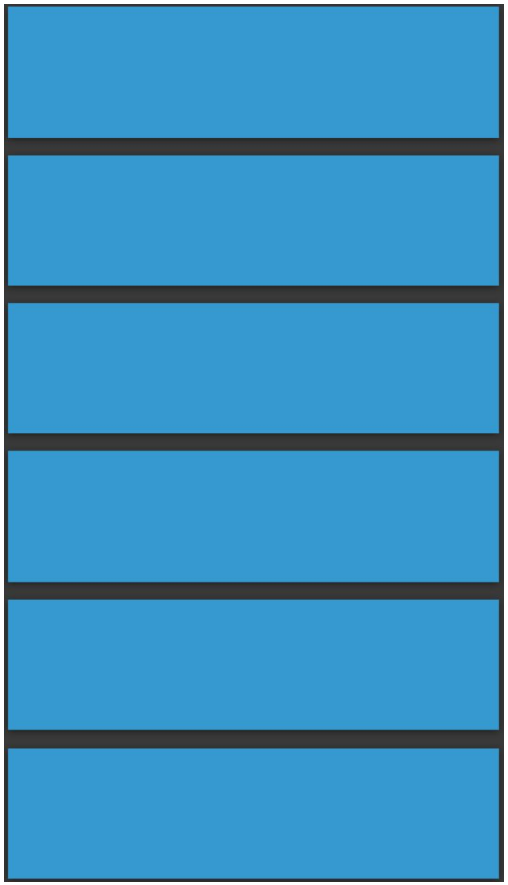


RecyclerView



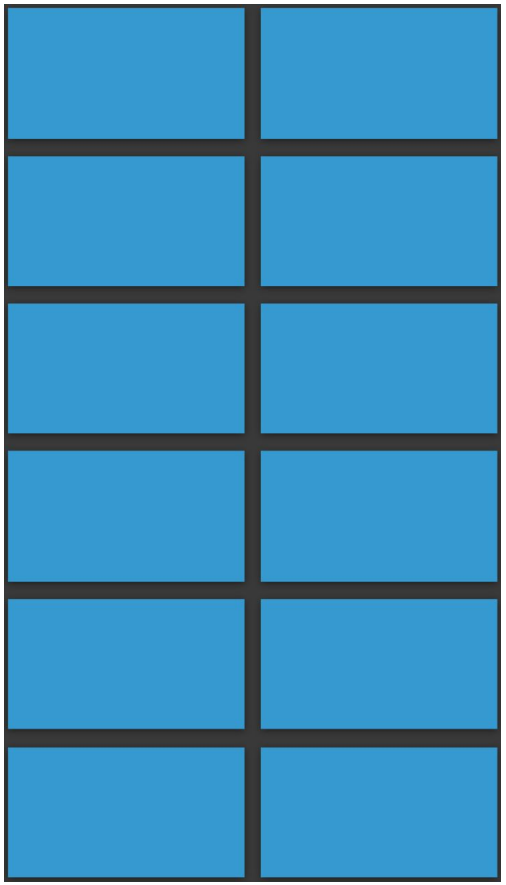


LinearLayoutManager



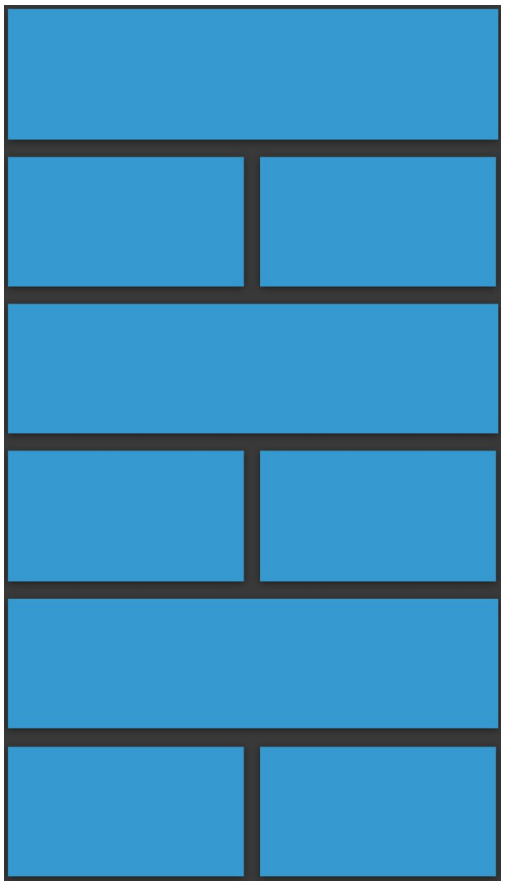
```
LinearLayoutManager manager = new LinearLayoutManager(this,
    LinearLayoutManager.VERTICAL,
    false);
```

GridLayoutManager



```
GridLayoutManager manager = new GridLayoutManager(this,
        2,
        GridLayoutManager.VERTICAL,
        false);
```

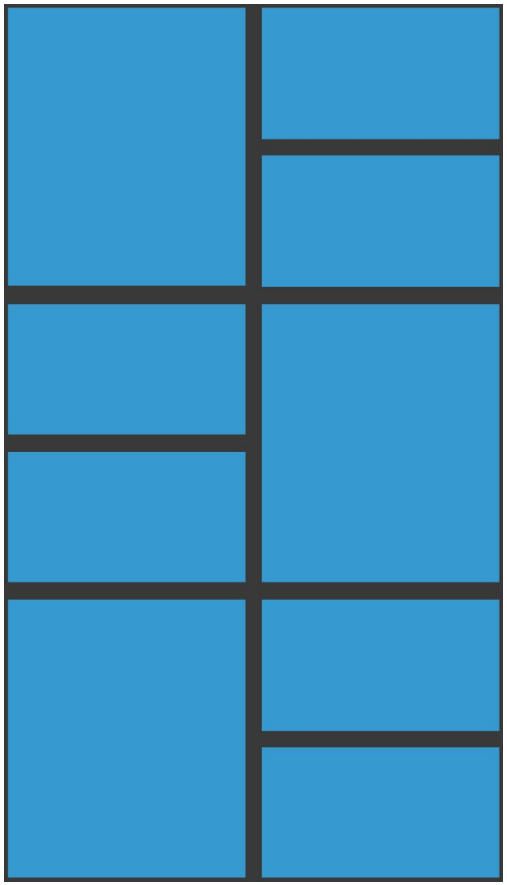
GridLayoutManager



```
GridLayoutManager manager = new GridLayoutManager(this,
        2,
        GridLayoutManager.VERTICAL,
        false);

manager.setSpanSizeLookup(
    new GridLayoutManager.SpanSizeLookup() {
        @Override
        public int getSpanSize(int position) {
            return (position % 3 == 0 ? 2 : 1); }
    });
```

StaggeredGridLayoutManager



```
StaggeredGridLayoutManager manager = new StaggeredGridLayoutManager(
    2,
    StaggeredGridLayoutManager.VERTICAL);
```

RecyclerView.Adapter<T>

```
public class SimpleRecyclerAdapter extends RecyclerView.Adapter<SimpleViewHolder> {

    @Override
    public SimpleViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

    }

    @Override
    public void onBindViewHolder(SimpleViewHolder holder, int position) {
    }

    @Override
    public int getItemCount() {
        return 0;
    }
}
```

RecyclerView.ViewHolder

```
public class SimpleViewHolder extends RecyclerView.ViewHolder {
    private TextView mTitle;
    private TextView mText;

    public void setTitle(String title) {
        mTitle.setText(title);
    }

    public void setText(String text) {
        mText.setText(text);
    }

    public SimpleViewHolder(View itemView) {
        super(itemView);

        mTitle = (TextView)itemView.findViewById(R.id.title);
        mText = (TextView)itemView.findViewById(R.id.text);
    }
}
```

Сервис

- Что такое сервис
- Чем сервис отличается от потока
- Какие бывают сервисы
- Как с ними работать
- Что такое AIDL

Сервис

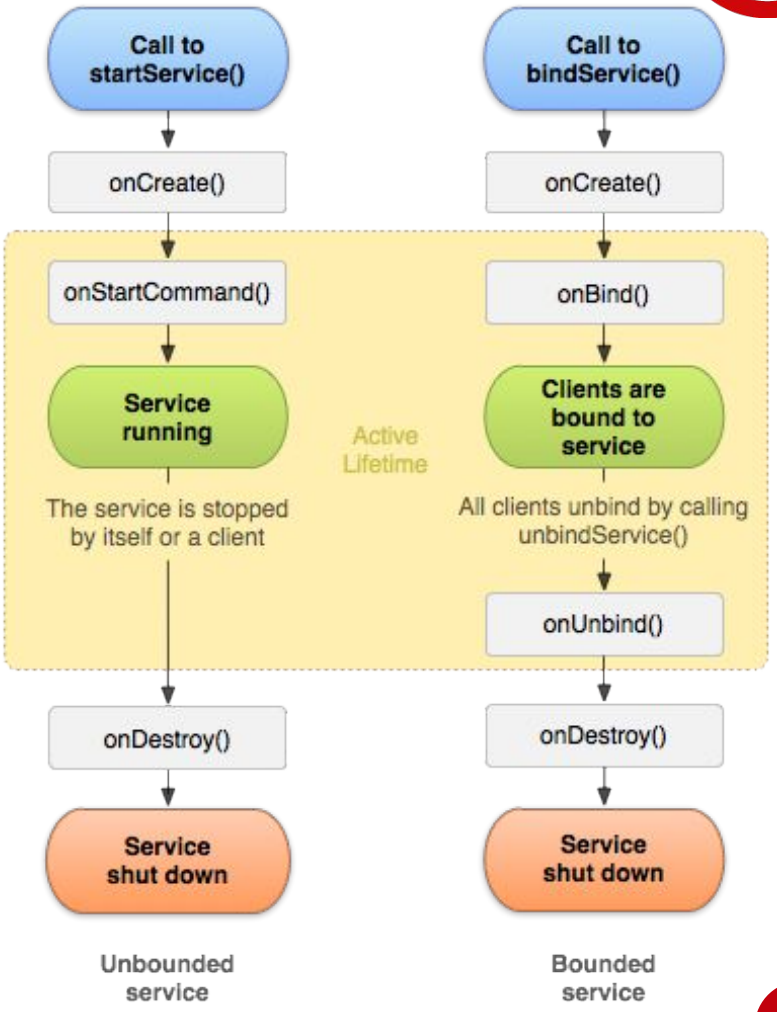
- Это компонент приложения
- Его нужно прописывать в манифесте
- Позволяет приложению осуществлять действия без взаимодействия с пользователем, в том числе в фоне
- Сервис по умолчанию запускается в основном потоке хост процесса (того кто его породил)
- **Сервис это не процесс** он запускается в рамках процесса хоста (есть исключения)
- **Сервис это не поток** он не создает отдельный поток

Сервис в манифесте

```
<service android:enabled=["true" | "false"]
         android:exported=["true" | "false"]
         android:icon="drawable resource"
         android:isolatedProcess=["true" | "false"]
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

Типы сервисов

- Started (запущенные) — сервисы которые запускаются любым другим компонентом приложения и работают пока не остановят сами себя или кто-то не остановит их.
- Bound (связанные) — сервис который выступает в роли сервера в клиент-серверной архитектуре. Такой сервис создается при первом соединении(запросе) от другого компонента приложения. Сервис останавливается, когда отсоединится последний клиент.
- Сервис может быть одновременно и Started и Bound. Такой сервис способен «жить вечно» и обслуживать запросы клиентов



Управление жизнью сервиса

@Override **public int** onStartCommand(Intent intent, **int** flags, **int** startId)

- START_STICKY
- START_NOT_STICKY
- START_REDELIVER_INTENT

Вызов метода

void startForeground(**int** id, Notification notification)

Класс сервиса

```
public class MyService extends Service {
  public void onCreate() {
    super.onCreate();
  }

  public int onStartCommand(Intent intent, int flags, int startId) {
    return super.onStartCommand(intent, flags, startId);
  }

  public void onDestroy() {
    super.onDestroy();
  }

  public IBinder onBind(Intent intent) {
    return null;
  }
}
```

IntentService

- Подкласс обычного Service
- Создает новый поток для работы
- Все Intent кидает в onHandlerIntent
- Когда очередь пустая сам завершает работу

```
public class MyService extends IntentService {
  public MyService() {
    super("myname");
  }

  public void onCreate() {
    super.onCreate();
  }

  @Override
  protected void onHandleIntent(Intent intent)
  {

    …
  }

  public void onDestroy() {
    super.onDestroy();
  }
}
```

AIDL

- В буквальном переводе – язык описания интерфейсов Android.
- Используется для описания композиции и декомпозиции Java объектов в примитивы ОС для непосредственно передачи между процесами.
- Импортировать нужно даже те aidl файлы, которые находятся в том же пакете.
- Ключевое слово oneway в декларации void метода означает что метод будет вызван асинхронно (клиент не дожидается его выполнения).
- Использовать можно только примитивы, String, List и Parcelable классы, объявленные в других aidl файлах.

Планирование задач

- Не требует выполнения немедленно

Планирование задач

- Не требует выполнения немедленно
- Периодическое выполнение

Планирование задач

- Не требует выполнения немедленно
- Периодическое выполнение
- Условное выполнение

Решение задачи планирования

- TimerTask

Решение задачи планирования

- TimerTask
- ScheduledExecutor

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager
- SyncAdapter

Решение задачи планирования

- TimerTask
- ScheduledExecutor
- Handler.postDelayed()
- AlarmManager
- SyncAdapter
- JobScheduler

Job Scheduler

Режим работы Android до API 21 (Android 1.6 - 4.4)

Job Scheduler

Режим работы Android начиная с API 21 (Android 5.0)
[Android Doze Mode]

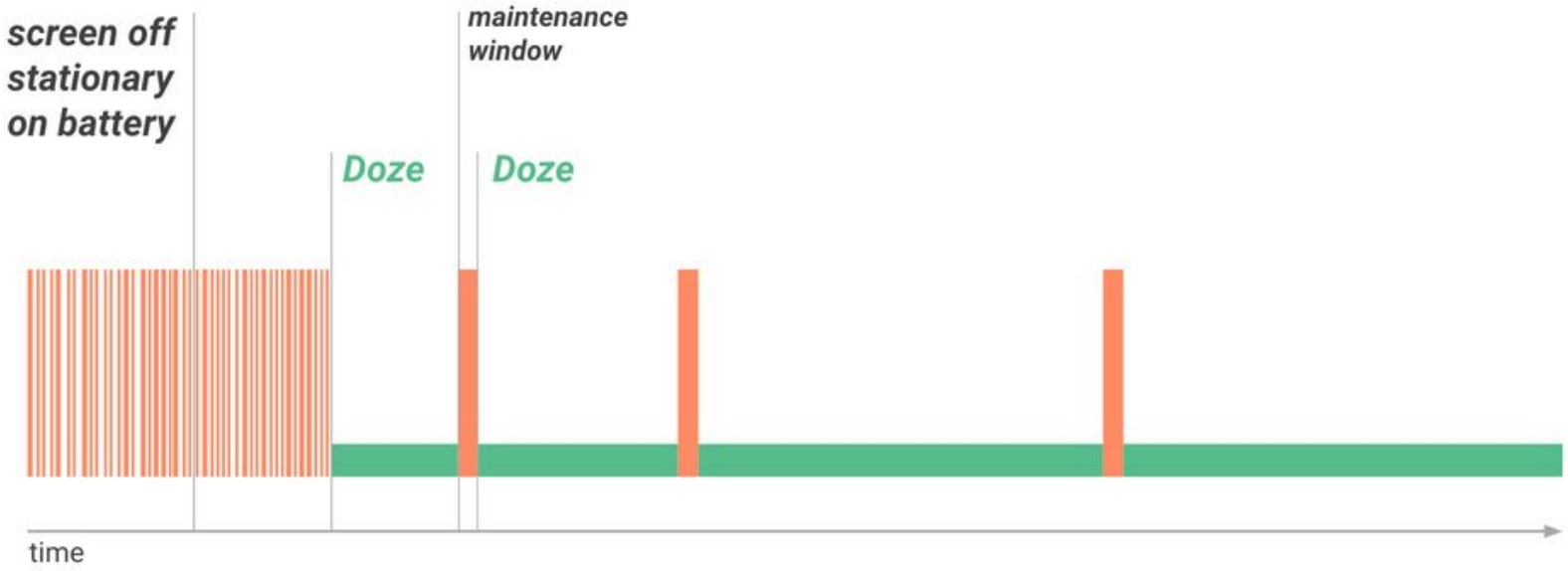


Figure 1. Doze provides a recurring maintenance window for apps to use the network and handle pending activities.

JobScheduler

Pros:

- Оптимизация расходования заряда батареи
- Оптимизация использования памяти
- Оптимизация сетевого взаимодействия

JobScheduler

Pros:

- Оптимизация расходования заряда батареи
- Оптимизация использования памяти
- Оптимизация сетевого взаимодействия

Cons: API >= 21

JobScheduler: условный запуск

- Устройство заряжается
- Сетевое подключение без ограничений
- Устройство не в роуминге
- Устройство в состоянии Idle
- Не ранее, чем через…
- Не позднее, чем…
- В течение...

Job

```
public class ServiceJob extends JobService {

    @Override
    public boolean onStartJob(JobParameters params) {
        // вызывается при начале выполнения задачи
        // !!! выполняется на Main Thread !!!
        // если выполнение задачи завершено в до вызова return
        // вернуть false, иначе true
        return false;
    }

    @Override
    public boolean onStopJob(JobParameters params) {
        // вызывается в случае остановки выполнения задачи
        // например, после вызова cancel или при изменении условий
        // если задачу необходимо перезапланировать, вернуть true
        return false;
    }
}
```

Job

AndroidManifest.xml:

```
<service android:name=".ServiceJob"
        android:permission="android.permission.BIND_JOB_SERVICE"/>
```

JobInfo

```
JobInfo.Builder builder = new JobInfo.Builder();
builder.setBackoffCriteria(initialBackoff, policy)
        .setExtras(bundle)
        .setMinimumLatency(latency)
        .setOverrideDeadline(maxDelay)
        .setPeriodic(period)
        .setPersisted(isPersisted)
        .setRequiredNetworkType(type)
        .setRequiresCharging(isRequiresCharging)
        .setRequiresDeviceIdle(isRequiresIdle)
        .setTriggerContentMaxDelay(maxDelay)
        .setTriggerContentUpdateDelay(delay);

JobInfo info = builder.build();
```

JobScheduler

```
JobScheduler scheduler = (JobScheduler)
context.getSystemService(JOB_SCHEDULER_SERVICE);

scheduler.schedule(info);
```

Demo

https://github.com/kirillF/JobSchedulerDemo

ТЕХНОТРЕК

Спасибо за внимание!

Кирилл Филимонов - Kirill.Filimonov@gmail.com

Юрий Береза - ybereza@gmail.com