



*Белорусский государственный университет
информатики и радиоэлектроники*

Эргономика мобильных приложений

Преподаватель: Меженная Марина Михайловна

К.Т.Н., доцент,

доцент кафедры инженерной психологии и эргономики
а 609-2

mezhennaya@bsuir.by



Кафедра инженерной психологии и эргономики

Лекция 7: Списки. Адаптеры

План лекции:

1. Класс `LayoutInflater`.
2. Список `List View`.
3. Создание собственного адаптера.

Класс `LayoutInflater`

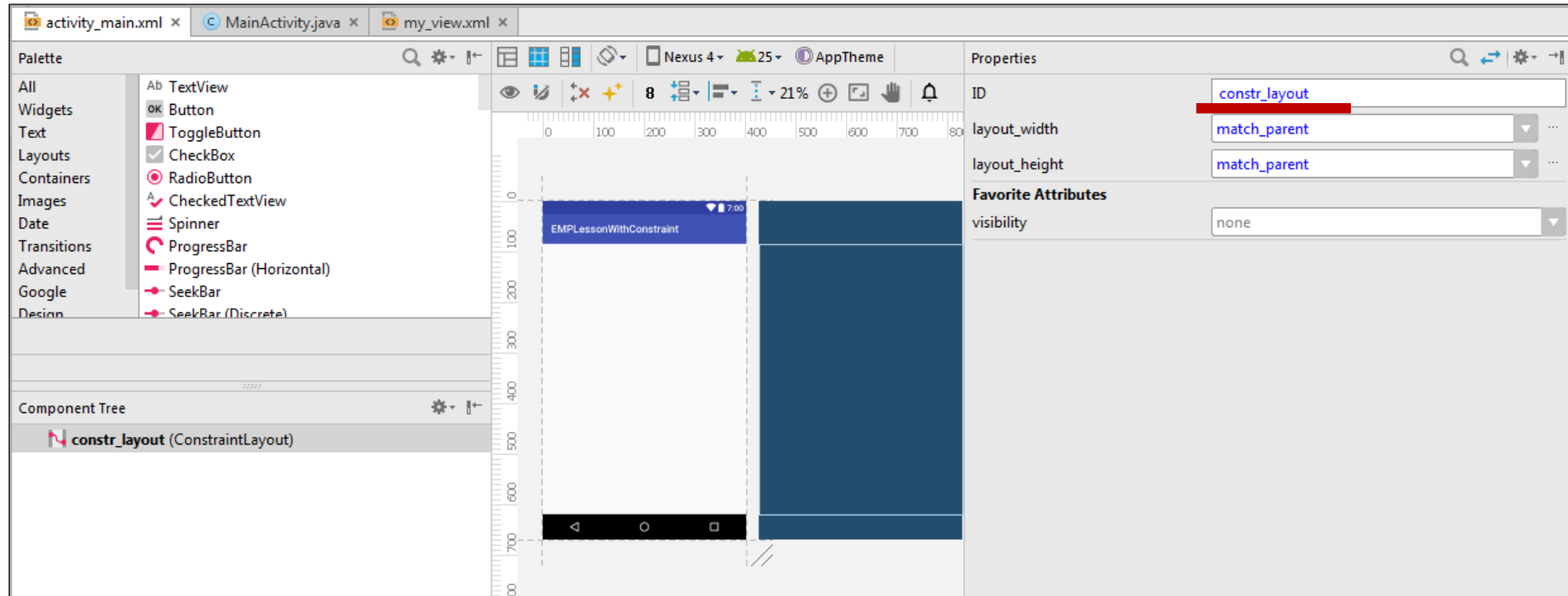
Класс `LayoutInflater` используется для создания View-элемента из содержимого layout-файла.

Последовательность действий:

1. Создать xml-файл с View-элементом/набором View-элементов (например, `my_view.xml`).
2. В Activity, к которому Вы хотите добавить содержимое `my_view.xml`, находим по id layout (в нем и разместиться View-элемент).
3. В методе `onCreate` создаем объект класса `LayoutInflater`:
`LayoutInflater inflater = LayoutInflater.from(this);`
4. Вызываем из-под объекта `inflater` метод `inflate`:
`View view = inflater.inflate(R.layout.my_view, lin_layout, true);`

```
public class MainActivity extends ActionBarActivity {  
  
    private LinearLayout lin_layout;  
    private LayoutInflater inflater;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        initView();  
  
        inflater = LayoutInflater.from(this);  
  
        View view = inflater.inflate(R.layout.my_view, lin_layout, true);  
    }  
  
    private void initView() {  
        lin_layout = (LinearLayout) findViewById(R.id.lin_layout);  
    }  
}
```

Пример создания id-параметра для layout:



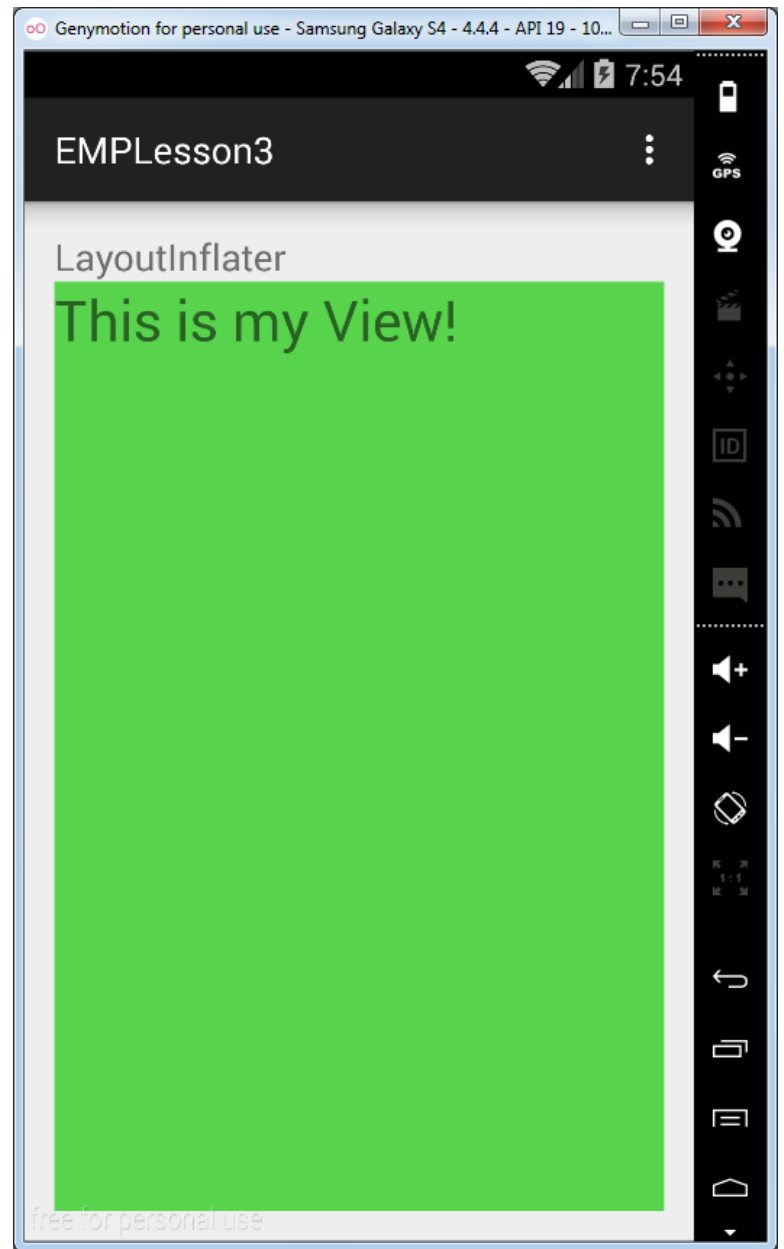
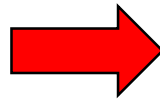
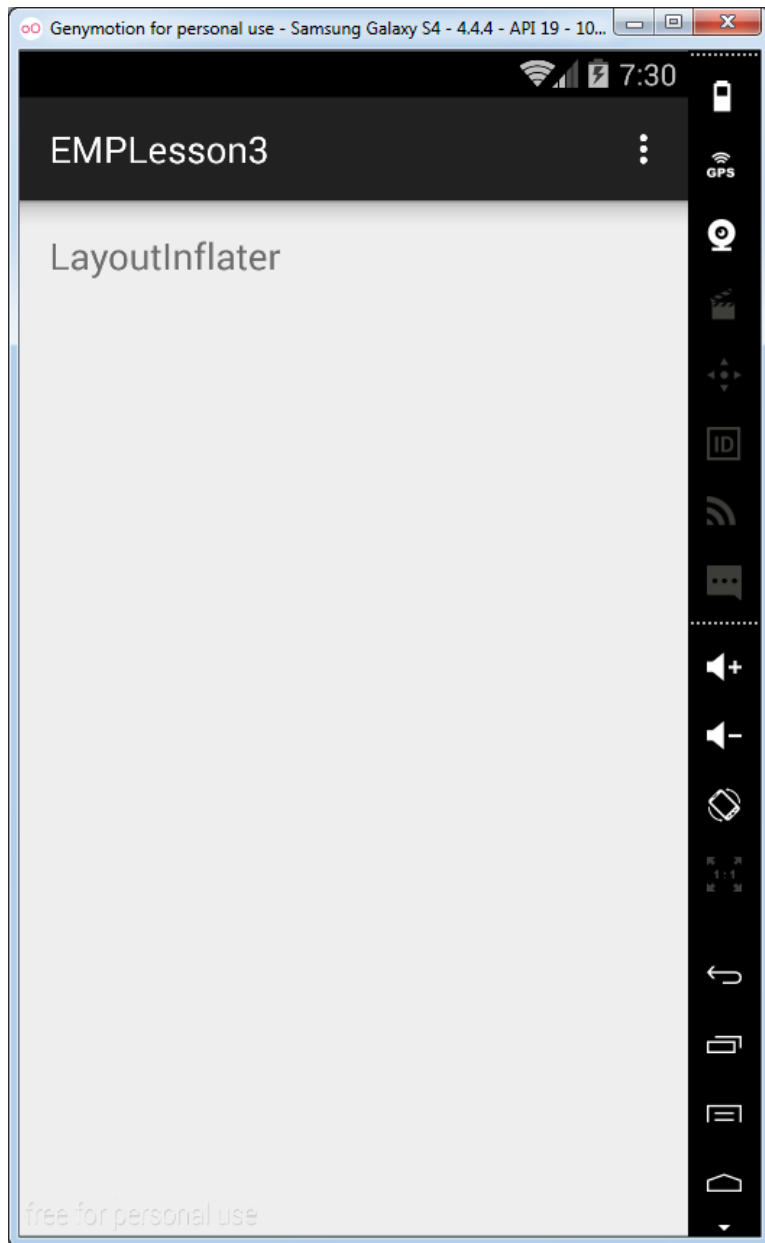
Метод inflater

```
public View inflate (int resource, ViewGroup root, boolean  
attachToRoot)
```

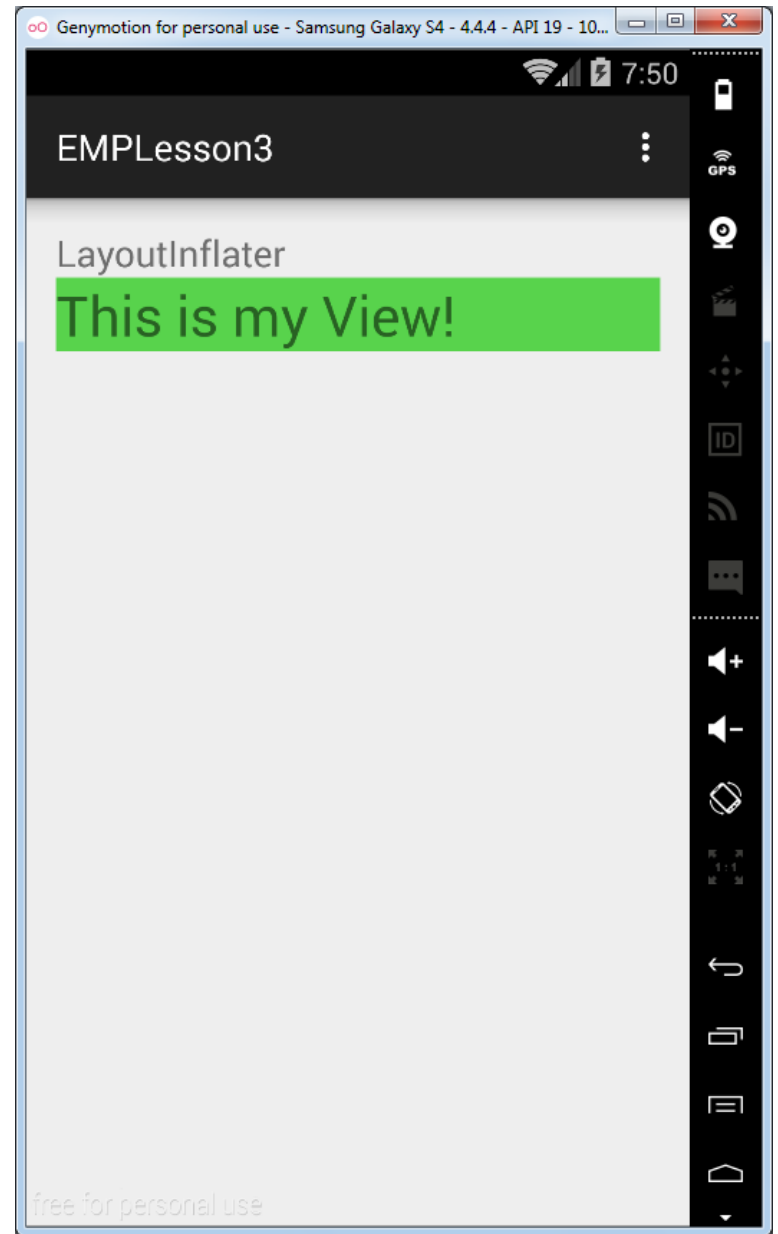
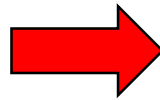
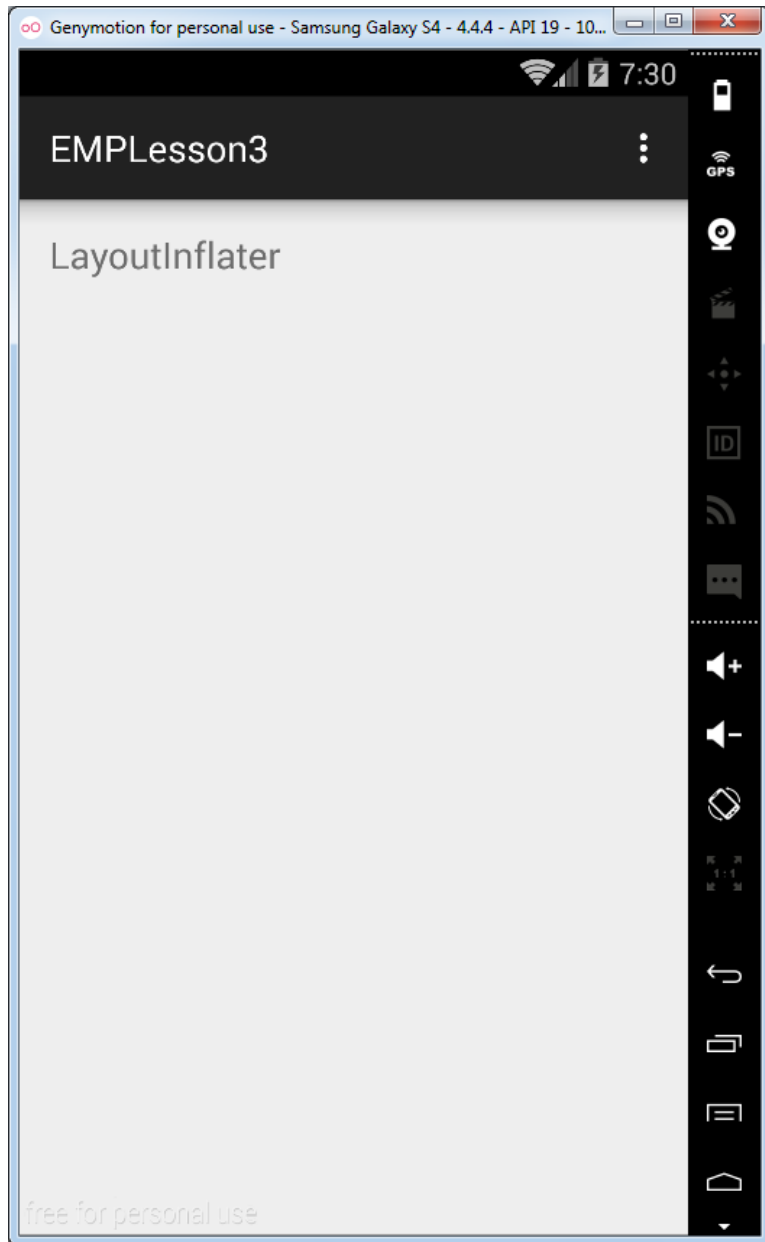
resource - ID layout-файла, который будет использован для создания View. Например - R.layout.my_view

root – родительский ViewGroup-элемент для создаваемого View. LayoutParams от этого ViewGroup присваиваются создаваемому View. Например - lin_layout.

attachToRoot – присоединять ли создаваемый View к root. Если true, то root становится родителем создаваемого View. Т.е. это равносильно команде root.addView(View). Если false – то создаваемый View просто получает LayoutParams от root, но его дочерним элементом не становится.



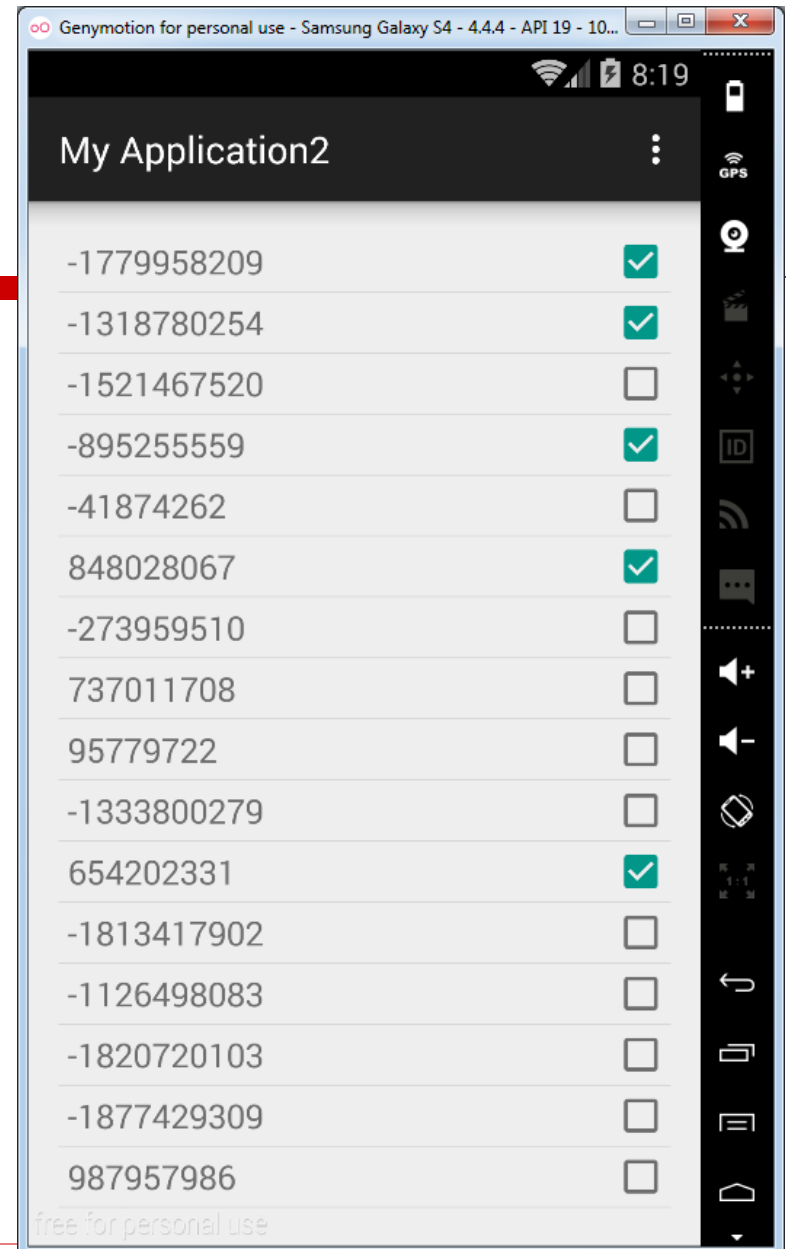
```
public class MainActivity extends ActionBarActivity {  
  
    private LinearLayout lin_layout;  
    private LayoutInflater inflater;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        initView();  
  
        inflater = LayoutInflater.from(this);  
  
        View view = inflater.inflate(R.layout.my_view, null, false);  
        lin_layout.addView(view);  
    }  
  
    private void initView() {  
        lin_layout = (LinearLayout) findViewById(R.id.lin_layout);  
    }  
}
```

Список List View

Список ListView содержит однотипные по дизайну пункты и логику обработки взаимодействия с элементами пунктов списка.

Каждый пункт представляет собой ViewGroup, Каждый пункт списка представляет собой ViewGroup, создаваемый в отдельном layout-файле.



Список List View

Стратегия:

1. Создаем в папке layout xml-файл с графическим отображением одного пункта списка. Для корректного отображения набора пунктов в результирующем списке в качестве layout_height указываем wrap_content или высоту в dp.
2. Создаем набор данных для последующего наполнения списка.

Список List View

Стратегия:

3.Программируем свой собственный адаптер.

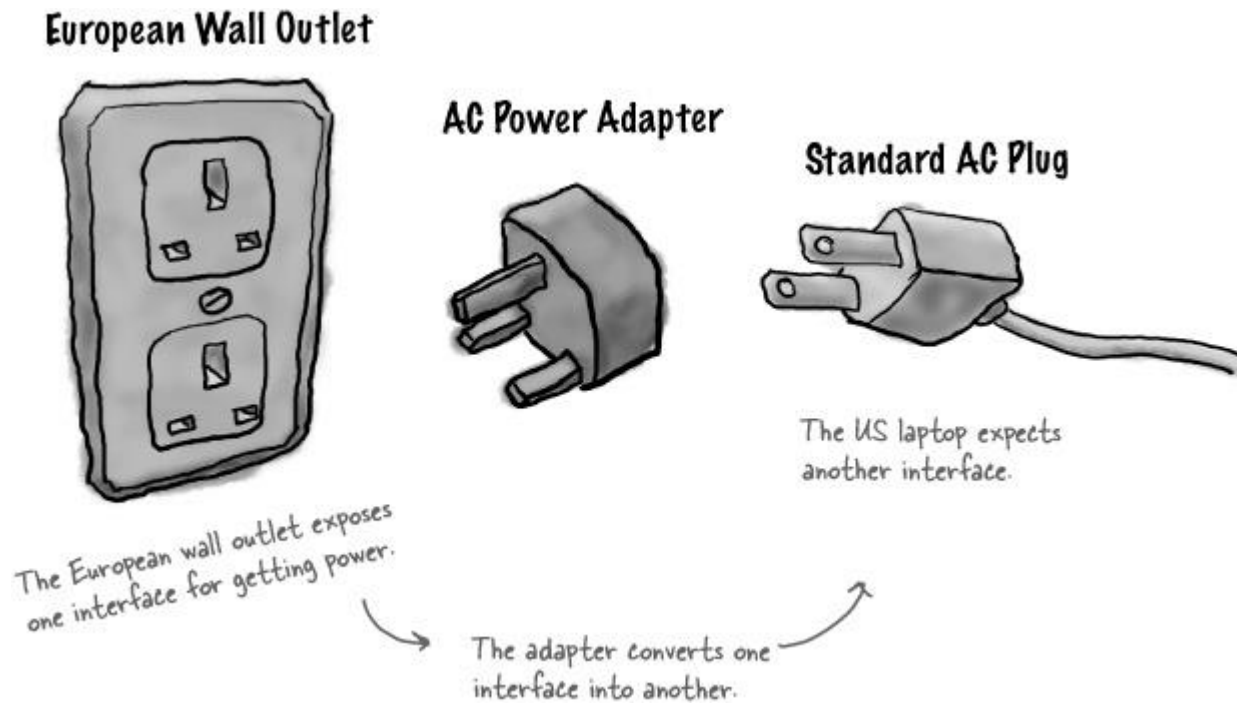
Адаптер — структурный шаблон проектирования, предназначенный для создания класса-оболочки с требуемым интерфейсом.

Адаптеру назначаем набор данных для наполнения списка и layout-ресурс с визуальным представлением одного пункта списка.

4.Присваиваем адаптер списку ListView. Список при построении запрашивает у адаптера пункты, адаптер их создает (используя данные и layout-файл) и возвращает списку.

ОСТОРОЖНО! ПАТТЕРН ПРОЕКТИРОВАНИЯ!!!

АДАПТЕР



Адаптер – это объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту.

При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о наличии первого. Например, вы можете обернуть объект, работающий в метрах, адаптером, который бы конвертировал данные в футы.

Адаптеры могут не только переводить данные из одного формата в другой, но и помогать объектам с разными интерфейсами работать сообща. Это работает так:

Адаптер имеет интерфейс, который совместим с одним из объектов.

Поэтому этот объект может свободно вызывать методы адаптера.

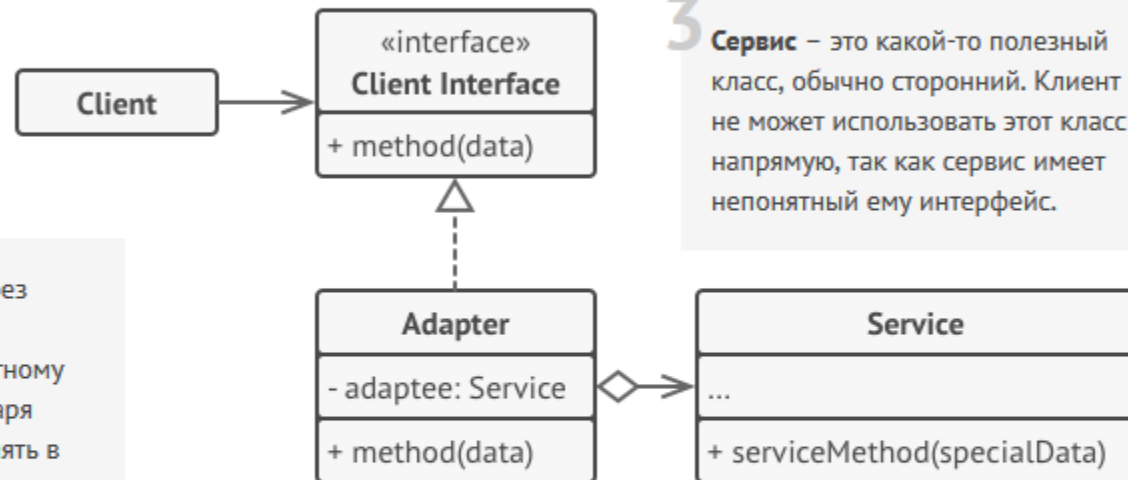
Адаптер получает эти вызовы и перенаправляет их второму объекту, но уже в том формате и последовательности, которые понятны второму объекту.

1 Клиент — это класс, который содержит существующую бизнес-логику программы.

2 Клиентский интерфейс описывает протокол, через который клиент может работать с другими классами.

3 Сервис — это какой-то полезный класс, обычно сторонний. Клиент не может использовать этот класс напрямую, так как сервис имеет непонятный ему интерфейс.

5 Работая с адаптером через интерфейс, клиент не привязывается к конкретному классу адаптера. Благодаря этому, вы можете добавлять в программу новые виды адаптеров, независимо от клиентского кода. Это может пригодиться, если интерфейс сервиса вдруг изменится, например, после выхода новой версии сторонней библиотеки.



```
specialData = convertToServiceFormat(data);
return adaptee.serviceMethod(specialData);
```

4 Адаптер — это класс, который может одновременно работать и с клиентом, и с сервисом. Он реализует клиентский интерфейс и содержит ссылку на объект сервиса. Адаптер получает вызовы от клиента через методы клиентского интерфейса, а затем переводит их в вызовы методов обёрнутого объекта в правильном формате.

Список List View: пример

Создадим модель данных – класс Good
с полями:

int id (номер товара),

String name (наименование товара),

boolean check (флаг, отображающий
выбор товара пользователем),

конструктором класса;

методами getId(), getName(), isCheck(),

setId(int id), setName(String name),
setCheck(boolean check).



Список List View: пример

Разработаем графическое представление одного пункта списка – файл `item_good.xml`, содержащий два `TextView` и один `CheckBox` для отображения параметров товара.

В `activity_main.xml` в корневой `layout` добавляем `ListView` с размерами `wrap content`.



Список List View: пример

В MainActivity.java находим
listView по id;

создаем динамический массив
ArrayList<Good> arr_goods и
имитируем интернет-магазин путем
генерации объектов класса Good.



Список List View: пример

Создаем класс GoodsAdapter для реализации кастомизированного адаптера.

В MainActivity.java создаем объект goodsAdapter от класса GoodsAdapter: для этого в конструктор класса передаем параметр context (т.е. this) и коллекцию объектов arr_goods.

Далее присваиваем goodsAdapter списку с помощью метода setAdapter.



MainActivity.java

```
public class MainActivity extends ActionBarActivity {  
  
    private ListView listView;  
    private ArrayList<Good> arr_goods = new ArrayList<Good>();  
    private final int SIZE_OF_ARR = 25;  
    private GoodsAdapter goodsAdapter;  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        initView();  
        createMyListView();  
    }  
}
```

MainActivity.java

```
private void initView() {  
    listView = (ListView) findViewById(R.id.listView);  
}  
private void createMyListView() {  
    fillData();  
    goodsAdapter = new GoodsAdapter(this, arr_goods);  
    listView.setAdapter(goodsAdapter);  
}  
private void fillData(){  
    int i=0;  
    while (i<SIZE_OF_ARR) {  
        i++;  
        arr_goods.add(new Good(i," " + "My good №" + i, false));  
    }  
}
```

Создаем свой адаптер

Рассмотрим реализацию класса `GoodsAdapter`. Наследуемся от базового адаптера `BaseAdapter`. Для обработки событий от чек-боксов реализуем интерфейс `CompoundButton.OnCheckedChangeListener`.

Создаем свой адаптер

```
public class GoodsAdapter extends BaseAdapter implements
CompoundButton.OnCheckedChangeListener{

    private Context context;
    private ArrayList<Good> arr_goods_adapter;
    private LayoutInflater inflater;

    public GoodsAdapter(Context context, ArrayList<Good>
arr_goods_adapter) {
        this.context = context;
        this.arr_goods_adapter = arr_goods_adapter;
        this.inflater = LayoutInflater.from(context);
    }
```

Создаем свой адаптер

```
// КОЛ-ВО ЭЛЕМЕНТОВ
```

```
@Override
```

```
public int getCount() {  
    return arr_goods_adapter.size();  
}
```

```
// ЭЛЕМЕНТ ПО ПОЗИЦИИ
```

```
@Override
```

```
public Object getItem(int position) {  
    return arr_goods_adapter.get(position);  
}
```

```
// id по позиции
```

```
@Override
```

```
public long getItemId(int position) {  
    return position;  
}
```


// пункт списка

@Override

public View getView(int position, View convertView, ViewGroup viewGroup)

{

View view = convertView;

if (view == null) {

view = inflater.inflate(R.layout.item_good, null, false);

}

Good good_temp = arr_goods_adapter.get(position);

TextView tv_goodId = (TextView) view.findViewById(R.id.tv_goodId);

tv_goodId.setText(Integer.toString(good_temp.getId()));

TextView tv_goodName = (TextView)

view.findViewById(R.id.tv_goodName);

tv_goodName.setText(good_temp.getName());

CheckBox cb_good = (CheckBox) view.findViewById(R.id.cb_good);

cb_good.setChecked(good_temp.isCheck());

cb_good.setTag(position);

cb_good.setOnCheckedChangeListener(this);

return view;

}

Создаем свой адаптер

@Override

```
public void onCheckedChanged(CompoundButton
compoundButton, boolean isChecked) {
    if (compoundButton.isShown()) {
        int i = (int) compoundButton.getTag();
        arr_goods_adapter.get(i).setCheck(isChecked);
        notifyDataSetChanged();
    }
}
```

Добавляем header и footer

EMPLesson4	
My Goods List	
1 My good №1	<input type="checkbox"/>
2 My good №2	<input checked="" type="checkbox"/>
3 My good №3	<input checked="" type="checkbox"/>
4 My good №4	<input type="checkbox"/>
5 My good №5	<input type="checkbox"/>
6 My good №6	<input type="checkbox"/>
7 My good №7	<input type="checkbox"/>
8 My good №8	<input type="checkbox"/>
9 My good №9	<input type="checkbox"/>
10 My good №10	<input type="checkbox"/>
11 My good №11	<input type="checkbox"/>
12 My good №12	<input type="checkbox"/>
13 My good №13	<input type="checkbox"/>

EMPLesson4	
11 My good №11	<input type="checkbox"/>
12 My good №12	<input type="checkbox"/>
13 My good №13	<input type="checkbox"/>
14 My good №14	<input type="checkbox"/>
15 My good №15	<input type="checkbox"/>
16 My good №16	<input type="checkbox"/>
17 My good №17	<input type="checkbox"/>
18 My good №18	<input type="checkbox"/>
19 My good №19	<input checked="" type="checkbox"/>
20 My good №20	<input type="checkbox"/>
21 My good №21	<input type="checkbox"/>
22 My good №22	<input type="checkbox"/>
23 My good №23	<input type="checkbox"/>
24 My good №24	<input type="checkbox"/>
25 My good №25	<input type="checkbox"/>
SHOW CHECKED ITEMS Count of goods = 3	

Добавляем header и footer

Header и Footer – это View-элементы, которые могут быть добавлены к списку сверху и снизу.

Для этого необходимо создать графические представления `header_mygoods.xml` и `footer_mygoods.xml`, преобразовать их в View-элементы и предоставить списку с помощью методов `addHeader` или `addFooter`.

Обязательным условием отображения Header и Footer является их добавление к списку до того, как присваивается адаптер.

Добавляем header и footer

Модифицируем код MainActivity.java:

```
public class MainActivity extends ActionBarActivity implements  
View.OnClickListener {
```

```
    // добавляем новые переменные класса  
    private LayoutInflater inflater;  
    private View view_header, view_footer;  
    private Button btnShow;  
    private TextView tv_count;
```

Добавляем header и footer

// добавляем Header и Footer

```
private void createMyListView() {  
    fillData();  
    goodsAdapter = new GoodsAdapter(this, arr_goods, this);  
  
    layoutInflater = LayoutInflater.from(this);  
    view_header = layoutInflater.inflate(R.layout.header_mygoods, null);  
    view_footer = layoutInflater.inflate(R.layout.footer_mygoods, null);  
    btnShow = (Button) view_footer.findViewById(R.id.btnShow);  
    btnShow.setOnClickListener(this);  
    tv_count = (TextView) view_footer.findViewById(R.id.tv_count);  
  
    listView.addHeaderView(view_header);  
    listView.addFooterView(view_footer);  
  
    listView.setAdapter(goodsAdapter);  
}
```

Реализуем обратную связь от GoodsAdapter в MainActivity

The diagram illustrates the implementation of a reverse connection from GoodsAdapter in MainActivity. It shows three components: EMPLesson4 (MainActivity), EMPLesson4 (GoodsAdapter), and SecondActivity.

EMPLesson4 (MainActivity): Displays a list of goods under the heading "My Goods List". The list includes items 1 through 13, each with a checkbox. Items 2 and 3 are checked.

EMPLesson4 (GoodsAdapter): Displays a list of goods from 11 to 25, each with a checkbox. Item 19 is checked. At the bottom, there is a button labeled "SHOW CHECKED ITEMS" and a label "Count of goods = 3" which is circled in red.

SecondActivity: Displays a message "В Вашей корзине 3 товара(ов):" followed by a list of goods: 2 My good №2, 3 My good №3, and 19 My good №19.

Annotations:

- A red arrow points from the "Count of goods = 3" label in the GoodsAdapter to the "В Вашей корзине 3 товара(ов):" text in SecondActivity.
- A yellow arrow labeled "Задача №1" points to the list of goods in SecondActivity.
- A yellow arrow labeled "Задача №2" points to the "SHOW CHECKED ITEMS" button in MainActivity.

```
private ArrayList<Good> arr_checked_goods_adapter = new ArrayList<Good>();
```

@Override

```
public void onCheckedChanged(CompoundButton compoundButton,
boolean isChecked) {
    if (compoundButton.isShown()) {
        int i = (int) compoundButton.getTag();
        arr_goods_adapter.get(i).setCheck(isChecked);
        notifyDataSetChanged();
        if(isChecked){
            arr_checked_goods_adapter.add(arr_goods_adapter.get(i));
        }else {
            arr_checked_goods_adapter.remove(arr_goods_adapter.get(i));
        }
    }
}
```



```
public ArrayList<Good> getCheckedGoods() {  
    return arr_checked_goods_adapter;  
}
```

MainActivity.java:

Задача
№1

```
private ArrayList<Good> arr_checked_goods = new ArrayList<Good>();
```

@Override

```
public void onClick(View view) {  
    arr_checked_goods = goodsAdapter.getCheckedGoods();
```

```
    Intent intent = new Intent(this, SecondActivity.class);  
    intent.putParcelableArrayListExtra("MyList", arr_checked_goods);  
    startActivity(intent);  
}
```

Good.java:

Задача
№1

```
public class Good implements Parcelable{
    private int id;
    private String name;
    private boolean check;

    // обычный конструктор
    public Good(int id, String name, boolean check){
        this.id = id;
        this.name = name;
        this.check = check;
    }

    // + реализация get-теров и set-теров
```

Good.java:

Задача
№1

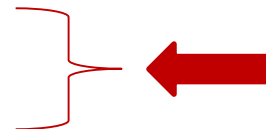
@Override

```
public int describeContents() {  
    return 0;  
}
```

// упаковываем объект в Parcel

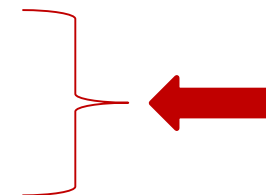
@Override

```
public void writeToParcel(Parcel parcel, int i) {  
    parcel.writeInt(id);  
    parcel.writeString(name);  
}
```



// конструктор, считывающий данные из Parcel

```
private Good(Parcel parcel) {  
    id = parcel.readInt();  
    name = parcel.readString();  
    check = false;  
}
```



```
public static final Parcelable.Creator<Good> CREATOR = new
Parcelable.Creator<Good>() {
    // распаковываем объект из Parcel
    public Good createFromParcel(Parcel in) {
        return new Good(in);
    }

    public Good[] newArray(int size) {
        return new Good[size];
    }
};
}
```

Задача №2

The screenshot displays three panels from an application:

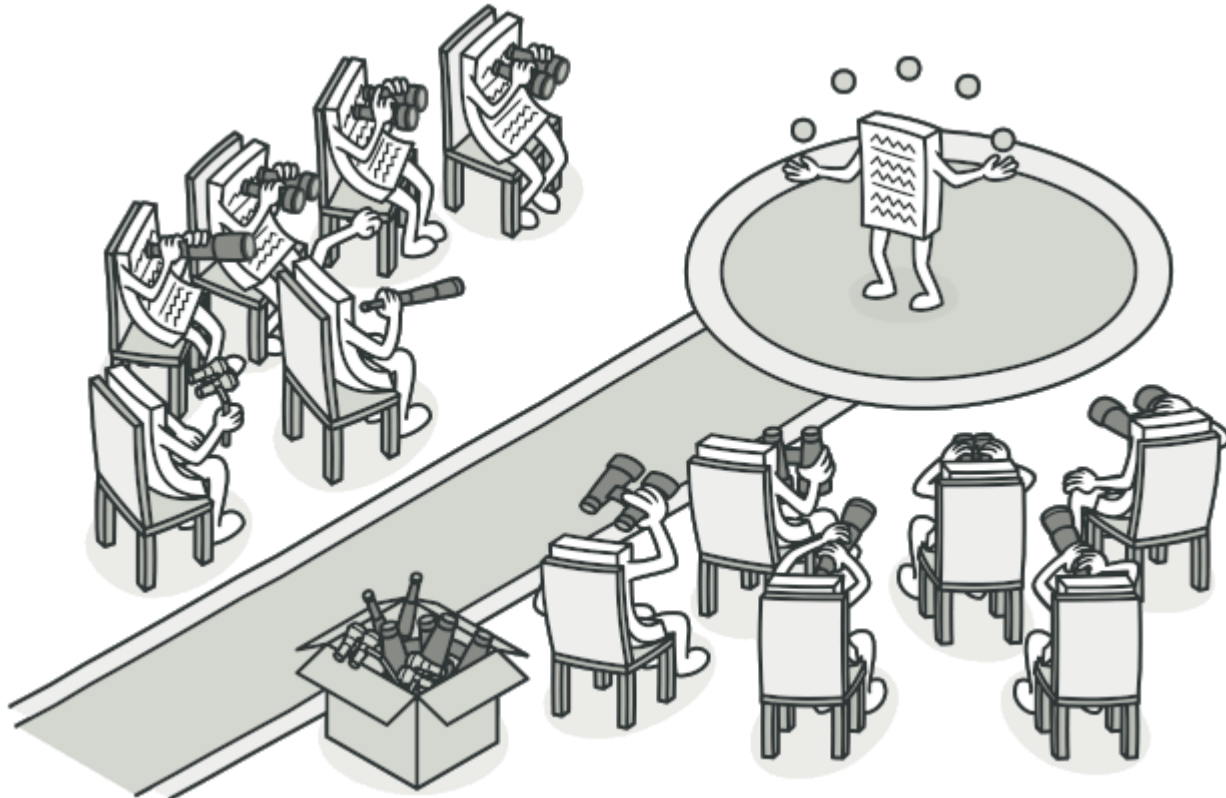
- EMPLesson4 (Left):** A list titled "My Goods List" with 13 items. Items 2 and 3 are checked.
- EMPLesson4 (Middle):** A list with 25 items. Items 19 and 20 are checked. At the bottom, a blue bar shows "SHOW CHECKED ITEMS" and "Count of goods = 3".
- SecondActivity (Right):** A summary titled "В Вашей корзине 3 товара(ов):" listing items 2, 3, and 19.

Yellow arrows indicate tasks:

- Задача №1:** Points to item 19 in the middle list.
- Задача №2:** Points to the "Count of goods = 3" label in the middle panel.

ВНИМАНИЕ! ЕЩЕ ОДИН ПАТТЕРН ПРОЕКТИРОВАНИЯ!!!

НАБЛЮДАТЕЛЬ



Наблюдатель — это поведенческий паттерн проектирования, который создаёт механизм подписки, позволяющий одним объектам следить и реагировать на события, происходящие в других объектах.

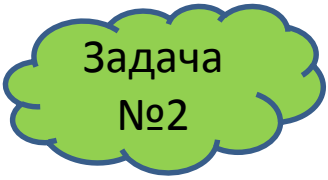
Давайте называть Издателем объект, который содержит важное или интересное для других состояние. Остальные объекты, которые хотят отслеживать изменения этого состояния, назовём Подписчиками.

Паттерн Наблюдатель предлагает хранить внутри объекта издателя список ссылок на объекты подписчиков. Когда в издателе будет происходить важное событие, он будет проходиться по списку подписчиков и оповещать их об этом, вызывая определённый метод объектов-подписчиков.

Издателю безразлично, какой класс будет иметь тот или иной подписчик, так как все они должны следовать общему интерфейсу и иметь единый метод оповещения.

(альтернативное название – интерфейс в обратной связи)

Создаем свой собственный интерфейс:



Задача
№2

```
public interface OnChangeListener {  
  
    public void onDataChanged();  
  
}
```

В **MainActivity.java** имплементируем данный интерфейс, а в методе `onDataChanged()` запрашиваем размер массива с выбранными пользователем товарами и выводим это число в соответствующий `TextView`.

```
public class MainActivity extends ActionBarActivity implements On-  
ChangeListener, View.OnClickListener {
```

```
    @Override
```

```
    public void onDataChanged() {  
        int size = goodsAdapter.getCheckedGoods().size();  
        tv_count.setText("Count of goods = " + size + "");  
    }
```

MainActivity.java

В конструктор GoodsAdapter.java необходимо передать объект от интерфейса OnChangeListener. Так как MainActivity.java имплементирует указанный интерфейс, то в качестве объекта от интерфейса OnChangeListener выступает собственно MainActivity.java , т. е. передаем this:

```
goodsAdapter = new GoodsAdapter(this, arr_goods, this);
```

GoodsAdapter.java:

```
private OnChangeListener onChangeListener;
```

```
public GoodsAdapter(Context context, ArrayList<Good>  
arr_goods_adapter, OnChangeListener onChangeListener) {  
    this.context = context;  
    this.arr_goods_adapter = arr_goods_adapter;  
    this.layoutInflater = LayoutInflater.from(context);  
    this.onChangeListener = onChangeListener;  
}
```

GoodsAdapter.java:

@Override

```
public void onCheckedChanged(CompoundButton compoundButton,  
boolean isChecked) {  
    if (compoundButton.isShown()) {  
        int i = (int) compoundButton.getTag();  
        arr_goods_adapter.get(i).setCheck(isChecked);  
        notifyDataSetChanged();  
  
        if(isChecked){  
            arr_checked_goods_adapter.add(arr_goods_adapter.get(i));  
        }else {  
            arr_checked_goods_adapter.remove(arr_goods_adapter.get(i));  
        }  
        onChangeListener.onDataChanged();  
    }  
}
```

Итоговая структура проекта:

