Recycler View

Тест

https://fintech.tinkoff.ru/contests

4 вопроса, Wi-Fi Tinkoff-Guest

План

- Чем плохи ListView и GridView?
- Что значит recycle?
- Как RecyclerView устроен?
- Что такое LayoutManager и какие они бывают ?
- Как устроен RecyclerView.Adapter
- ItemTouchHelper
- NotifyDataSetChanged или DiffUtil ?
- Recycler Dich

Чем плохи ListView и GridView?

Вопрос к вам

Что значит Recycle?

Recycle

=

Переиспользовать

Что значит Recycle?

RecycleView

=

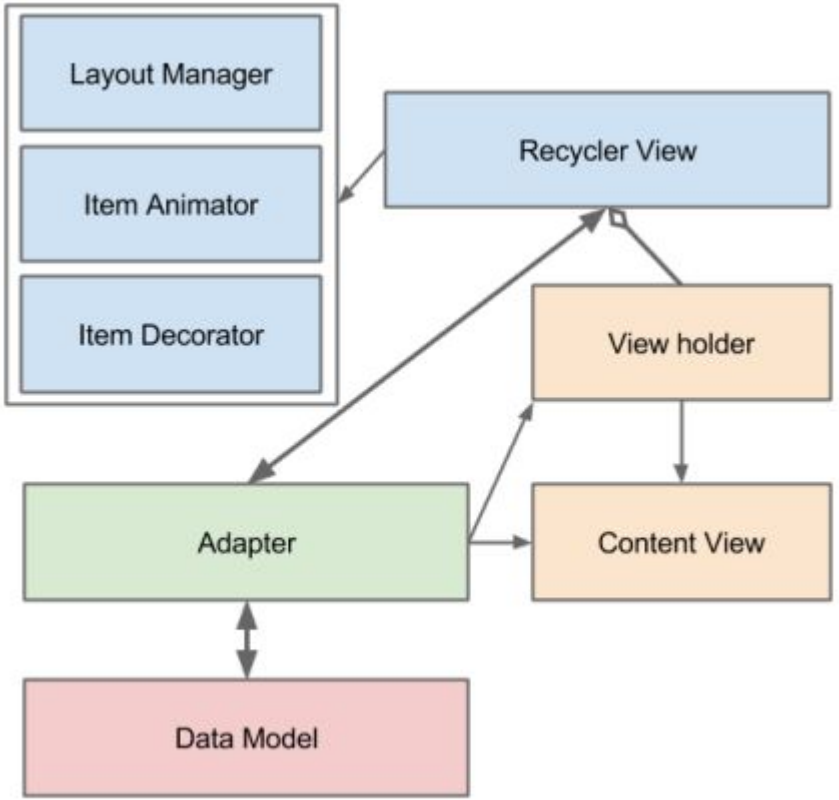
Переиспользовать view

Как RecyclerView устроен?

- View которое умеет отрисовываться
- LayoutManager управляющий состоянием
- Adapter с данными
- ViewHolder — объект, держащий в памяти созданную view

Компоненты RecyclerView

- **LayoutManager - размещает элементы**
- **Item Animator - анимирует элементы**
- **Item Decorator - дорисовывает элементы**
- **Adapter - создает элементы**
- **ViewHolder - кеширует findViewById**



ViewHolder

```
/**
* Реализация класса ViewHolder, хранящего ссылки на виджеты.
*/

class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {

    private val name: TextView
    private val icon: ImageView

    init {
        name = itemView.findViewById<TextView>(R.id.recyclerViewItemName)
        icon = itemView.findViewById<ImageView>(R.id.recyclerViewItemIcon)
    }
}
```

Обязаности LayoutManager’a:

- размещает элементы
- отвечает за размер элементов (measure)
- отвечает за то, какие элементы больше не нужны
- отвечает за скроллинг
- отвечает за View Focusing (В случае ListView отвечал сам ListView); т.е. на каком элементе сфокусироваться.
- отвечает за Accessibility; для людей с ограниченными возможностями.

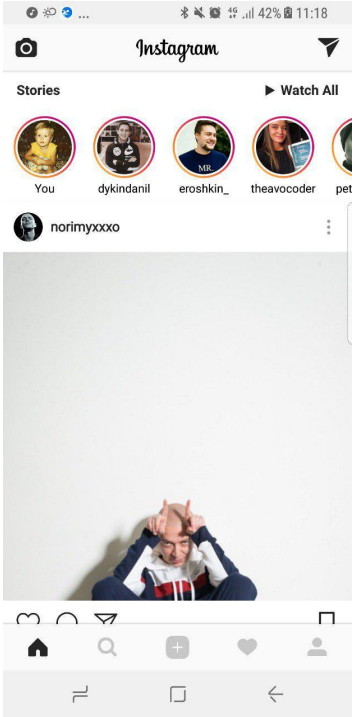
Какие бывают LayoutManager’ы?

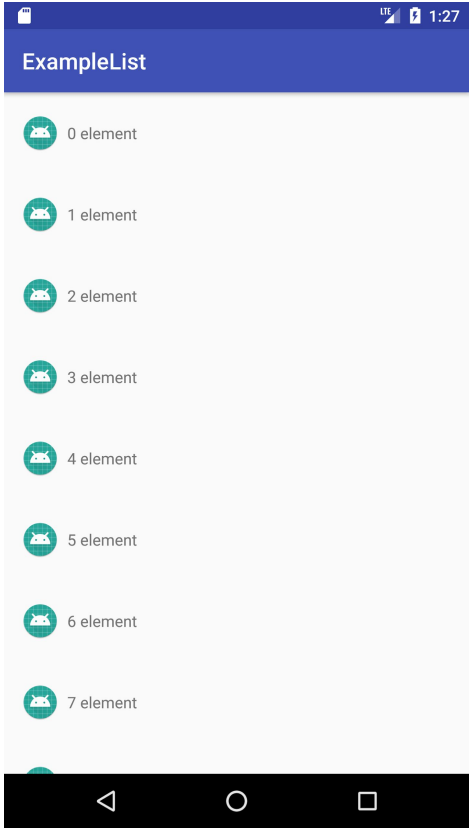
1) LinearLayoutManager
2) GridLayoutManager
3) StaggeredGridLayoutManager

https://habrahabr.ru/company/eastbanctech/blog/267497/

LinearLayoutManager

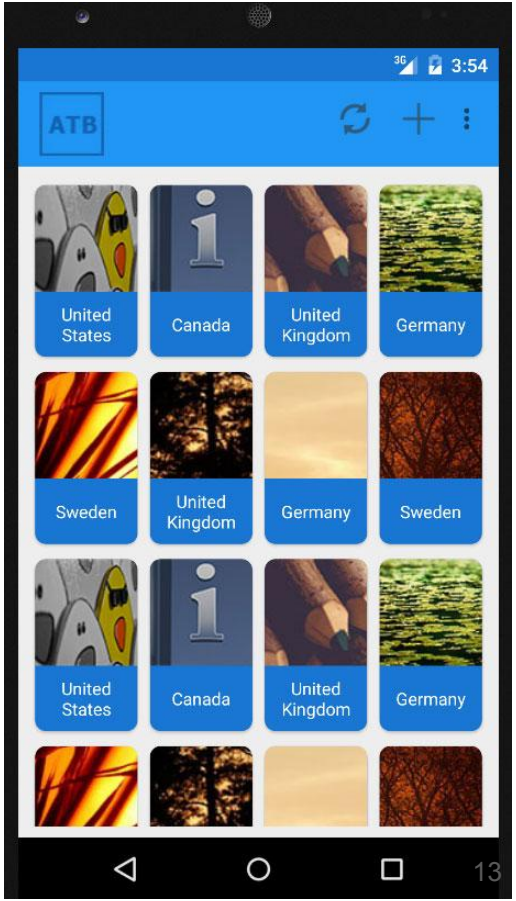
Отображает список в вертикальном или горизонтальном виде





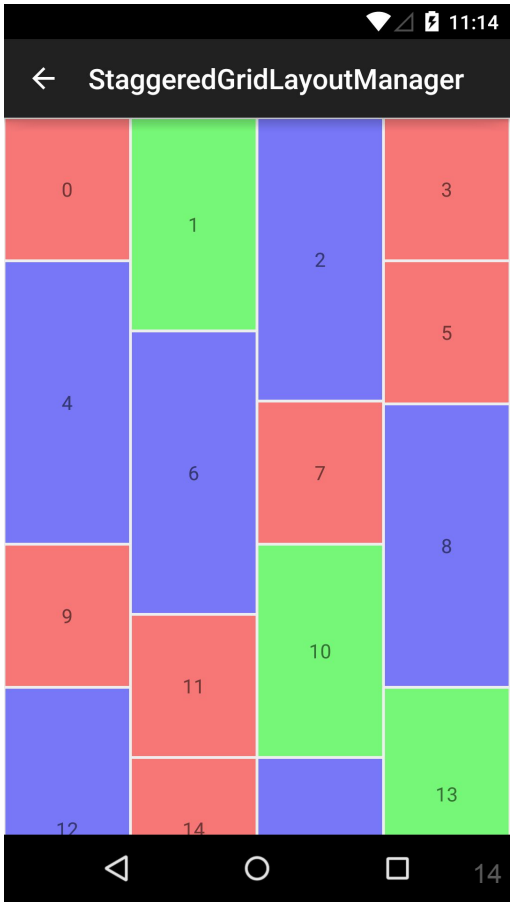
GridLayoutManager

Grid - сетка



StaggeredGridLayoutManager

Staggered Grid - шахматная сетка



Обязанности Adapter’а:

- создание ViewHolder’ов
- заполнение ViewHolder’ов информацией
- уведомление RecyclerView о том, какие элементы изменились.
- частичное обновление данных
- управление количеством ViewType’ов
- информация о переиспользовании ViewHolder’а

Как устроен RecyclerView.Adapter

```
class MyViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
    val name: TextView
    val age: TextView
    val position: TextView
    val photo: AppCompatImageView
    init {
        name = itemView.findViewById(R.id.name)
        age = itemView.findViewById(R.id.age)
        position = itemView.findViewById(R.id.position)
        photo = itemView.findViewById(R.id.photo)
    }
}
```

Как устроен RecyclerView.Adapter

```
class MyAdapter(private val workerList: List<Scheduler.Worker>)
: RecyclerView.Adapter<MyAdapter.MyViewHolder>() {

override fun onCreateViewHolder(parent: ViewGroup,
 viewType: Int): MyViewHolder {
        val v = LayoutInflater.from(parent.context)
                .inflate(R.layout.item_main_grid, parent, false)
        return MyViewHolder(v)
    }
```

Как устроен RecyclerView.Adapter

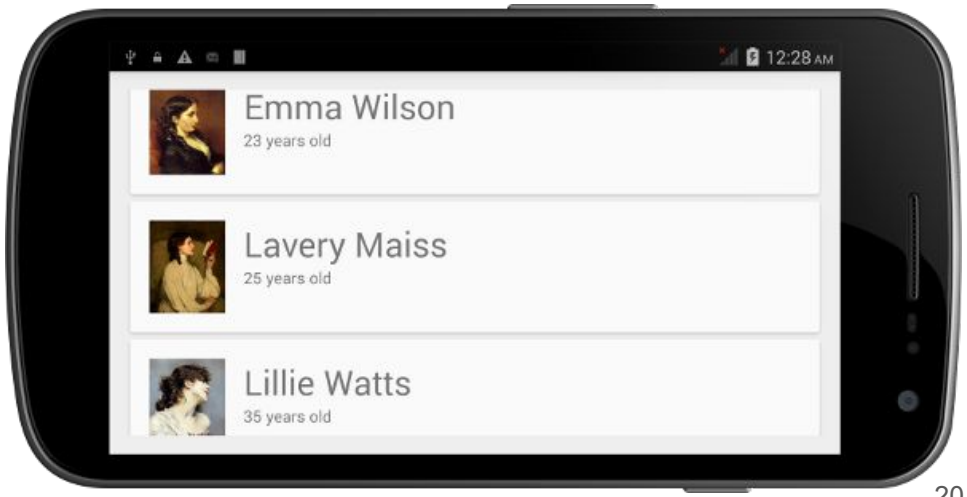
```
override fun onBindViewHolder(holder: MyViewHolder, position: Int) {
    val worker = workerList[position]
    holder.name.setText(worker.getName())
    holder.age.setText("Age: " + worker.getAge())
    holder.position.setText("Age: " + worker.getPosition())
    holder.photo.setImageResource(worker.getPhoto())
}
```

Как устроен RecyclerView.Adapter

```
override fun getItemCount(): Int {
    return workerList.size()
}
```

Как устроен RecyclerView.Adapter

```
val recyclerView = findViewById<RecyclerView>(R.id.recycler_view)

val layoutManager = LinearLayoutManager(this)

recyclerView.layoutManager = layoutManager

val workers = data.getWorkers()

val adapter = MyAdapter(workers)

recyclerView.adapter = adapter
```



Жизнь и смерть ViewHolder

В случае создания элемента:

1. LayoutManager дает запрос RecyclerView на элемент: getViewForPosition
2. RecyclerView обращается в Cache
3. Если в Cache нет элемента, RecyclerView идет в Adapter и просит у него viewType
4. Получив ViewType RecyclerView идет в Recycler Pool: getViewHolderByType, в котором может хранится элемент с неправильными данными, которые надо потом заполнить.
5.
    a. Если Recycler Pool не вернул элемент, то он идет в Adapter и просит создать элемент.
    b. Если Recycler Pool вернул элемент, то RecyclerView просит Adapter сделать onBindViewHolder (заполнить данные) и возвращает элемент LayoutManager’y

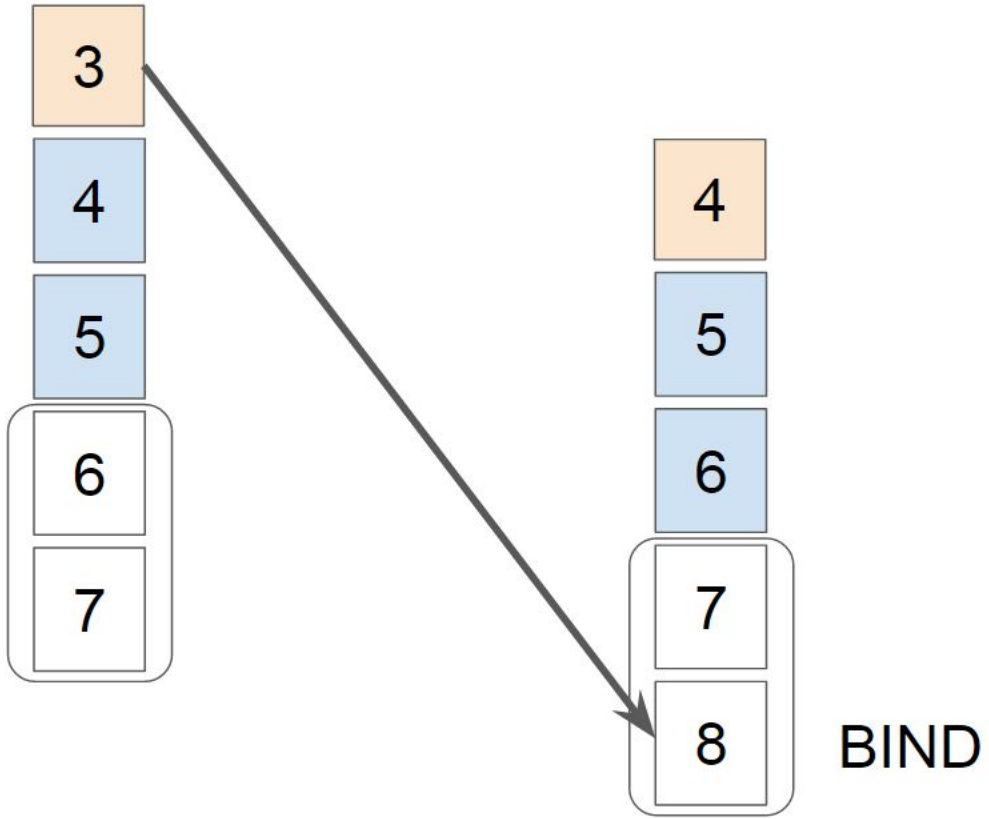
Жизнь и смерть ViewHolder

В случае удаления элемента:

1. LayoutManager сообщает RecyclerView о том, что удаляет элемент: removeAndRecycleView
2. RecyclerView посылает запрос Adapter'у: onViewDettachedToWindow
3. RecyclerView идет в Cache, если элемент валидный (isValid? А если не valid то сразу идет в Recycler Pool) для этой позиции, помещает туда элемент
4. Cache помещает старые элементы в Recycler Pool (recycle)
5. Recycled Pool сообщает Adapter'у, что элемент (view) был переиспользован и Adapter может сделать с ним что хочет: onViewRecycled (те элементы которые попадают в Recycled Pool, понадобятся нескоро, в отличии от тех, которые находятся в Cache)

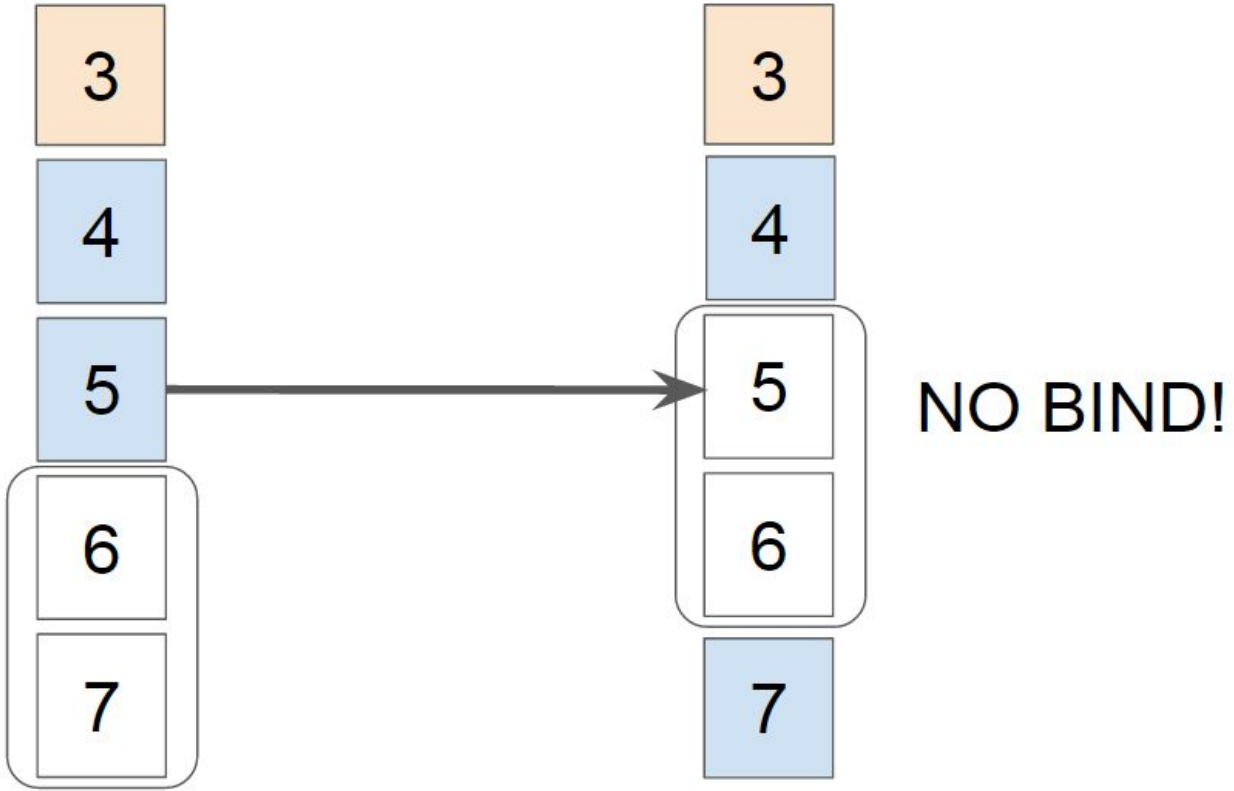
Переиспользуемые холдеры

Pool



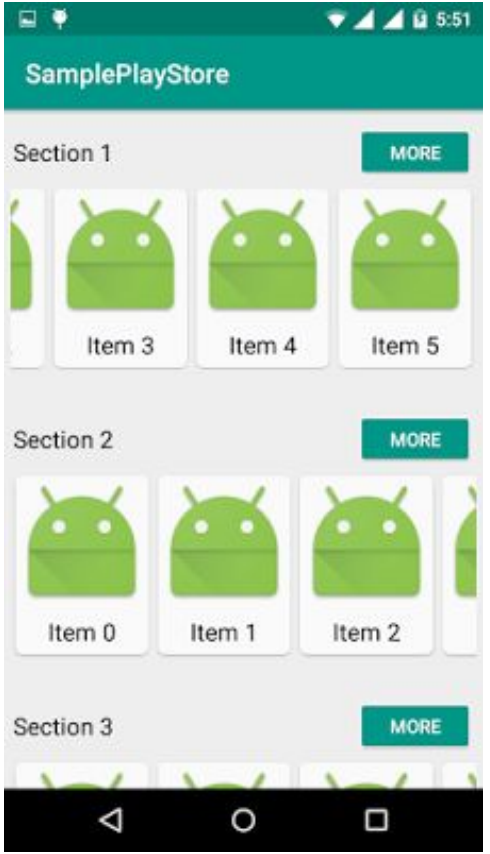
Переиспользуемые холдеры

Cache



ItemViewType

```
override fun getItemViewType(position: Int):
Int {
  return if (position == <условие>)
       return TYPE_ITEM1;
   else
       return TYPE_ITEM2;
}
```



ItemViewType

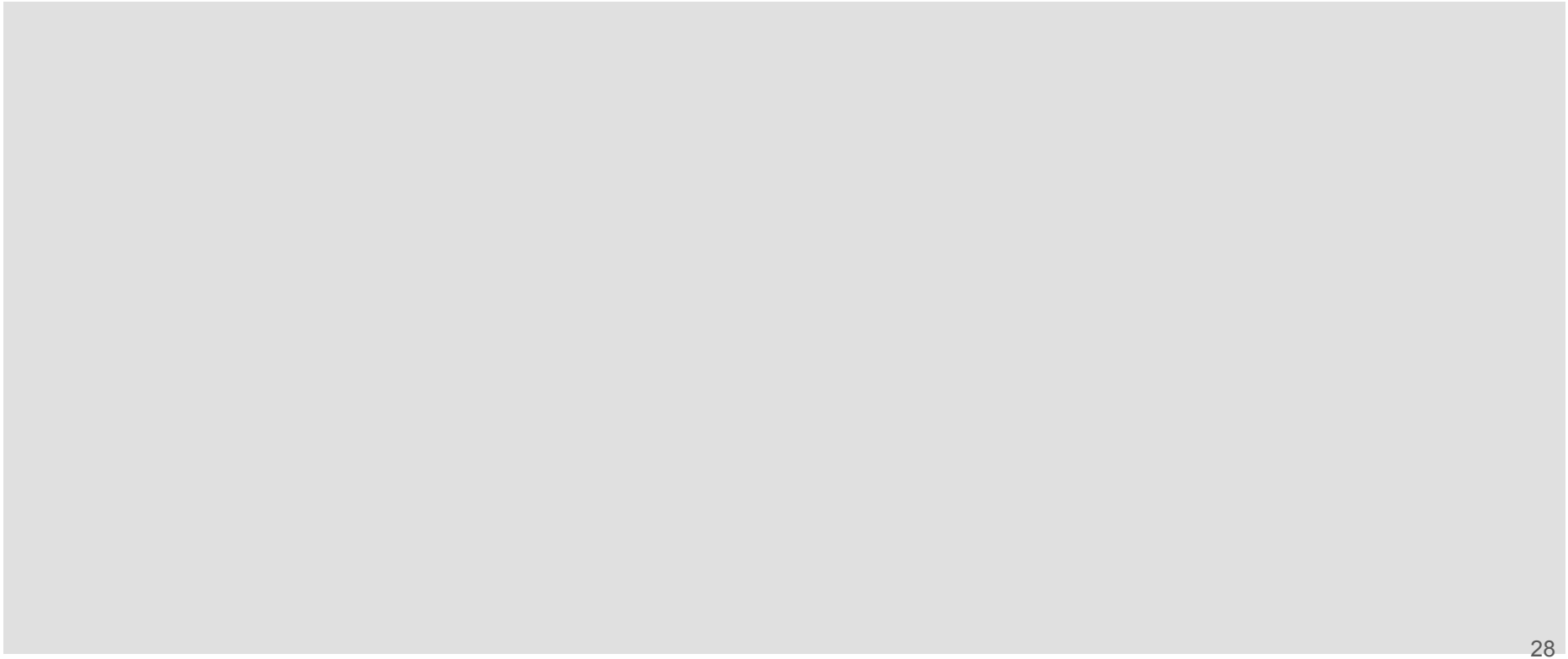
```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {
    val v: View = if (viewType == WITHOUT_PHOTO_TYPE) {
        LayoutInflater.from(parent.context)
                .inflate(R.layout.item_main_grid_small, parent, false)
    } else {
        LayoutInflater.from(parent.context)
                .inflate(R.layout.item_main_grid, parent, false)
    }
    return MyViewHolder(v)
}
```

ItemViewType

```
fun onBindViewHolder(holder: MyViewHolder, position: Int) {
    val type = getItemViewType(position)
    val worker = workerList[position]
    holder.apply {
        name.setText(worker.getName())
        age.setText("Age: " + worker.getAge())
    }
    when (type) {
        WITHOUT_PHOTO_TYPE -> {
            holder.photo.setVisibility(View.GONE)
            }
        WITH_PHOTO_TYPE -> {
            holder.photo.setVisibility(View.VISIBLE)
            Glide.with(ctx)
                    .load(worker.getPhoto())
                    .diskCacheStrategy(DiskCacheStrategy.SOURCE)
                    .into(holder.photo)
        }
    }
}
```

ItemTouchHelper

Drag & Swipe with RecyclerView



ItemTouchHelper

```
public abstract static class Callback {

        ……….

public abstract int getMovementFlags(RecyclerView recyclerView,
ViewHolder viewHolder);

public abstract boolean onMove(RecyclerView recyclerView,  ViewHolder
viewHolder, ViewHolder target);

public abstract void onSwiped(ViewHolder viewHolder, int direction);

        ……….

}
```

ItemTouchHelper

```
val item = object : ItemTouchHelper.SimpleCallback(ItemTouchHelper.UP or
ItemTouchHelper.DOWN,ItemTouchHelper.LEFT) {
    override fun onSwiped(viewHolder: RecyclerView.ViewHolder?, direction: Int) {
        // remove from adapter
    }
    override fun onMove(recyclerView: RecyclerView,
                        viewHolder: RecyclerView.ViewHolder,
                        target: RecyclerView.ViewHolder): Boolean {
            val fromPos = viewHolder.adapterPosition;
            val toPos = target.adapterPosition;
            // move item in `fromPos` to `toPos` in adapter.
            return true;// true if moved, false otherwise

        }
}
```

ItemTouchHelper

```
fun getMovementFlags(recyclerView: RecyclerView, viewHolder:
ViewHolder): Int {
    val dragFlags = ItemTouchHelper.UP or ItemTouchHelper.DOWN
    val swipeFlags = ItemTouchHelper.START or ItemTouchHelper.END

    return makeMovementFlags(dragFlags, swipeFlags)
}
```

ItemTouchHelper

```
interface ItemTouchHelperAdapter {

    fun onItemMove(fromPosition: Int, toPosition: Int)

    fun onItemDismiss(position: Int)

}
```

ItemTouchHelper

```
class SimpleItemTouchHelperCallback(private val mAdapter:
ItemTouchHelperAdapter) : ItemTouchHelper.Callback() {
```

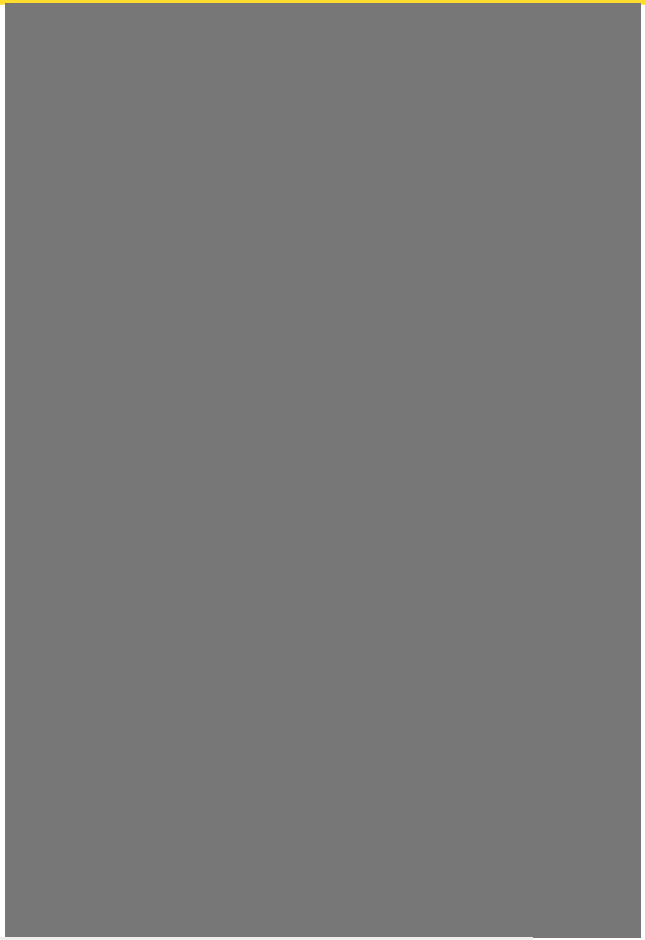
ItemTouchHelper

```
override fun onMove(recyclerView: RecyclerView,
                    viewHolder: RecyclerView.ViewHolder,
                    target: RecyclerView.ViewHolder): Boolean {
    mAdapter.onItemMove(viewHolder.adapterPosition, target.adapterPosition)
    return true
}

override fun onSwiped(viewHolder: RecyclerView.ViewHolder, direction: Int) {
    mAdapter.onItemDismiss(viewHolder.adapterPosition)
}
```

ItemTouchHelper

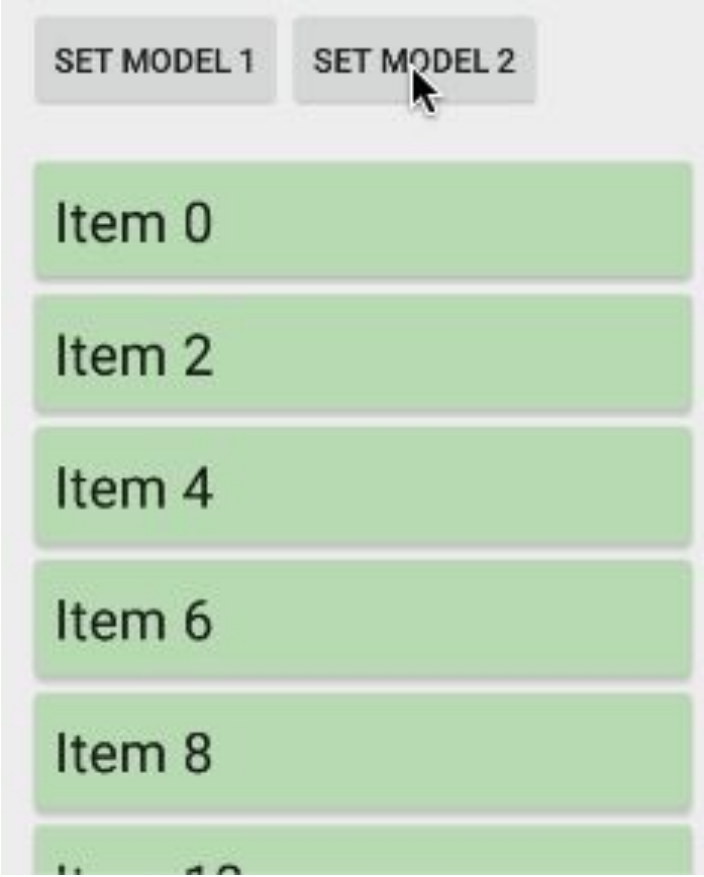
```
val callback = SimpleItemTouchHelperCallback(adapter)
val touchHelper = ItemTouchHelper(callback)
touchHelper.attachToRecyclerView(recyclerView)
```

ItemTouchHelper



ItemTouchHelper

А за счёт чего получается такая анимация?



NotifyDataSetChanged или DiffUtil?

- notifyItemChanged();
- notifyItemInserted();
- notifyItemMoved();
- notifyItemRemoved();
- notifyItemRangeChanged();
- notifyItemRangeInserted();
- notifyItemRangeMoved();
- notifyItemRangeRemoved();

VS *notifyDataSetChanged*()

DiffUtil

- *DiffUtil* находит разницу между двумя списками и предоставляет обновленный лист на выходе. Этот класс используется для оповещения *RecyclerView* Adapter об изменениях.
- Он использует Eugene W. Myers’s difference algorithm для подсчёта минимального количества обновлений.

DiffUtil

```
class EmployeeDiffCallback(private val mOldEmployeeList: List<Employee>,
                           private val mNewEmployeeList: List<Employee>) :
DiffUtil.Callback()
```

DiffUtil

```
override fun getOldListSize(): Int {
    return mOldEmployeeList.size
}

override fun getNewListSize(): Int {
    return mNewEmployeeList.size
}
```

DiffUtil

```
override fun areItemsTheSame(oldItemPosition: Int, newItemPosition: Int):
Boolean {
    return mOldEmployeeList[oldItemPosition].getId() ===
                                    mNewEmployeeList[newItemPosition].getId()
}
```

DiffUtil

```
override fun areContentsTheSame(oldItemPosition: Int,
newItemPosition: Int): Boolean {
    val oldEmployee = mOldEmployeeList[oldItemPosition]
    val newEmployee = mNewEmployeeList[newItemPosition]
    return oldEmployee.getName()
            .equals(newEmployee.getName())
}
```

DiffUtil

```
class CustomRecyclerViewAdapter :
RecyclerView.Adapter<CustomRecyclerViewAdapter.ViewHolder>() {

    fun updateEmployeeListItems(employees: List<Employee>) {
        val diffCallback = EmployeeDiffCallback(this.mEmployees, employees)
        val diffResult = DiffUtil.calculateDiff(diffCallback, false)

        diffResult.dispatchUpdatesTo(this)
    }
}
```

DiffUtil

ВАЖНО:

- Считать Diff на UI треде - зло
- Максимальный размер листа — 2^{26}

ItemAnimator

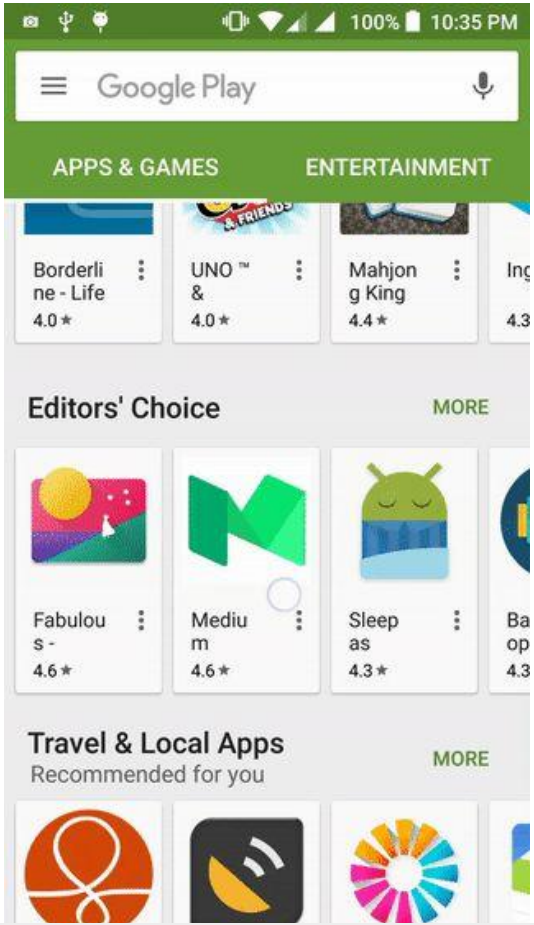
```
RecyclerView.ItemAnimator itemAnimator = new DefaultItemAnimator();
recyclerView.setItemAnimator(itemAnimator);
```

ItemAnimator

Recycler Dich

Nested RecyclerViews

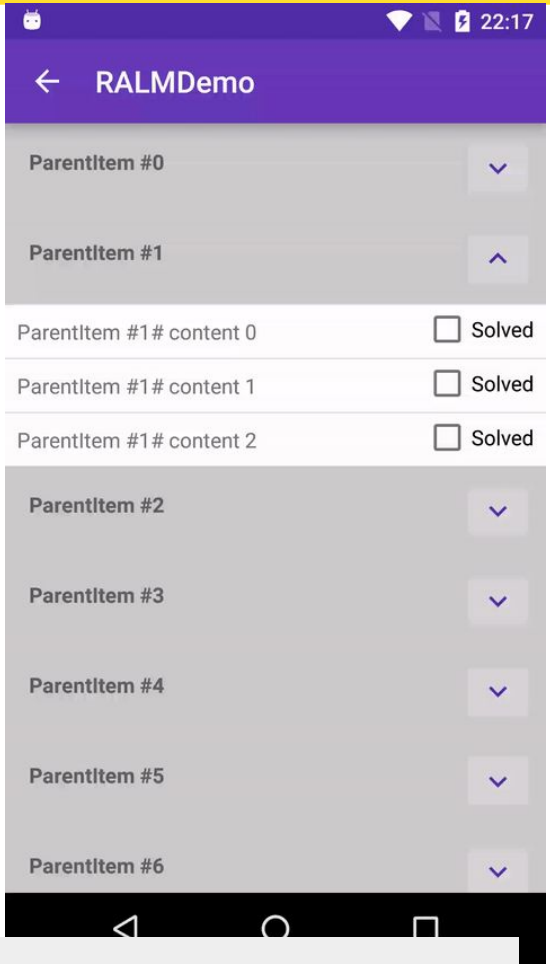
```
<RecyclerView> //vertical
    <RecyclerView/> //horizontal
</RecyclerView>
```



Recycler Dich

ExpandableRecyclerHeaderView

* Collapsed:
* - Group Header
*
* Expanded:
* - Group Header
* - Child
* - Child
* - etc



Резюме

- Recycler — круто
- LayoutManager’ы — гибкость
- RecyclerView.Adapter — просто
- ViewType — изменичивость
- ItemTouchHelper - удобно
- DiffUtil — научно
- Recycler Dich — бывает

Ссылки

- https://developer.android.com/training/material/lists-cards.html?hl=ru
- https://developer.android.com/guide/topics/ui/layout/recyclerview.html
- https://developer.android.com/reference/android/support/v7/widget/RecyclerView.html
- http://www.fandroid.info/primer-ispolzovaniya-cardview-i-recyclerview-v-android/
- https://medium.com/master-of-code-global/recyclerview-tips-and-recipes-476410fa12fd
- https://www.youtube.com/watch?v=860_Y5TXh1Y (смотреть обязательно)
- https://www.youtube.com/watch?v=h5afEeuI0GQ (смотреть обязательно)

Домашнее задание

- Реализовать adapter, выбрать любой LayoutManager и прикрутить это всё к RecyclerView
- Нажатием на fab можно добавить новые данные
- Тестовые данные для отображения генерятся WorkerGenerator
- Реализовать ItemTouchHelper (не использовать SimpleCallback) с Move и Swipe
- Реализовать ItemDecorator

https://github.com/Jacks0N23/homework_3_recyclerview

План семинара

- Viewtype’s
- LayoutManager’s
- bind во ViewHolder

https://github.com/Jacks0N23/lesson_3_recycler_view

Спасибо за внимание