Clayton Turner
CSIS 603 - Design Patterns

Strategy Pattern Assignment

Android Reference: http://developer.android.com/reference/android/support/v7/widget/StaggeredGridLayoutManager.html
Source Code: https://android.googlesource.com/platform/frameworks/support/+/0b1059b/v7/recyclerview/src/android/support/v7/widget/StaggeredGridLayoutManager.java

The strategy pattern is a pattern whose motivation to be used when different strategies, or algorithms, want to be used dynamically. For example, sales tax computation for invoices would need different strategies chosen as tax computations are different in different areas. As per Dr Moore's slides, the strategy pattern can be thought of as changing the "guts" of an object (as opposed to a decorator pattern which changes the "skin" of an object). The guts of the object that are changed is thought of as adding a strategy to a component which the client manages. The definitive intent of the strategy pattern is to define a family of algorithms, encapsulate those algorithms, and make them interchangeable, allowing the algorithm to vary independently from the clients that use it, according to the slides.

The strategy pattern is applicable, according to the slides, when:

- You want to provide a way to configure a class with one of many behaviors
- You need different variants of an algorithm for time/space tradeoffs
- An algorithm uses data that clients should not know about
- A class defines many behaviors and conditionals are to be avoided

The StaggeredGridLayoutManager is a subclass of LayoutManager, within the Android OS, which lays out children items in a staggered grid formation, according to the Android Reference. The strategy pattern used by this StaggeredGridLayoutManager is for managing gaps at the edges of the layout. The strategy involves offsetting gaps independently and/or moving items between different spans. Strategies are set using "public void setGapStrategy(int gapStrategy)" and the gapStrategy passed in is selected from a set of public static final ints defined in the StaggeredGridLayoutManager class (e.g. public static final int GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS = 2;) The many behaviors condition is satisfied as there are different manners in which you can handle gaps with StaggeredGridLayoutManager, which means that the strategy pattern is applicable to this situation as Android apps are built very differently all the time and gaps on layouts need to be handled differently, even within the same app.

```
/* Sets the gap handling strategy for StaggeredGridLayoutManager. If the gapStrategy parameter
* is different than the current strategy, calling this method will trigger a layout request.
*
* @param gapStrategy The new gap handling strategy. Should be {@link #GAP_HANDLING_LAZY}
* , {@link #GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS} or
* {@link #GAP_HANDLING_NONE}
* @see #getGapStrategy()
*/
public void setGapStrategy(int gapStrategy) {
    assertNotInLayoutOrScroll(null);
    if (mPendingSavedState != null && mPendingSavedState.mGapStrategy != gapStrategy) {
        mPendingSavedState.mGapStrategy = gapStrategy;
    }
    if (gapStrategy == mGapStrategy) {
        return;
    }
    if (gapStrategy != GAP_HANDLING_LAZY && gapStrategy != GAP_HANDLING_NONE &&
        gapStrategy != GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS) {
        throw new IllegalArgumentException("invalid gap strategy. Must be GAP_HANDLING_NONE "
            + ", GAP_HANDLING_LAZY or GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS");
    }
    mGapStrategy = gapStrategy;
    requestLayout();
}


/**
    * Returns the current gap handling strategy for StaggeredGridLayoutManager.
    * <p>
```

```
* Staggered grid may have gaps in the layout as items may have different sizes. To avoid gaps,

* StaggeredGridLayoutManager provides 3 options. Check {@link #GAP_HANDLING_NONE},

* {@link #GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS}, {@link #GAP_HANDLING_LAZY} for details.

* <p>

* By default, StaggeredGridLayoutManager uses {@link #GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS}.

*

* @return Current gap handling strategy.

* @see #setGapStrategy(int)

* @see #GAP_HANDLING_NONE

* @see #GAP_HANDLING_LAZY

* @see #GAP_HANDLING_MOVE_ITEMS_BETWEEN_SPANS

*/

public int getGapStrategy() {

    return mGapStrategy;

}
```