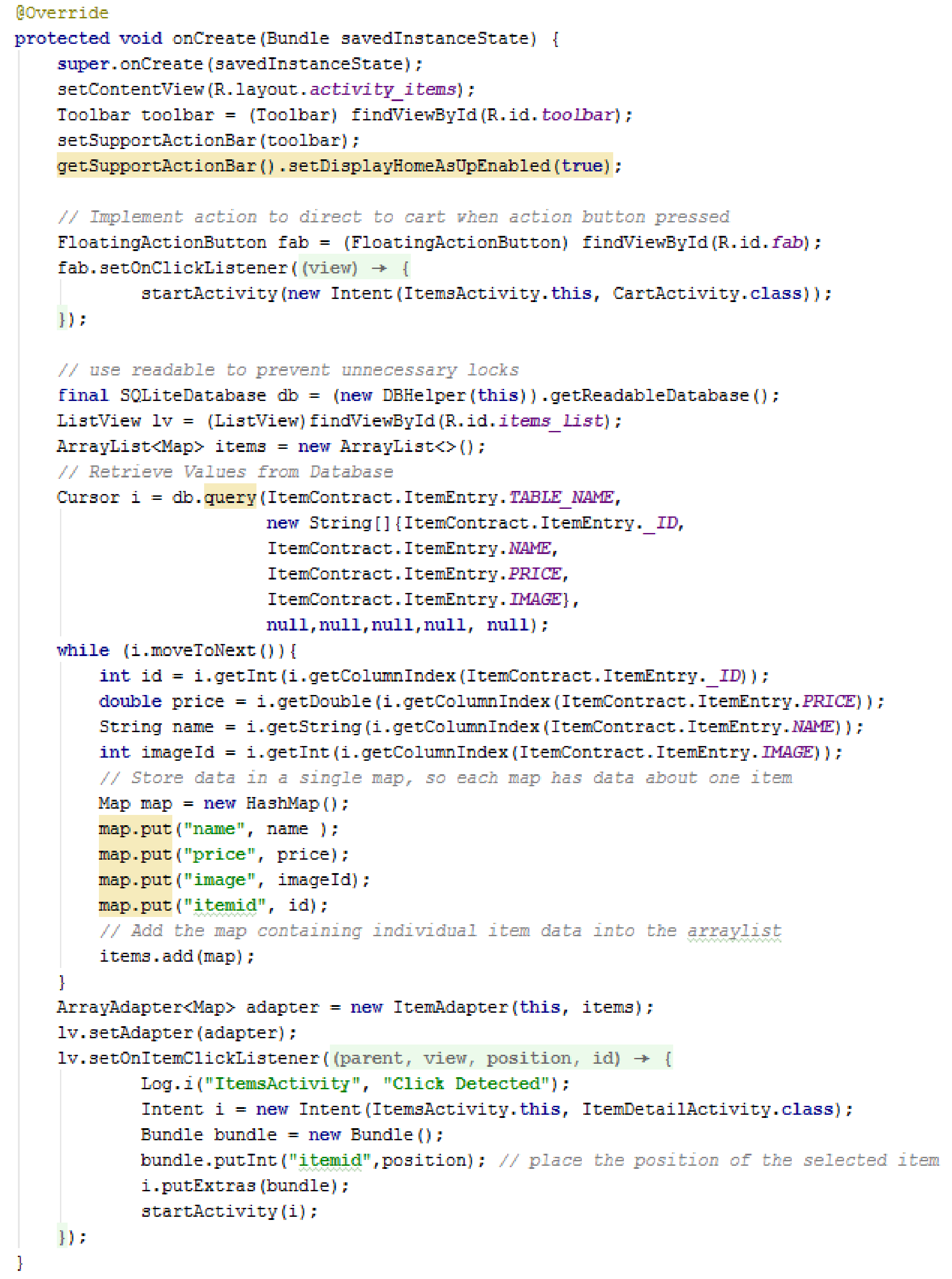
COMP3275
Lab 6
Threads & Databases

Kyle De Freitas

Department of Computing and Information Technology

University of the West Indies



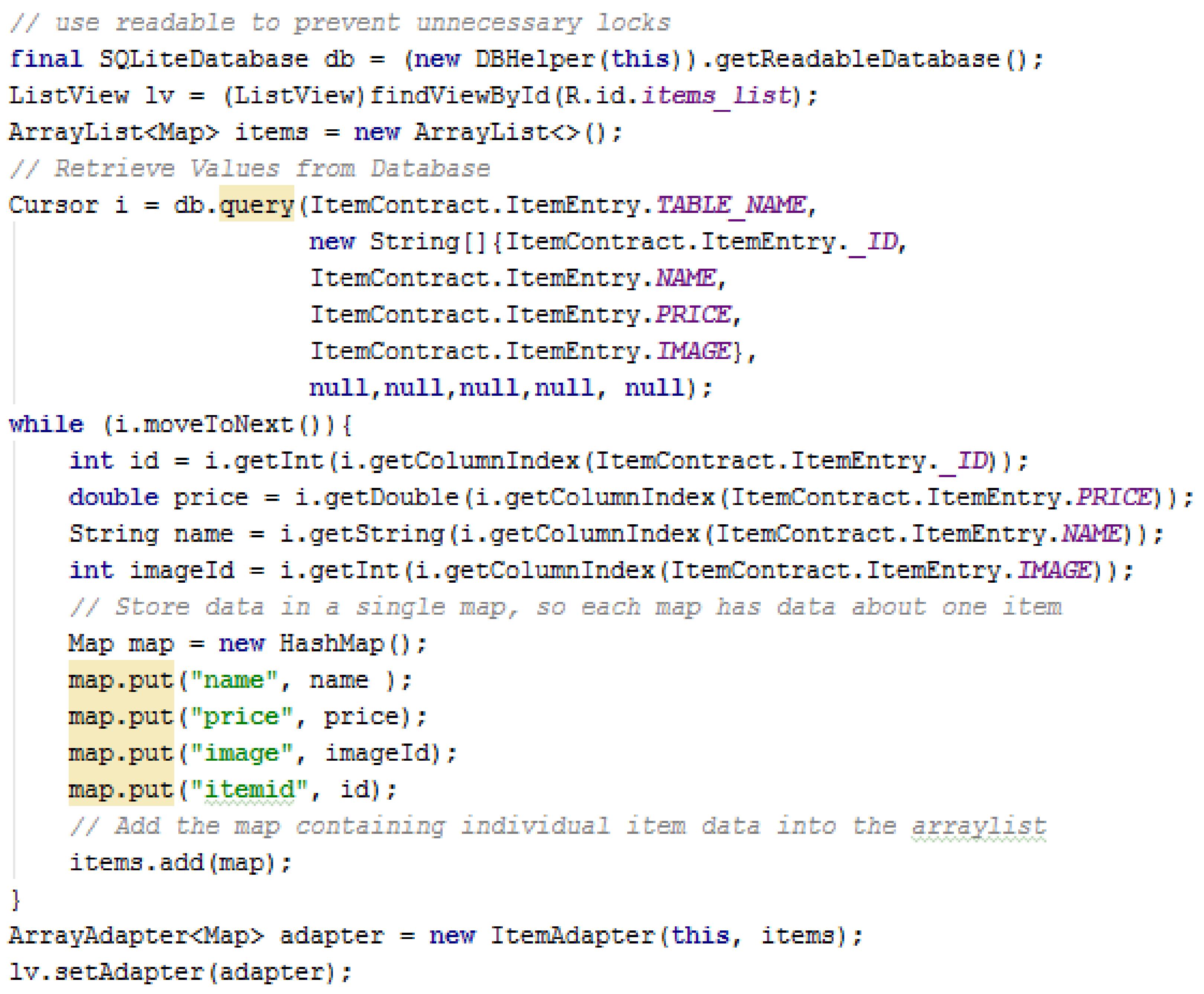
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_items);
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);

    // Implement action to direct to cart when action button pressed
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener((view) → {
            startActivity(new Intent(ItemsActivity.this, CartActivity.class));
    });

    // use readable to prevent unnecessary locks
    final SQLiteDatabase db = (new DBHelper(this)).getReadableDatabase();
    ListView lv = (ListView)findViewById(R.id.items_list);
    ArrayList<Map> items = new ArrayList<>();
    // Retrieve Values from Database
    Cursor i = db.query(ItemContract.ItemEntry.TABLE_NAME,
                        new String[]{ItemContract.ItemEntry._ID,
                        ItemContract.ItemEntry.NAME,
                        ItemContract.ItemEntry.PRICE,
                        ItemContract.ItemEntry.IMAGE},
                        null,null,null,null, null);
    while (i.moveToNext()){
        int id = i.getInt(i.getColumnIndex(ItemContract.ItemEntry._ID));
        double price = i.getDouble(i.getColumnIndex(ItemContract.ItemEntry.PRICE));
        String name = i.getString(i.getColumnIndex(ItemContract.ItemEntry.NAME));
        int imageId = i.getInt(i.getColumnIndex(ItemContract.ItemEntry.IMAGE));
        // Store data in a single map, so each map has data about one item
        Map map = new HashMap();
        map.put("name", name );
        map.put("price", price);
        map.put("image", imageId);
        map.put("itemid", id);
        // Add the map containing individual item data into the arraylist
        items.add(map);
    }
    ArrayAdapter<Map> adapter = new ItemAdapter(this, items);
    lv.setAdapter(adapter);
    lv.setOnItemClickListener((parent, view, position, id) → {
            Log.i("ItemsActivity", "Click Detected");
            Intent i = new Intent(ItemsActivity.this, ItemDetailActivity.class);
            Bundle bundle = new Bundle();
            bundle.putInt("itemid",position); // place the position of the selected item
            i.putExtras(bundle);
            startActivity(i);
    });
}
```

Currently the ItemsActivity class onCreate method is given on the left. It currently perform the following general tasks:

1. Associates the layout xml ‘activity_items’ with this class
2. Sets up the toolbars and the action bar
3. Specifies the command to run when the floating button is selected
4. Accesses the database
5. Perform a select operation on the database
6. Load each record as a hashmap
7. Store each hashmap (representing the record) into an array list
8. Use the array list to populate the custom adapater to display on the UI



```java
// use readable to prevent unnecessary locks
final SQLiteDatabase db = (new DBHelper(this)).getReadableDatabase();
ListView lv = (ListView)findViewById(R.id.items_list);
ArrayList<Map> items = new ArrayList<>();
// Retrieve Values from Database
Cursor i = db.query(ItemContract.ItemEntry.TABLE_NAME,
                    new String[]{ItemContract.ItemEntry._ID,
                    ItemContract.ItemEntry.NAME,
                    ItemContract.ItemEntry.PRICE,
                    ItemContract.ItemEntry.IMAGE},
                    null,null,null,null, null);
while (i.moveToNext()){
    int id = i.getInt(i.getColumnIndex(ItemContract.ItemEntry._ID));
    double price = i.getDouble(i.getColumnIndex(ItemContract.ItemEntry.PRICE));
    String name = i.getString(i.getColumnIndex(ItemContract.ItemEntry.NAME));
    int imageId = i.getInt(i.getColumnIndex(ItemContract.ItemEntry.IMAGE));
    // Store data in a single map, so each map has data about one item
    Map map = new HashMap();
    map.put("name", name );
    map.put("price", price);
    map.put("image", imageId);
    map.put("itemid", id);
    // Add the map containing individual item data into the arraylist
    items.add(map);
}
ArrayAdapter<Map> adapter = new ItemAdapter(this, items);
lv.setAdapter(adapter);
```

Focusing on the database related code:

1. As we mentioned before database operations can be considered computationally expensive and may take a while to complete (depends on the number of items in the database)
2. These computations and time for the tasks to complete can result in the UI becoming responsive which is undesirable for the user.
3. The solution is to place this code (that retrieves from the database and displays on the UI) within a background thread.

Threads

- A thread of execution is the sequence of programmed instructions managed by the OS’s scheduler.
- Within the android system, a launched application posses a thread of execution called “main”.
- This “main” thread also referred to as the “UI thread” is used to control the states for all components instantiated within itself.
  - Therefore all activities and onclick listeners are executed within this thread.

Threads

- When your app performs intensive work in response to user interaction, this single thread model can yield poor performance unless you implement your application properly.
- Specifically, if everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI.
- When the thread is blocked, no events can be dispatched, including drawing events.
- From the user's perspective, the application appears to hang. Even worse, if the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog.
- The user might then decide to quit your application and uninstall it if they are unhappy.

Extracted from verbatim from: http://developer.android.com/guide/components/processes-and-threads.html

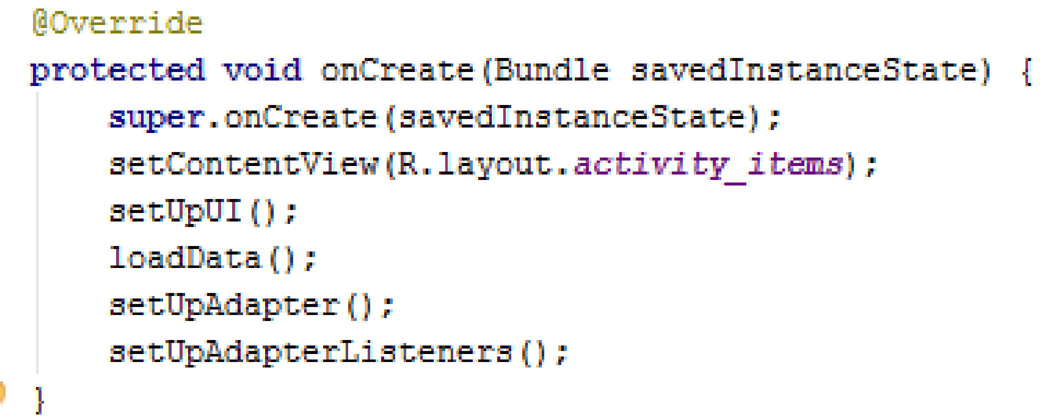
Threads

- Within the Android System, it provides two main ways of creating and utilizing threads within the application:
  - Worker threads
  - Async Task
- Async Tasks:
  - Allow you to perform asynchronous work on your user interface. It performs the blocking operations in a worker thread (doInBackground method) and then publishes the results on the UI without requiring you to handle threads or handlers yourself.
  - For an example of the difference you can refer to the documentation
- For this lab we will be focusing on the worker thread, we will cover async task at another time.

Threads

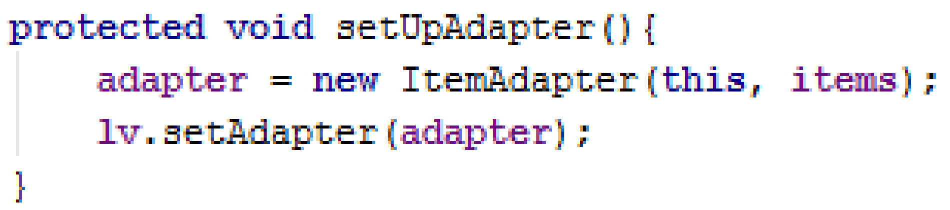
- Additionally, the Android UI toolkit is *not* thread-safe. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread. Thus, there are simply two rules to Android's single thread model:
  - Do not block the UI thread
  - Do not access the Android UI toolkit from outside the UI thread

Extracted from verbatim from: http://developer.android.com/guide/components/processes-and-threads.html

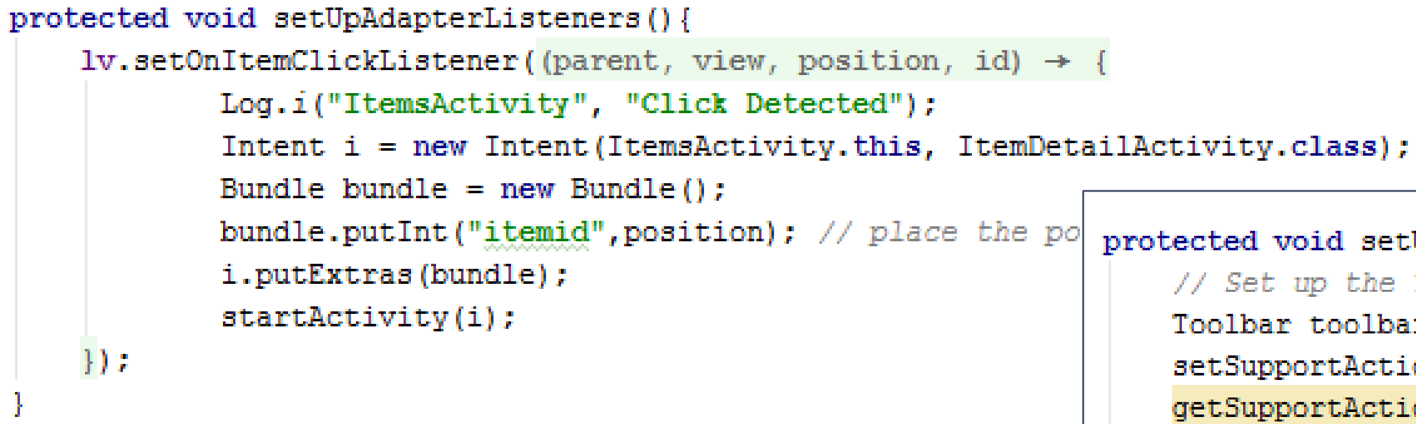


```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_items);
    setUpUI();
    loadData();
    setUpAdapter();
    setUpAdapterListeners();
}
```

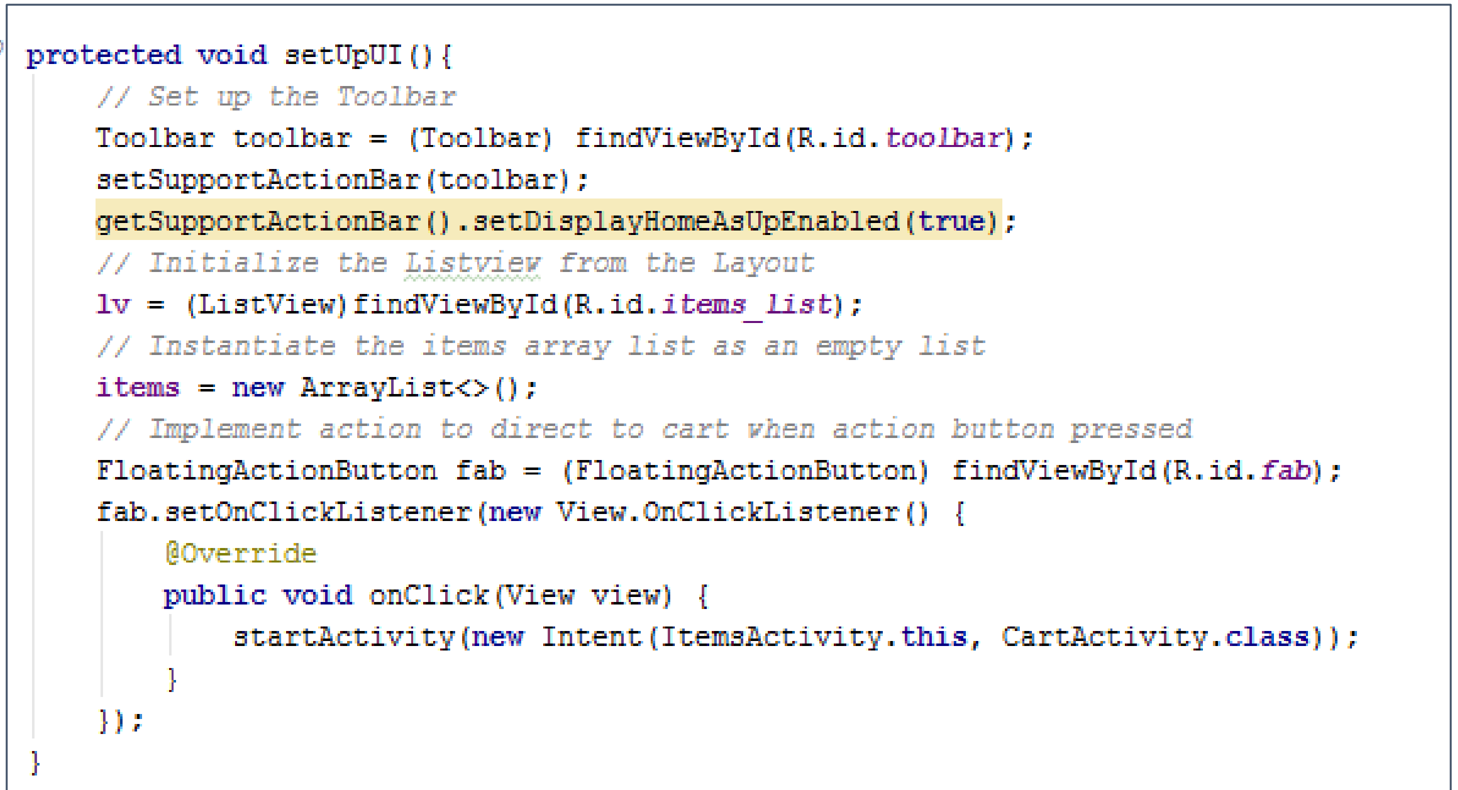
We want to break up the onCreate within the ItemsActivity into smaller parts (based on functionality) to make the code easier to understand and easier to manage.



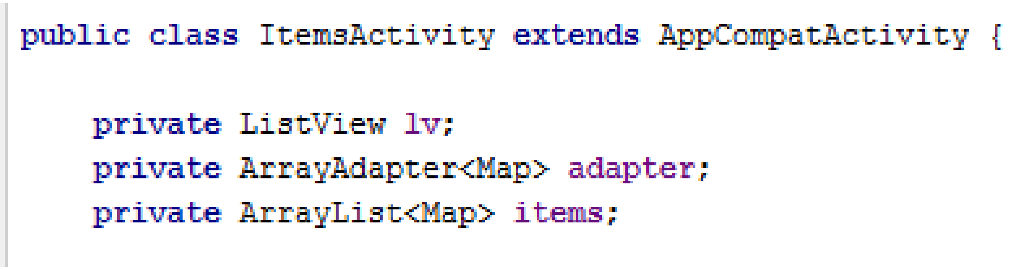
```java
protected void setUpAdapter(){
    adapter = new ItemAdapter(this, items);
    lv.setAdapter(adapter);
}
```



```java
protected void setUpAdapterListeners(){
    lv.setOnItemClickListener((parent, view, position, id) → {
            Log.i("ItemsActivity", "Click Detected");
            Intent i = new Intent(ItemsActivity.this, ItemDetailActivity.class);
            Bundle bundle = new Bundle();
            bundle.putInt("itemid",position); // place the po
            i.putExtras(bundle);
            startActivity(i);
    });
}
```



```java
protected void setUpUI(){
    // Set up the Toolbar
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    // Initialize the Listview from the Layout
    lv = (ListView)findViewById(R.id.items_list);
    // Instantiate the items array list as an empty list
    items = new ArrayList<>();
    // Implement action to direct to cart when action button pressed
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            startActivity(new Intent(ItemsActivity.this, CartActivity.class));
        }
    });
}
```

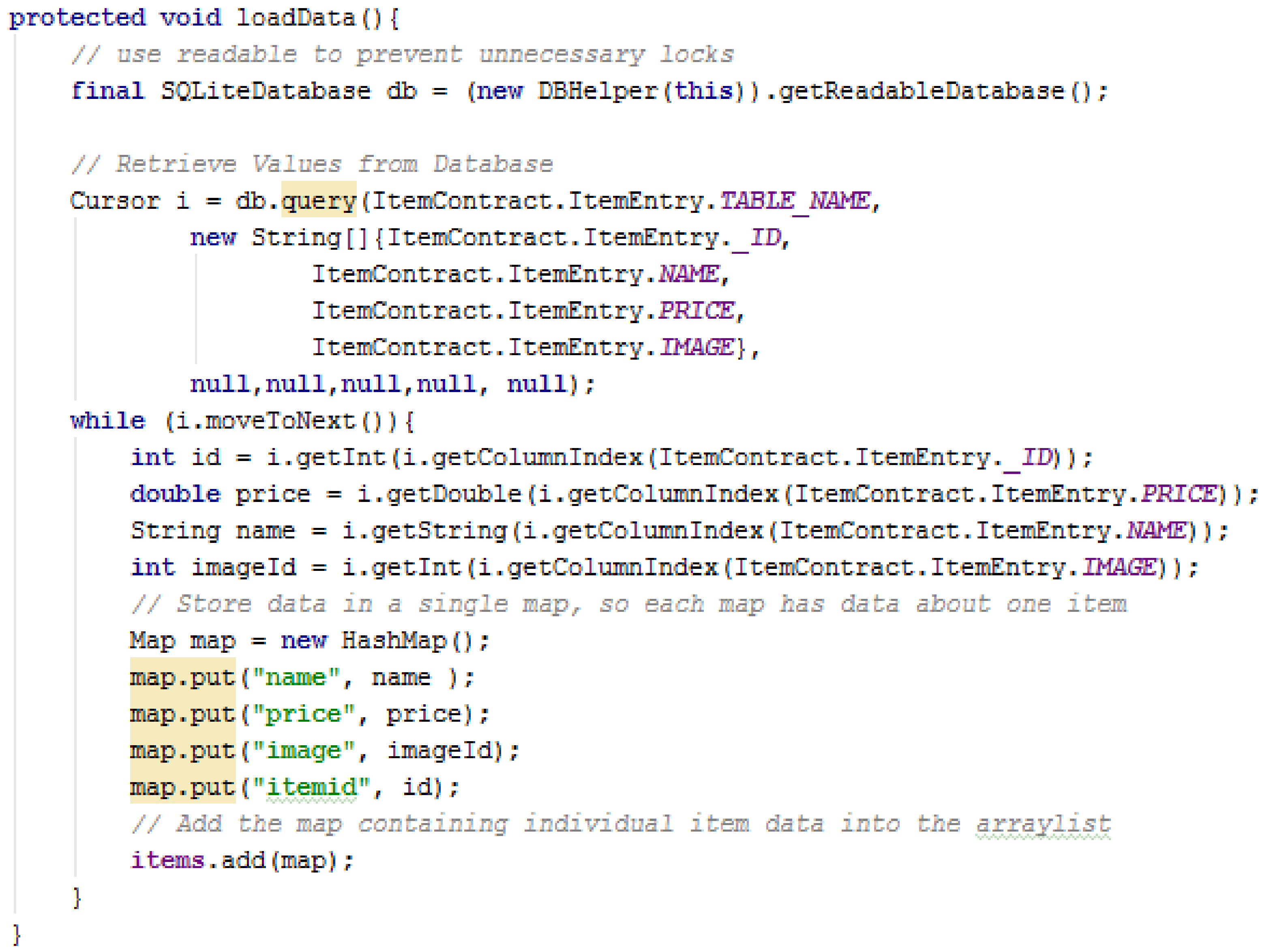


```java
public class ItemsActivity extends AppCompatActivity {

    private ListView lv;
    private ArrayAdapter<Map> adapter;
    private ArrayList<Map> items;
```

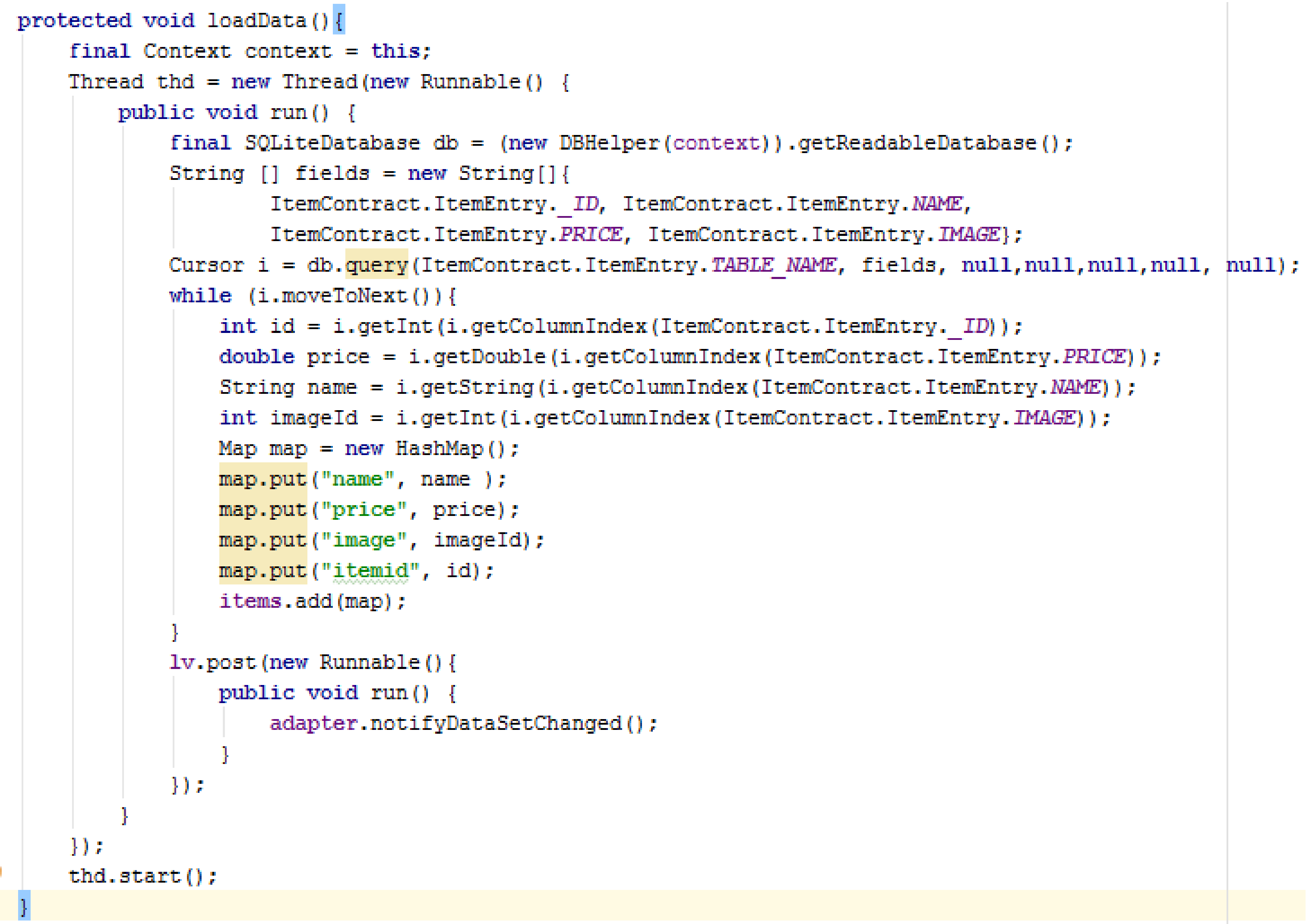
Moved from variables in the onCreate method to properties of the class

Importantly we separate the code to load the data into its own method



```java
protected void loadData(){
    // use readable to prevent unnecessary locks
    final SQLiteDatabase db = (new DBHelper(this)).getReadableDatabase();

    // Retrieve Values from Database
    Cursor i = db.query(ItemContract.ItemEntry.TABLE_NAME,
            new String[]{ItemContract.ItemEntry._ID,
                    ItemContract.ItemEntry.NAME,
                    ItemContract.ItemEntry.PRICE,
                    ItemContract.ItemEntry.IMAGE},
            null,null,null,null, null);
    while (i.moveToNext()){
        int id = i.getInt(i.getColumnIndex(ItemContract.ItemEntry._ID));
        double price = i.getDouble(i.getColumnIndex(ItemContract.ItemEntry.PRICE));
        String name = i.getString(i.getColumnIndex(ItemContract.ItemEntry.NAME));
        int imageId = i.getInt(i.getColumnIndex(ItemContract.ItemEntry.IMAGE));
        // Store data in a single map, so each map has data about one item
        Map map = new HashMap();
        map.put("name", name );
        map.put("price", price);
        map.put("image", imageId);
        map.put("itemid", id);
        // Add the map containing individual item data into the arraylist
        items.add(map);
    }
}
```



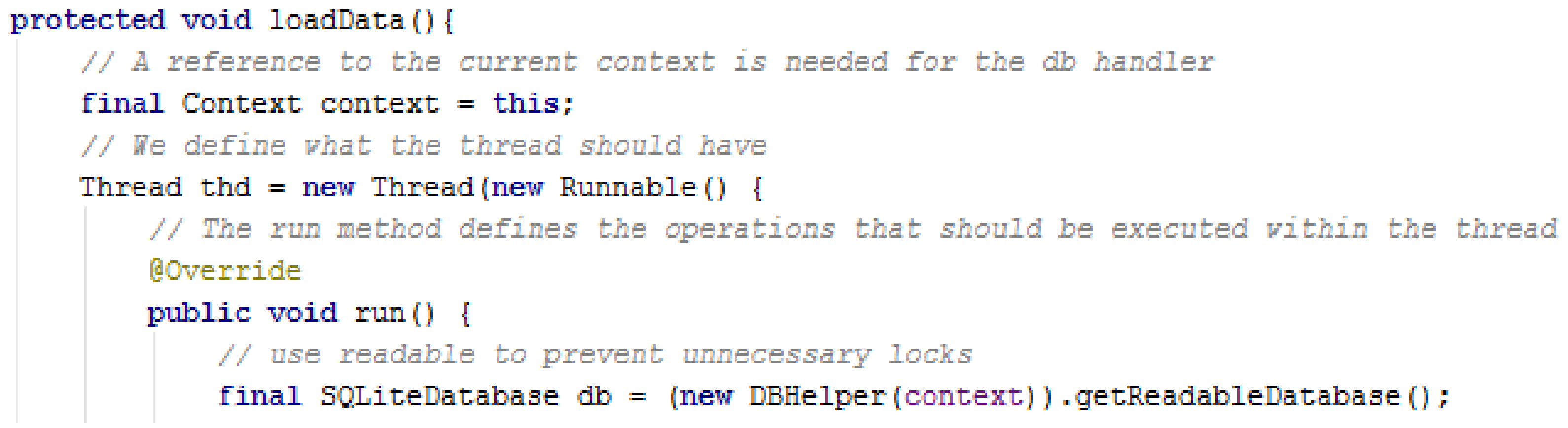
```java
protected void loadData(){
    final Context context = this;
    Thread thd = new Thread(new Runnable() {
        public void run() {
            final SQLiteDatabase db = (new DBHelper(context)).getReadableDatabase();
            String [] fields = new String[]{
                    ItemContract.ItemEntry._ID, ItemContract.ItemEntry.NAME,
                    ItemContract.ItemEntry.PRICE, ItemContract.ItemEntry.IMAGE};
            Cursor i = db.query(ItemContract.ItemEntry.TABLE_NAME, fields, null,null,null,null, null);
            while (i.moveToNext()){
                int id = i.getInt(i.getColumnIndex(ItemContract.ItemEntry._ID));
                double price = i.getDouble(i.getColumnIndex(ItemContract.ItemEntry.PRICE));
                String name = i.getString(i.getColumnIndex(ItemContract.ItemEntry.NAME));
                int imageId = i.getInt(i.getColumnIndex(ItemContract.ItemEntry.IMAGE));
                Map map = new HashMap();
                map.put("name", name );
                map.put("price", price);
                map.put("image", imageId);
                map.put("itemid", id);
                items.add(map);
            }
            lv.post(new Runnable(){
                public void run() {
                    adapter.notifyDataSetChanged();
                }
            });
        }
    });
    thd.start();
}
```

Now we can utilize the worker thread to load the data in a background operation.

We do this by creating a new thread with a new runnable class defined within it.

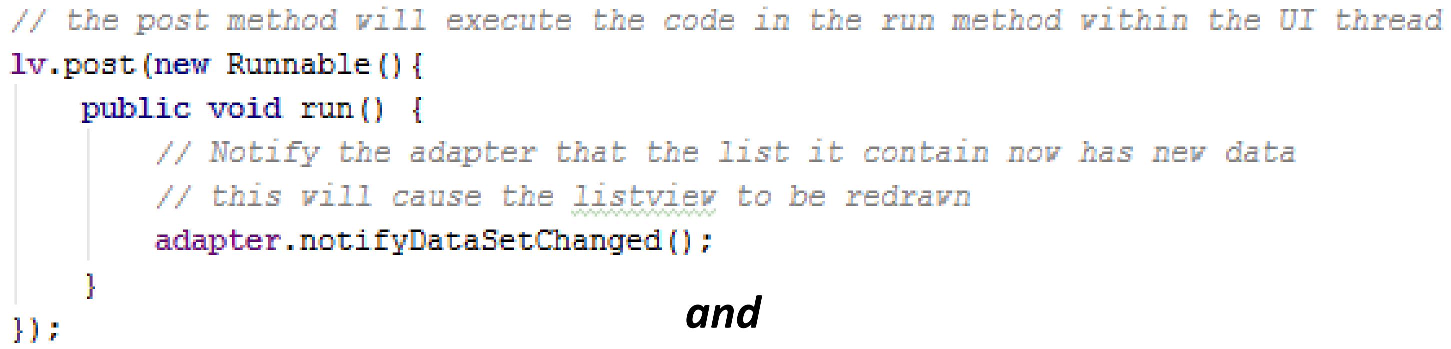
This is illustrated to the left

Importantly:



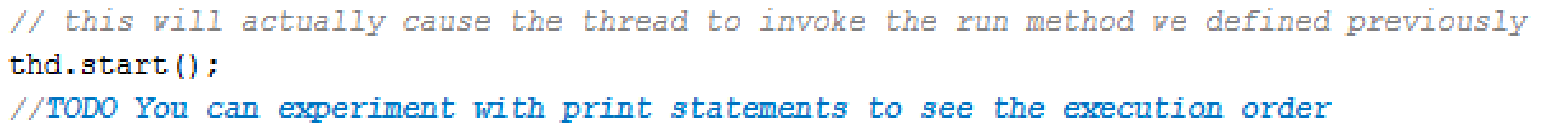
```java
protected void loadData(){
    // A reference to the current context is needed for the db handler
    final Context context = this;
    // We define what the thread should have
    Thread thd = new Thread(new Runnable() {
        // The run method defines the operations that should be executed within the thread
        @Override
        public void run() {
            // use readable to prevent unnecessary locks
            final SQLiteDatabase db = (new DBHelper(context)).getReadableDatabase();
```

and

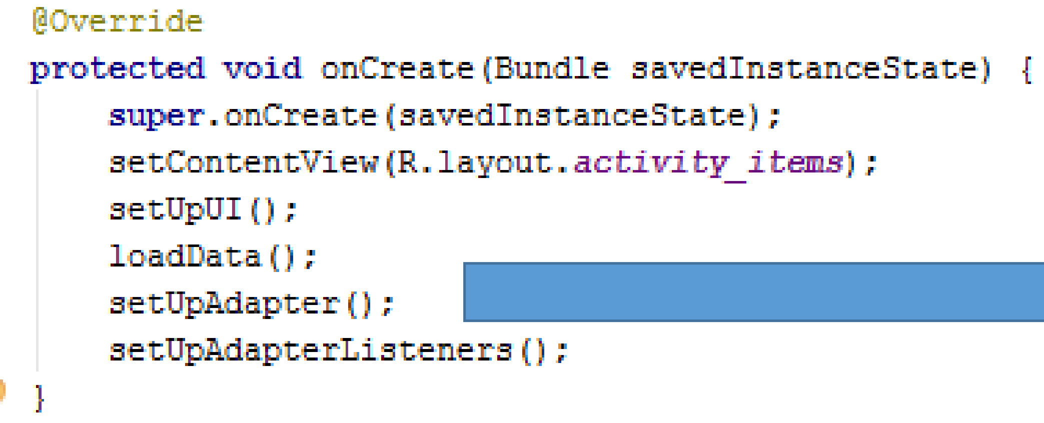


```java
// the post method will execute the code in the run method within the UI thread
lv.post(new Runnable(){
    public void run() {
        // Notify the adapter that the list it contain now has new data
        // this will cause the listview to be redrawn
        adapter.notifyDataSetChanged();
    }
});
```

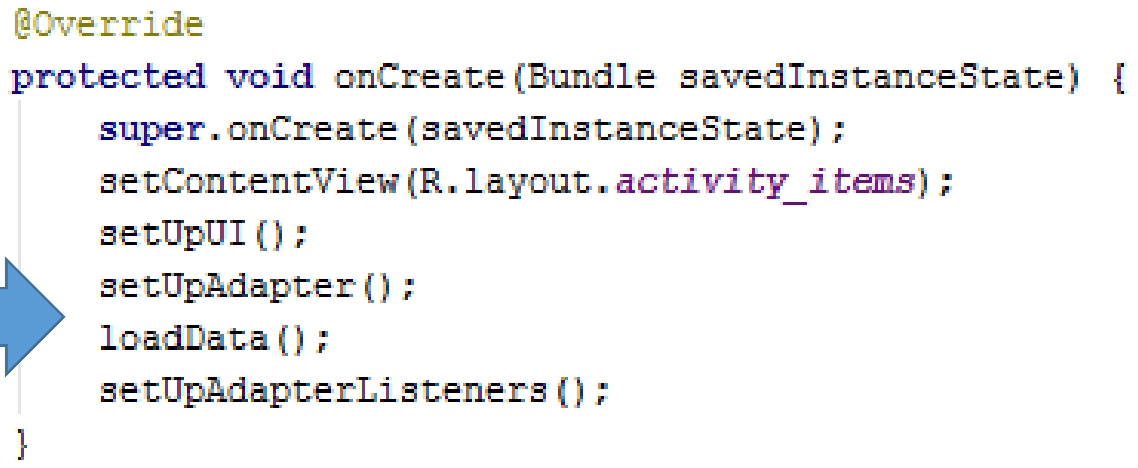
and



```java
// this will actually cause the thread to invoke the run method we defined previously
thd.start();
//TODO You can experiment with print statements to see the execution order
```

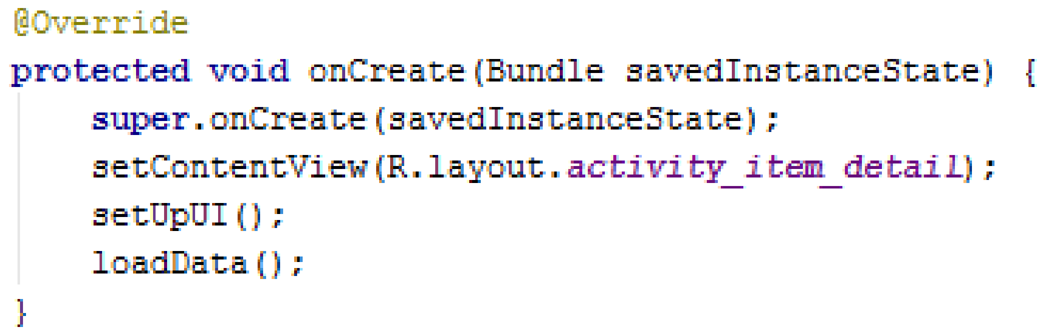


```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_items);
    setUpUI();
    loadData();
    setUpAdapter();
    setUpAdapterListeners();
}
```

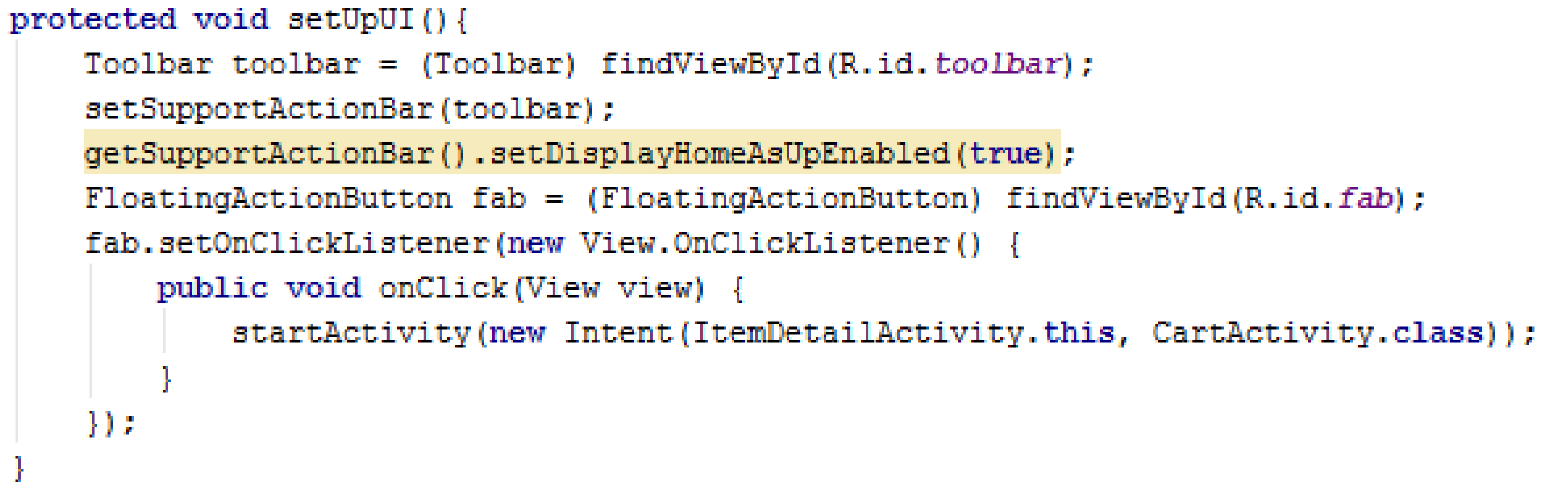


```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_items);
    setUpUI();
    setUpAdapter();
    loadData();
    setUpAdapterListeners();
}
```

NB: Ideally the setUpAdapter should come before the loadData method (within the onCreate method), this helps to ensure that the adapter is created before the thread tries to called its notifyDataChanged method



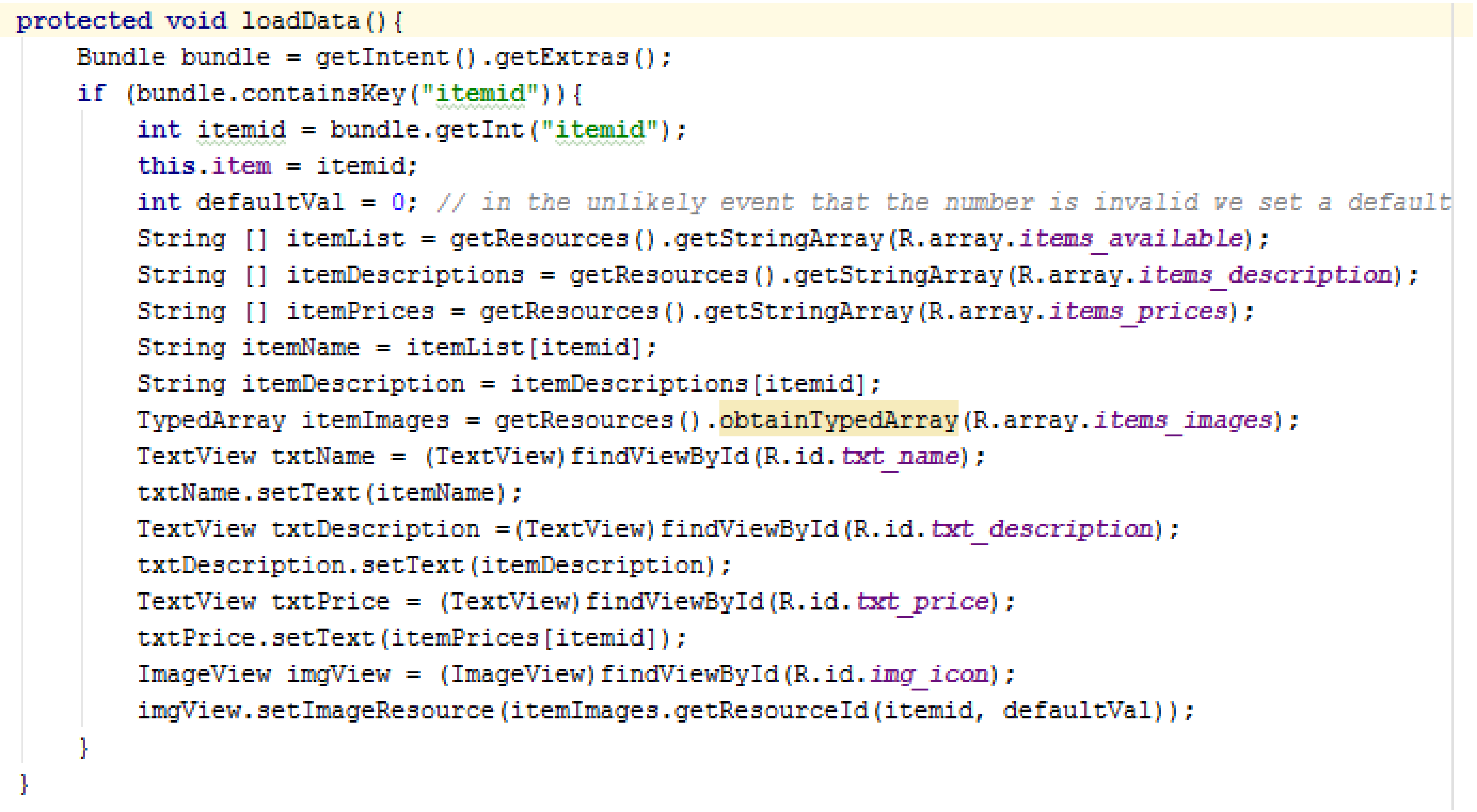
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_item_detail);
    setUpUI();
    loadData();
}
```



```
protected void setUpUI(){
    Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
    setSupportActionBar(toolbar);
    getSupportActionBar().setDisplayHomeAsUpEnabled(true);
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
            startActivity(new Intent(ItemDetailActivity.this, CartActivity.class));
        }
    });
}
```

Within the ItemDetailActivity, it still uses the incorrect strategy of the XML files, we now want to convert this strategy to use the database within threads

First we break up the onCreate method into other methods similar to action in ItemsActivity

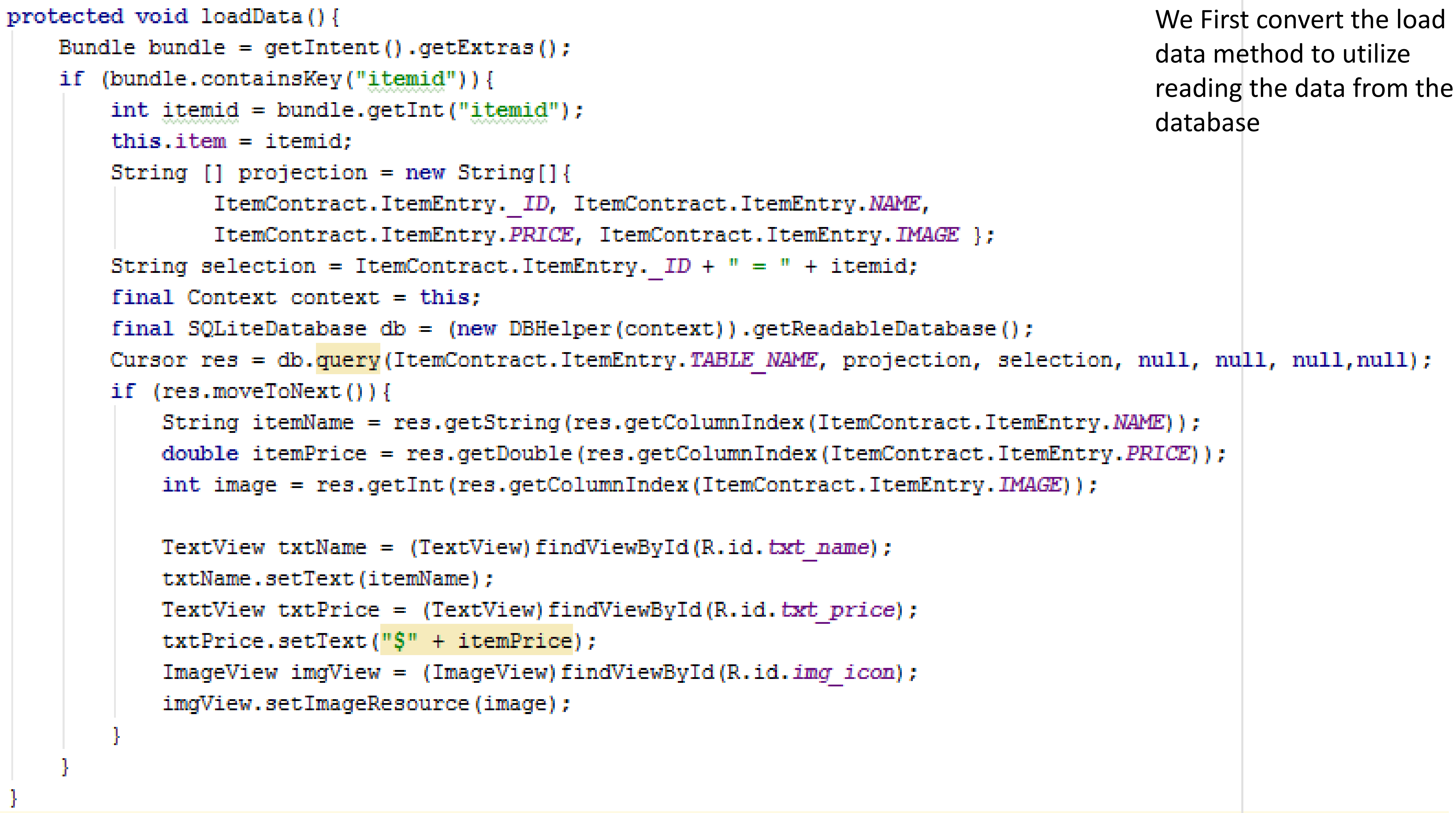


```
protected void loadData(){
    Bundle bundle = getIntent().getExtras();
    if (bundle.containsKey("itemid")){
        int itemid = bundle.getInt("itemid");
        this.item = itemid;
        int defaultVal = 0; // in the unlikely event that the number is invalid we set a default
        String [] itemList = getResources().getStringArray(R.array.items_available);
        String [] itemDescriptions = getResources().getStringArray(R.array.items_description);
        String [] itemPrices = getResources().getStringArray(R.array.items_prices);
        String itemName = itemList[itemid];
        String itemDescription = itemDescriptions[itemid];
        TypedArray itemImages = getResources().obtainTypedArray(R.array.items_images);
        TextView txtName = (TextView)findViewById(R.id.txt_name);
        txtName.setText(itemName);
        TextView txtDescription =(TextView)findViewById(R.id.txt_description);
        txtDescription.setText(itemDescription);
        TextView txtPrice = (TextView)findViewById(R.id.txt_price);
        txtPrice.setText(itemPrices[itemid]);
        ImageView imgView = (ImageView)findViewById(R.id.img_icon);
        imgView.setImageResource(itemImages.getResourceId(itemid, defaultVal));
    }
}
```

Within the ItemDetailActivity, it still uses the incorrect strategy of the XML files, we now want to convert this strategy to use the database within threads

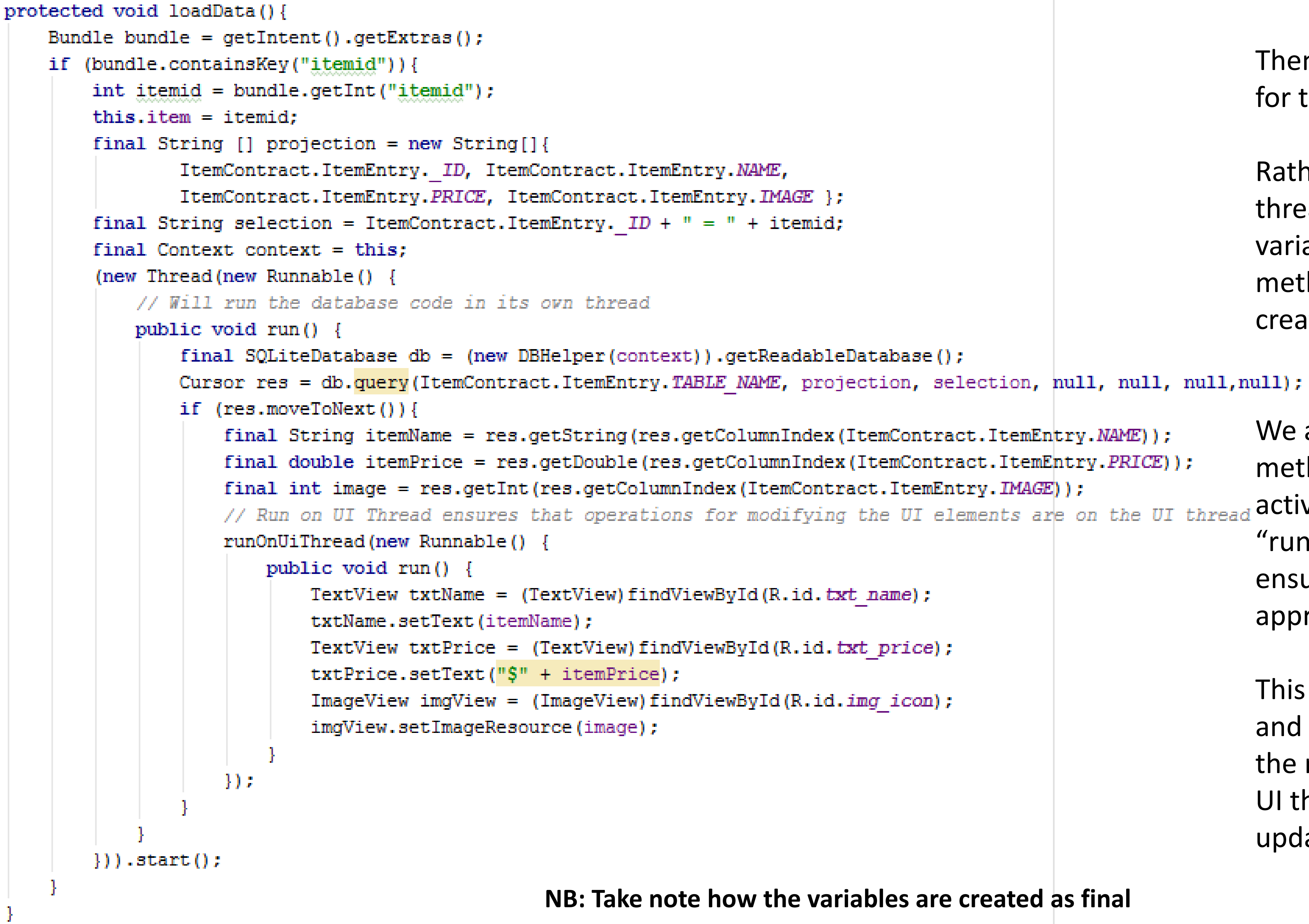
First we break up the onCreate method into other methods similar to action in ItemsActivity

We First convert the load data method to utilize reading the data from the database



```java
protected void loadData(){
    Bundle bundle = getIntent().getExtras();
    if (bundle.containsKey("itemid")){
        int itemid = bundle.getInt("itemid");
        this.item = itemid;
        String [] projection = new String[]{
                ItemContract.ItemEntry._ID, ItemContract.ItemEntry.NAME,
                ItemContract.ItemEntry.PRICE, ItemContract.ItemEntry.IMAGE };
        String selection = ItemContract.ItemEntry._ID + " = " + itemid;
        final Context context = this;
        final SQLiteDatabase db = (new DBHelper(context)).getReadableDatabase();
        Cursor res = db.query(ItemContract.ItemEntry.TABLE_NAME, projection, selection, null, null, null,null);
        if (res.moveToNext()){
            String itemName = res.getString(res.getColumnIndex(ItemContract.ItemEntry.NAME));
            double itemPrice = res.getDouble(res.getColumnIndex(ItemContract.ItemEntry.PRICE));
            int image = res.getInt(res.getColumnIndex(ItemContract.ItemEntry.IMAGE));

            TextView txtName = (TextView)findViewById(R.id.txt_name);
            txtName.setText(itemName);
            TextView txtPrice = (TextView)findViewById(R.id.txt_price);
            txtPrice.setText("$" + itemPrice);
            ImageView imgView = (ImageView)findViewById(R.id.img_icon);
            imgView.setImageResource(image);
        }
    }
}
```



```java
protected void loadData(){
    Bundle bundle = getIntent().getExtras();
    if (bundle.containsKey("itemid")){
        int itemid = bundle.getInt("itemid");
        this.item = itemid;
        final String [] projection = new String[]{
                ItemContract.ItemEntry._ID, ItemContract.ItemEntry.NAME,
                ItemContract.ItemEntry.PRICE, ItemContract.ItemEntry.IMAGE };
        final String selection = ItemContract.ItemEntry._ID + " = " + itemid;
        final Context context = this;
        (new Thread(new Runnable() {
            // Will run the database code in its own thread
            public void run() {
                final SQLiteDatabase db = (new DBHelper(context)).getReadableDatabase();
                Cursor res = db.query(ItemContract.ItemEntry.TABLE_NAME, projection, selection, null, null, null,null);
                if (res.moveToNext()){
                    final String itemName = res.getString(res.getColumnIndex(ItemContract.ItemEntry.NAME));
                    final double itemPrice = res.getDouble(res.getColumnIndex(ItemContract.ItemEntry.PRICE));
                    final int image = res.getInt(res.getColumnIndex(ItemContract.ItemEntry.IMAGE));
                    // Run on UI Thread ensures that operations for modifying the UI elements are on the UI thread
                    runOnUiThread(new Runnable() {
                        public void run() {
                            TextView txtName = (TextView)findViewById(R.id.txt_name);
                            txtName.setText(itemName);
                            TextView txtPrice = (TextView)findViewById(R.id.txt_price);
                            txtPrice.setText("$" + itemPrice);
                            ImageView imgView = (ImageView)findViewById(R.id.img_icon);
                            imgView.setImageResource(image);
                        }
                    });
                }
            }
        })).start();
    }
}
```

Then we add the strategy for the thread workers

Rather than storing the thread in a separate variable, we call the start method upon the actual creation of the thread.

We also now utilize another method (this time of the activity class) called “runOnUiThread” that will ensure code runs appropriately

This also accepts a runnable and will execute the code in the run method within the UI thread where it is safe to update the UI components

NB: Take note how the variables are created as final

Extra (optional)

- Convert the thread worker strategies to utilize the Async task instead.
  - Using the following as a guide:
    - http://developer.android.com/guide/components/processes-and-threads.html#AsyncTask
    - http://developer.android.com/reference/android/os/AsyncTask.html
- Rather than using a map to store the data create a class that will store the data for the Cart and Item.
  - Create only getters, values are passed in through the constructor, properties should be protected
  - Modify the adapters and load data methods to utilize this created class

HW

- Highly recommend converting the code in the CartActivity, to utilize the thread strategy we discussed during the tutorial.