A202
Android Development

Oliver Bell - 6393896

https://bitbucket.org/freshollie/a202

App Icon

App design

Introduction

Whenever I need to go to the shop I find that I don't remember what items I need to put on my shopping list. Even when if I make a list, I miss items off.

I also never know what I can have for meals based on what's in my house. There is so much choice that I can't remember everything I can make.

The idea is to make an app which stores what I currently have in my kitchen, and stores the meals I can make out of those items. All user defined.

Market research

After doing some research there are apps that already perform my function:

"Out of milk" is an app that stores the items in your house and gives you a shopping list for the items you need, but this app's design is not very good at all and doesn't seem to use the material design guidelines which android 5.0+ is built for. The app is also designed for america as the currency is in $. However the app does sync across multiple devices.

"Our Groceries" also performs the same function, and also allows for compilation of meals. It however does not export to google keep for cross platform integration which is what I need as I am currently using an iPhone.

Specification

1. App should allow the user to input grocery items
2. App should allow the user to input how many of that item they have
3. App should allow the user to be able to use up that item as they use it
4. App should export a shopping list to google keep of items that they are running low on
5. App should either get the price of each item from the user or try and pull the price from a stores api (tesco, asda)
6. App should allow the user to define meals using the items that the user has in their inventory
7. App should calculate the price of said meal
8. App should conform the material design specification

Build

I have chosen the package name: "uk.ac.coventry.bello.myinventory" as I feel it incorporates the university aspect of this project. I have also chosen the target API level 24, with a minimum 19, to build for the near latest android devices.

Interface

I designed my interface to use the latest material design for android. I use animations and colours all conforming to the material design specification.

Navigation Drawer

Firstly I used a Navigation Drawer ("Creating a Navigation Drawer") and App Bar ("Adding the App Bar") as my main interface , both of which were not taught.

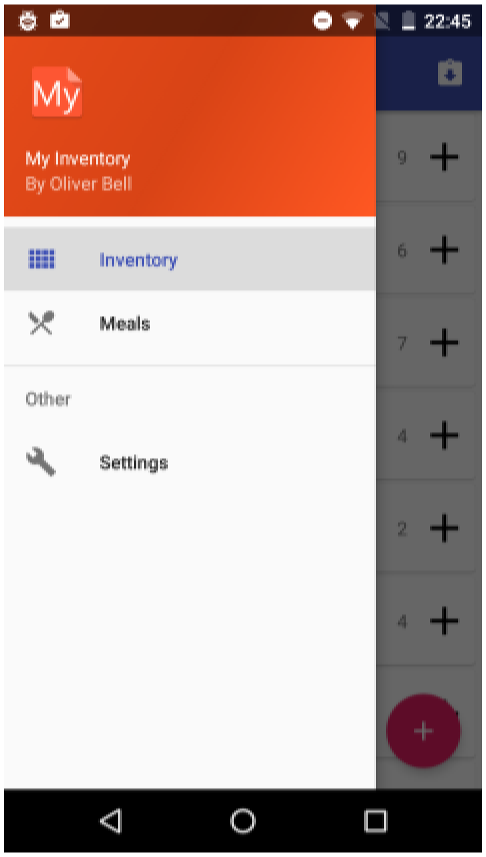


Figure 1 opened navigation drawer

The app is split into separate lists for meals and inventory. With the settings in its own activity.

The user can chose either the inventory tab or the meals tab.

When the user presses the back button it will return to the inventory tab.

When the user switches tabs, each tab has its own colour for the action bar. When the action bar colour is updated it's colour fades between the two.

I made sure that the navigation drawer was shown behind the status bar but in front of the app bar when I designed the interface. This was also to conform to the material design guidelines.

Card View

To show the users items, and meals, I used Card View's ("Creating Lists and Cards") not taught under the module.

Each card displays information about that item or meal. In the case of the inventory, the card displays name price and the quantity. It also contains buttons to increase or decrease the quantity.

Card views are implemented using a RecyclerView Adapter ("RecyclerView.Adapter")

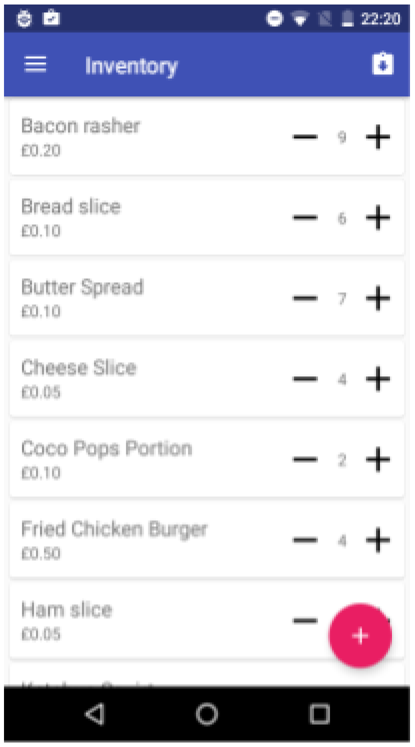

22:20
Inventory
Bacon rasher £0.20 9
Bread slice £0.10 6
Butter Spread £0.10 7
Cheese Slice £0.05 4
Coco Pops Portion £0.10 2
Fried Chicken Burger £0.50 4
Ham slice £0.05

Figure 2 inventory tab

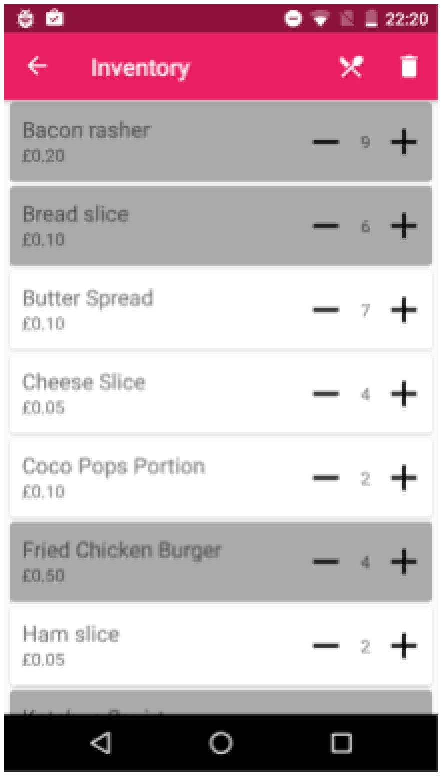

22:20
Inventory
Bacon rasher £0.20 9
Bread slice £0.10 6
Butter Spread £0.10 7
Cheese Slice £0.05 4
Coco Pops Portion £0.10 2
Fried Chicken Burger £0.50 4
Ham slice £0.05 2

Figure 3 inventory tab in selection mode

Each card is interactive, holding down on the card will launch selection mode where each card can be selected and then an action can be performed on those selected cards from the app bar.

When not in selection mode clicking on a card will produce a dialog to update the item represented by the card. Seen in Figure 7.

In the case of the inventory tab, the user can delete items or make a meal out of those selected items. While in the Meals tab the user can only delete the selected meals.

Inventory Tab

The inventory tab can be seen in Figure 2. It has a button in the menu that will make a list of items which the user has “0” of, and then post that list to a “Google Keep” list which the user can save.

The dialog shown, see Figure 4, is actually created by Google Keep when an intent is sent to it.

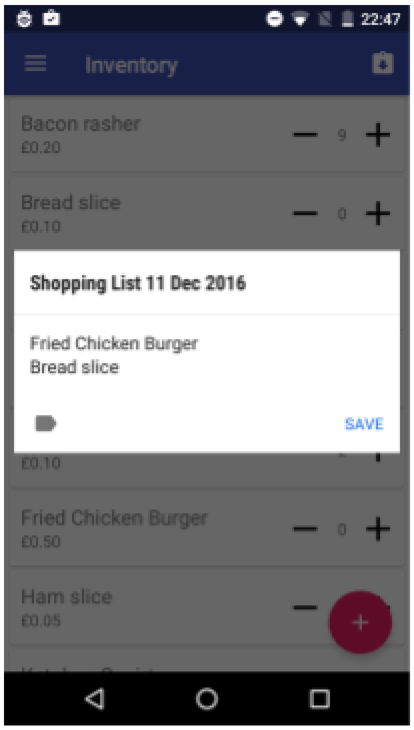


Figure 4 google keep generated UI

Meals Tab

The meals tab, shown in Figure 5, is the list of available user defined meals. The user adds these meals, shown in Figure 6, by clicking the floating action button.

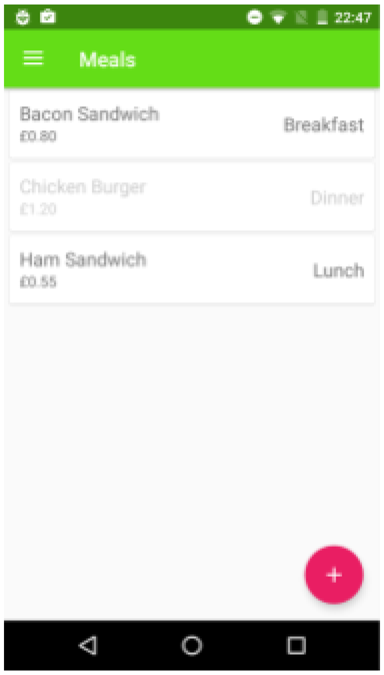


Figure 5 Meals Tab

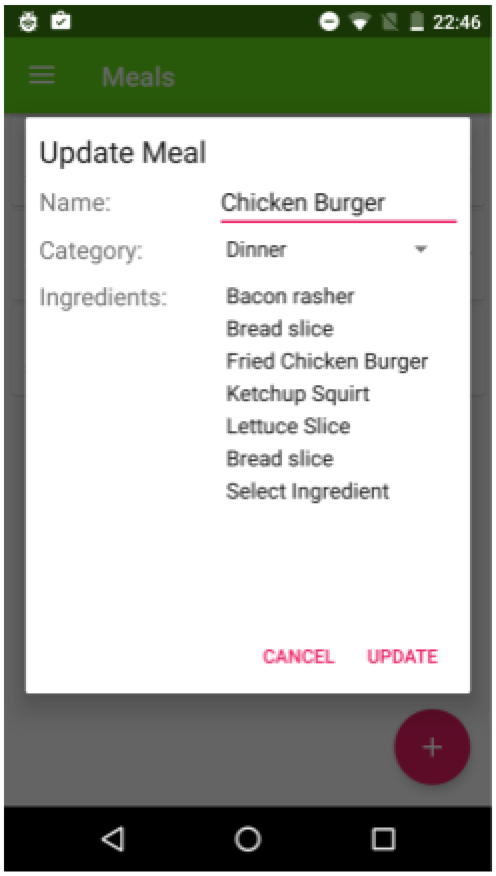
The meal is “unavailable” if the user’s inventory contains less than 1 of the ingredients that make the meal.

Floating Action Button

I use a Floating Action Button ("Buttons: Floating Action Button"), as per the material design specification, in both the Inventory Tab and the Meals Tab as the main action to add a meal or an Item to the lists.

When the user clicks the button, a Custom Dialog opens and allows the user to input information, see Figure 6.

When the floating action button is not used, it is removed with a removal animation. This happens when in selection mode, see Figure 3.



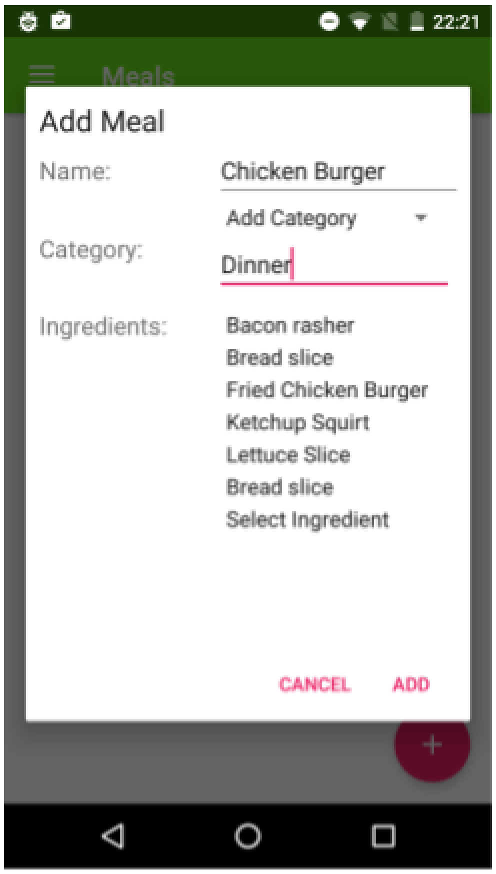
22:46
Meals

Update Meal

Name: Chicken Burger
Category: Dinner
Ingredients: Bacon rasher
Bread slice
Fried Chicken Burger
Ketchup Squirt
Lettuce Slice
Bread slice
Select Ingredient

CANCEL UPDATE

Figure 7 update meal dialog



22:21
Meals

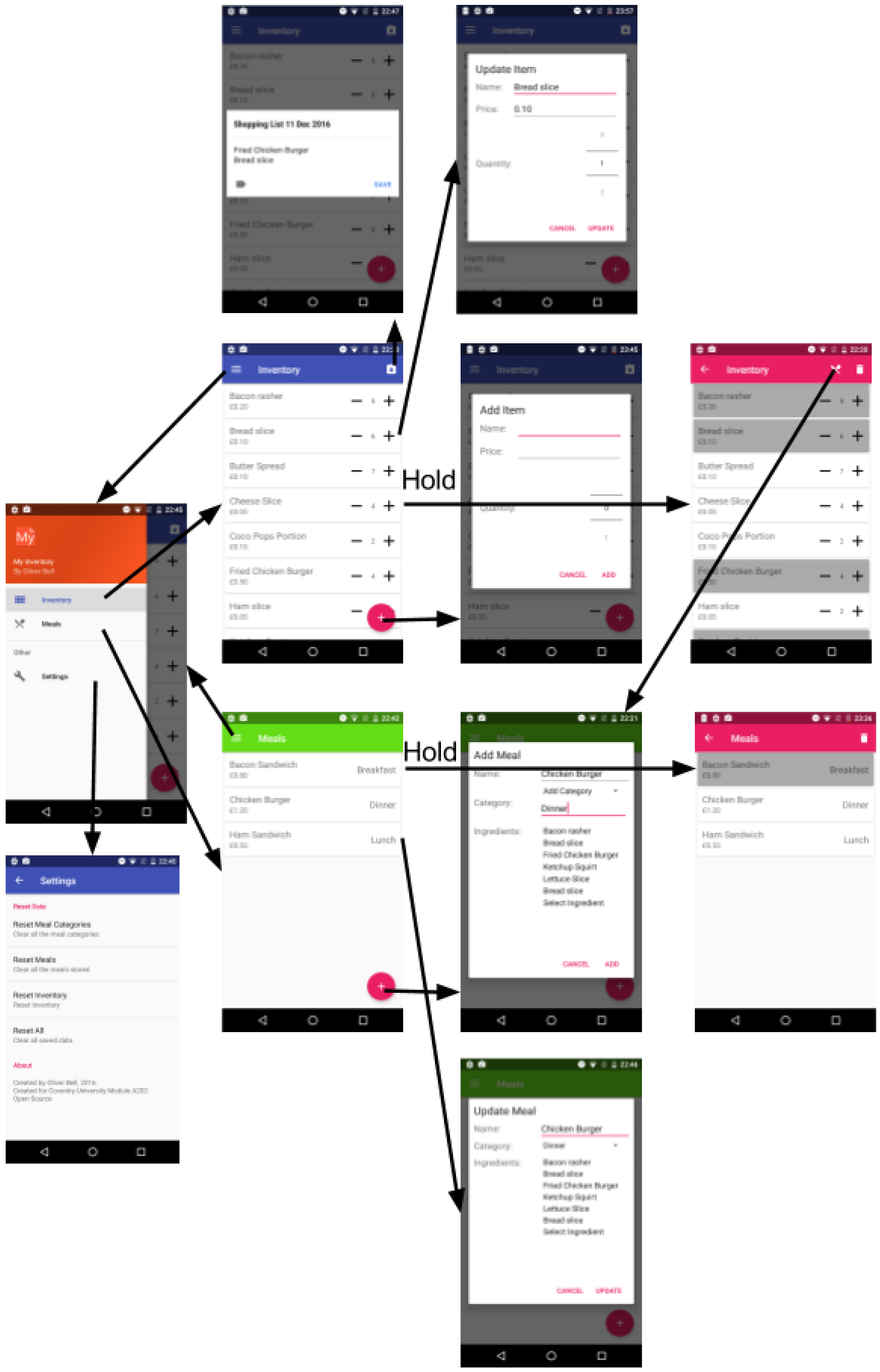
Add Meal

Name: Chicken Burger
Add Category
Category: Dinner
Ingredients: Bacon rasher
Bread slice
Fried Chicken Burger
Ketchup Squirt
Lettuce Slice
Bread slice
Select Ingredient

CANCEL ADD

Figure 6 add meal dialog

Interface Flowchart



Data

Data in my app is all handled via objects with attributes. These objects are then converted to json strings containing the object attributes and stored in SharedPreferences as string sets.

Objects

InventoryItem

Every item is an object “InventoryItem”

Attributes:

- Name
- Price

The InventoryItem contains methods to change data and get data about the item’s name and price. It also contains methods to get a json string containing the object values.

Inventory

The Inventory object is an aggregation of the user’s InventoryItems.

The object contains a HashMap which stores the quantity of the item object with the key as the item object, same as it would be as a stock list in real life.

It also contains methods to:

- Add and remove items from the inventory
- Update the items quantities
- Getting a list of items
- Checking if an item name already exists
- Save and load the inventory from shared preferences

Meal

Every Meal is stored as an object “Meal”

Attributes:

- Name
- Category
- Ingredients

The Meal object stores the name, category, and InventoryItems ArrayList that makes up that meal.

It contains methods to:

- Update those attributes
- Get and calculate the price of the meal using the prices of the items
- Get a json string containing the attributes

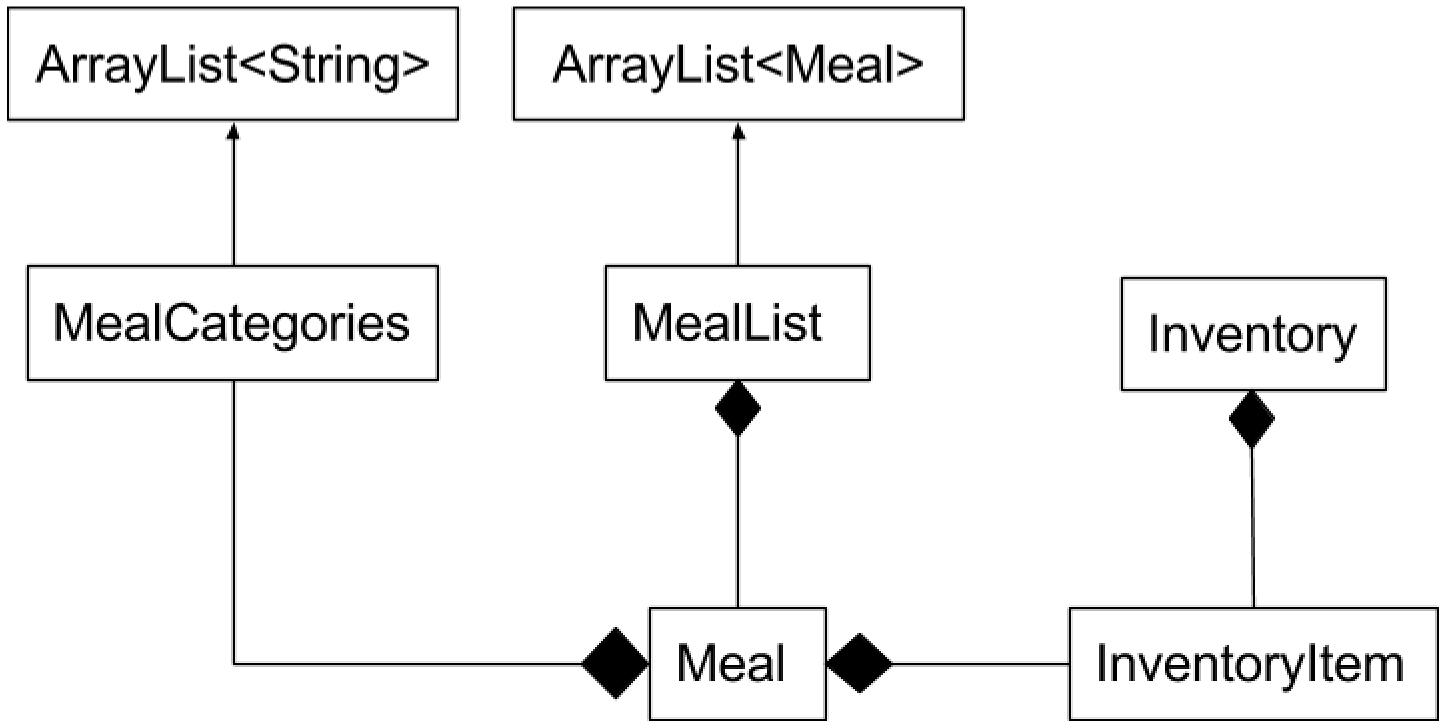
MealsList
MealList object extends ArrayList and is an aggregation of the user's meals stored in their inventory.

It contains Meal objects and has methods to:
- Check if a meal name already exists
- Get a copy of the list
- Save the list
- Load the list

CategoriesList
CategoriesList extends ArrayList and contains a list of category name strings.

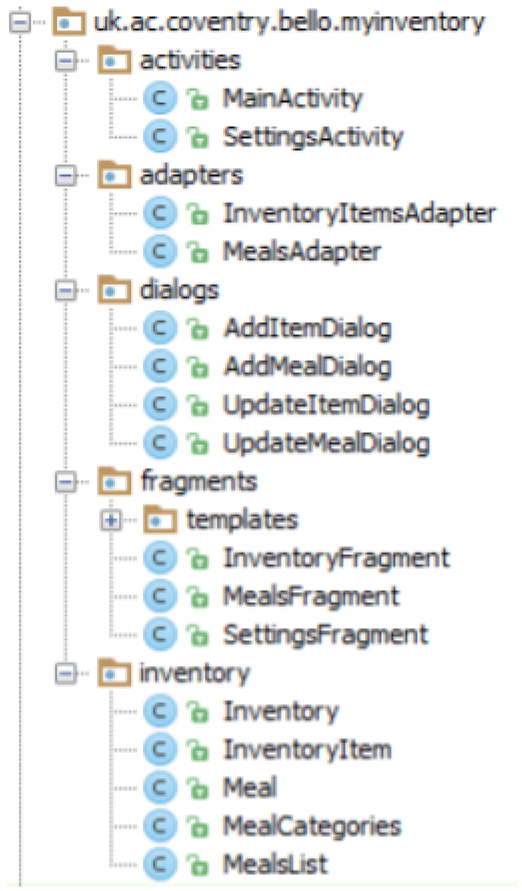
It's made into its own object so that it can have methods to:
- Save itself into a string set
- Load itself
- Check if a category already exists irrelevant of case

Relationships



Coding

The project was structured so that it was as easy to identify class types as possible.

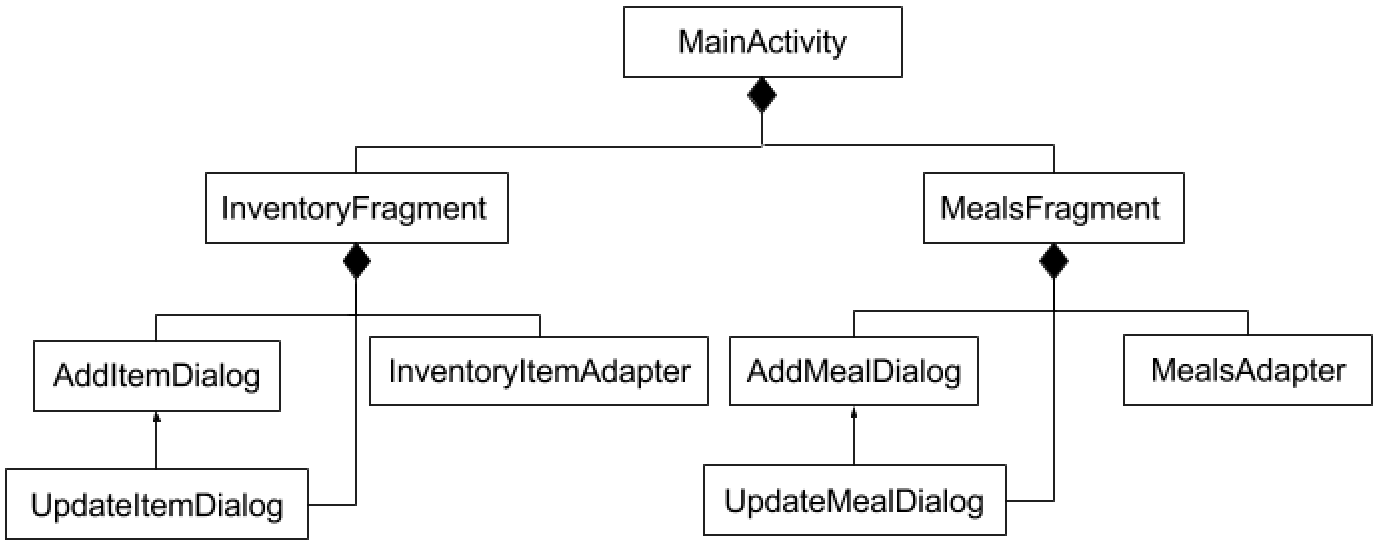


```
uk.ac.coventry.bello.myinventory
    activities
        MainActivity
        SettingsActivity
    adapters
        InventoryItemsAdapter
        MealsAdapter
    dialogs
        AddItemDialog
        AddMealDialog
        UpdateItemDialog
        UpdateMealDialog
    fragments
        templates
        InventoryFragment
        MealsFragment
        SettingsFragment
    inventory
        Inventory
        InventoryItem
        Meal
        MealCategories
        MealsList
```

Figure 14 project structure

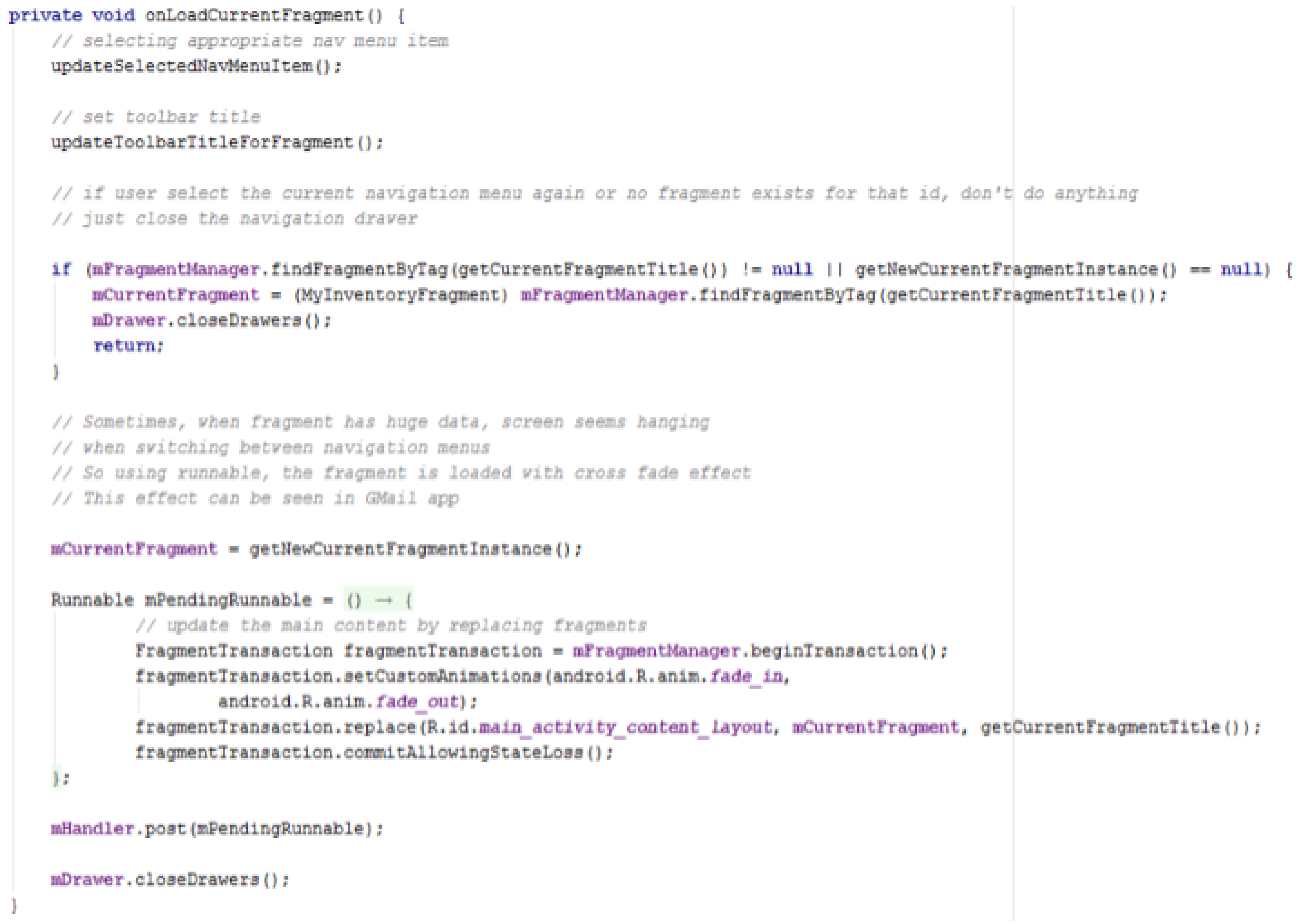
Interface

The interface is written with two activities, and provides plenty of material design animations. This is the data structure of the interface:



Main Activity

The main activity holds the navigation drawer and the main content tabs of the app. The app bar is constant but the content under the app bar is replaced by the relevant fragment when the tab is selected. This is accompanied by a fade animation. See Figure 8.

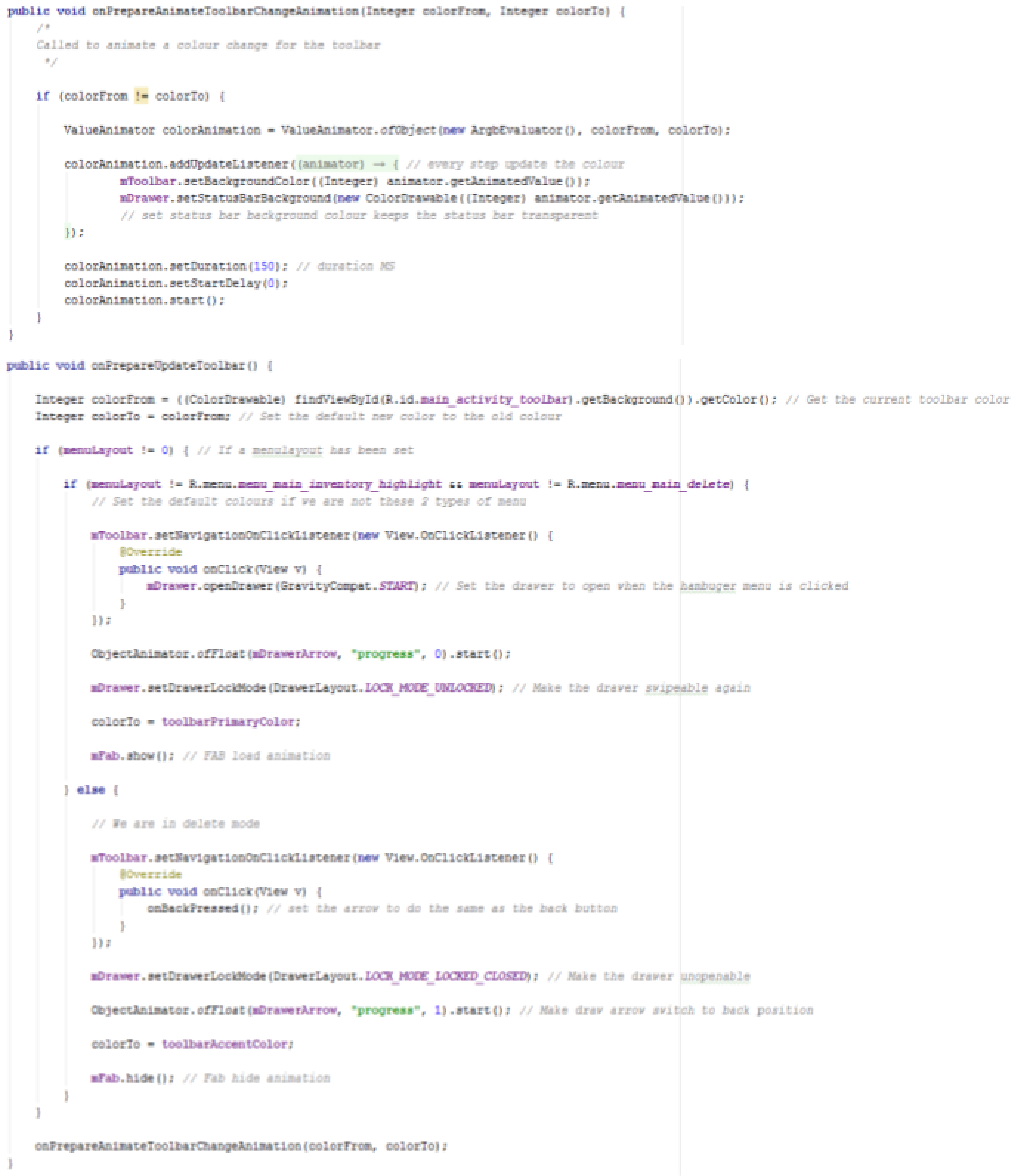


```
private void onLoadCurrentFragment() {
    // selecting appropriate nav menu item
    updateSelectedNavMenuItem();

    // set toolbar title
    updateToolbarTitleForFragment();

    // if user select the current navigation menu again or no fragment exists for that id, don't do anything
    // just close the navigation drawer

    if (mFragmentManager.findFragmentByTag(getCurrentFragmentTitle()) != null || getNewCurrentFragmentInstance() == null) {
        mCurrentFragment = (MyInventoryFragment) mFragmentManager.findFragmentByTag(getCurrentFragmentTitle());
        mDrawer.closeDrawers();
        return;
    }

    // Sometimes, when fragment has huge data, screen seems hanging
    // when switching between navigation menus
    // So using runnable, the fragment is loaded with cross fade effect
    // This effect can be seen in GMail app

    mCurrentFragment = getNewCurrentFragmentInstance();

    Runnable mPendingRunnable = () → {
            // update the main content by replacing fragments
            FragmentTransaction fragmentTransaction = mFragmentManager.beginTransaction();
            fragmentTransaction.setCustomAnimations(android.R.anim.fade_in,
                    android.R.anim.fade_out);
            fragmentTransaction.replace(R.id.main_activity_content_layout, mCurrentFragment, getCurrentFragmentTitle());
            fragmentTransaction.commitAllowingStateLoss();
    };

    mHandler.post(mPendingRunnable);

    mDrawer.closeDrawers();
}
```

Figure 8

If the fragment then changes the app bar colours of the main activity, a fading animation is run. It also switches the drawer button to an arrow if needed.

uk.ac.coventry.myinventory.activities.MainActivity



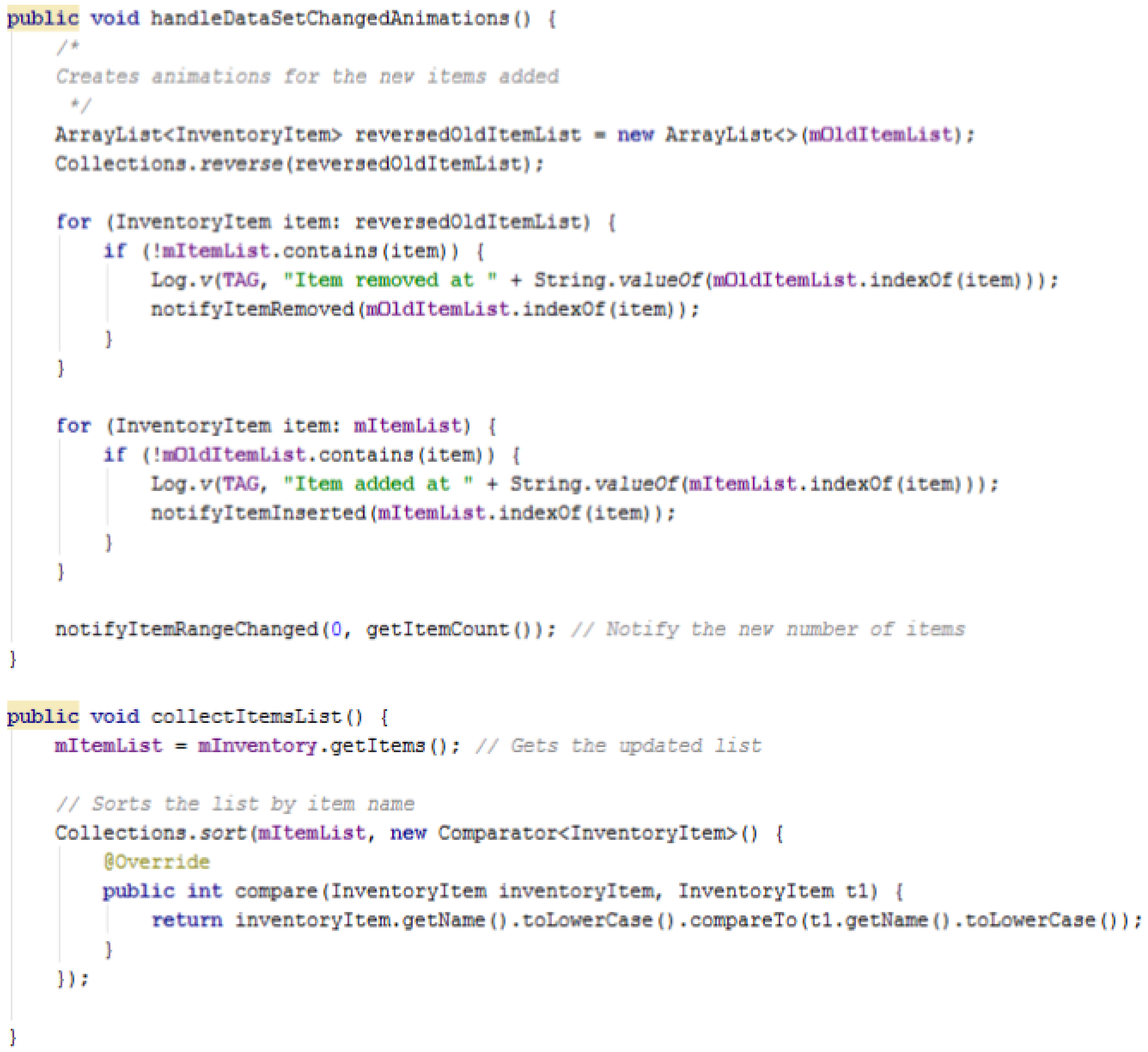
```java
public void onPrepareAnimateToolbarChangeAnimation(Integer colorFrom, Integer colorTo) {
    /*
    Called to animate a colour change for the toolbar
     */

    if (colorFrom != colorTo) {

        ValueAnimator colorAnimation = ValueAnimator.ofObject(new ArgbEvaluator(), colorFrom, colorTo);

        colorAnimation.addUpdateListener((animator) → { // every step update the colour
                mToolbar.setBackgroundColor((Integer) animator.getAnimatedValue());
                mDrawer.setStatusBarBackground(new ColorDrawable((Integer) animator.getAnimatedValue()));
                // set status bar background colour keeps the status bar transparent
        });

        colorAnimation.setDuration(150); // duration MS
        colorAnimation.setStartDelay(0);
        colorAnimation.start();
    }
}

public void onPrepareUpdateToolbar() {

    Integer colorFrom = ((ColorDrawable) findViewById(R.id.main_activity_toolbar).getBackground()).getColor(); // Get the current toolbar color
    Integer colorTo = colorFrom; // Set the default new color to the old colour

    if (menuLayout != 0) { // If a menulayout has been set

        if (menuLayout != R.menu.menu_main_inventory_highlight && menuLayout != R.menu.menu_main_delete) {
            // Set the default colours if we are not these 2 types of menu

            mToolbar.setNavigationOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    mDrawer.openDrawer(GravityCompat.START); // Set the drawer to open when the hambuger menu is clicked
                }
            });

            ObjectAnimator.ofFloat(mDrawerArrow, "progress", 0).start();

            mDrawer.setDrawerLockMode(DrawerLayout.LOCK_MODE_UNLOCKED); // Make the drawer swipeable again

            colorTo = toolbarPrimaryColor;

            mFab.show(); // FAB load animation

        } else {

            // We are in delete mode

            mToolbar.setNavigationOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    onBackPressed(); // set the arrow to do the same as the back button
                }
            });

            mDrawer.setDrawerLockMode(DrawerLayout.LOCK_MODE_LOCKED_CLOSED); // Make the drawer unopenable

            ObjectAnimator.ofFloat(mDrawerArrow, "progress", 1).start(); // Make draw arrow switch to back position

            colorTo = toolbarAccentColor;

            mFab.hide(); // Fab hide animation
        }
    }

    onPrepareAnimateToolbarChangeAnimation(colorFrom, colorTo);
}
```

Figure 9

Fragments

Both fragments implement a RecyclerView.Adapter to hold a list of cards. When the data list is changed a method is called, to check where the cards are removed or added, and produces an animation for those added or removed cards. See figure 10

uk.ac.coventry.bello.myinventory.adapters.InventoryAdapter



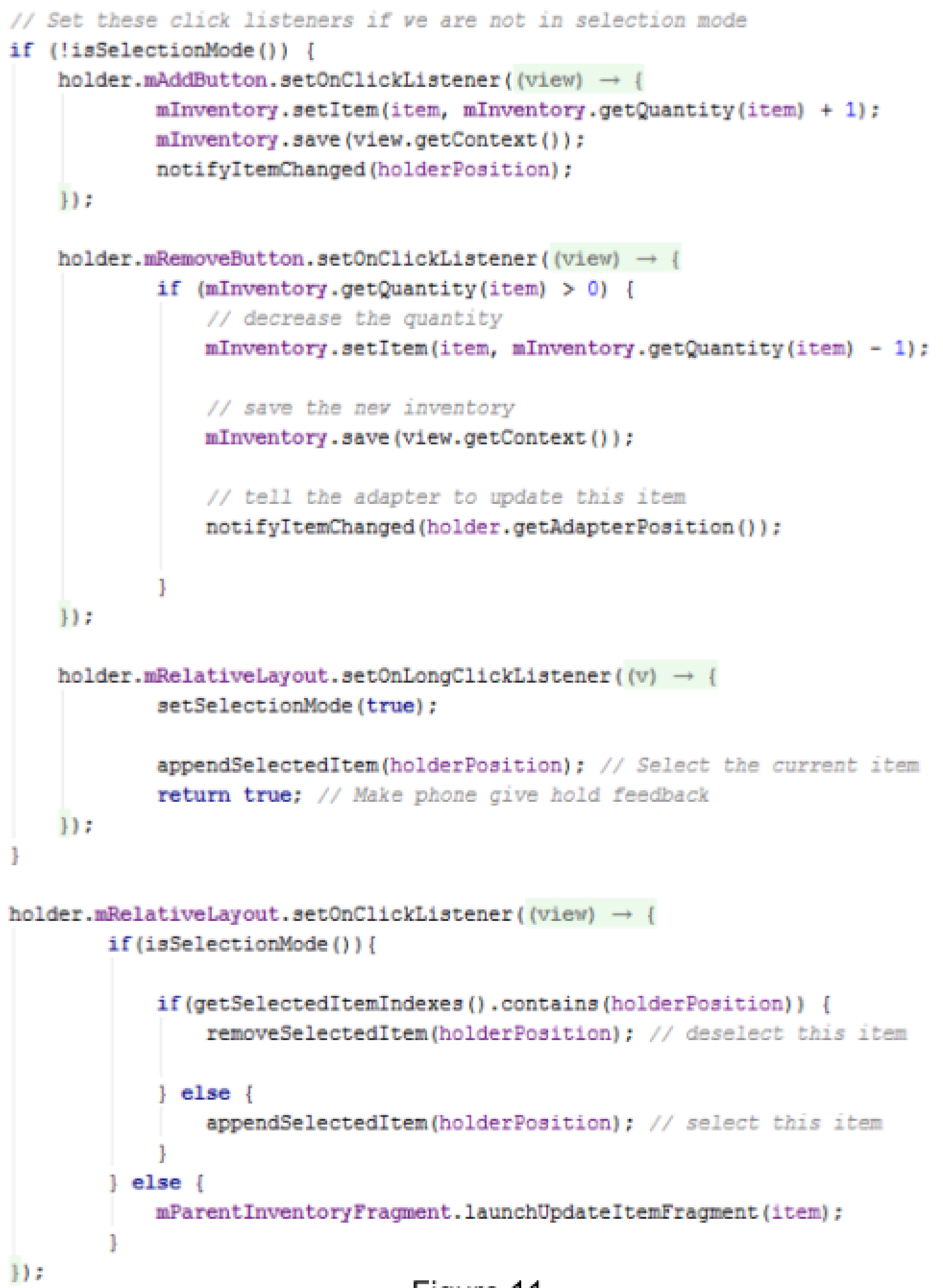
```java
public void handleDataSetChangedAnimations() {
    /*
    Creates animations for the new items added
     */
    ArrayList<InventoryItem> reversedOldItemList = new ArrayList<>(mOldItemList);
    Collections.reverse(reversedOldItemList);

    for (InventoryItem item: reversedOldItemList) {
        if (!mItemList.contains(item)) {
            Log.v(TAG, "Item removed at " + String.valueOf(mOldItemList.indexOf(item)));
            notifyItemRemoved(mOldItemList.indexOf(item));
        }
    }

    for (InventoryItem item: mItemList) {
        if (!mOldItemList.contains(item)) {
            Log.v(TAG, "Item added at " + String.valueOf(mItemList.indexOf(item)));
            notifyItemInserted(mItemList.indexOf(item));
        }
    }

    notifyItemRangeChanged(0, getItemCount()); // Notify the new number of items
}

public void collectItemsList() {
    mItemList = mInventory.getItems(); // Gets the updated list

    // Sorts the list by item name
    Collections.sort(mItemList, new Comparator<InventoryItem>() {
        @Override
        public int compare(InventoryItem inventoryItem, InventoryItem t1) {
            return inventoryItem.getName().toLowerCase().compareTo(t1.getName().toLowerCase());
        }
    });

}
```

Figure 10

The adapter list also handles the selection of items using onclicklisteners, see figure 11.

uk.ac.coventry.bello.myinventory.adapters.InventoryAdapter.onBindViewHolder

...

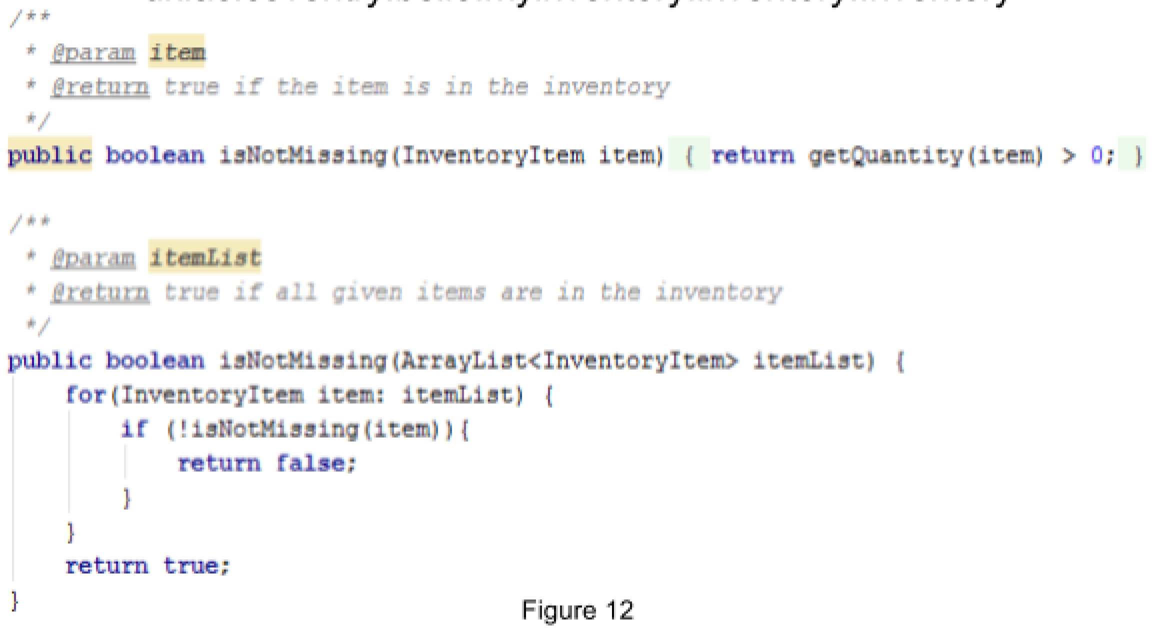


```
// Set these click listeners if we are not in selection mode
if (!isSelectionMode()) {
    holder.mAddButton.setOnClickListener((view) → {
            mInventory.setItem(item, mInventory.getQuantity(item) + 1);
            mInventory.save(view.getContext());
            notifyItemChanged(holderPosition);
    });

    holder.mRemoveButton.setOnClickListener((view) → {
            if (mInventory.getQuantity(item) > 0) {
                // decrease the quantity
                mInventory.setItem(item, mInventory.getQuantity(item) - 1);

                // save the new inventory
                mInventory.save(view.getContext());

                // tell the adapter to update this item
                notifyItemChanged(holder.getAdapterPosition());

            }
    });

    holder.mRelativeLayout.setOnLongClickListener((v) → {
            setSelectionMode(true);

            appendSelectedItem(holderPosition); // Select the current item
            return true; // Make phone give hold feedback
    });
}

holder.mRelativeLayout.setOnClickListener((view) → {
        if(isSelectionMode()){

            if(getSelectedItemIndexes().contains(holderPosition)) {
                removeSelectedItem(holderPosition); // deselect this item

            } else {
                appendSelectedItem(holderPosition); // select this item
            }
        } else {
            mParentInventoryFragment.launchUpdateItemFragment(item);
        }
});
```

Figure 11

The meals adapter tab will "Grey out" meals which the user does not have the inventory to cook. This can be seen in Figure 5. Figure 12 shows the method of the inventory object to check if the ingredients passed are missing from the stock.

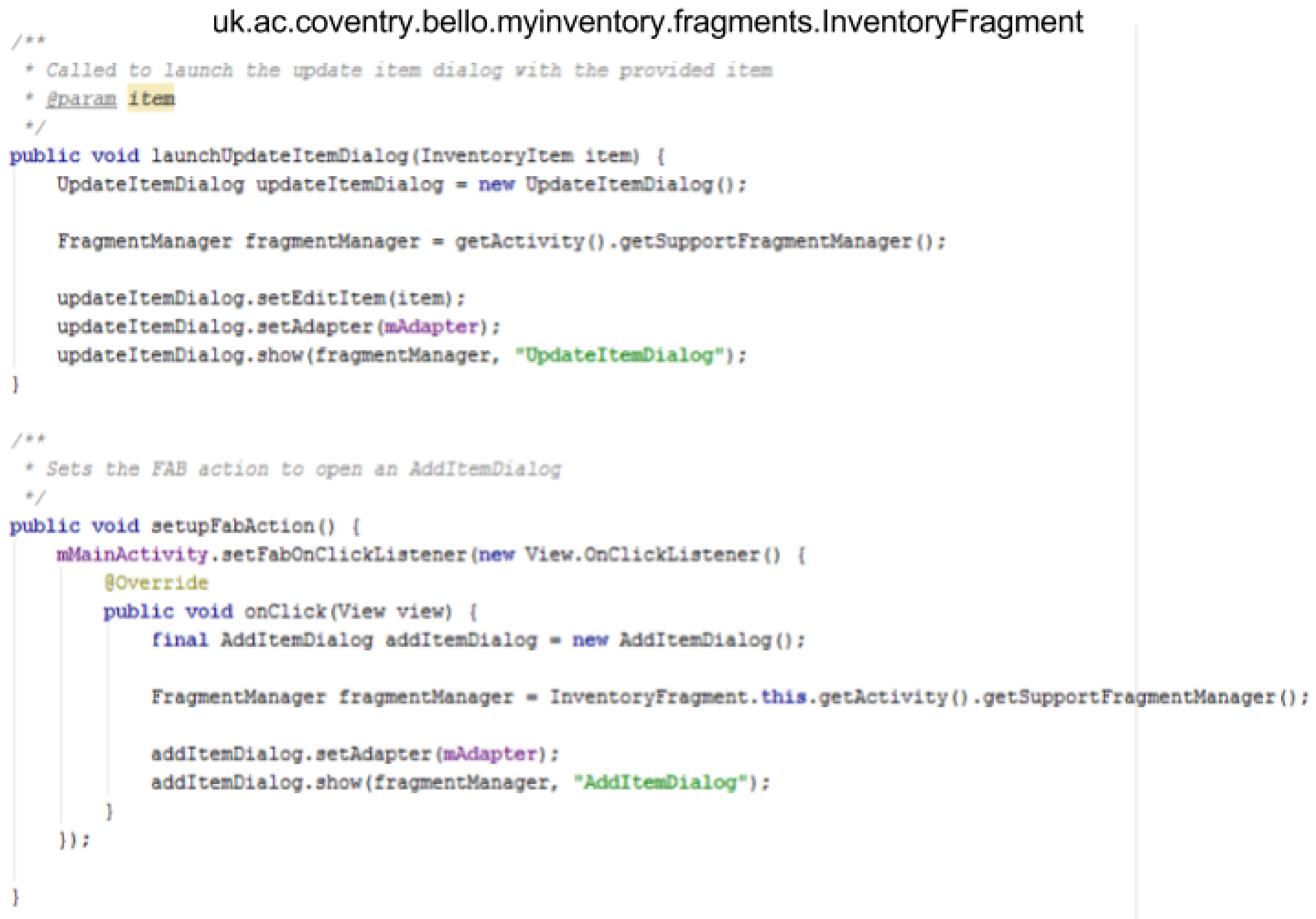
uk.ac.coventry.bello.myinventory.inventory.Inventory



```java
/**
 * @param item
 * @return true if the item is in the inventory
 */
public boolean isNotMissing(InventoryItem item) { return getQuantity(item) > 0; }

/**
 * @param itemList
 * @return true if all given items are in the inventory
 */
public boolean isNotMissing(ArrayList<InventoryItem> itemList) {
    for(InventoryItem item: itemList) {
        if (!isNotMissing(item)){
            return false;
        }
    }
    return true;
}
```

Figure 12

The Fragments launch custom dialogs for editing or adding user defined information. Figure 13 shows the functions to launch these.



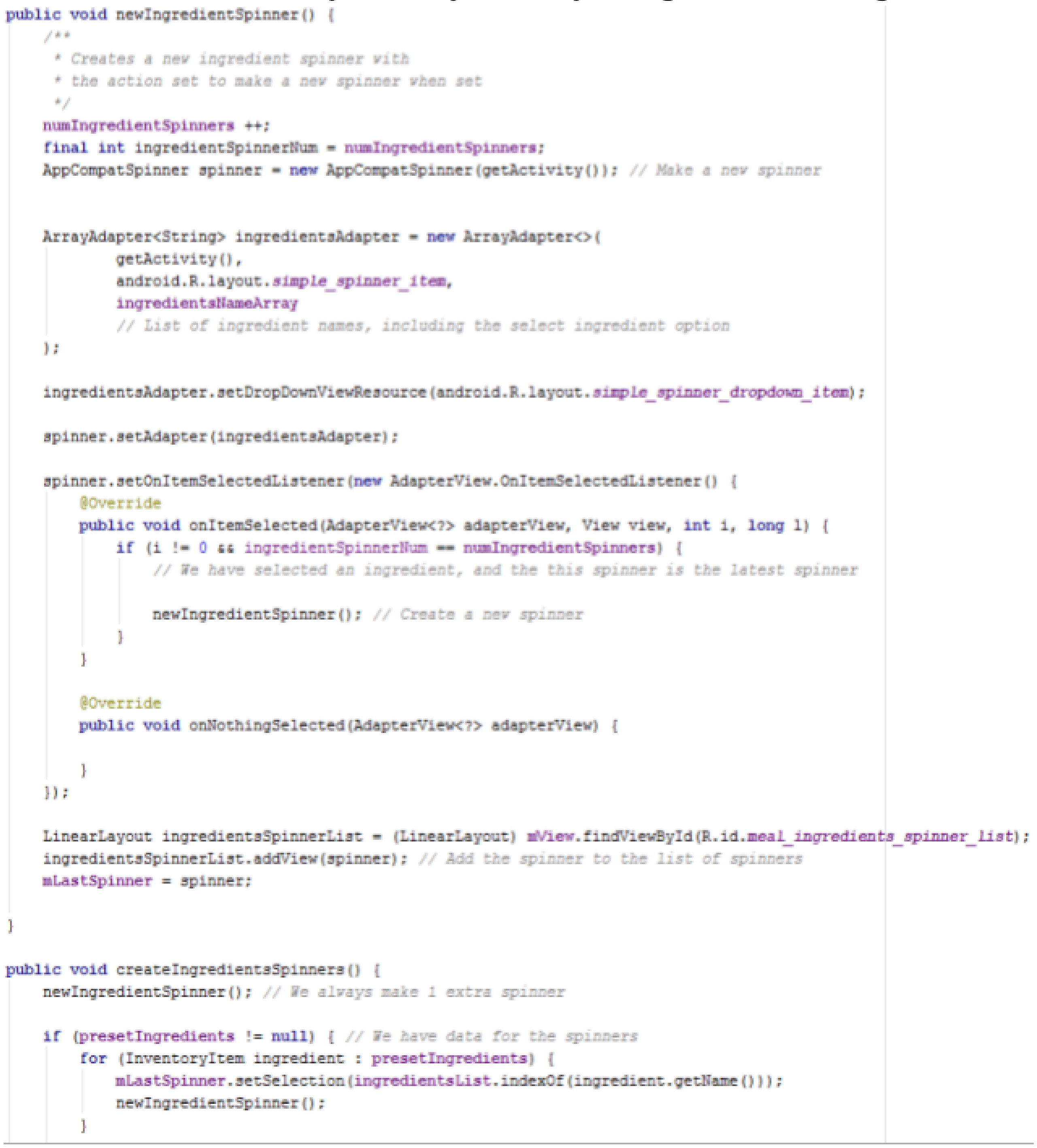
uk.ac.coventry.bello.myinventory.fragments.InventoryFragment

```java
/**
 * Called to launch the update item dialog with the provided item
 * @param item
 */
public void launchUpdateItemDialog(InventoryItem item) {
    UpdateItemDialog updateItemDialog = new UpdateItemDialog();

    FragmentManager fragmentManager = getActivity().getSupportFragmentManager();

    updateItemDialog.setEditItem(item);
    updateItemDialog.setAdapter(mAdapter);
    updateItemDialog.show(fragmentManager, "UpdateItemDialog");
}

/**
 * Sets the FAB action to open an AddItemDialog
 */
public void setupFabAction() {
    mMainActivity.setFabOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            final AddItemDialog addItemDialog = new AddItemDialog();

            FragmentManager fragmentManager = InventoryFragment.this.getActivity().getSupportFragmentManager();

            addItemDialog.setAdapter(mAdapter);
            addItemDialog.show(fragmentManager, "AddItemDialog");
        }
    });

}
```

Figure 13 Lauching of custom Dialogs

Add meal Fragment makes a new spinner for each new ingredient that the user wishes to add. See figure 14.

uk.ac.coventry.bello.myinventory.dialog.AddMealDialog

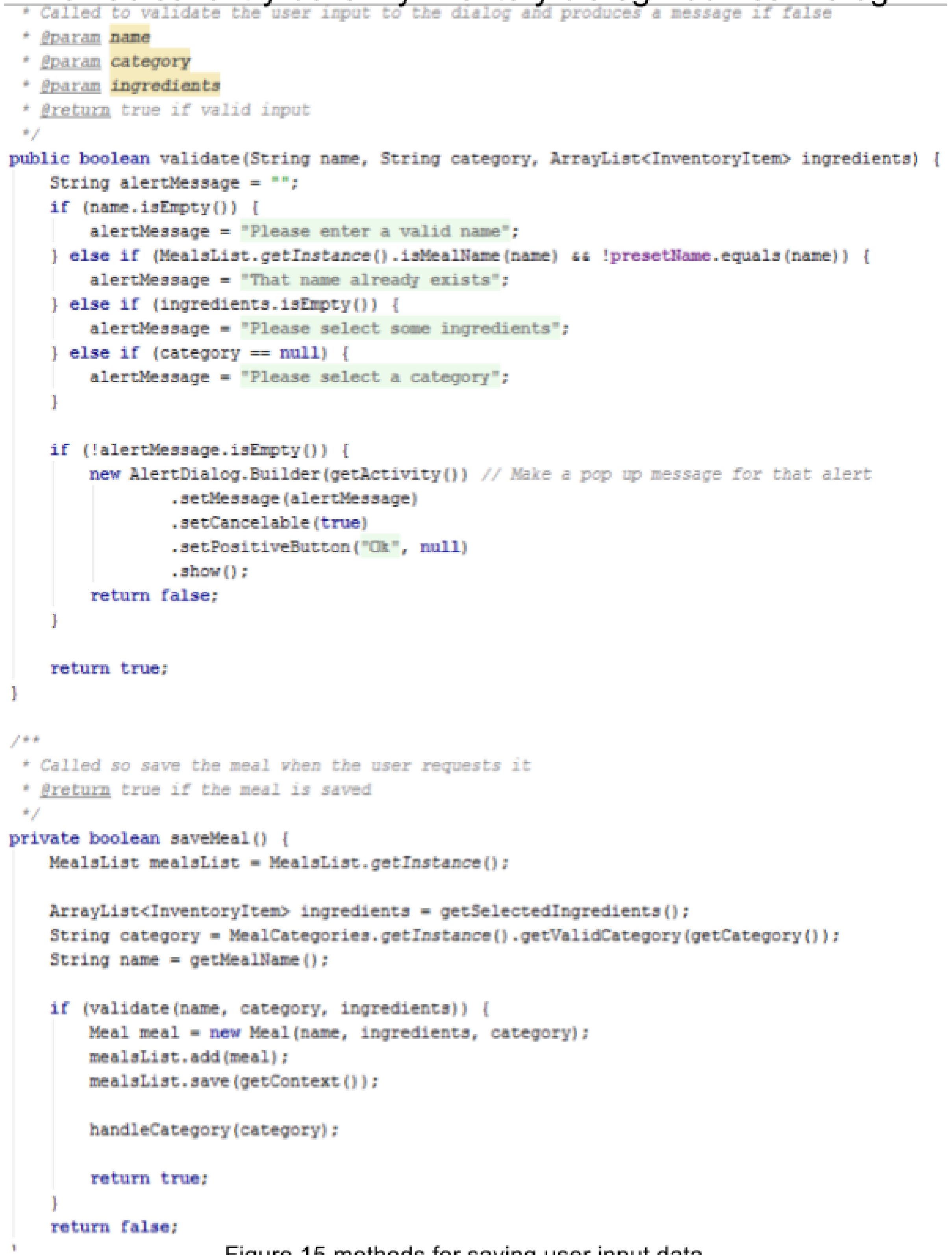


```java
public void newIngredientSpinner() {
    /**
     * Creates a new ingredient spinner with
     * the action set to make a new spinner when set
     */
    numIngredientSpinners ++;
    final int ingredientSpinnerNum = numIngredientSpinners;
    AppCompatSpinner spinner = new AppCompatSpinner(getActivity()); // Make a new spinner


    ArrayAdapter<String> ingredientsAdapter = new ArrayAdapter<>(
            getActivity(),
            android.R.layout.simple_spinner_item,
            ingredientsNameArray
            // List of ingredient names, including the select ingredient option
    );

    ingredientsAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);

    spinner.setAdapter(ingredientsAdapter);

    spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
            if (i != 0 && ingredientSpinnerNum == numIngredientSpinners) {
                // We have selected an ingredient, and the this spinner is the latest spinner

                newIngredientSpinner(); // Create a new spinner
            }
        }

        @Override
        public void onNothingSelected(AdapterView<?> adapterView) {

        }
    });

    LinearLayout ingredientsSpinnerList = (LinearLayout) mView.findViewById(R.id.meal_ingredients_spinner_list);
    ingredientsSpinnerList.addView(spinner); // Add the spinner to the list of spinners
    mLastSpinner = spinner;

}

public void createIngredientsSpinners() {
    newIngredientSpinner(); // We always make 1 extra spinner

    if (presetIngredients != null) { // We have data for the spinners
        for (InventoryItem ingredient : presetIngredients) {
            mLastSpinner.setSelection(ingredientsList.indexOf(ingredient.getName()));
            newIngredientSpinner();
        }
```

Figure 14 methods to create new spinners

When the user adds a new item, or meal, a save function is called which validates the data and adds the information. See Figure 15.

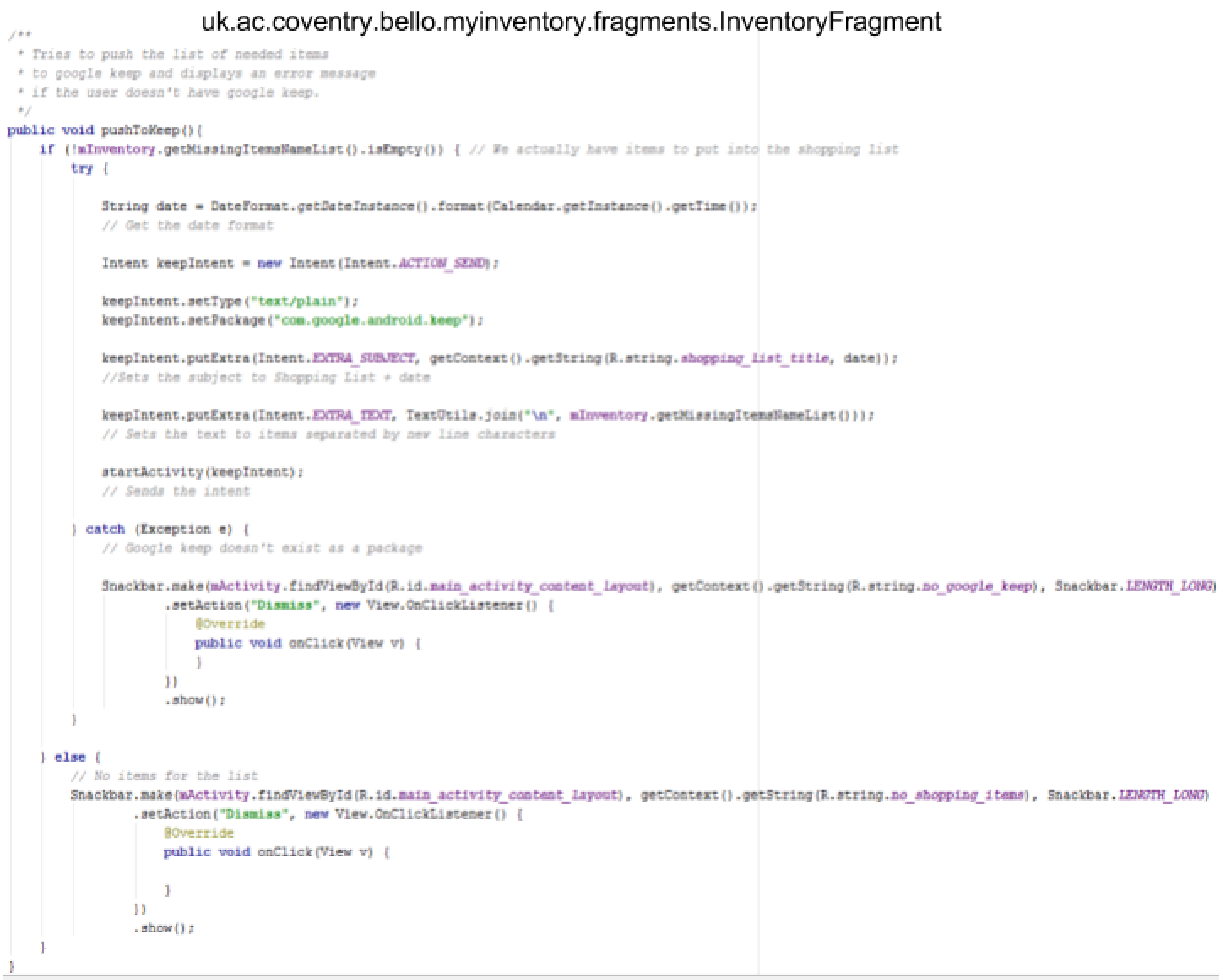
uk.ac.coventry.bello.myinventory.dialog.AddMealDialog



```java
 * Called to validate the user input to the dialog and produces a message if false
 * @param name
 * @param category
 * @param ingredients
 * @return true if valid input
 */
public boolean validate(String name, String category, ArrayList<InventoryItem> ingredients) {
    String alertMessage = "";
    if (name.isEmpty()) {
        alertMessage = "Please enter a valid name";
    } else if (MealsList.getInstance().isMealName(name) && !presetName.equals(name)) {
        alertMessage = "That name already exists";
    } else if (ingredients.isEmpty()) {
        alertMessage = "Please select some ingredients";
    } else if (category == null) {
        alertMessage = "Please select a category";
    }

    if (!alertMessage.isEmpty()) {
        new AlertDialog.Builder(getActivity()) // Make a pop up message for that alert
                .setMessage(alertMessage)
                .setCancelable(true)
                .setPositiveButton("Ok", null)
                .show();
        return false;
    }

    return true;
}

/**
 * Called so save the meal when the user requests it
 * @return true if the meal is saved
 */
private boolean saveMeal() {
    MealsList mealsList = MealsList.getInstance();

    ArrayList<InventoryItem> ingredients = getSelectedIngredients();
    String category = MealCategories.getInstance().getValidCategory(getCategory());
    String name = getMealName();

    if (validate(name, category, ingredients)) {
        Meal meal = new Meal(name, ingredients, category);
        mealsList.add(meal);
        mealsList.save(getContext());

        handleCategory(category);

        return true;
    }
    return false;
}
```

Figure 15 methods for saving user input data

A main feature of the app is to save a shopping list to google keep. This is done through an intent in the inventory fragment. See figure 16.



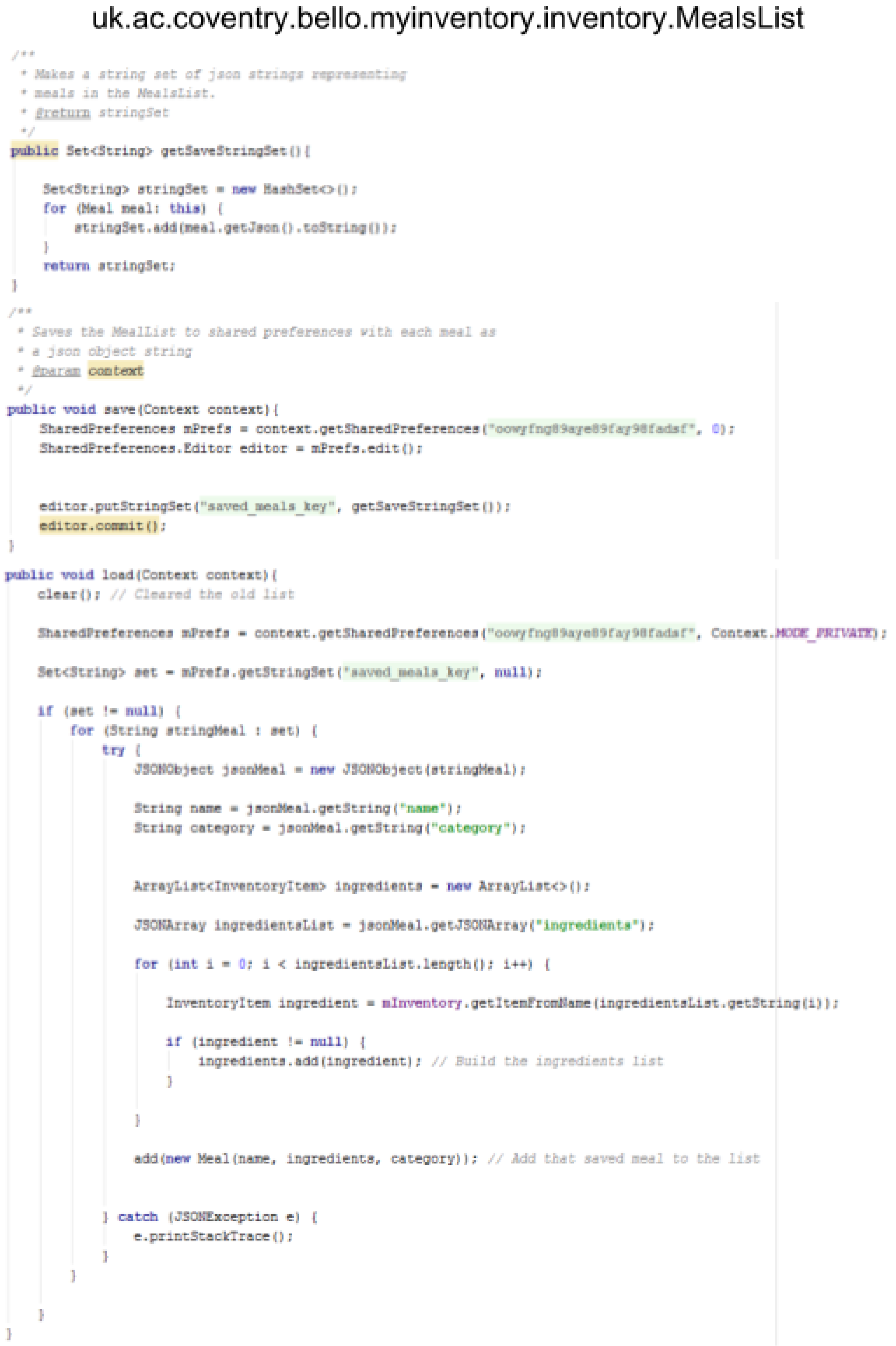
uk.ac.coventry.bello.myinventory.fragments.InventoryFragment

```java
/**
 * Tries to push the list of needed items
 * to google keep and displays an error message
 * if the user doesn't have google keep.
 */
public void pushToKeep(){
    if (!mInventory.getMissingItemsNameList().isEmpty()) { // We actually have items to put into the shopping list
        try {

            String date = DateFormat.getDateInstance().format(Calendar.getInstance().getTime());
            // Get the date format

            Intent keepIntent = new Intent(Intent.ACTION_SEND);

            keepIntent.setType("text/plain");
            keepIntent.setPackage("com.google.android.keep");

            keepIntent.putExtra(Intent.EXTRA_SUBJECT, getContext().getString(R.string.shopping_list_title, date));
            //Sets the subject to Shopping List + date

            keepIntent.putExtra(Intent.EXTRA_TEXT, TextUtils.join("\n", mInventory.getMissingItemsNameList()));
            // Sets the text to items separated by new line characters

            startActivity(keepIntent);
            // Sends the intent

        } catch (Exception e) {
            // Google keep doesn't exist as a package

            Snackbar.make(mActivity.findViewById(R.id.main_activity_content_layout), getContext().getString(R.string.no_google_keep), Snackbar.LENGTH_LONG)
                    .setAction("Dismiss", new View.OnClickListener() {
                        @Override
                        public void onClick(View v) {
                        }
                    })
                    .show();
        }

    } else {
        // No items for the list
        Snackbar.make(mActivity.findViewById(R.id.main_activity_content_layout), getContext().getString(R.string.no_shopping_items), Snackbar.LENGTH_LONG)
                .setAction("Dismiss", new View.OnClickListener() {
                    @Override
                    public void onClick(View v) {

                    }
                })
                .show();
    }
}
```

Figure 16 methods to add items to google keep

Data saving and retrieval

Data in my app is saved to shared preferences. List objects save each individual object as a json string in a string set. The data is saved every change to the object or list. See Figure 17

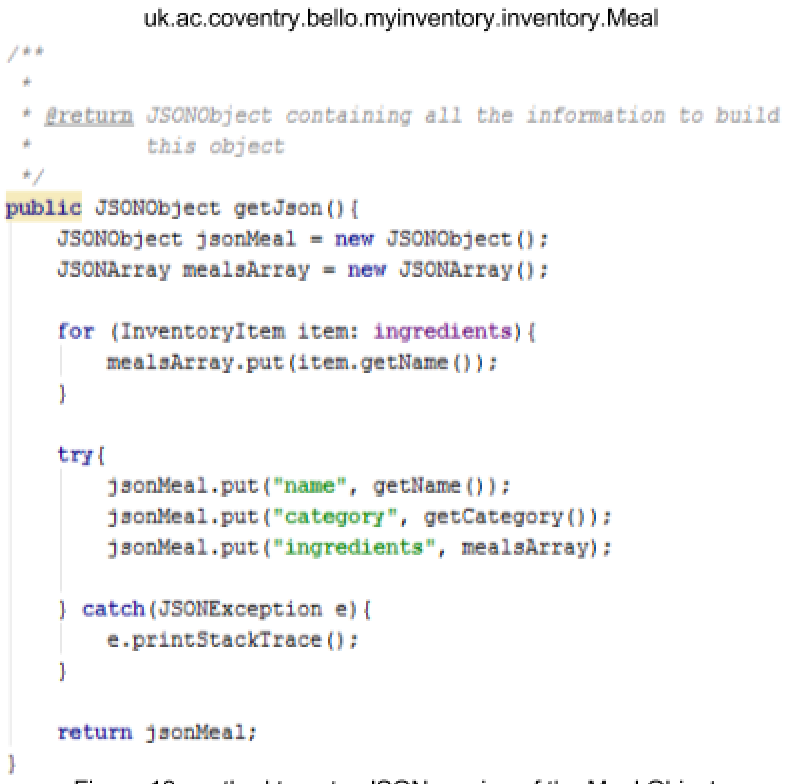


uk.ac.coventry.bello.myinventory.inventory.MealsList

```java
/**
 * Makes a string set of json strings representing
 * meals in the MealsList.
 * @return stringSet
 */
public Set<String> getSaveStringSet(){

    Set<String> stringSet = new HashSet<>();
    for (Meal meal: this) {
        stringSet.add(meal.getJson().toString());
    }
    return stringSet;
}
/**
 * Saves the MealList to shared preferences with each meal as
 * a json object string
 * @param context
 */
public void save(Context context){
    SharedPreferences mPrefs = context.getSharedPreferences("oowyfng89aye89fay98fadsf", 0);
    SharedPreferences.Editor editor = mPrefs.edit();


    editor.putStringSet("saved_meals_key", getSaveStringSet());
    editor.commit();
}
public void load(Context context){
    clear(); // Cleared the old list

    SharedPreferences mPrefs = context.getSharedPreferences("oowyfng89aye89fay98fadsf", Context.MODE_PRIVATE);

    Set<String> set = mPrefs.getStringSet("saved_meals_key", null);

    if (set != null) {
        for (String stringMeal : set) {
            try {
                JSONObject jsonMeal = new JSONObject(stringMeal);

                String name = jsonMeal.getString("name");
                String category = jsonMeal.getString("category");


                ArrayList<InventoryItem> ingredients = new ArrayList<>();

                JSONArray ingredientsList = jsonMeal.getJSONArray("ingredients");

                for (int i = 0; i < ingredientsList.length(); i++) {

                    InventoryItem ingredient = mInventory.getItemFromName(ingredientsList.getString(i));

                    if (ingredient != null) {
                        ingredients.add(ingredient); // Build the ingredients list
                    }

                }

                add(new Meal(name, ingredients, category)); // Add that saved meal to the list


            } catch (JSONException e) {
                e.printStackTrace();
            }
        }

    }
}
```

Figure 17, methods to save the MealsList

Each object has a method to convert itself to a JSONObject. See Figure 18.



uk.ac.coventry.bello.myinventory.inventory.Meal

```java
/**
 *
 * @return JSONObject containing all the information to build
 *         this object
 */
public JSONObject getJson(){
    JSONObject jsonMeal = new JSONObject();
    JSONArray mealsArray = new JSONArray();

    for (InventoryItem item: ingredients){
        mealsArray.put(item.getName());
    }

    try{
        jsonMeal.put("name", getName());
        jsonMeal.put("category", getCategory());
        jsonMeal.put("ingredients", mealsArray);

    } catch(JSONException e){
        e.printStackTrace();
    }

    return jsonMeal;
}
```

Figure 18, method to get a JSON version of the Meal Object

Evaluation

Overall I am very happy with how the app has turned out. All points of the specification have been met perfectly and I generalised lots of the programming within the app so that it can easily be expanded upon.

The module has made me research a vast amount about android development, and I now feel confident that I would be ready to produce an android app of excellent quality for the Playstore.

References

HeadCode (2016) *Our Groceries* [online] available from <https://play.google.com/store/apps/details?id=com.headcode.ourgroceries> [12 December 2016]

Capigami Inc (2016) *Out of milk* [online] available from <https://play.google.com/store/apps/details?id=com.capigami.outofmilk> [12 December 2016]

Google (2016) *Creating a Navigation Drawer* [online] available from <https://developer.android.com/training/implementing-navigation/nav-drawer.html> [12 December 2016]

Google (2016) *Adding the App Bar* [online] available from <https://developer.android.com/training/appbar/index.html> [12 December 2016]

Google (2016) *Creating Lists and Cards* [online] available from <https://developer.android.com/training/material/lists-cards.html> [12 December 2016]

Google (2016) *RecyclerView.Adapter* [online] available from <https://developer.android.com/reference/android/support/v7/widget/RecyclerView.Adapter.html> [12 December 2016]

Google (2016) *Buttons: Floating Action Button* [online] available from <https://material.google.com/components/buttons-floating-action-button.html> [12 December 2016]

Google (2016) *Google Keep* [online] available from <https://play.google.com/store/apps/details?id=com.google.android.keep> [12 December 2016]