CS385 Individual Contribution Document

App name: SeeTheCapital

Team Member: Sinead O'Rourke

Student Number: 10335141

Allocated Work

The idea for *SeeTheCapital* had already formed before the Militant Fireballs joined as a team. The work that originally needed to be done was as follows:

-Holiday List (database)

-Shopping List + Spending List (database)

-Simulated Bank Account (database)

-Saving Status/Countdown

-Theme Changer

-Designs (Splash Screen, 4x Themes + Logo)

Sinead had already begun researching into SQLite, therefore took control of database management and the Saving Status activity. Her partner expressed interest in graphic design, therefore took control of the design/layout aspects and the Theme Changer. Sinead's partner had also shown interest in using Google maps, and it was decided that he would add an extra maps feature.

The Database

Sinead began by integrating the SQLite database connection into the application. SQLite was used as it does not require an external server, but instead saves the information directly to the mobile device. SQLite also provides the DbHelper class, which provides an easy way to access and then close the database after any execution statements are complete. The Holiday.java class was created to hold all the database implementations (create, insert, update, delete statements).

-Create

Below is the first SQL execution statement written by Sinead in the Holiday.java class (i.e. the DbHelper). The statement creates a new table (DATABASE_TABLE) which holds ten attributes of the entity holiday. First is the KEY_ROWID which holds the primary key of each holiday item. This holiday ID is auto-generated by the database (auto-increment) which means that the user does not need to choose an ID for any of the holidays they wish to create.

The destination name is saved as a varchar data type (the same as 'char') and has a limit of 15 characters per destination name. The rest of the columns have an integer data type and overall they store the departure date, the return date, the travel price and the ID of the holiday profile theme.

```
private class DbHelper extends SQLiteOpenHelper {

    public DbHelper(Context context){
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        db.execSQL("CREATE TABLE " + DATABASE_TABLE + " (" +
                KEY_ROWID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
                KEY_DESTINATION + " VARCHAR(15) NOT NULL, " +
                KEY_DEPARTURE_DAY + " INTEGER NOT NULL, " +
                KEY_DEPARTURE_MONTH + " INTEGER NOT NULL, " +
                KEY_DEPARTURE_YEAR + " INTEGER NOT NULL, " +
                KEY_RETURN_DAY + " INTEGER NOT NULL, " +
                KEY_RETURN_MONTH + " INTEGER NOT NULL, " +
                KEY_RETURN_YEAR + " INTEGER NOT NULL, " +
                KEY_THEME + " INTEGER NOT NULL, " +
                KEY_FLIGHT + " INTEGER NOT NULL" + "); ");
```

When the app is first compiled and run, the onCreate() method for the DbHelper creates all of the tables in the .execSQL statements, but after that the tables are not created again. Whenever a create table statement needs to be updated, the app itself needs to be uninstalled from the mobile device and then reinstalled with the new table.

In total, Holiday.java needs to create four separate tables in the same database:

-Table to store the holiday list

-Table to store the shopping list items (see Group Assignment for further code)

-Table to store the spending money items

-Table to store the simulated bank account balance

-Insert

Below is the insert code for adding a holiday item to the holiday list table. After various user defined Strings have been gathered from EditText boxes, CalendarViews and a Spinner (see Group Assignment for further code) the addHoliday() method is called with these Strings as the parameters.

An object referencing the ContentValues class is used to add all the elements into the appropriate table with just one insert statement. 'cv' acts as a container to store information about what is being put in to the table and in which column it should be placed.

→cv.put(Table_column_name, elementToBeInsertedIntoColumn); //syntax

```
public long addHoliday(String destination, String departureDay,
                       String departureMonth, String departureYear,
                       String returnDay, String returnMonth, String returnYear,
                       String theme, String flight){
    ContentValues cv = new ContentValues();
    cv.put(KEY_DESTINATION, destination);
    cv.put(KEY_DEPARTURE_DAY, departureDay);
    cv.put(KEY_DEPARTURE_MONTH, departureMonth);
    cv.put(KEY_DEPARTURE_YEAR, departureYear);
    cv.put(KEY_RETURN_DAY, returnDay);
    cv.put(KEY_RETURN_MONTH, returnMonth);
    cv.put(KEY_RETURN_YEAR, returnYear);
    cv.put(KEY_THEME, theme);
    cv.put(KEY_FLIGHT, flight);
    return ourDatabase.insert(DATABASE_TABLE, null, cv);
}
```

After all the elements have been put into the ContentValues, the .insert() method can be run on the SQLite database which tells the database which table is the information is being inserted in to, tells the database to put 'null' into any empty values, and to fill the row with the elements from the ContentValues container.

→ return SQLiteDatabase.insert(Table_name, nullColumnHack, ContentValues) //syntax

Insert methods where needed for all four tables:

-insert holiday items into the holiday list

-insert shopping items into the shopping list (See Group Assignment for further code)

-insert spending money items into the spending money budget

-insert '0' into the bank account table [this insert statement is only run once in the onCreate() method of Holiday.java when the bank account table is initially created]

-Delete

Below is the delete code for removing an item from the holiday list. The user inputs the holiday ID of the holiday item they wish to delete (into an EditText). This information is passed through deleteEntry() when the “delete” button is pressed.

The database finds the holiday item from the holiday list which corresponds to the holiday ID and deletes it. The method then calls for the database to look through the shopping list table and spending money table for any items that have the same holiday ID (i.e. shopping/spending items that correspond to the deleted holiday) and also permanently remove them to save storage space.

```
public void deleteEntry(int LRow1) throws SQLException {
    ourDatabase.delete(DATABASE_TABLE, KEY_ROWID + " = " + LRow1, null);
    ourDatabase.delete(DATABASE_TABLE_SHOPPING, KEY_SHOP_HOLIDAY + " = " + LRow1, null);
    ourDatabase.delete(DATABASE_TABLE_SPENDING, KEY_SPEND_HOLIDAY + " = " + LRow1, null);
}
```

→SQLiteDatabase.delete(Table_name, condition(where statement), whereArgs[]) //Syntax

Delete methods are needed for only three out of the four tables:

-delete a holiday item from the holiday list

-delete a shopping item from the shopping list

-delete a spending item from the spending list (See Group Assignment for further code)

Simulated Bank Account

After the table has been created and ‘0’ is inserted into its only column, the simulated bank account only needs a total of three methods to manage it:

-getAmount() to return a String of the current bank balance

-addBankAccount() to add the inputted amount to the balance

-deleteBankAccount() to subtract the inputted amount from the balance

-Query

getAmount() is a method that executes an SQL select/query statement on the bank account table. It starts by declaring a String array of the columns it wants to show in the result set (bank account table only has one column).

Next a Cursor is initialized to run the query statement on the SQLiteDatabase, checking for the one column in the bank account table (DATABASE_BANK). There are no other conditions (where, order by, group by statements etc.) so leave the rest of the parameters as null.

If the result set is not empty (i.e. c!=null), then move the Cursor to the first element in the result set. Get the String version of the column in position 0 in the String array and store it in the variable called "total". This is now the end of the query statement and the String version of the bank balance is returned.

```
public String getAmount(){
    String[] columns = new String[]{KEY_BANK_TOTAL};
    Cursor c = ourDatabase.query(DATABASE_BANK, columns, null, null, null, null, null);

    if( c!=null){
        c.moveToFirst();
        String total = c.getString(0);
        return total;
    }

    return null;
}
```

-Update

The addBankAccount() and deleteBankAccount() methods are almost identical apart from one character ('+' or '-'). They both take in the amount inputted by the user, and they both start by using the same query statement as above [*in hindsight, getAmount() could have just been called instead of repeating the code].

Once the String version of the bank balance is obtained, it needs to be converted into an integer. Next the calculation is made (either add the inputted amount to the current balance, or subtract it).

Finally, the database needs to be updated using an SQL execution statement for the addition/subtraction to be made permanent.

```
public void addBankAccount(int amount){
    String[] columns = new String[]{KEY_BANK_TOTAL};
    Cursor c = ourDatabase.query(DATABASE_BANK, columns, null, null, null, null, null);
    String total="";
    if( c!=null){
        c.moveToFirst();
        total = c.getString(0);
    }

    int current = Integer.parseInt(total);
    current = current+amount;
    ourDatabase.execSQL("UPDATE " + DATABASE_BANK + " SET " + KEY_BANK_TOTAL + " = " + current + ";");
}
```

Savings Status/Countdown

Sinead configured the holiday countdown in the main holiday profile page and the Bank Savings Status activity which can be reached from the Navigation Drawer in the holiday profile.

Many features from the HolidaySavingStatus.java file are already explained in the Group Assignment, therefore this Individual Assignment won't focus heavily on the code specifics.

The custom methods implemented for the Savings Status to work are as follows:

-getBalanceOutput() to get the balance of the money needed minus the amount already in the savings account.

-setDates() to get the current date and to retrieve the departure date of the selected holiday from the database.

-getDateDiff() to get the countdown to the departure date.

-getTotalHolidayAmount() to get the total cost from the selected holiday's shopping items, spending money items and travel price (in Holiday.java file)

//Screen shot of HolidaySavingStatus.java in action

Additional Work

After finishing the work that she initially agreed to do, Sinead researched into more android features in the attempt to make *SeeTheCapital* flow better and more user-friendly. Those implemented features are as follows:

-Alert Dialogs to explain particular aspects of the app to the user

-Dialog style activities

-Notifications to appear when the user increases the amount in their bank account

-Navigation Drawer to help the flow of the holiday profile section

-Theme Changer to allow the user to define the background of the holiday profile

Alert Dialogs

The purpose of the alert dialog is to help the user to figure out what to do in the activity. Below is the code for one out of the eight of the alert dialogs implemented.

An ImageButton (with background set to a small .jpg of a grey question mark) is created in the xml file connected to the HolidayChangeTheme.java file. When this button is pressed, the custom alert dialog method is called (themeDialog).

This method creates a new Alert Dialog object, sets the title (shown at top of dialog box), sets the message (main text), sets an icon to display beside the title (question mark .jpg), and sets the dialog box to have one “OK” button, that closes the Alert Dialog when clicked.

```
public void themeDialog (View view){
    AlertDialog alertDialog = new AlertDialog.Builder(HolidayChangeTheme.this).create();
    alertDialog.setTitle("Change the Theme");
    alertDialog.setMessage("Here you can change the background of your holiday profile." +
                           "Drag the slider right and left, and select which holiday"+
                           " theme you think best suits your destination.");
    alertDialog.setIcon(R.mipmap.qmark);
    alertDialog.setButton("OK", new DialogInterface.OnClickListener() {
        @Override
        public void onClick(DialogInterface dialog, int which) {

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

Dialog activity

Sinead changed the theme of two java files (AddShopping.java, AddSpending.java) to display them as pop-up dialog boxes instead of full paged activities. This is done in three lines of code.

First in the onCreate() method of the java file, set the theme of the activity to a different style resource. Sinead chose that the user must press on of the buttons for activities to change (instead of touching the screen outside of the dialog box).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    setTheme(android.R.style.Theme_DeviceDefault_Dialog_NoActionBar);
    this.setFinishOnTouchOutside(false);

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_add_shopping);
```

Finally, in the Android Manifest file, one line of code is needed inside the activity:

→android:theme=”@android:style/Theme.Dialog”

```
<activity
    android:name=".AddShopping"
    android:theme="@android:style/Theme.Dialog" >
    <intent-filter>
        <action android:name="com.example.sinead.seethecapital.AddShopping" />

        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
<!-- AddShopping is in the style of a dialog box/ -->
```

Notification

The purpose of the Notification is to inform the user when there has been an increase in their savings account, and encourage them to revisit *SeeTheCapital* to check their target.

The code to this feature was already explained in the Real-World Application section in the Group Assignment.

Navigation Menu

The purpose of the Navigation Menu is to allow the holiday profile section of the app flow better. The user can slide the screen from left to right to open a menu. When an option in the menu is selected, the user is brought to the Activity that corresponds.

Although Android Studio auto-generates the code for a Navigation Menu, it was a task in itself to read and understand this code, and make tweaks to the numerous navigation activities to customize SeeTheCapital. Some code is explained already in the Group Assignment.

Theme Changer

The purpose of the Theme Changer is to allow the user to choose a background suitable for the selected holiday profile. It is an activity in the Navigation Menu and implements the use of an ImageView and SeekBar.

A more detailed explanation can be seen in the Group assignment.

Sinead's overall contribution

- Came up with the idea for *SeeTheCapital*
- Contributed to half of the Project Submission assignment and designed the mockflow layout
- Wrote and presented two out of three slides for the class presentation
- Splash screen code (thread, timer + closing notification)
- Complete management of the SQLite database
- Implemented simulated bank account and Saving Status calculations
- Navigation menu
- Alert dialogs + dialog styled activities
- Notifications
- Theme Changer
- Helped with minor adjustments to the Google maps elements
- Pieced the app together (added her partner's Google maps code, images and Splash screen music to her own work in a fresh project)
- Commented all of *SeeTheCapital*'s code (apart from the main Google maps part and Splash Screen music [as this is her partner's contribution])
- Contributed towards the Group Assignment (Outline, Screen Flow, Methodology, Theme Changer, Database, Savings Status, Navigation Drawer, Real-World Application, Future Projection)