ChipRider
AQA GCSE Computer Science
Mobile Assignment

1. Design of solution
2. Solution Development
3. Programming Techniques Used
4. Testing and Evaluation

Design of Solution

What the problem involves

This project has four separate parts:

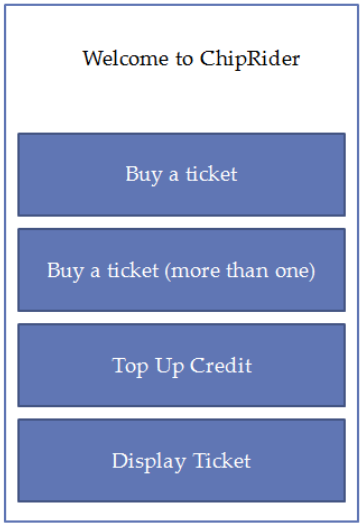
- Buying a ticket for one passenger (single or return)
- Buying single tickets for a group of passengers
- Topping up credit up to 30 credits
- Viewing the most recently bought ticket

Each one of these parts will be a separate menu on my app and the first menu will have buttons that link to all of them.

When a passenger buys a ticket the app must know how many credits they have left and also it needs to record when they bought a ticket, how much it cost and how many passengers it is for. I will use a database for this.

MIT AppInventor has all of the features I need because you can use it to create screens and write the code for all of the buttons, labels and text boxes and you can also use it to design the screens as well. AppInventor also has a web-based database so I can save all of the information from the app to the web so there it doesn't matter if the app is closed after the ticket is bought because the information is still there. This also means the drivers code doesn't have to be stored in the phone which is more secure.

The first screen will look like this:



Buying a ticket for one passenger

The general program is like this:

1. The driver enters their code
2. The passenger ticks a box if they want a return
3. The passenger presses the buy button
4. The app finds out how many credits the passenger has
5. If the number of credits is less than the cost of the ticket then the passenger is taken to the top up screen
6. But if the number of credits is more than or equal to the cost of the ticket then the time and date of the ticket, the cost of ticket and the number of passengers (1) is saved to the web and the cost of the ticket is deducted from the total number of credits.
7. The screen goes back to the main screen.

Buying a ticket for more than one passenger

You can only buy singles for a group so the general program is like this:

1. The driver enters their code
2. The passenger enters the number of passenger
3. The passenger presses the buy button
4. The app finds out how many credits the passenger has
5. If the number of credits is less than the cost of the ticket x the number of passengers then the passenger is taken to the topup screen
6. But if the number of credits is more than or equal to the cost of the tickets then the time and date of the ticket, the cost of ticket and the number of passengers is saved to the web and the cost of the ticket is deducted from the total number of credits.
7. The screen goes back to the main screen.

Topping Up Credit

The app can only have a maximum of 30 credits so if the students tries to top up more than that they should have a warning message. You can get to this screen directly and you can also be directed here from the two buying screens if you don't have enough credit.

The general program is simple and looks like this:

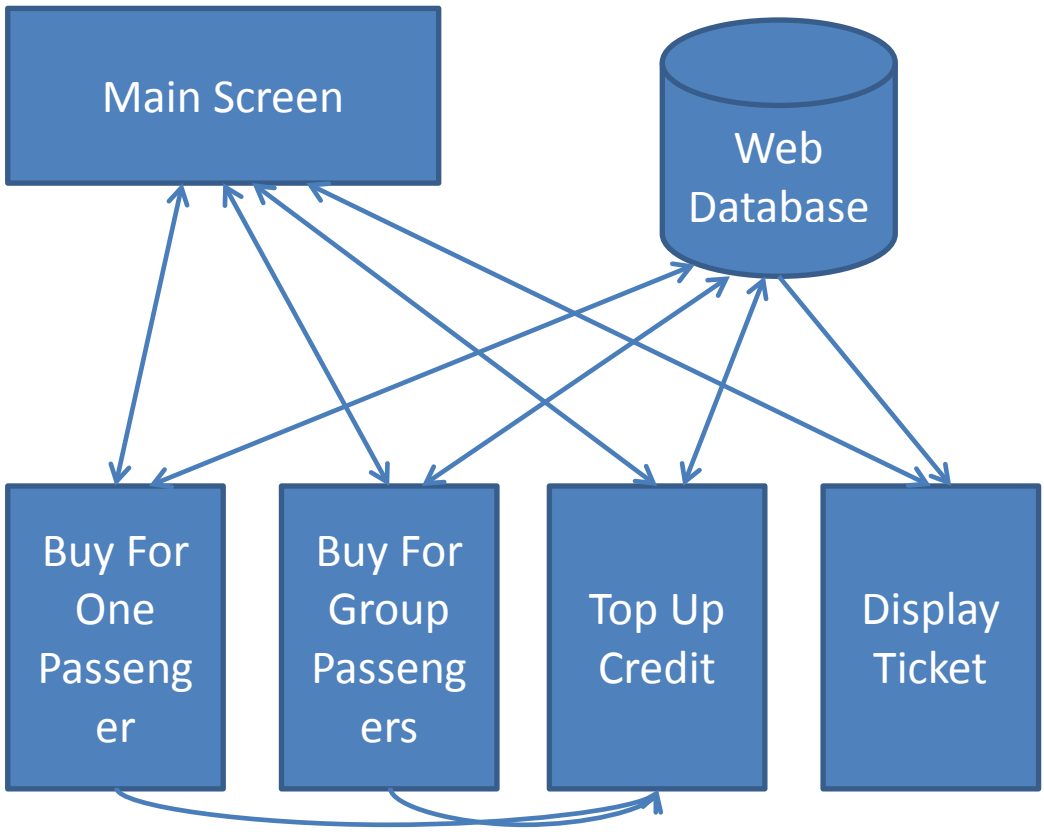
1. The current number of credits is found out from the web and this is displayed at the top of the screen
2. The driver enters their code
3. The passenger enters the number of credits and then they press buy
4. If the original number of credits + the number to buy is less than or equal to 30 then the topping up goes ahead
5. Else a warning message is displayed
6. The screen returns to the previous one.

Displaying the ticket

This is the simplest screen because all that has to happen is the values of the ticket type, the date and time and the cost of the ticket are found out from the web database and they are displayed.

Overview

This diagram shows how you can get between all of the screens, it also shows how the database gets information from screens and sends information to screens.



Pseudocode

The pseudocode for **buying a ticket for one passenger** is like:

```
DriversCode ← GetFromDatabase(DriversCode)
Credits ← GetFromDatabase(Credits)
# this could come from a database
IF DriversCode = 1234
THEN
     # check that its not a return
     IF TicketType ≠ 'Return'
     THEN
          # check enough credits
          IF Credits ≥ 3
          THEN
               Credits ← Credits - 3
               Cost ← 3
          ENDIF
     ELSE
          IF Credits ≥ 5
          THEN
               Credits ← Credits - 5
               Cost ← 5
          ENDIF
     ENDIF
# save all the data to a database
SaveToDatabase(Credits)
SaveToDatabase(TimeRightNow)
NumberOfTickets ← 1
SaveToDatabase(NumberOfTickets)
ENDIF
```

The pseudocode for **buying a ticket for more than one passenger** is like this:

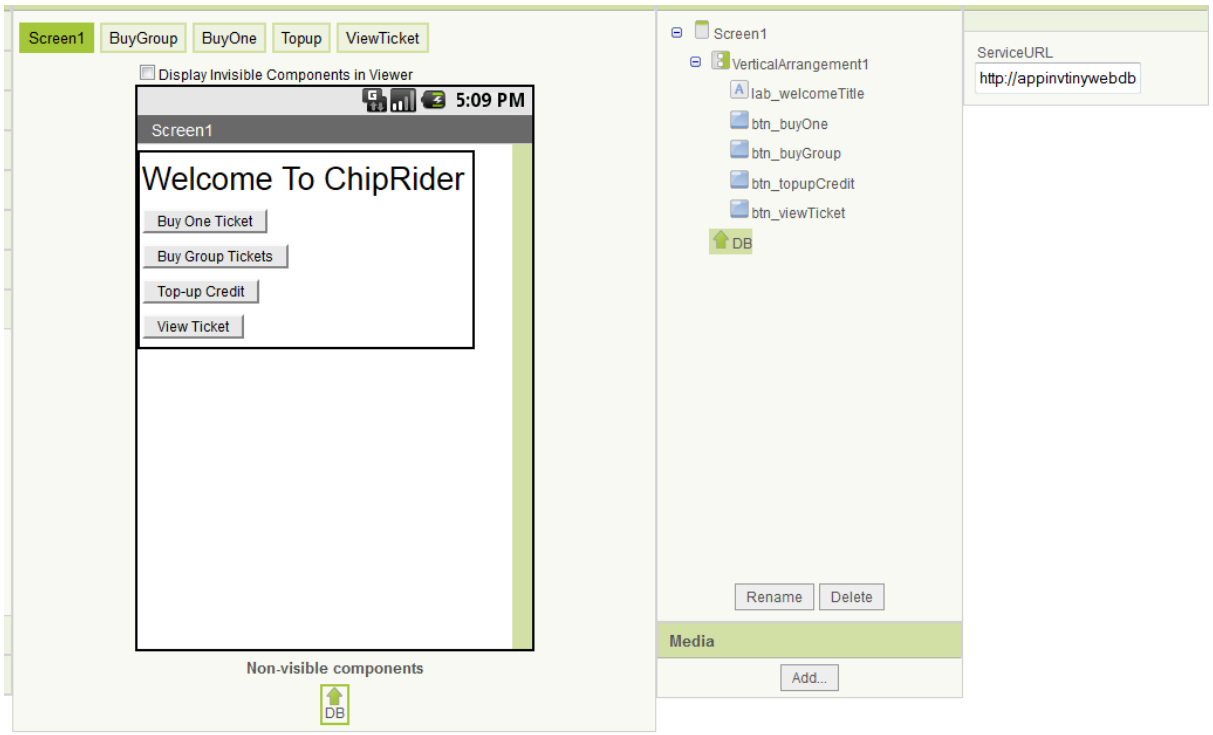
```
DriversCode ← GetFromDatabase(DriversCode)
Credits ← GetFromDatabase(Credits)
IF DriversCode = 1234
THEN
     # check enough credits
     IF (NumberOfPassengers * 3) ≤ Credits
     THEN
          # update credits and record time bought
          Credits ← Credits - (NumberOfPassengers * 3)
          BoughtOn ← RightNow
     ENDIF
ENDIF
SaveToDatabase(Credits)
SaveToDatabase(BoughtOn)
SaveToDatabase(NumberOfPassengers)
```

The pseudocode for **topping up credit** looks like this:

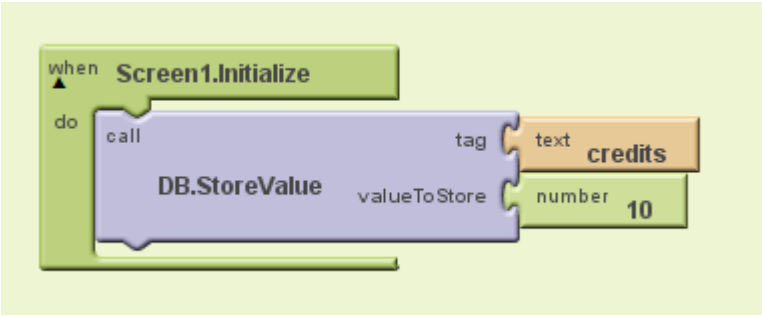
```
DriversCode ← GetFromDatabase(DriversCode)
Credits ← GetFromDatabase(Credits)
IF DriversCode = 1234
THEN
     # check that doesn't go over 30
     IF CreditsToAdd + Credits ≤ 30
     THEN
          # update credits
          Credits ← Credits + CreditsToAdd
     ENDIF
ENDIF
```

Solution Development

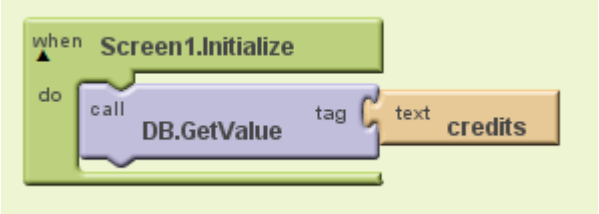
This is the development of the main screen (showing the four buttons that take you to the four different screens)



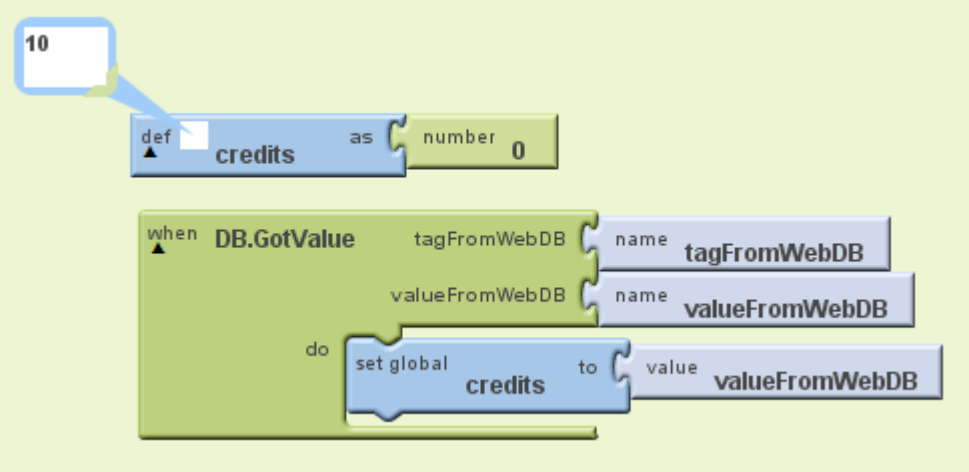
This is the section of code that is run when the application starts – this is only run once before I deleted it (in order to give the credits a starting value)



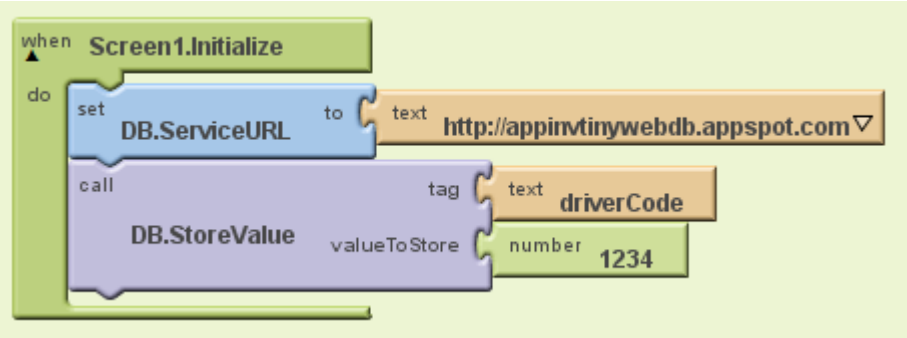
This is the actual code that will be run every time the screen starts.



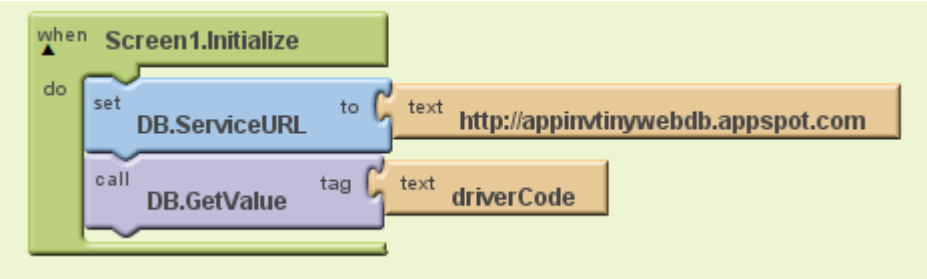
This is the code that gets the credits when the answer from the database is returned (the 10 is the variable watcher)



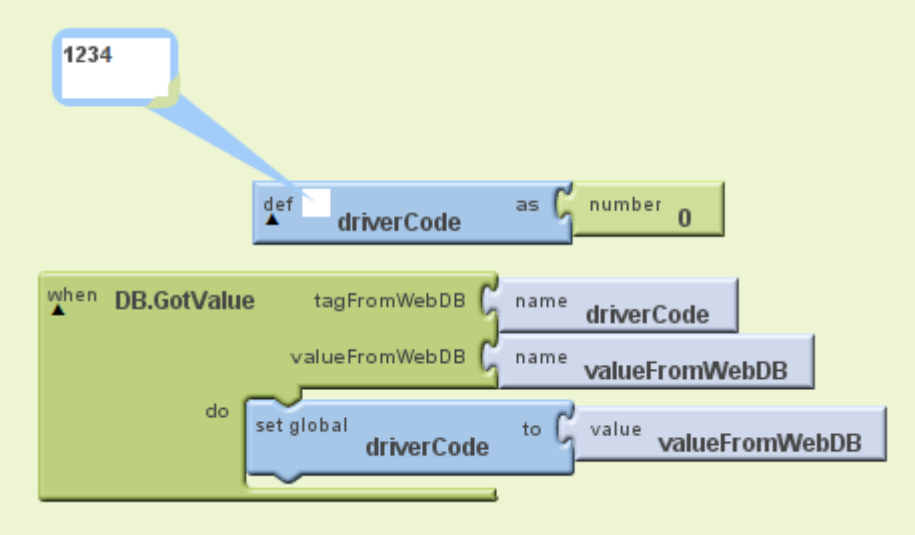
I'm doing exactly the same for the drivers code – this is only run once before I will delete this.



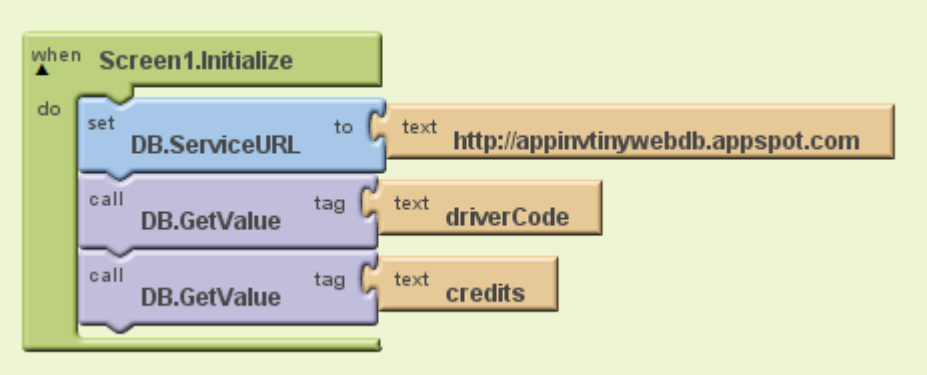
This is the final code for getting the driverCode.



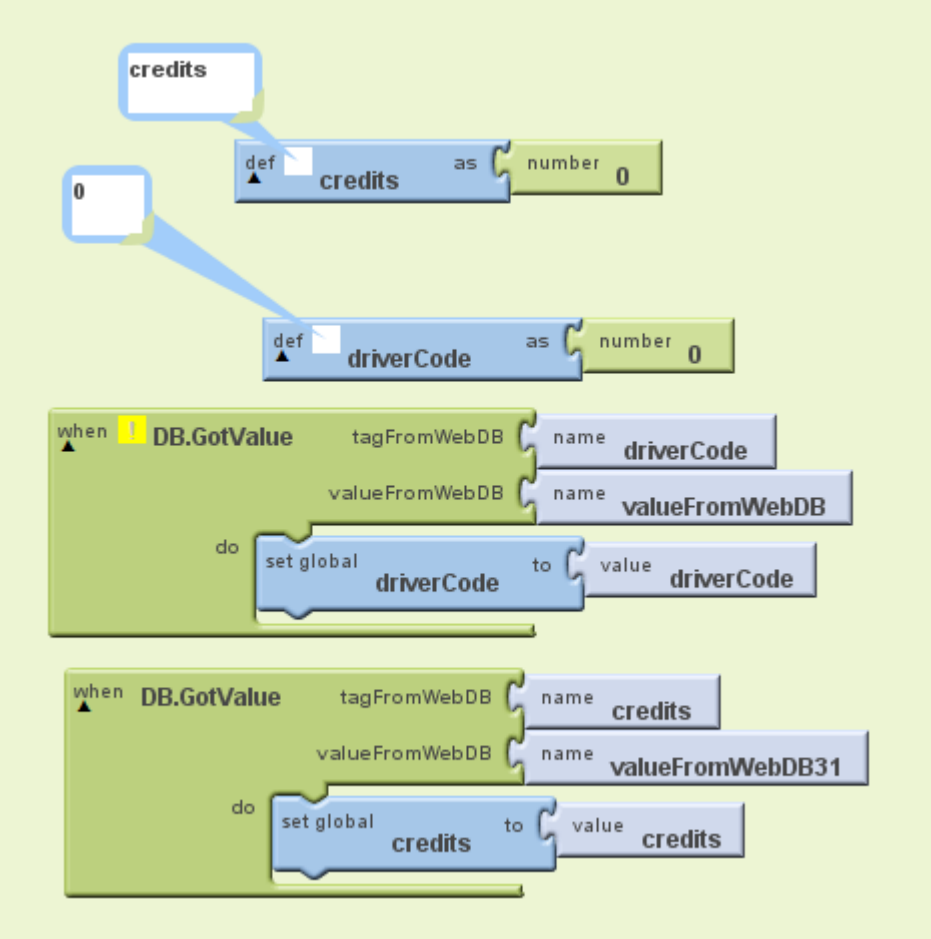
This is watcher again showing that it has worked.



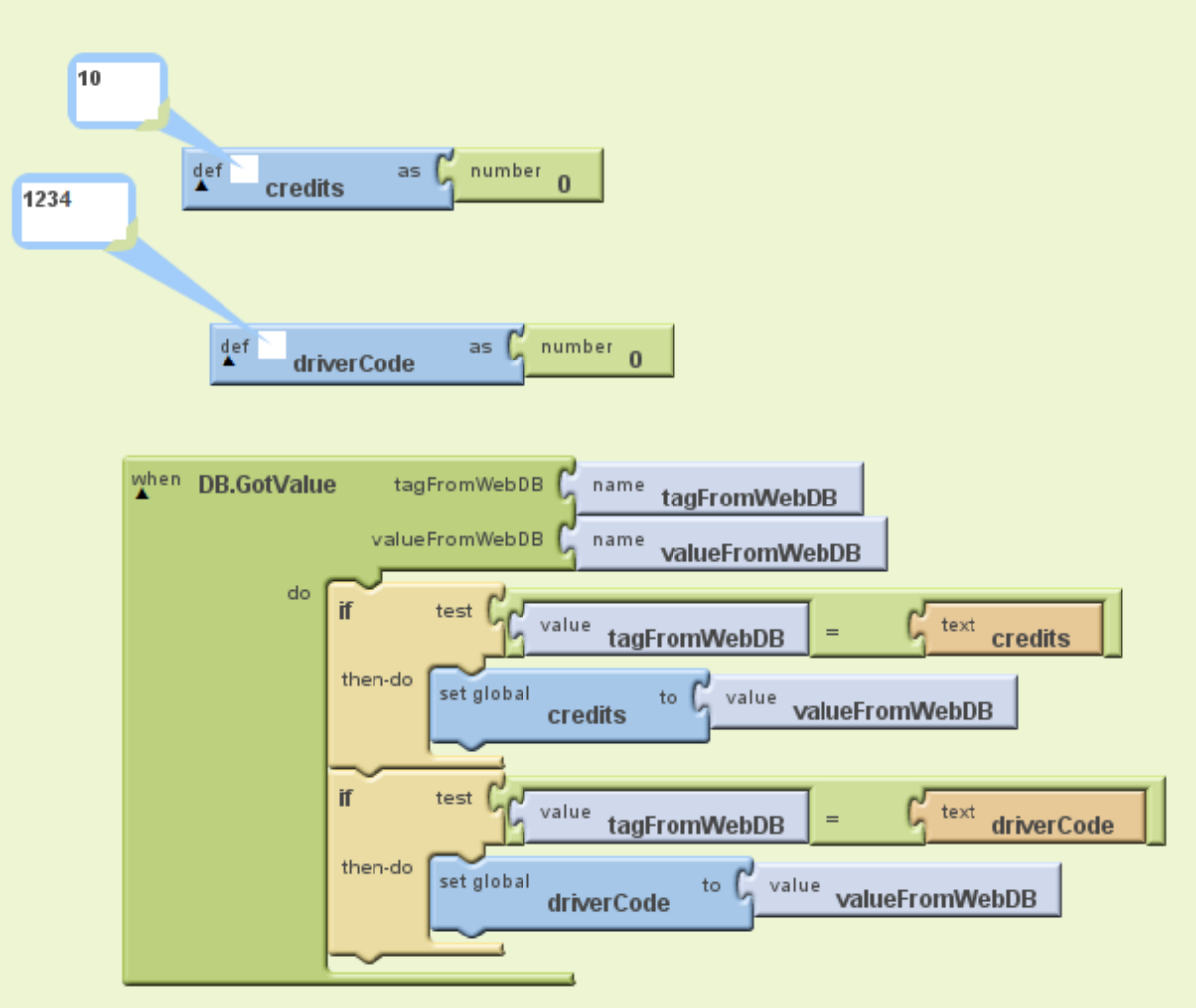
This is the final version that puts the two database calls together.



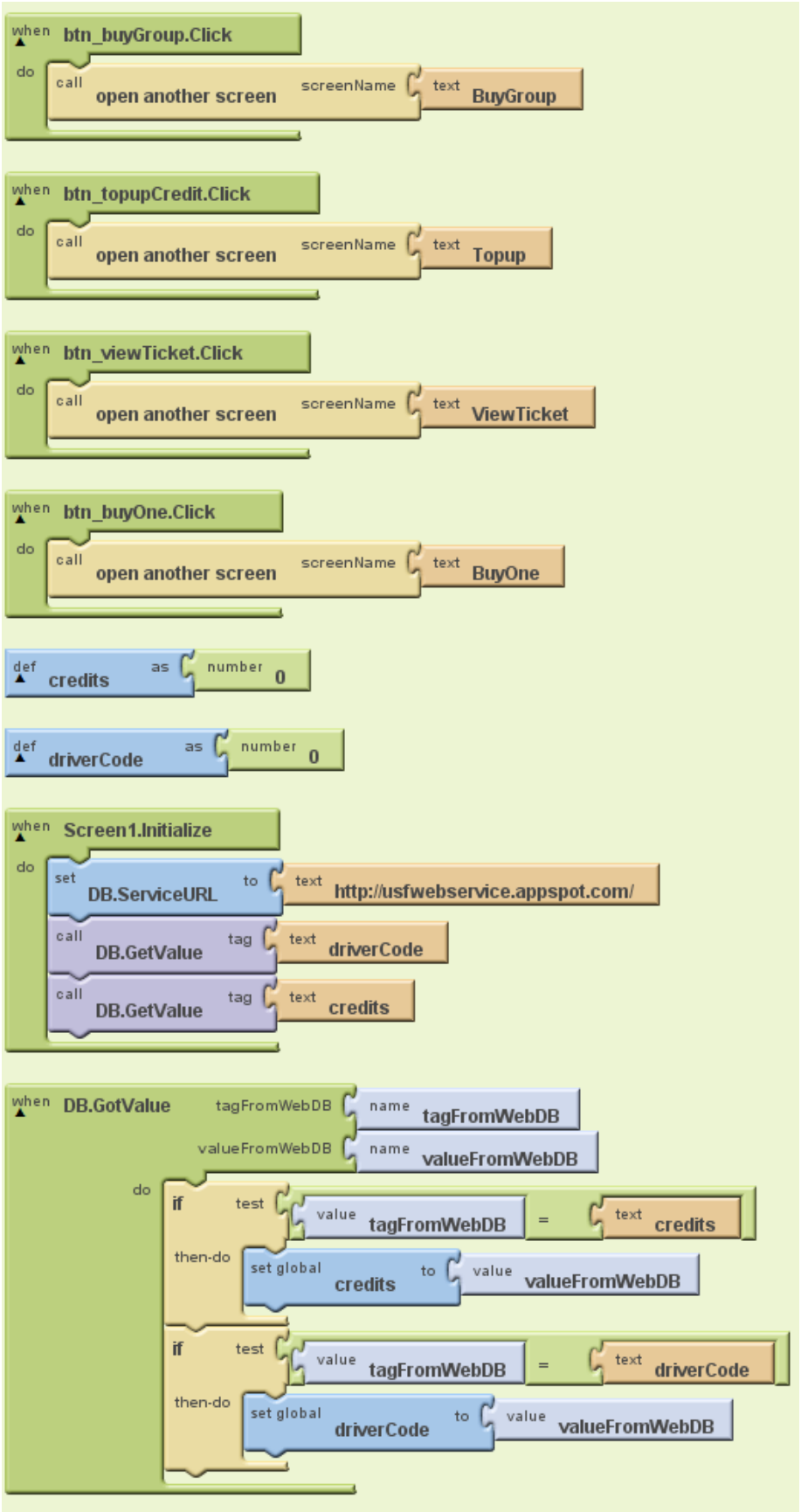
This was my first attempt at putting the two database returning parts of the code together but you can see that it didn't work.



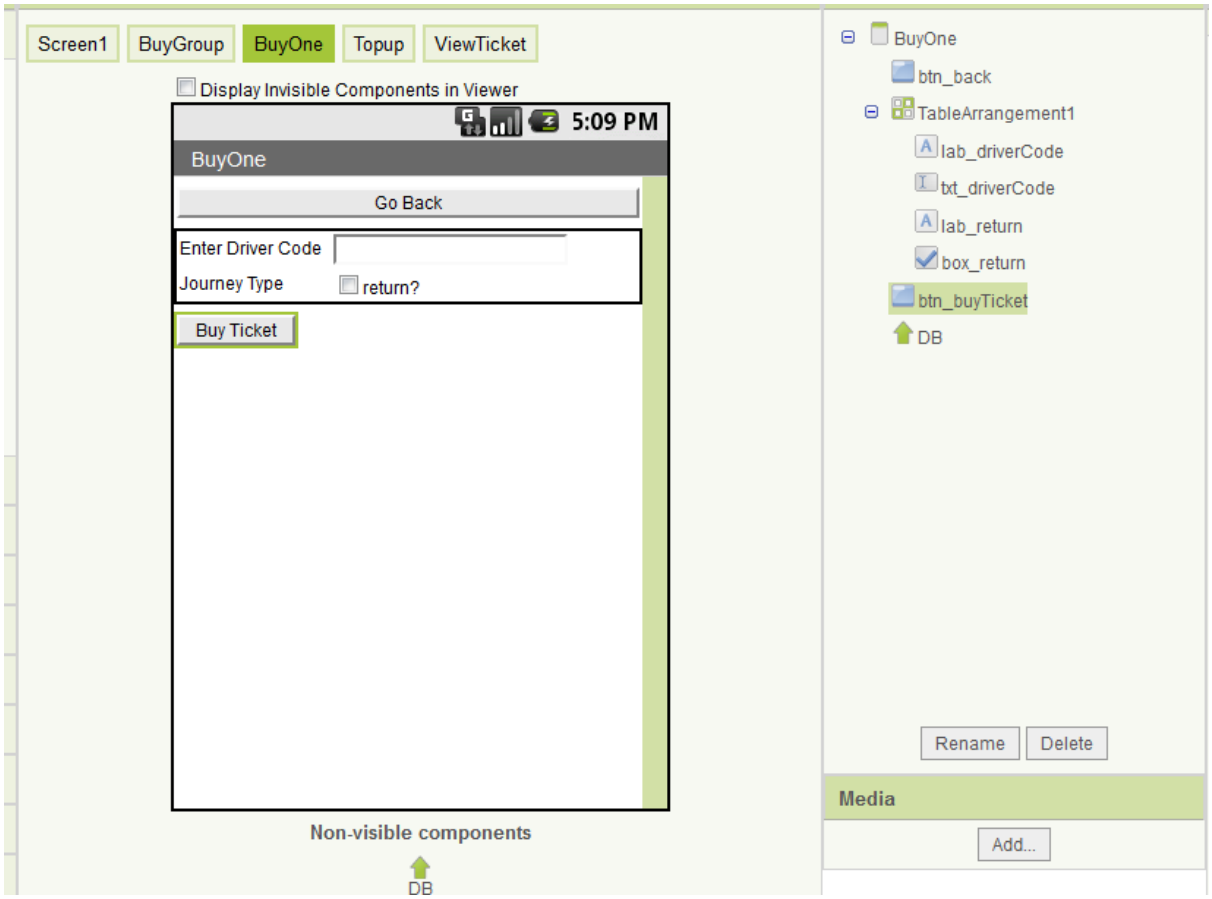
I hadn't realised the correct way to get information from this web database; this is the final version that works out what the 'tag' from the web is and puts it into the correct value.



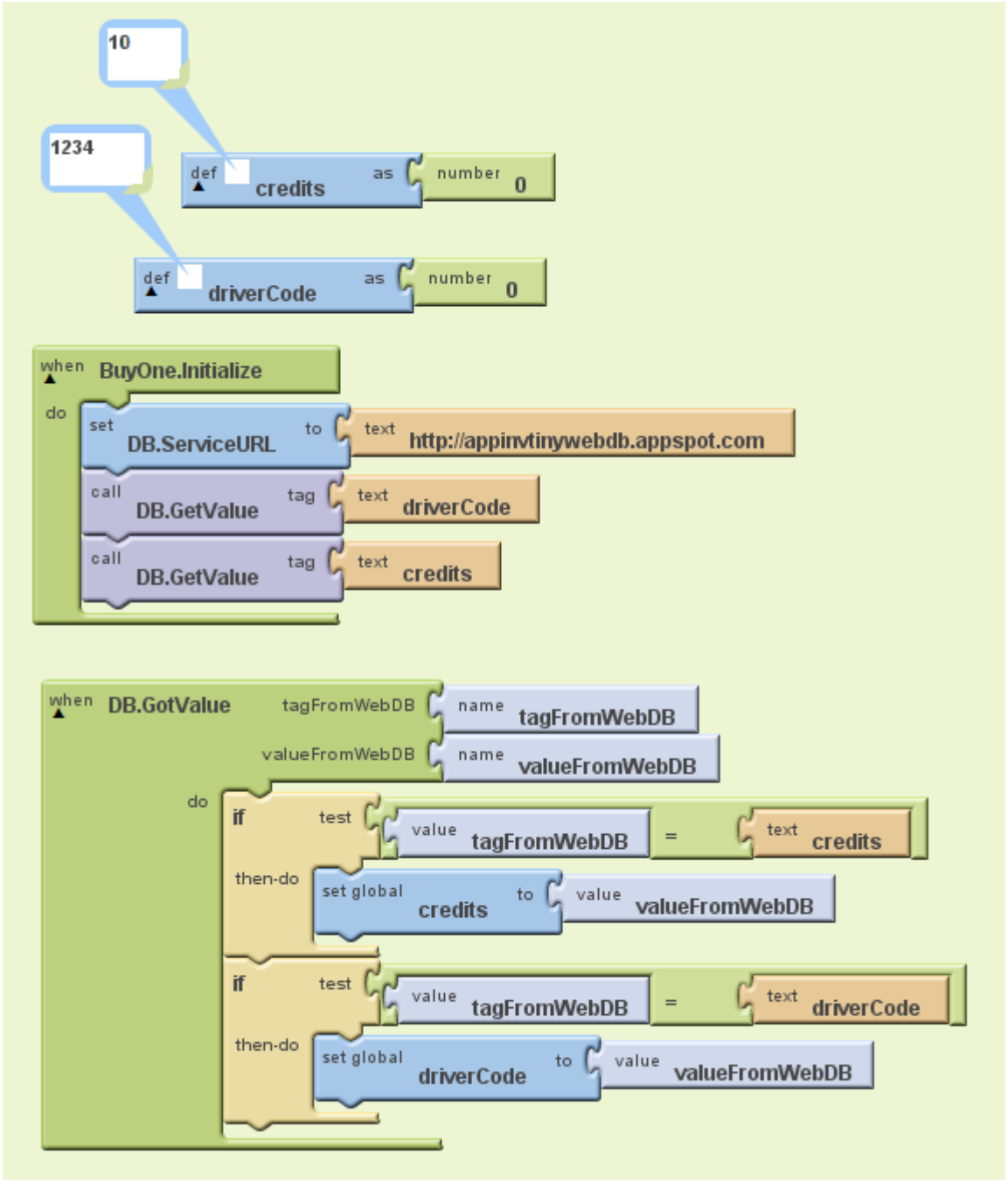
This is the completed code for the first screen showing all of the things that happen when the four different buttons are pressed.



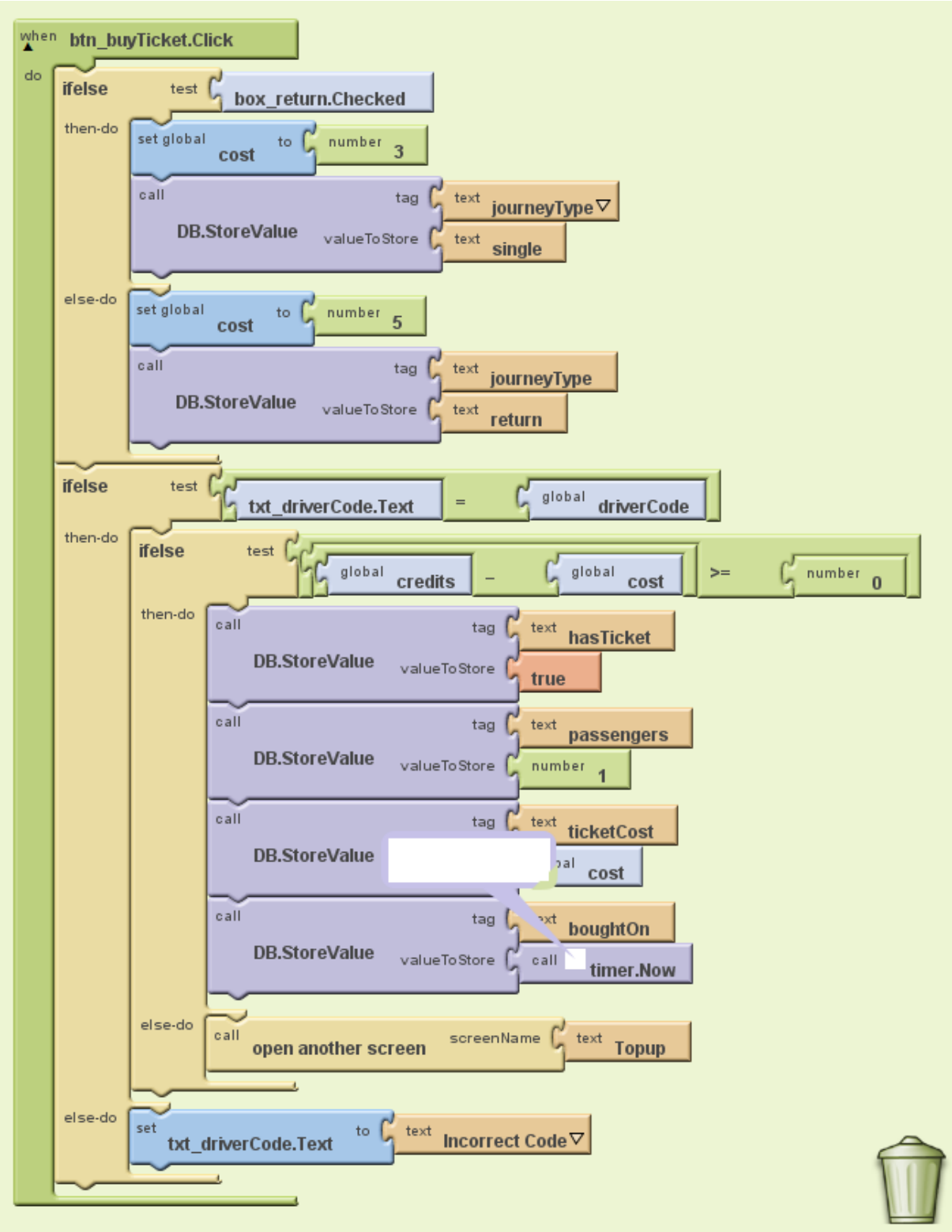
This is the screen showing the development of the screen to buy a single ticket.



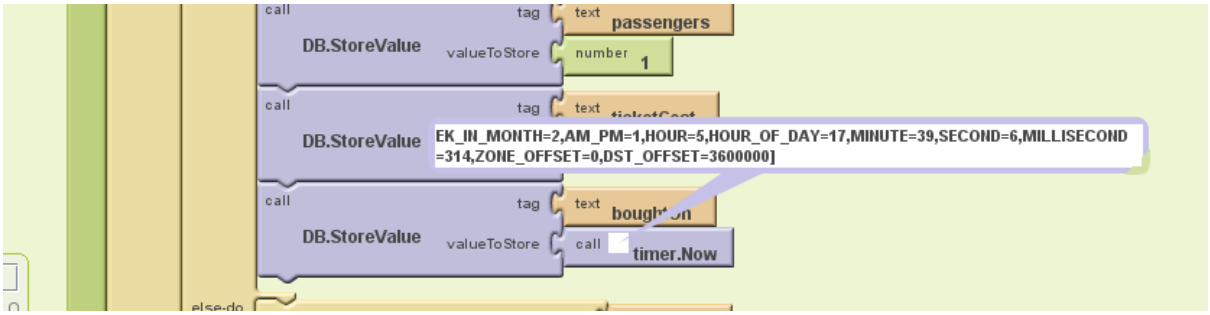
This part of the code is very similar to the code on the first screen (the watchers show that it works).



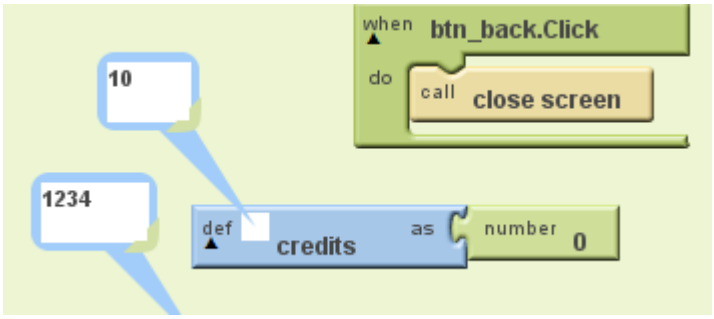
This is what happens when the buy ticket button is pressed. First of all the cost of the ticket is 3 or 5 depending on whether the return check box is ticked (by default it is not). Then the drivers code is checked to see if it is correct. If it is then the total number of credits – the cost is checked to see that it will be 0 or above and if all of that is correct then the details of the ticket are saved to the database.



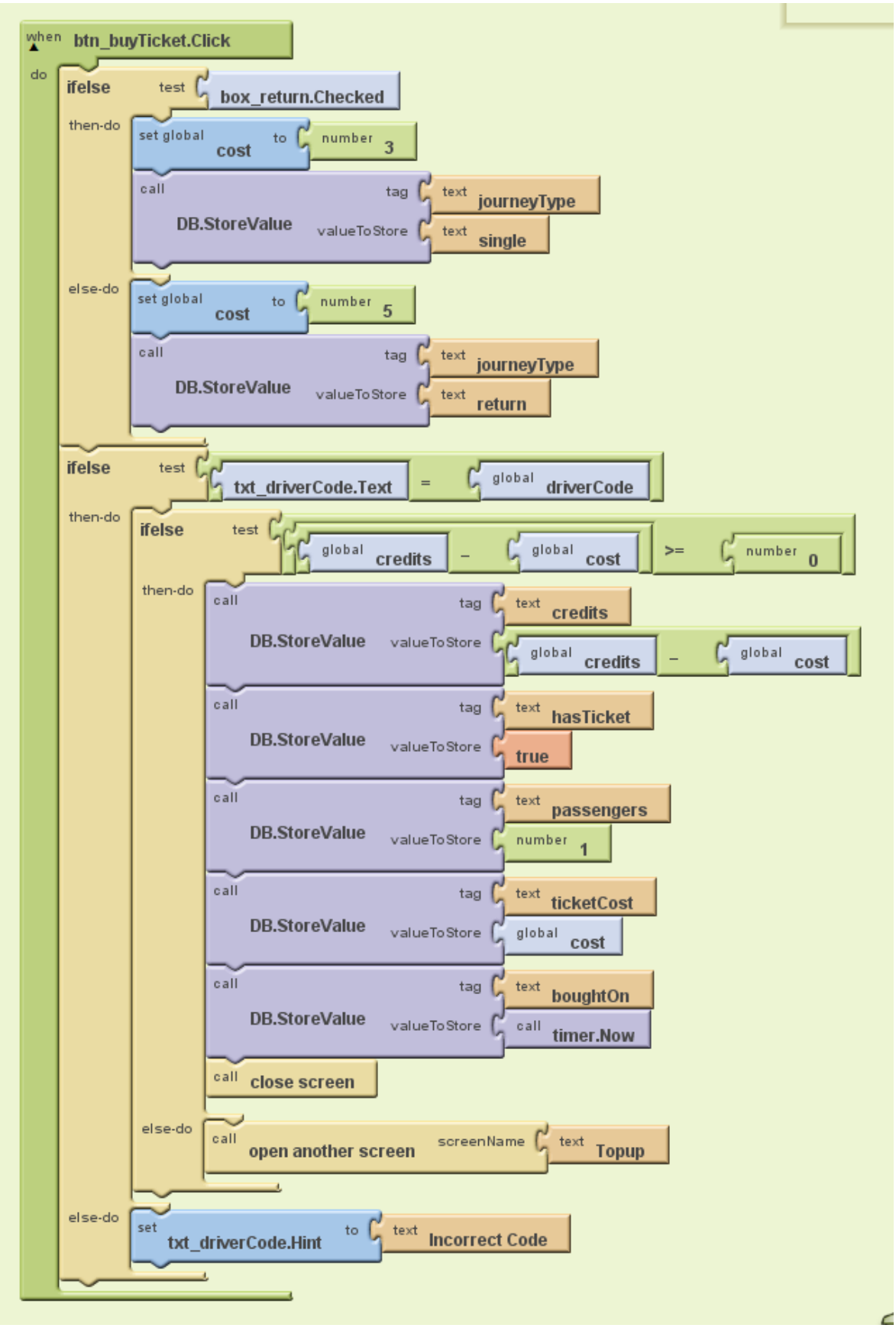
This shows the bit of code after a ticket has been bought and you can see that the timer works.



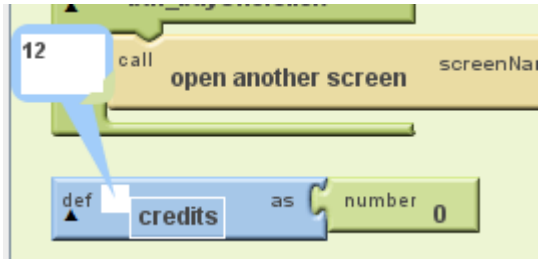
This shows the credits and the drivers code but unfortunately the credits didn't update. This is because I forgot to save the credits to the web database.



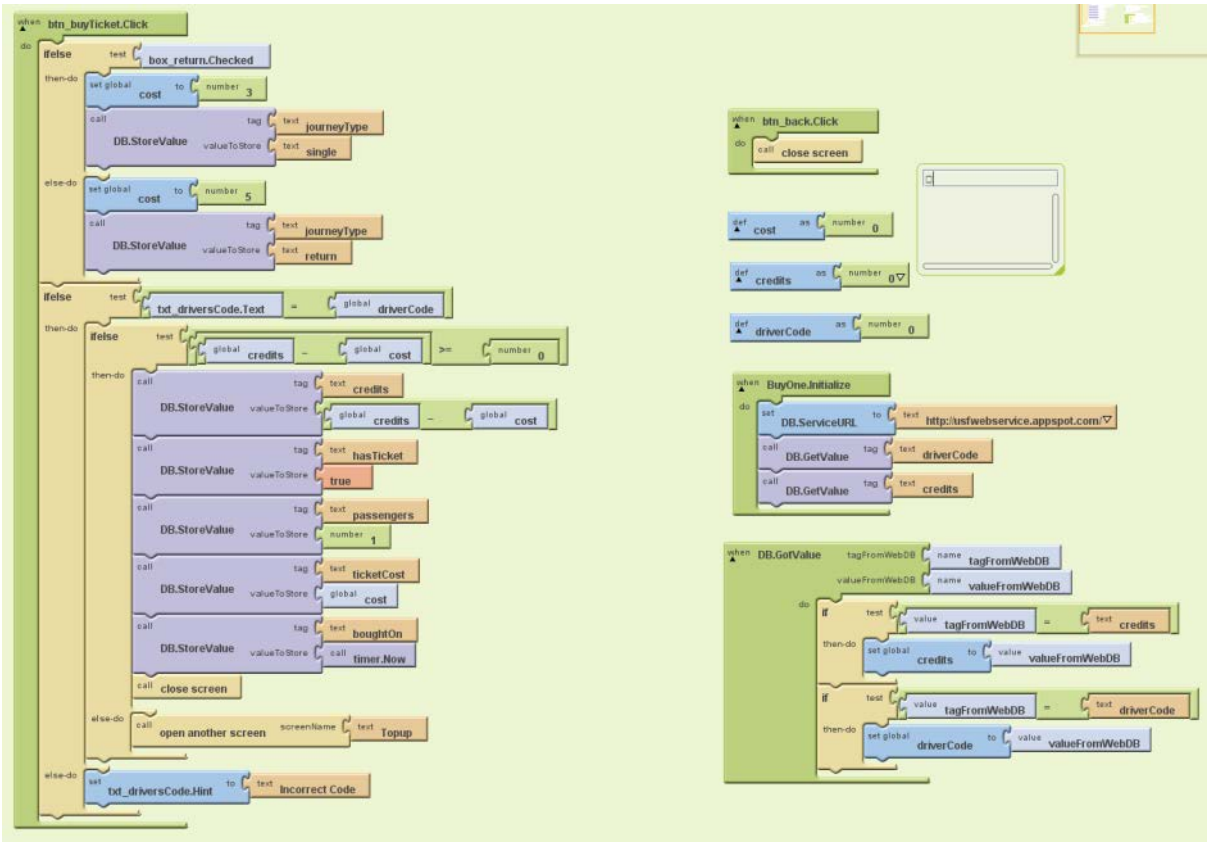
This is the completed code that also saves the credits (minus the cost of the ticket).



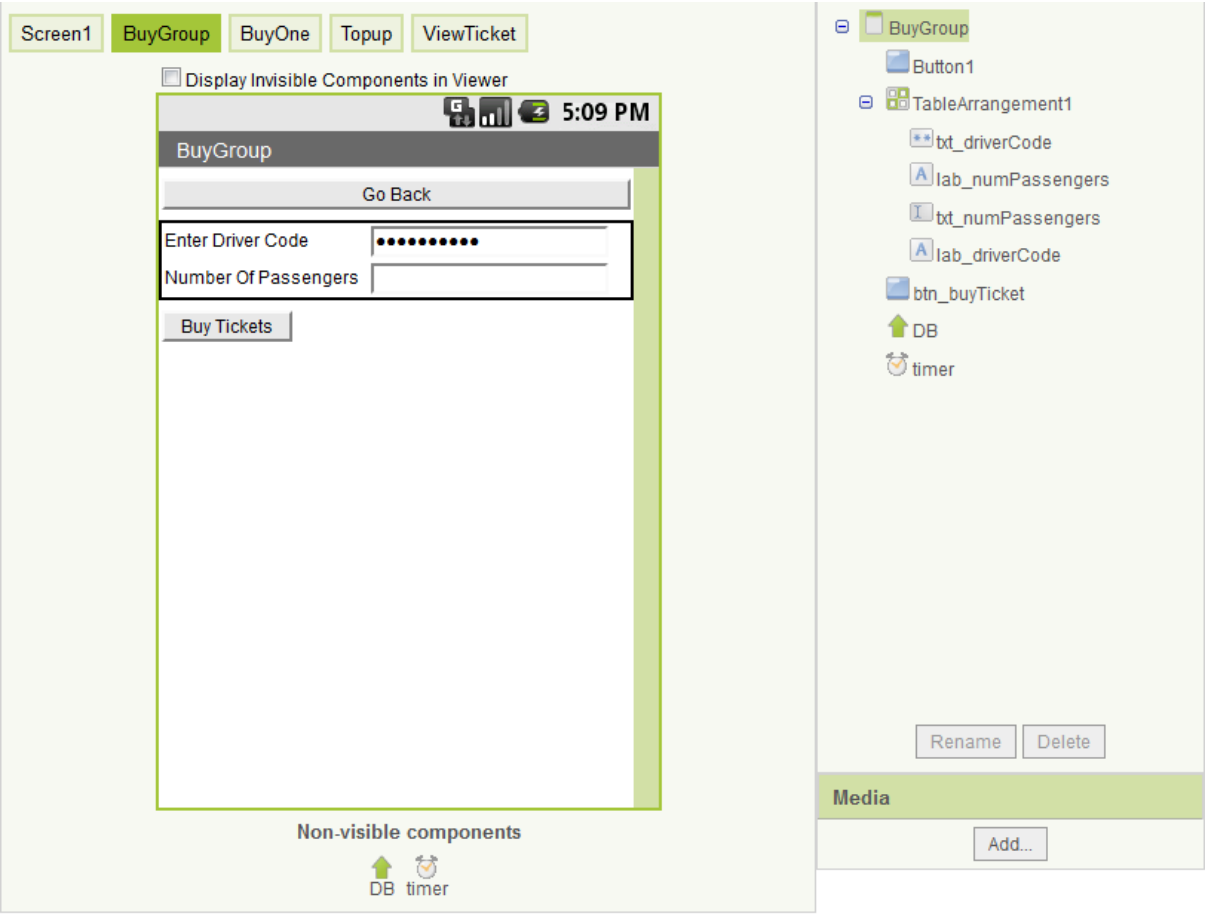
This shows the number of credits after a ticket has been bought – this is shown more in the testing section.



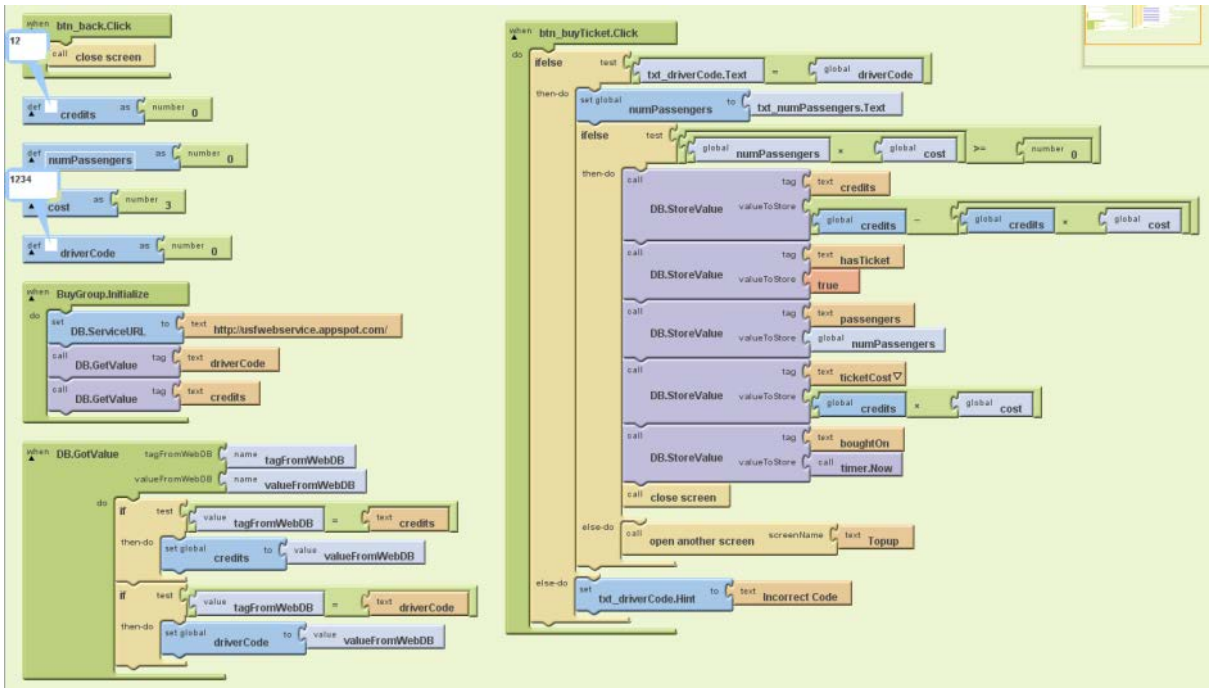
This is the final code.



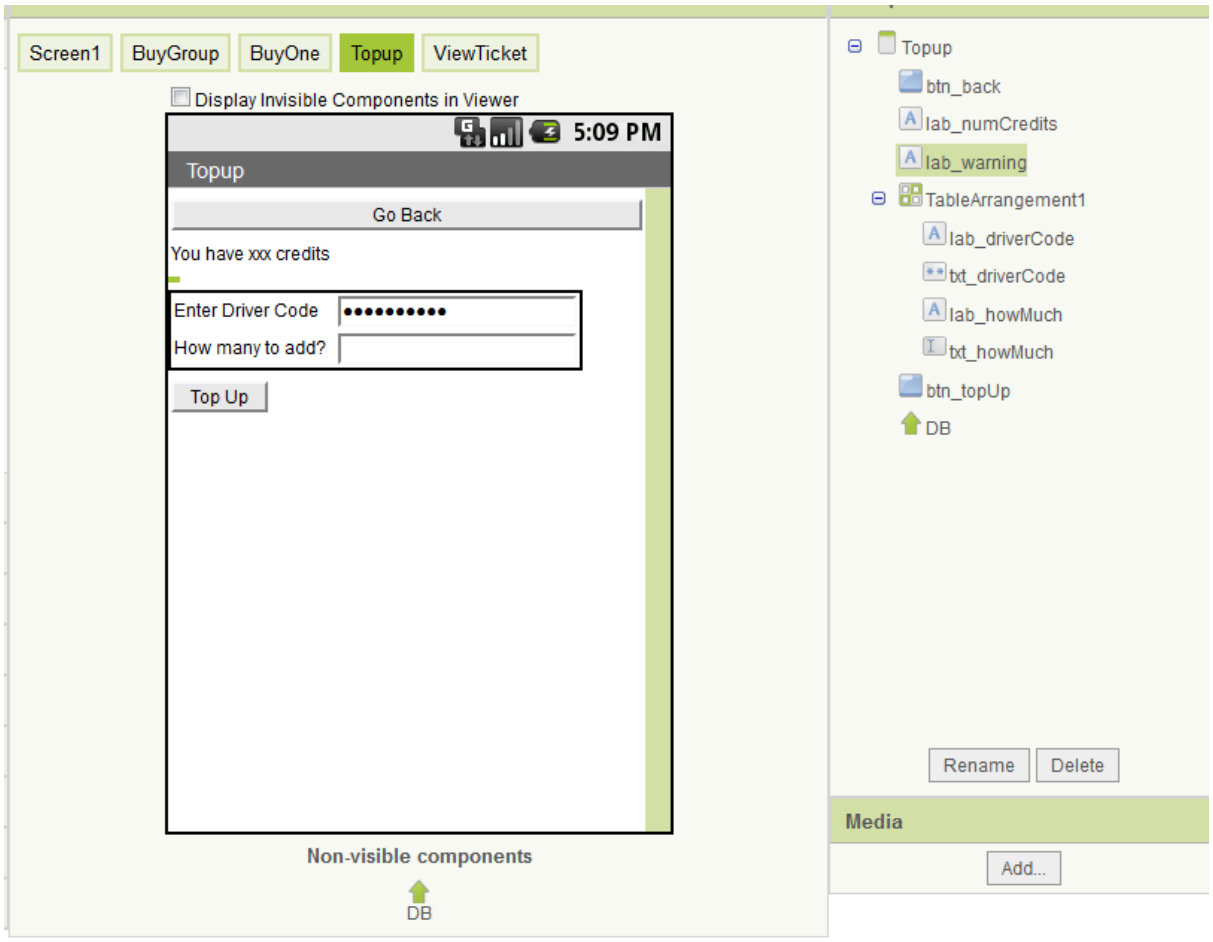
This shows the development of the screen to buy tickets for groups of passengers. The field to enter the driver details is a password field that will show the input as small black balls and not the actual characters; this is like all of the similar screens.



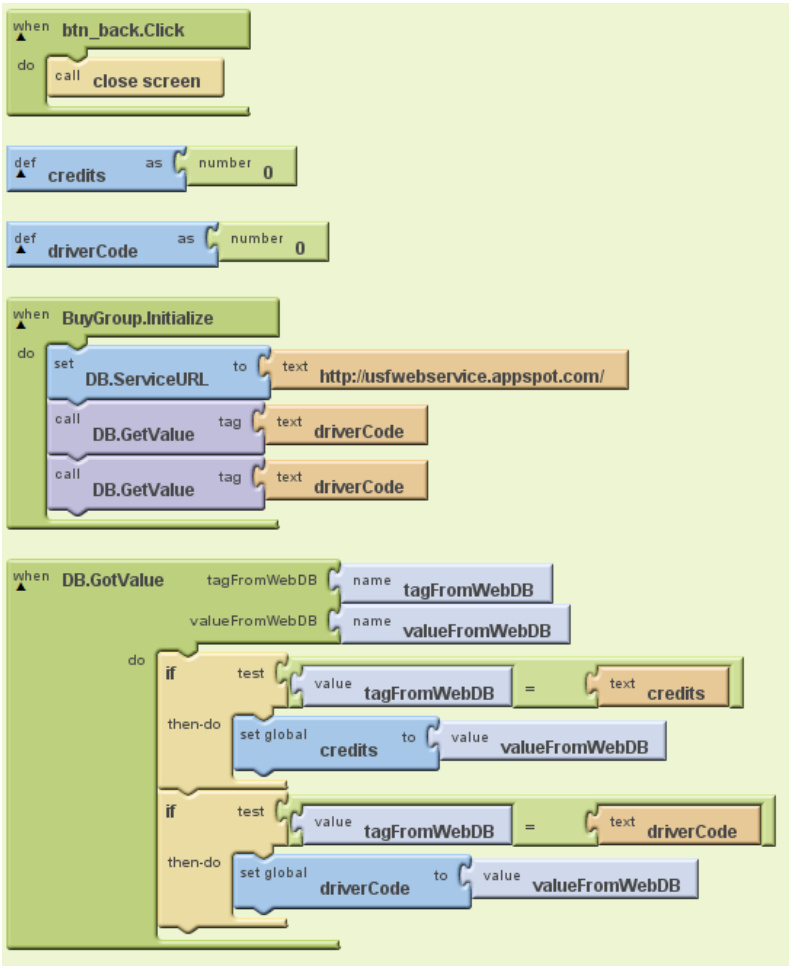
This is the complete code for this screen. The left hand side is very similar to the other screens in that it makes calls to the web database to find out the credits and the drivers code and also creates the variables numPassengers (which is a number) and cost (which is also a number). The code on the right is what happens when the buy button is pressed. Firstly the drivers code is checked and if it is correct the credits are checked to see if they are enough (the cost variable is always 3 because you cannot have return fares with groups of passengers). If the student has enough credits then the database is updated (credits, passengers, the Boolean hasTicket, numPassengers, ticketCost and boughtOn).



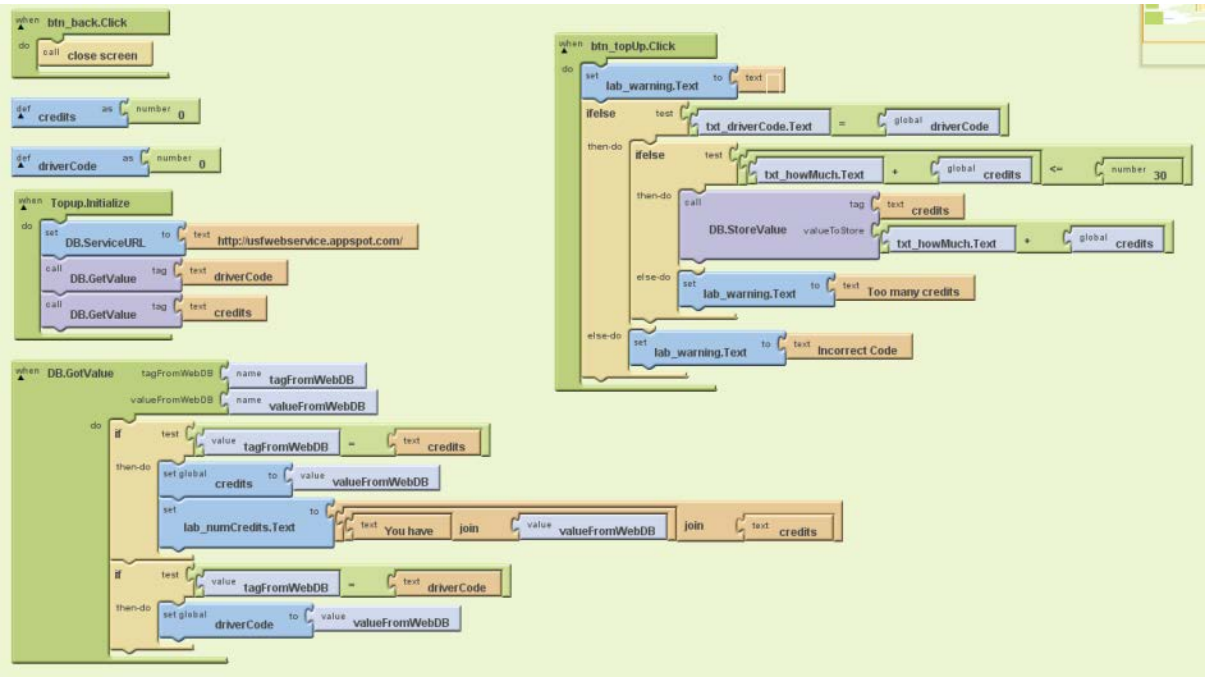
The top up screen displays information on the number of credits at the top of the screen (it says xxx at the moment because this is the development, actually this will be replaced with the number).



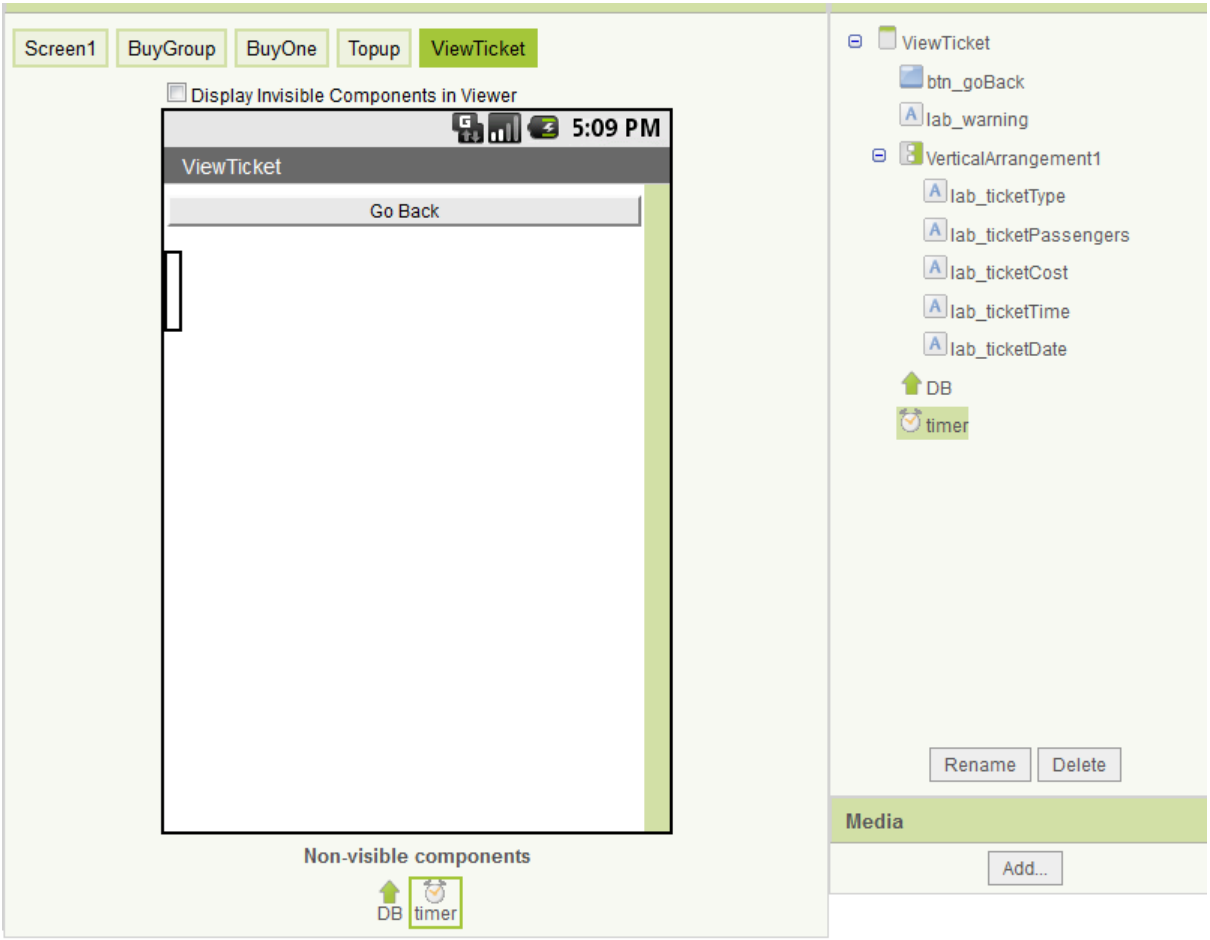
The set-up code is very similar because this screen needs to know the driver code and also how many credits the student has (this involves calls to the web database).



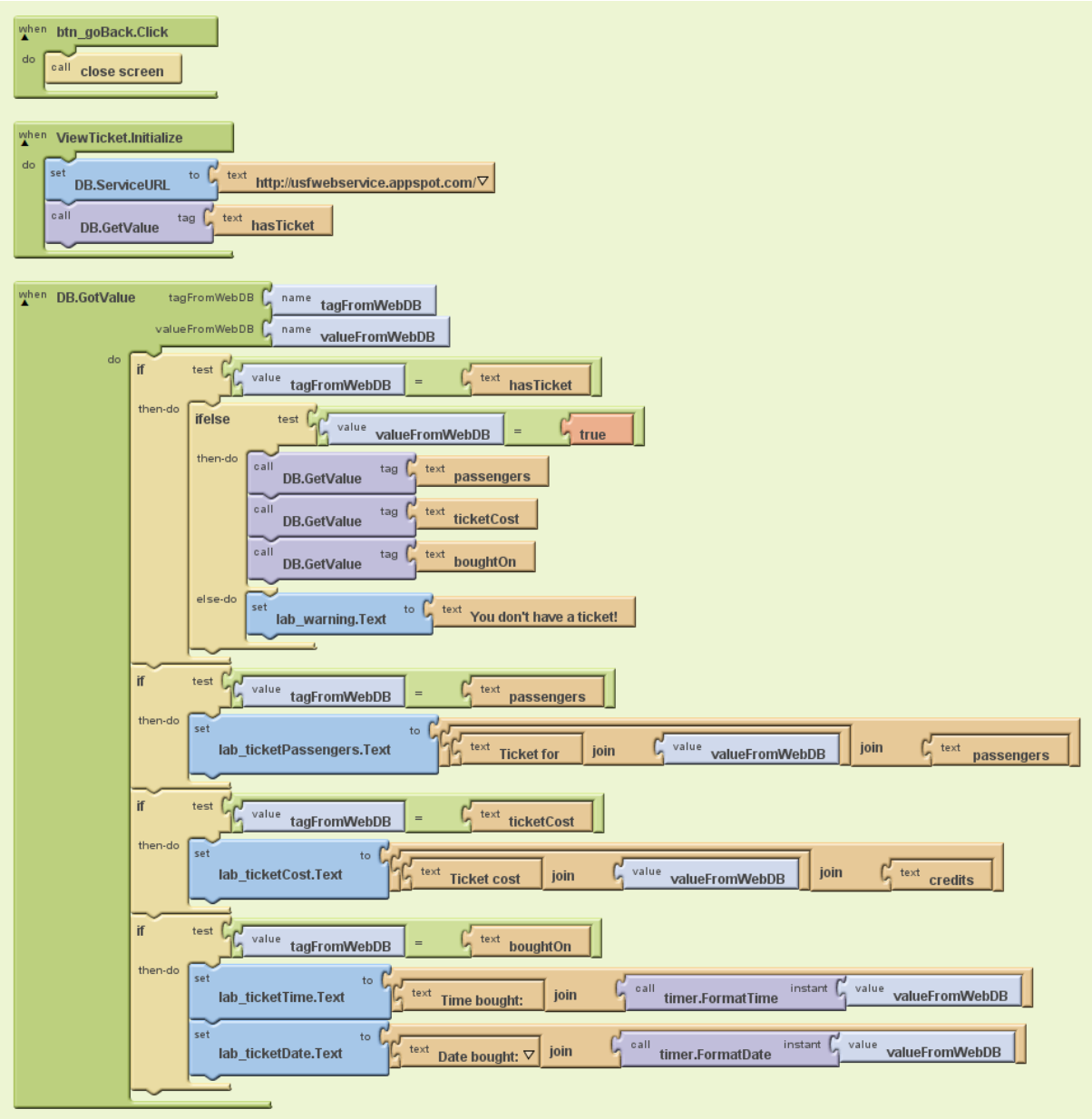
This is the whole code. I changed what happens when the database values are returned; now it creates the string at the top of the screen that displays the amount of credits the passenger has left. The right hand side of the code checks the driver's code and then checks that the credits to be added plus the actual credits the student has is not more than 30 (which is the limit). If it is 30 or under then the database is updated. If it is over 30 credits or the driver's code is wrong then a warning text is displayed.



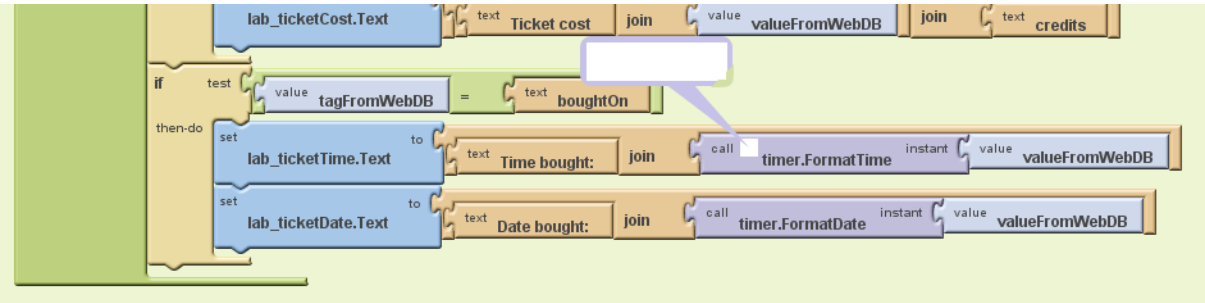
This is the final screen for displaying the ticket. It looks blank because there is no ticket to be displayed but if a ticket had been bought it would display when the ticket was bought (time and date), how much the ticket cost, whether it is a single or return and how many people the ticket is for.



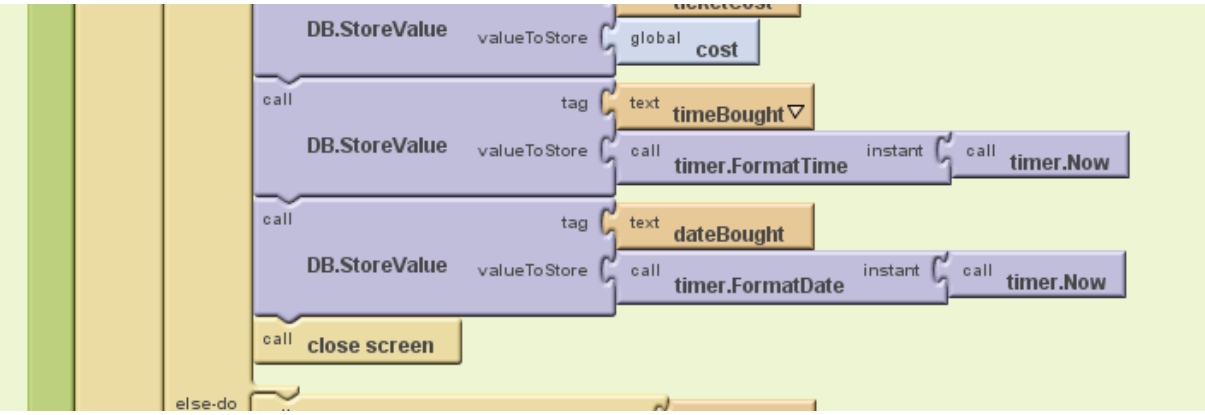
This is the completed code. The first thing that happens is the Boolean variable hasTicket is checked with the database – if it is true then a ticket is displayed otherwise the message “You don’t have a ticket!“ is displayed. If it is true then it in turn asks the database for the passengers, ticketCost and boughtOn values. When each one of these are returned they are made into a string with appropriate labels and they are displayed.

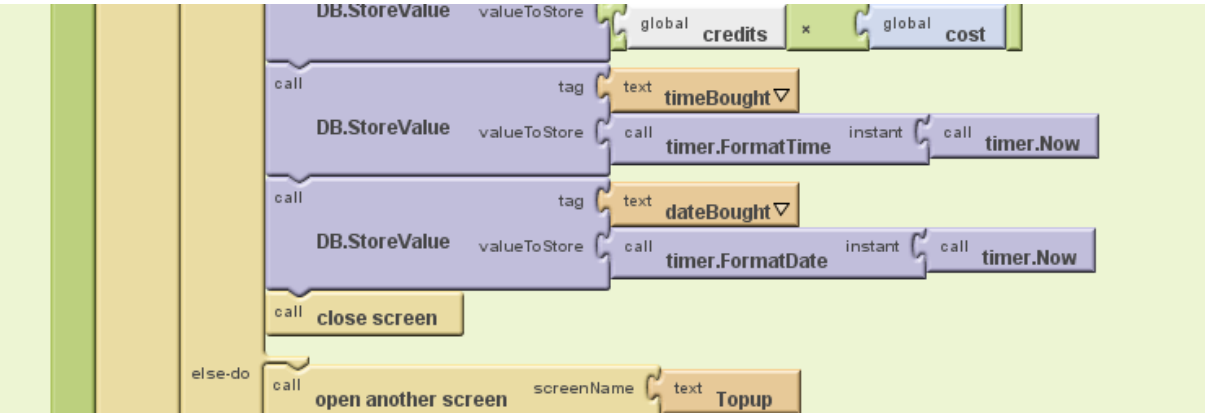


When I ran this the value of the formatted time was blank so I knew there was an error.

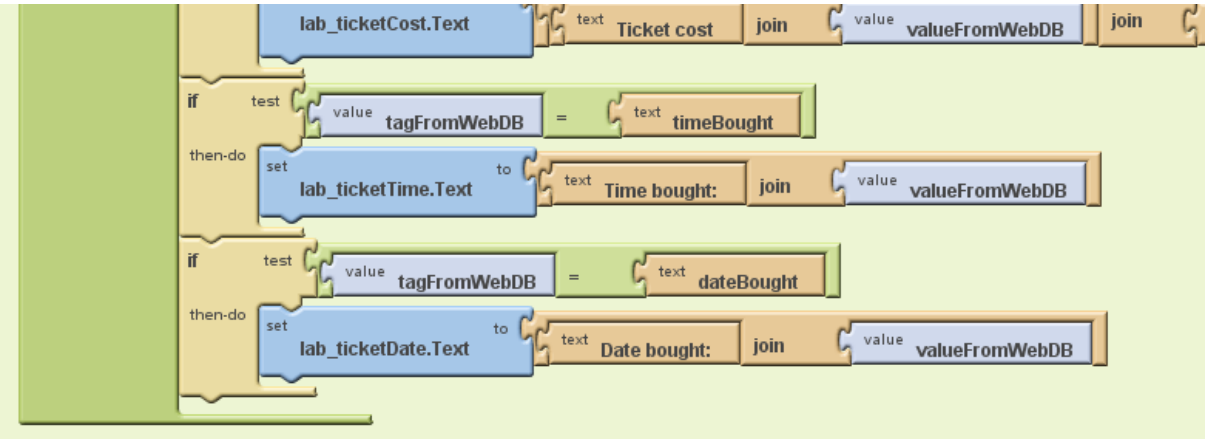


I changed the code from where the ticket was bought (single passenger and group code) to create two variables called timeBought and dateBought and formatted them before saving them.

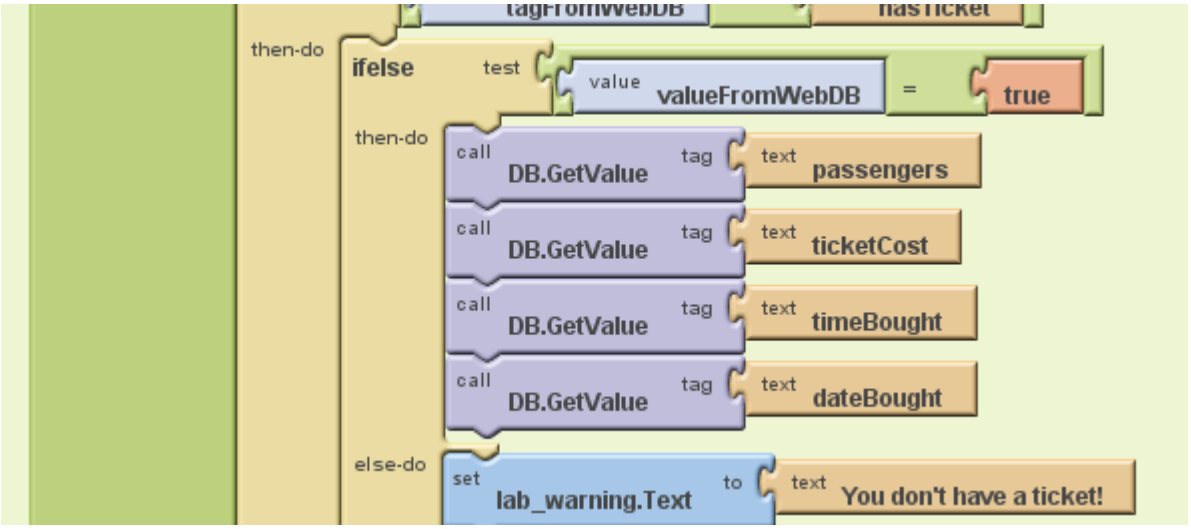




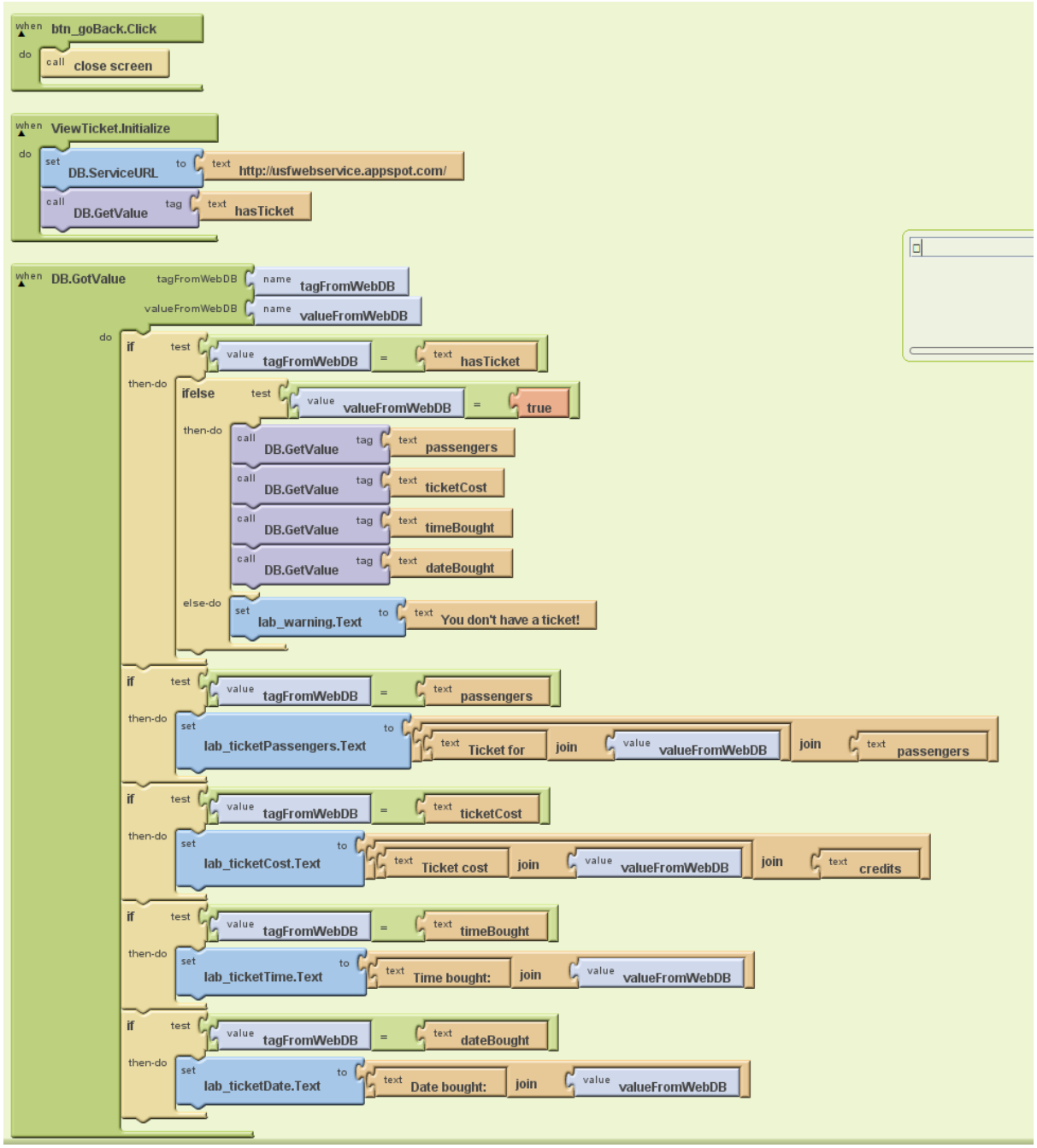
This meant that I didn't have to format the string returned from the database and instead I could use it straight away in the messages.



I also had to change what was called from the database.



This is the completed code for the print ticket section:

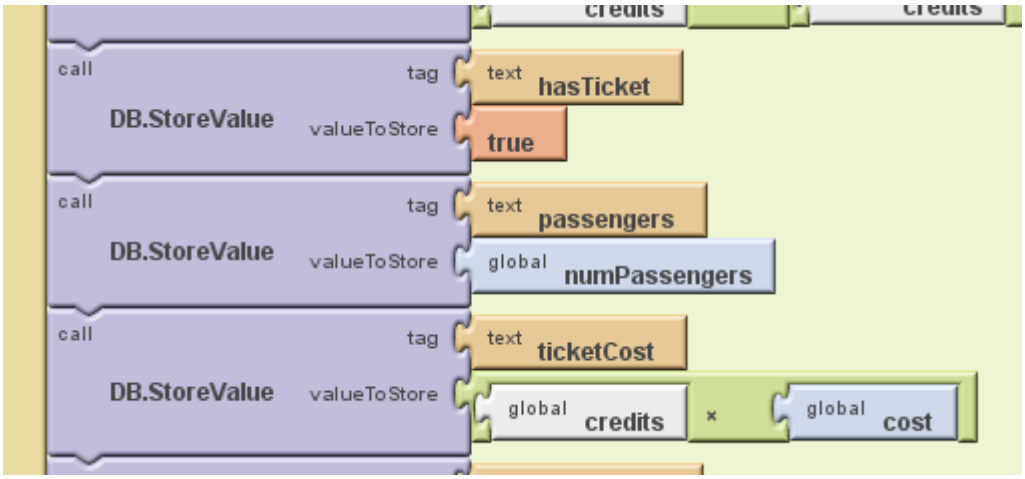


Programming Techniques Used

Use of a database

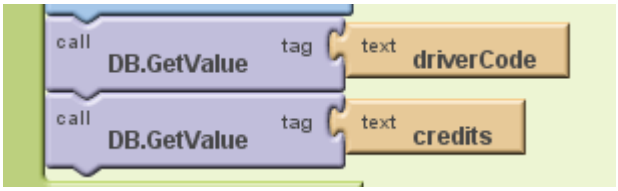
I have used an external web based database (this is not very secure and if this app was to be developed for real then it would need a more secure version). The database can be seen here: http://usfwebservice.appspot.com/ This database just contains 'tags' and their values so it is like a dictionary in Python and not like a relational database but it works for what I want.

Values are saved to the database using the StoreValue procedure of the database:

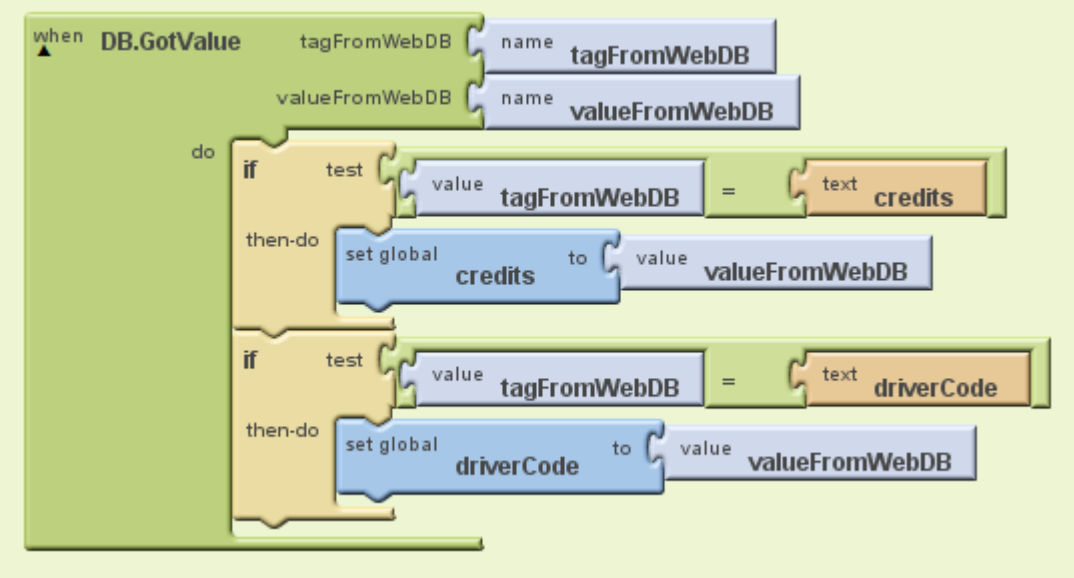


This associates the value of tag with the value of valueToStore.

Values are found out from the database in two parts. Firstly the call is made to the database to find a value:



However, this doesn't mean they will come straight away or in the order I asked so I have to have another procedure of my database called GotValue. This checks what the tag is and then you do stuff with the valueFromWebDB:

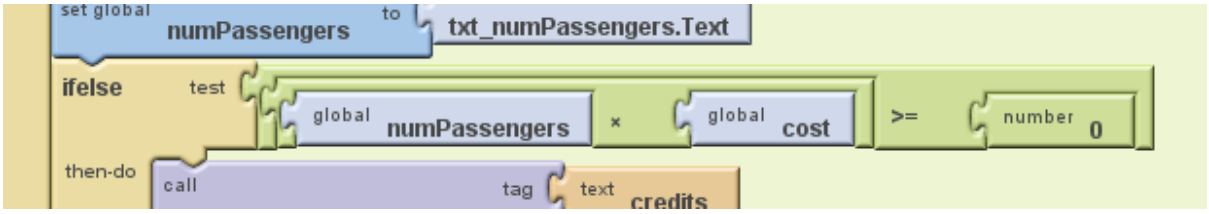


I have had to repeat quite a lot of code throughout my project because the different screens do not share global variables so every new screen needs another call to the database. This is quite inefficient but I don't know another way I could have made this work.

This is the line of code that tells the program where the database can be found:

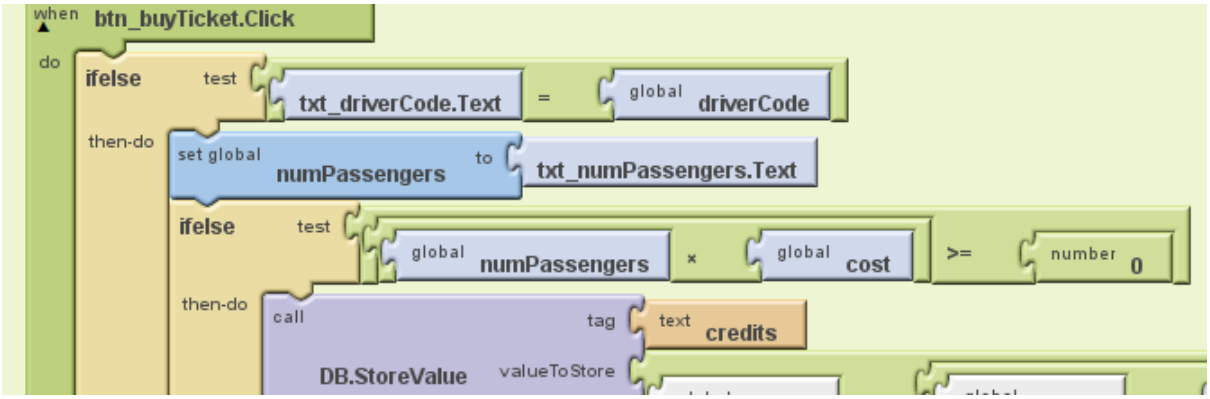
Control Statements

Throughout my project I have had to use selection statements to decide what direction the program should take. For instance this one here checks that the number of passengers multiplied by the cost is greater than or equal to 0:



I've used IFELSE and IF statements for when I don't need an alternative if the answer is false.

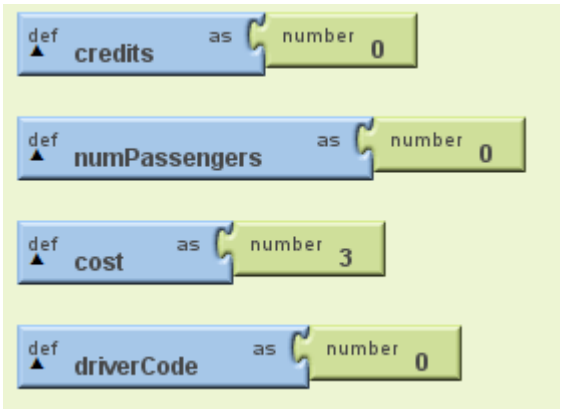
This is a nested conditional:



This means that if the first Boolean expression is true then the second one also has to be true for it to happen. If the second IFELSE was just underneath this then I would have to test the driverCode again.

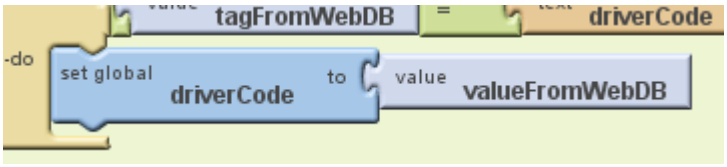
There was no need to use any iteration in my program.

I've made use of variables throughout my program. In AppInventor they need to be declared and given a first value, even if that value is going to be changed straight away:



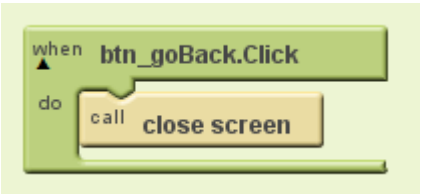
This means I can use all of these variables in this screen. Unfortunately like I said earlier there is no way I can see to make global variables have more scope than one screen so I have had to declare these variables when every screen is initialised.

This is how you assign a new value to a variable in AppInventor:



Procedures

Almost every procedure in my program is called in response to a button being pressed, such as this one which closes the screen when the 'back' button is pressed:



This means that there is no need for loops that listen for events like button pressed because this happens automatically.

All five of my screens are completely self-contained, the only thing they can do is open other screens up and close themselves. This makes the parts of the program work together very easily. The only thing I had to be careful of was making sure I used the same tags for the database throughout the program.

Working Solution

These images show the app working on my Android phone:

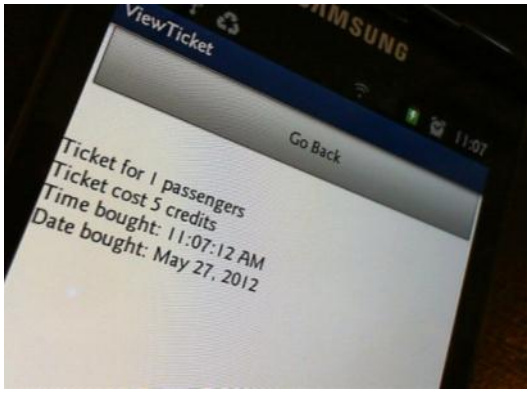
The screen shot below shows the main screen with buttons taking you to the other four sections:

This is the screen that allows the driver to enter their code and the return journey box to be ticked before you buy:

This is a similar screen but instead of a return journey the user gets to choose how many passengers:

This screen allows the user to top up credits:

The final screen shows the ticket information:



Testing and Evaluation

Test Plan

Test Number	Test Description	Input Data / User Action	Expected Result	Actual Outcome	Change Required?
General					
1.1	Application opens correctly	Open app on phone	Application opens	App opens (see image)	No
Main Screen					
2.1	Buy One Ticket Button	Press the Buy One Ticket button	The Buy One screen opens	Screen opens (see image)	No
2.2	Buy Group Ticket Button	Press the Buy Group Ticket button	The Buy Group screen opens	Screen opens (see image)	No
2.3	Top-up Credit Button	Press the Top-up Credit button	The Topup credit screen opens	Screen opens (see image)	No
2.4	View Ticket Button	Press the View Ticket button	The ViewTicket screen opens	Screen opens (see image)	No
Buy One Screen					
3.1	Go Back button	Press Go Back button	Return to main screen	Works as expected	No
3.2	Buy ticket with incorrect driver code	Driver Code: 1212 and press Buy Ticket (make sure that total credits is greater than 3)	Warning message is displayed saying incorrect code	No error message is displayed	Display error message
3.3	Buying ticket with not enough credits	Driver Code: 1234, make sure credits is less than 3 and press Buy Ticket	Taken to top up screen without buying a ticket	Taken to top up screen (see image)	No
3.4	Buying a single ticket	Driver Code: 1234, make sure the return check box isn't checked and press Buy Ticket	Check that the display ticket shows the correct time and date and journey type and that the credits has been reduced by 3	The ticket is purchased and the ticket is displayed but it has taken away 5 credits instead of 3	Adjust program to make sure it takes away 3 credits for a single and not 5

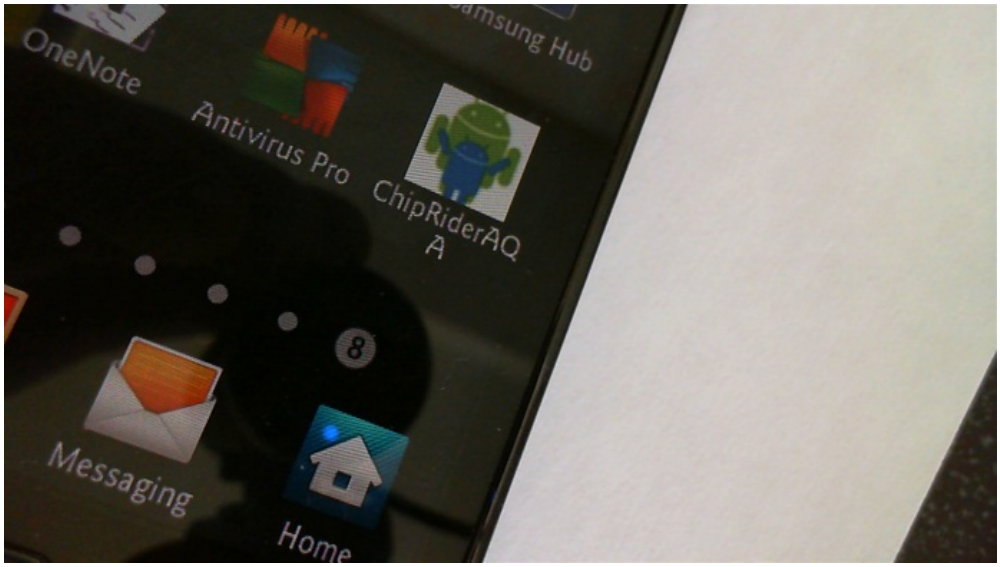
3.5	Buying a return ticket	Driver Code: 1234, make sure the return check box is checked and press Buy Ticket	Check that the display ticket shows the correct time and date and journey type and that the credits has been reduced by 5	The ticket is purchased and the ticket is displayed but it has taken away 3 credits instead of 5 – this is probably linked to problem in test 3.4	Adjust program to make sure it takes away 5 credits for a return and not 3
Buy Group Tickets					
4.1	Go Back button	Press Go Back button	Return to main screen	Works as expected	No
4.2	Buy ticket with incorrect driver code	Driver Code: 1212, number of passengers: 2 and press Buy Ticket (make sure that total credits is greater than 3)	Warning message is displayed saying incorrect code	No error message is displayed	Display error message
4.3	Buying ticket with not enough credits	Driver Code: 1234, make sure credits is less than 6, number of passengers: 2 and press Buy Ticket	Taken to top up screen without buying a ticket	Application crashes with error message "Bad arguments to x"	Find out why the crash occurred and fix it
4.4	Buying a ticket for 2 passengers	Driver Code: 1234, number of passengers: 2, make sure credits available is at least 6 and press Buy Tickets	Check that the display ticket shows the correct time and date and journey type and the number of passengers is 2 and that the credits has been reduced by 6	The ticket is purchased but the cost is 30 credits and not 6 credits.	Fix the cost of the tickets.

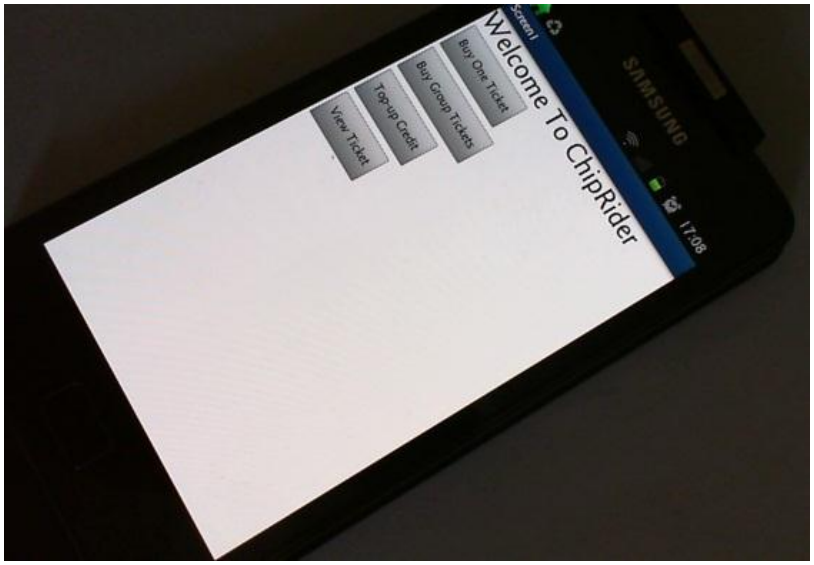
4.5	Buying a ticket for 3 passengers	Driver Code: 1234, number of passengers: 3, make sure credits available is at least 9 and press Buy Tickets	Check that the display ticket shows the correct time and date and journey type and the number of passengers is 3 and that the credits has been reduced by 9	Works as expected (correct amount deducted (9 to 0) and ticket displays correctly)	No
Top-up Screen					
5.1	Go Back button	Press Go Back button	Return to main screen	Works as expected	No
5.2	Check the display credit works	Inspect the watcher in the code and record the credits, open this screen and compare values	The two values are the same	Both values display 10 credits	No
5.3	Buying credit with the wrong driver code	Driver Code: 1212, How many to add: 10. Check credit before and press TopUp button.	Error message displayed and the amount of credits remains unchanged	Error message displayed: "Incorrect code"	None (although change the previous screens to show this error message)
5.4	Buying credit that exceeds a total of 30	Driver Code: 1234, check credit level and enter How Many to Add as 31 minus credits and press TopUp button.	Error message displayed that it is too many credits.	Error message displayed: "Too many credits"	None (although change the previous screens to show this error message)
5.5	Buying credit that makes it up to 30	Driver Code: 1234, check credit level and enter How Many to Add as 30 minus credits and press TopUp button	Go back to the topup screen and check that current credits is 30	Current credit was 10, topped up 20 credits and the result was 30.	No

View Ticket Screen					
6.1	Go Back button	Press Go Back button	Return to main screen	Works as expected	No
6.2	Check display works (single)	Buy a single ticket for one passenger and record the time and date then open the view ticket screen	The time and date are accurate and the cost is 3 and the number of passengers is 1	(evidenced in test 3.4)	No
6.3	Check display works (return)	Buy a return ticket for one passenger and record the time and date then open the view ticket screen	The time and date are accurate and the cost is 5 and the number of passengers is 1	(evidenced in test 3.5)	No
6.4	Check display works (2 passengers)	Buy a single ticket for two passengers and record the time and date then open the view ticket screen	The time and date are accurate and the cost is 6 and the number of passengers is 2	(evidenced in test 4.4)	No

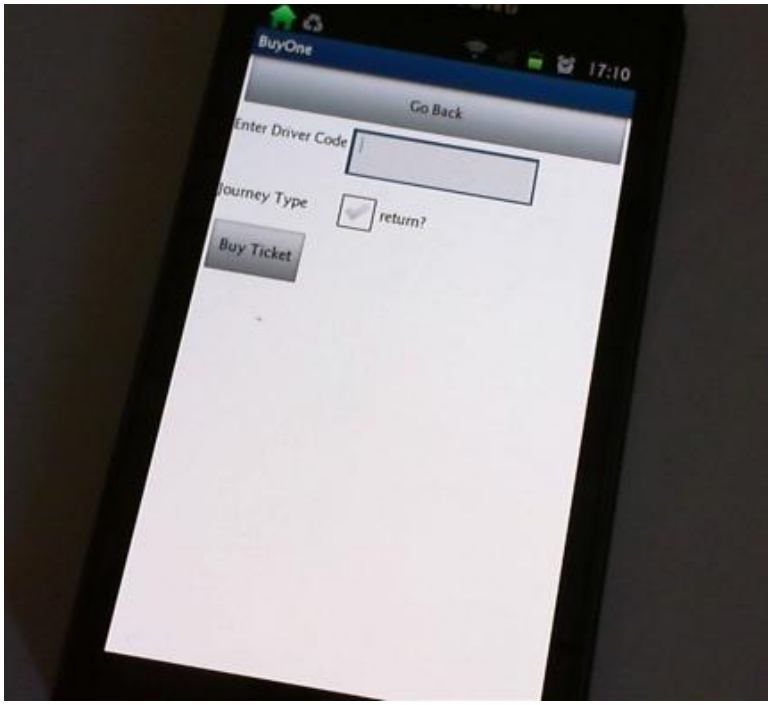
Evidence

1.1


OneNote
Antivirus Pro
ChipRiderAQ
A
Messaging
Home


Welcome To ChipRider
Buy One Ticket
Buy Group Tickets
Top-up Credit
View Ticket
SAMSUNG
17:08

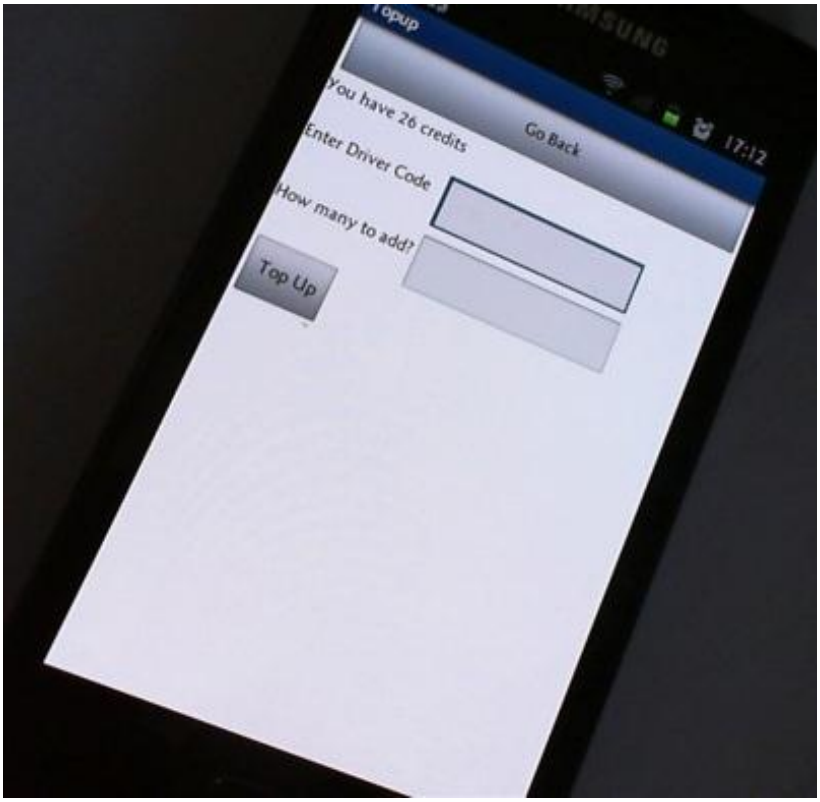
2.1


BuyOne
17:10
Go Back
Enter Driver Code
Journey Type
return?
Buy Ticket

2.2


BuyGroup
17:11
Go Back
Enter Driver Code
Number Of Passengers
Buy Tickets

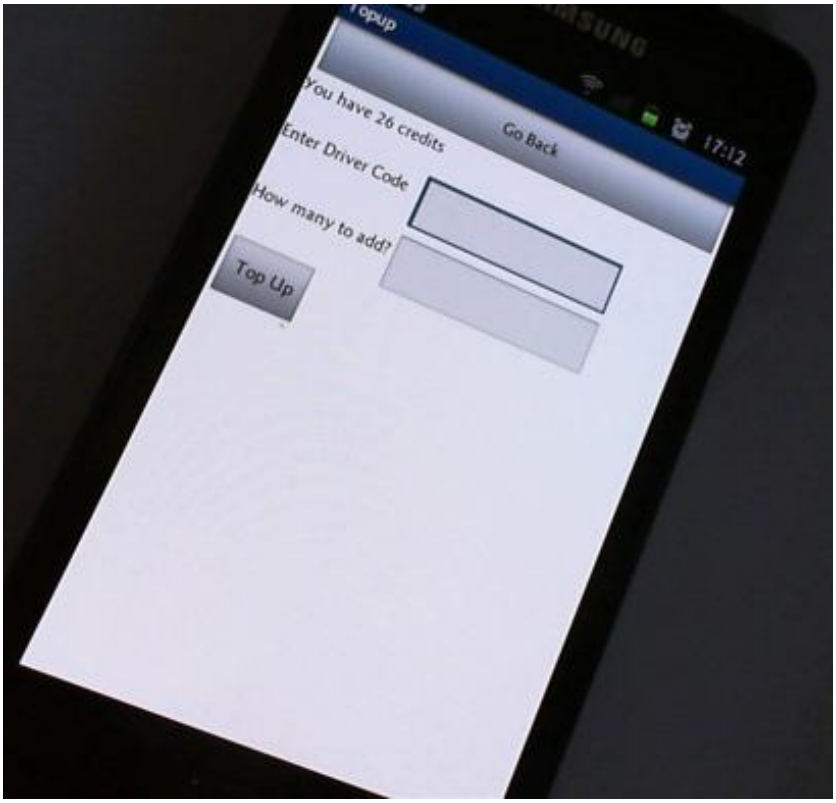
2.3


Topup
Go Back
17:12
You have 26 credits
Enter Driver Code
How many to add?
Top Up

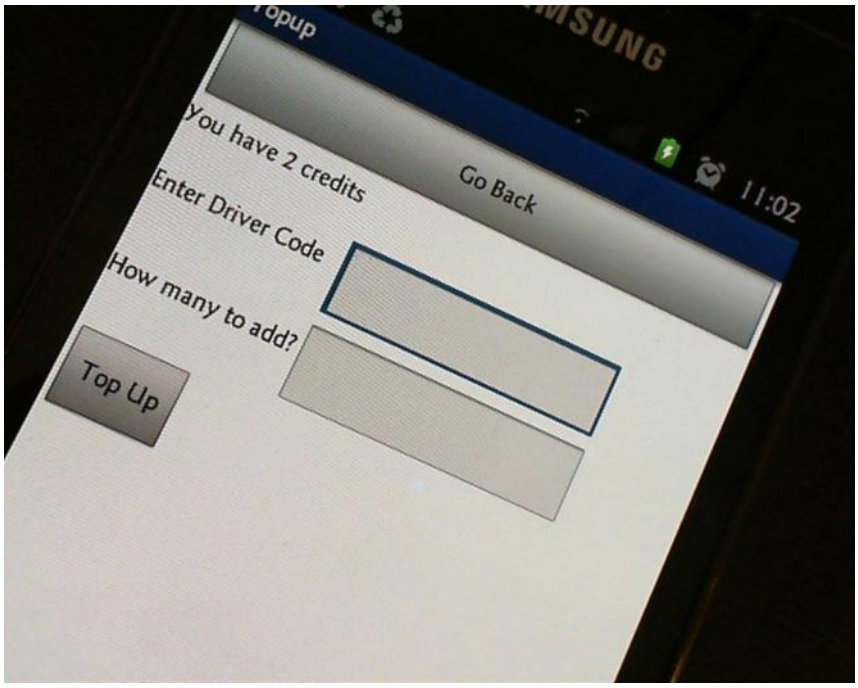
2.4

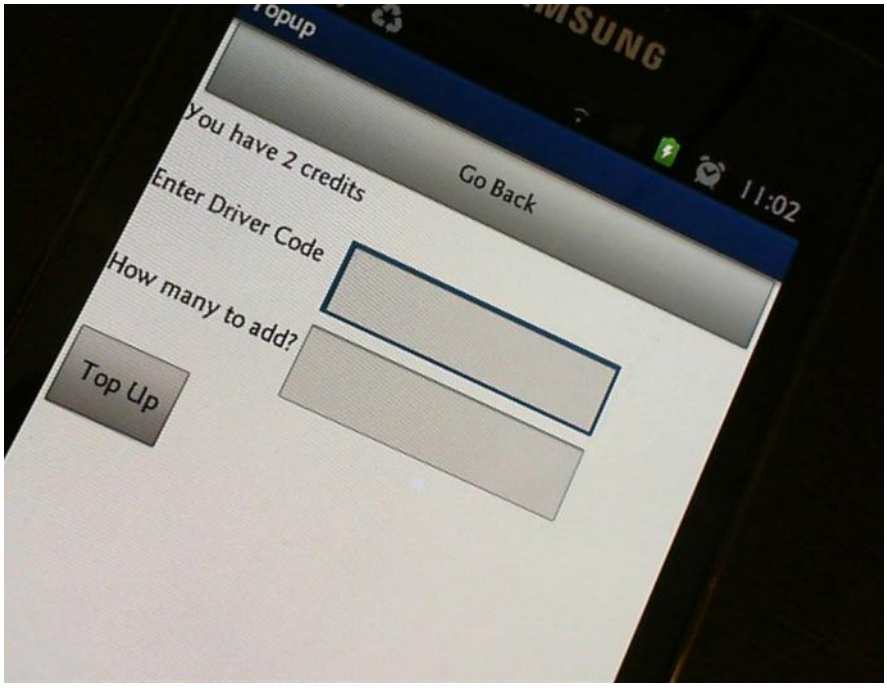

Topup
Go Back
17:12
You have 26 credits
Enter Driver Code
How many to add?
Top Up

3.2

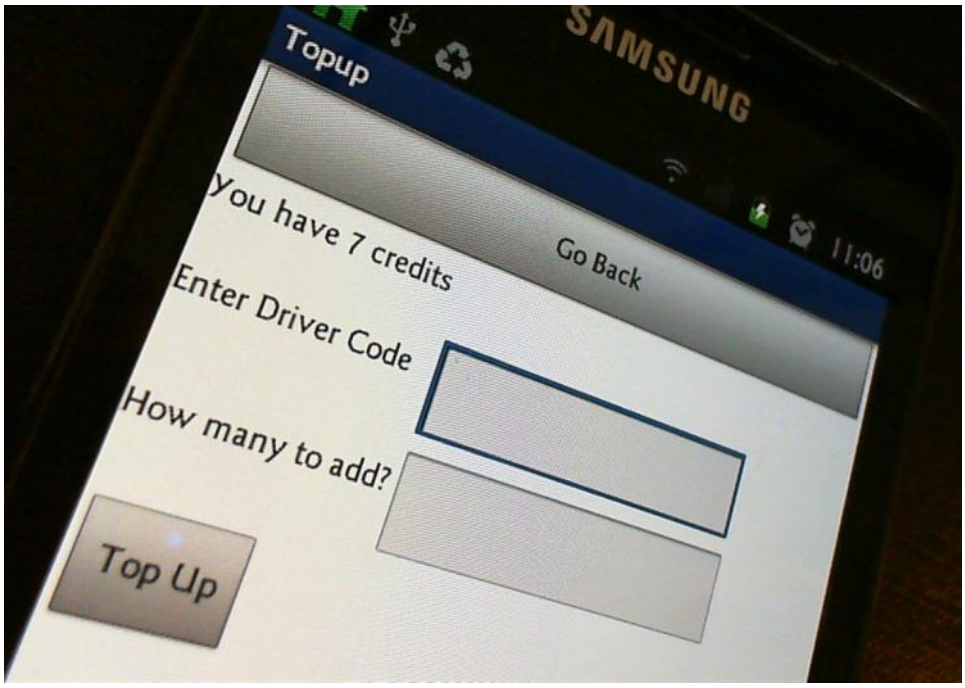

Popup
Go Back
17:12
You have 26 credits
Enter Driver Code
How many to add?
Top Up

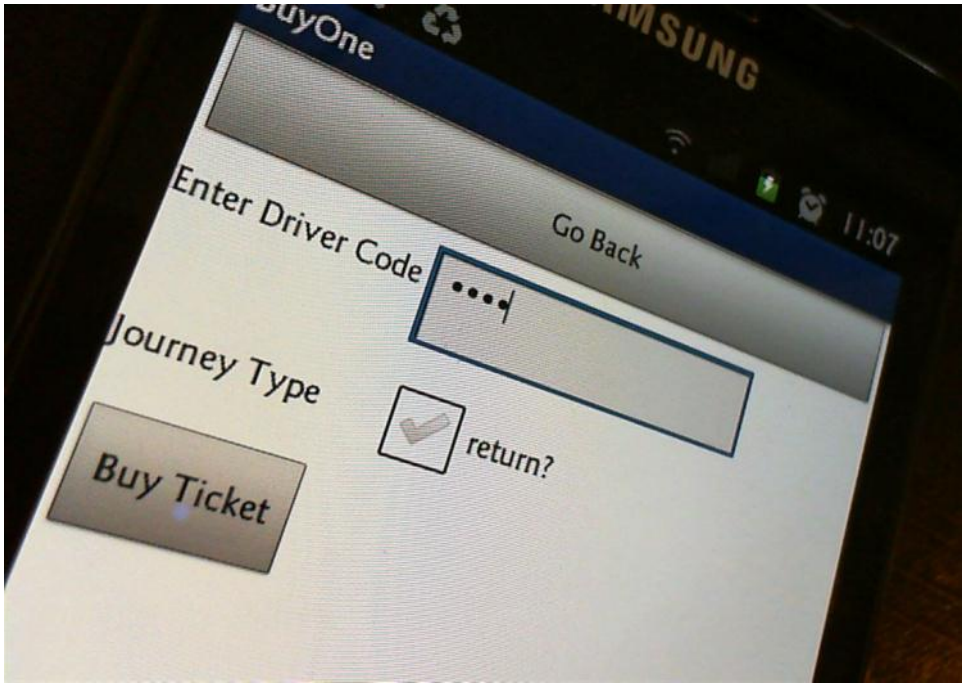
3.3

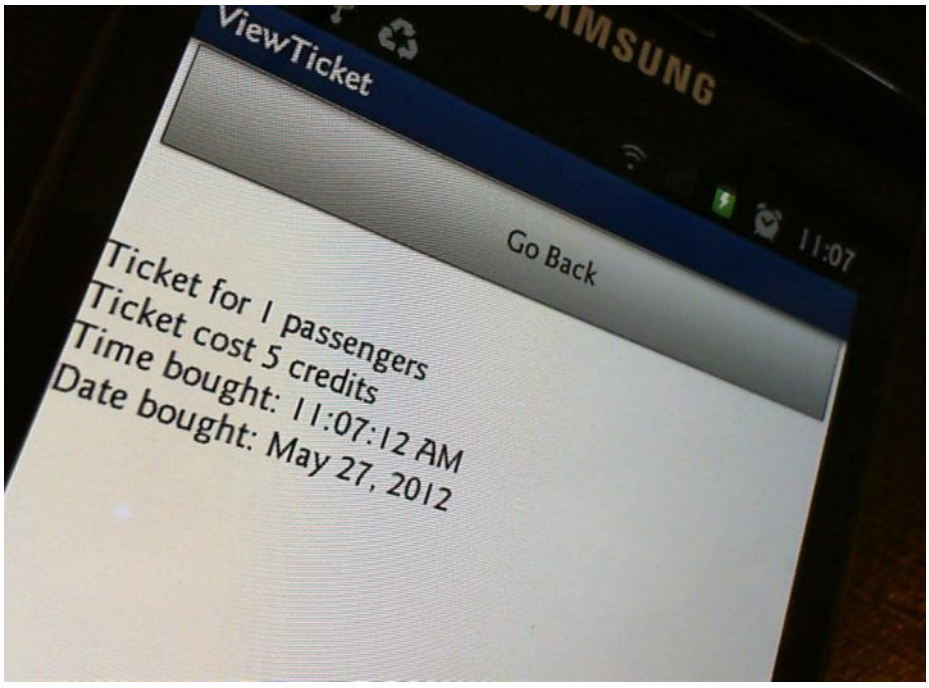

Popup
Go Back
11:02
You have 2 credits
Enter Driver Code
How many to add?
Top Up

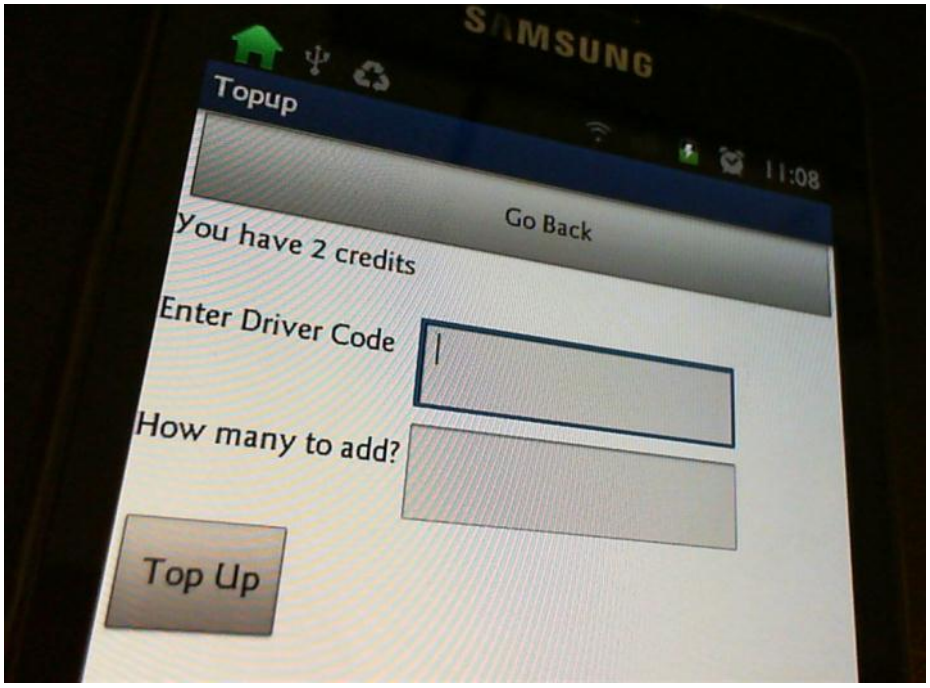

Topup
SUNG
You have 2 credits
Go Back
11:02
Enter Driver Code
How many to add?
Top Up

3.4

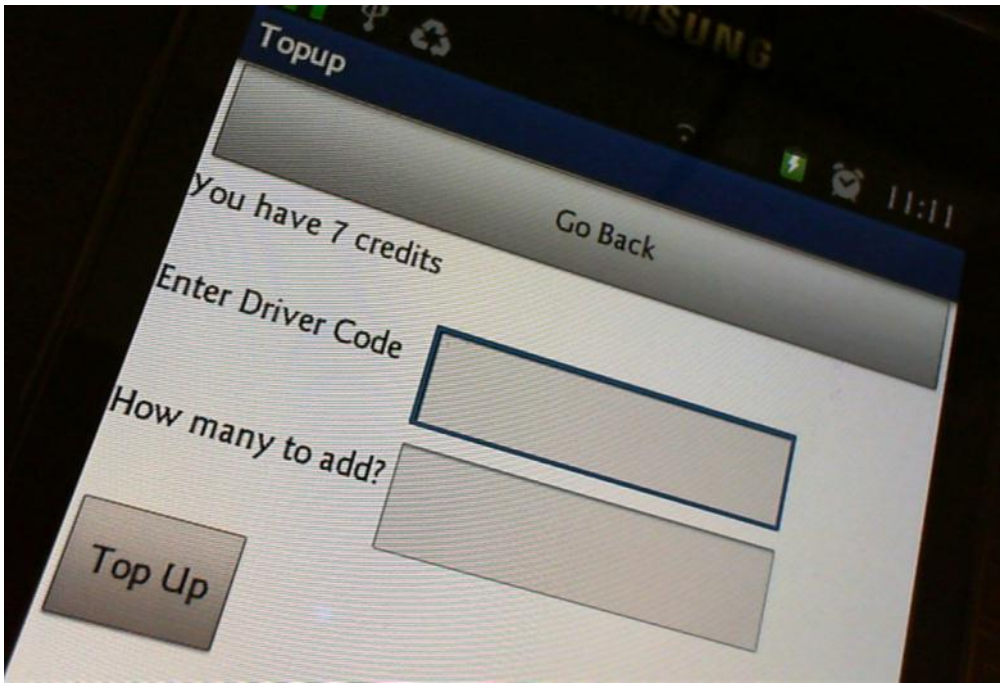

Topup
SAMSUNG
You have 7 credits
Go Back
11:06
Enter Driver Code
How many to add?
Top Up

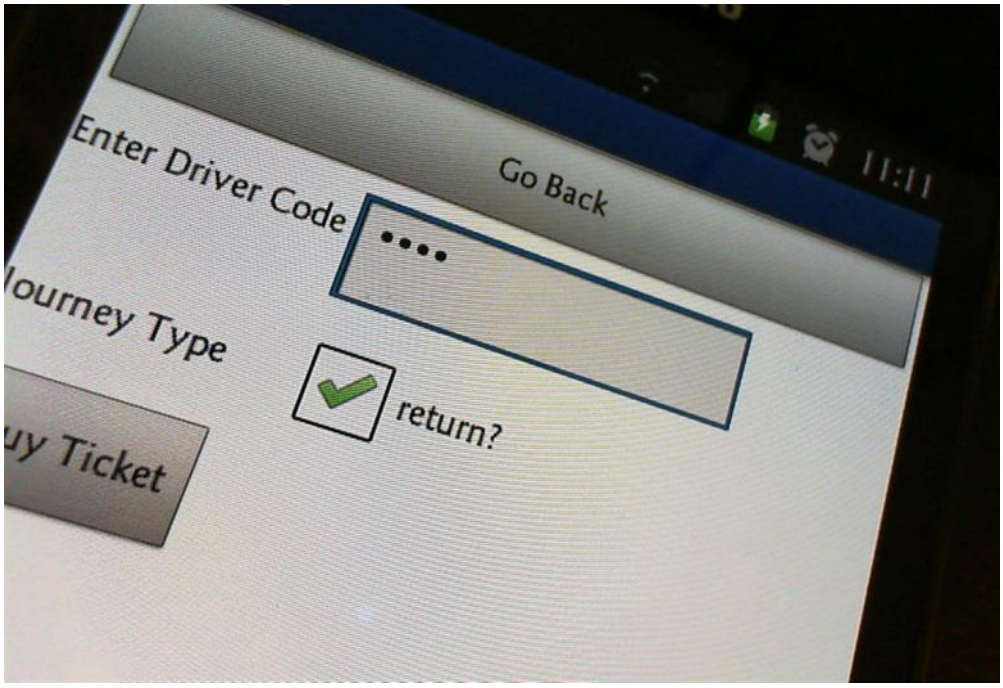

BuyOne
SAMSUNG
Go Back
11:07
Enter Driver Code
Journey Type
return?
Buy Ticket

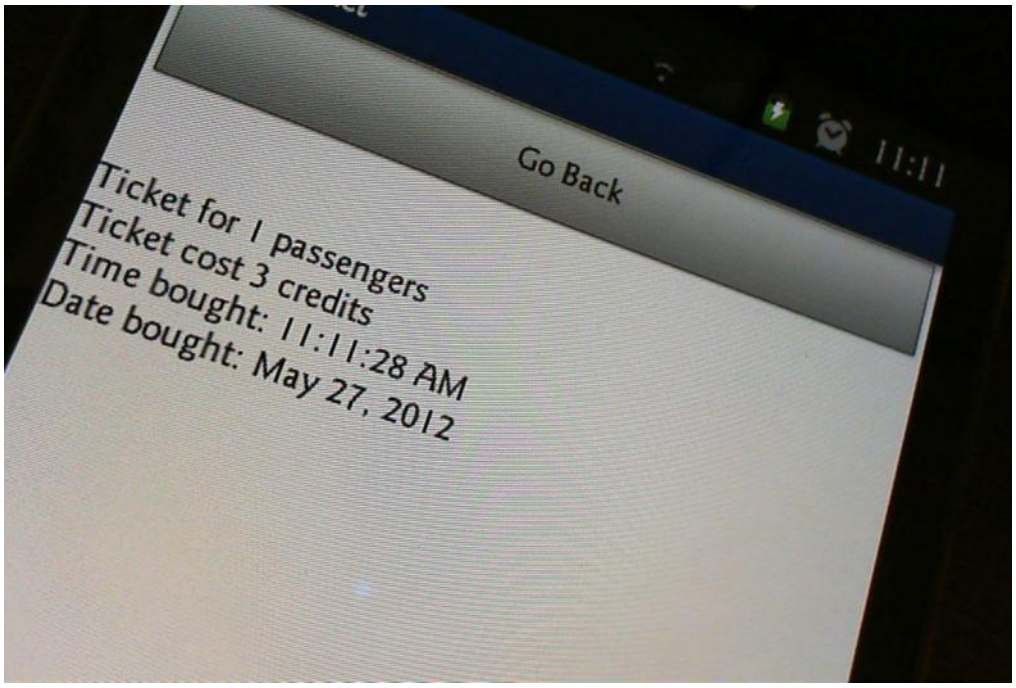

ViewTicket
SAMSUNG
Go Back
11:07
Ticket for 1 passengers
Ticket cost 5 credits
Time bought: 11:07:12 AM
Date bought: May 27, 2012

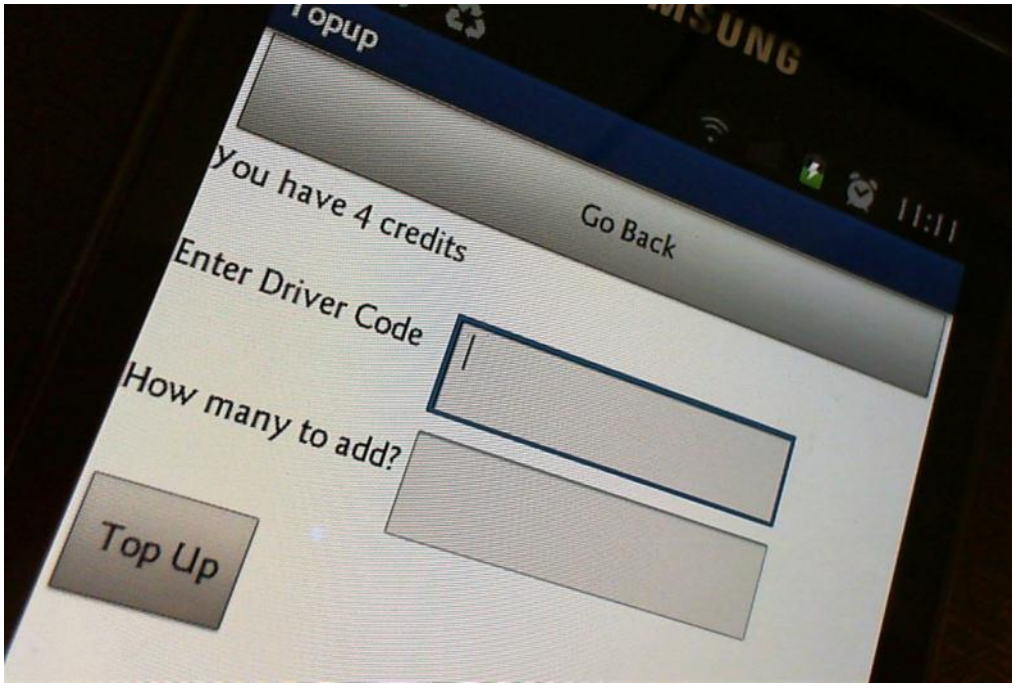

SAMSUNG
Topup
11:08
Go Back
You have 2 credits
Enter Driver Code
How many to add?
Top Up

3.5

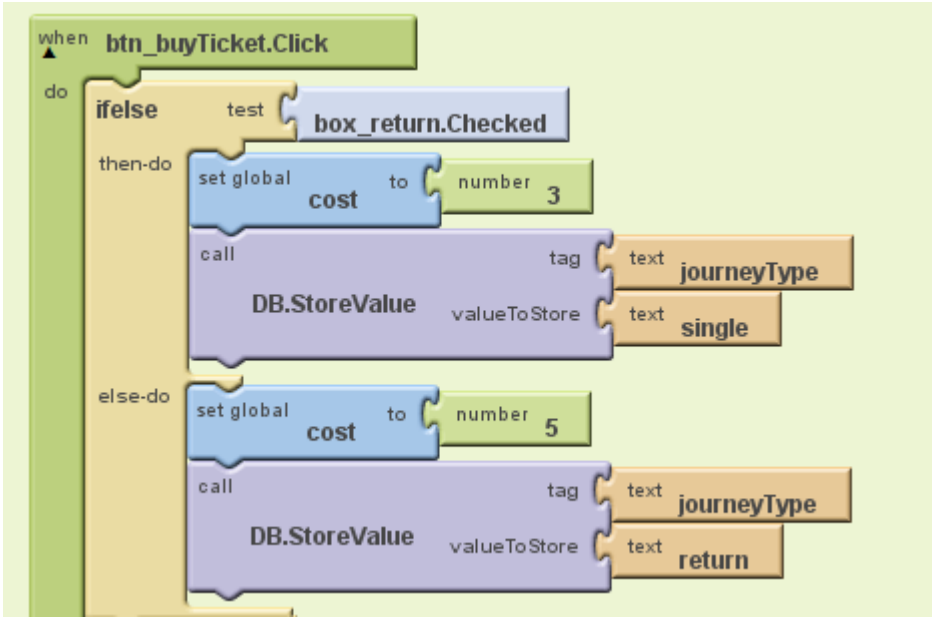

Topup
11:11
Go Back
You have 7 credits
Enter Driver Code
How many to add?
Top Up


Go Back
11:11
Enter Driver Code
••••
ourney Type
return?
uy Ticket

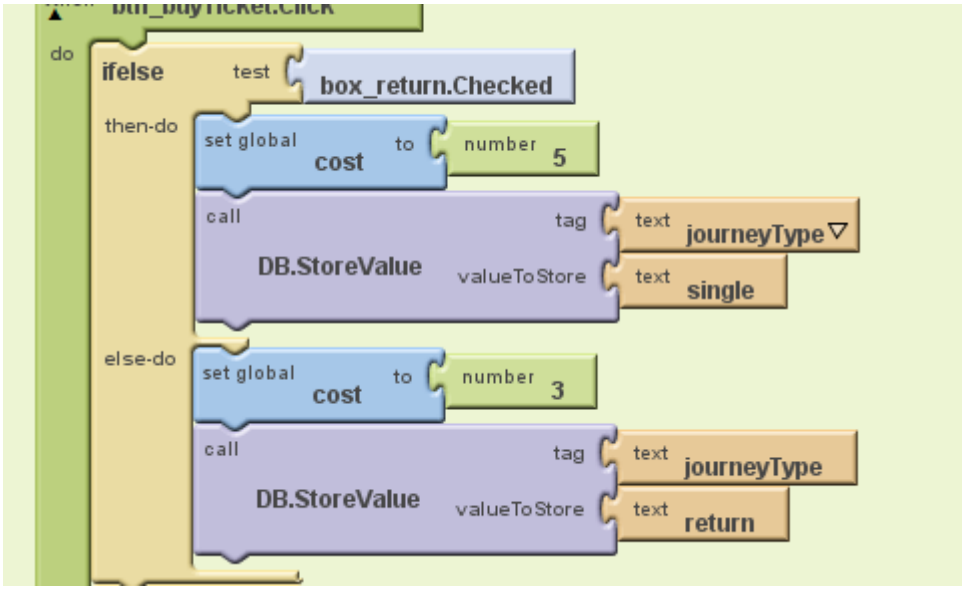

Go Back
11:11
Ticket for 1 passengers
Ticket cost 3 credits
Time bought: 11:11:28 AM
Date bought: May 27, 2012



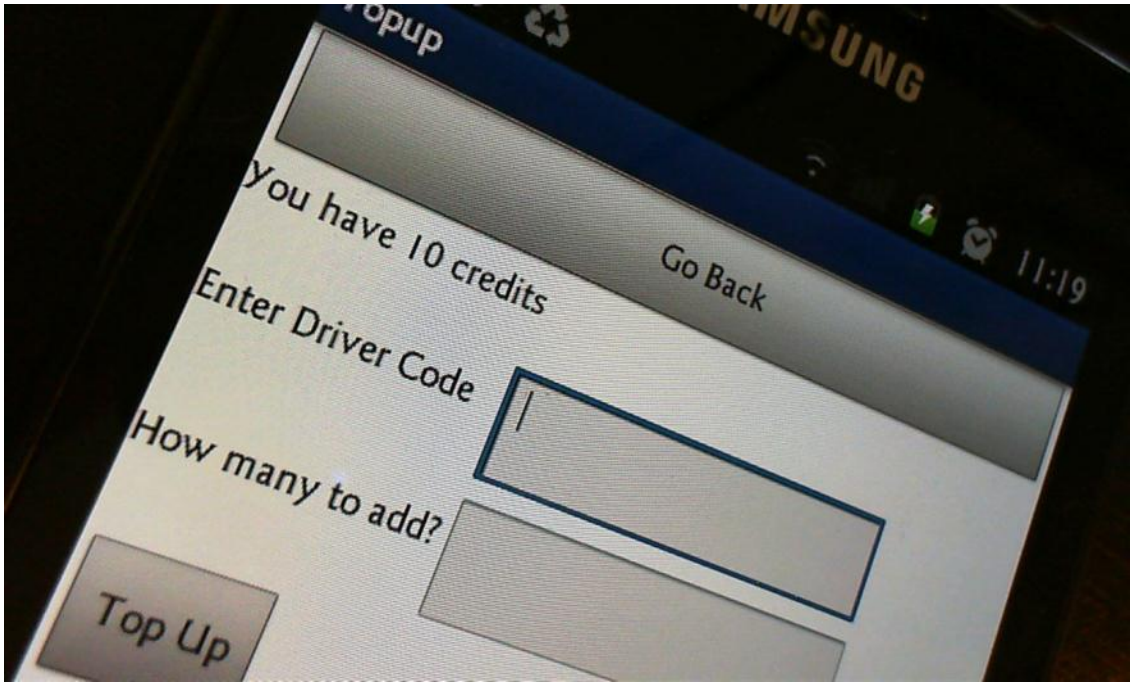
This section of code shows the cost is set to 3 and 5 in the wrong order:

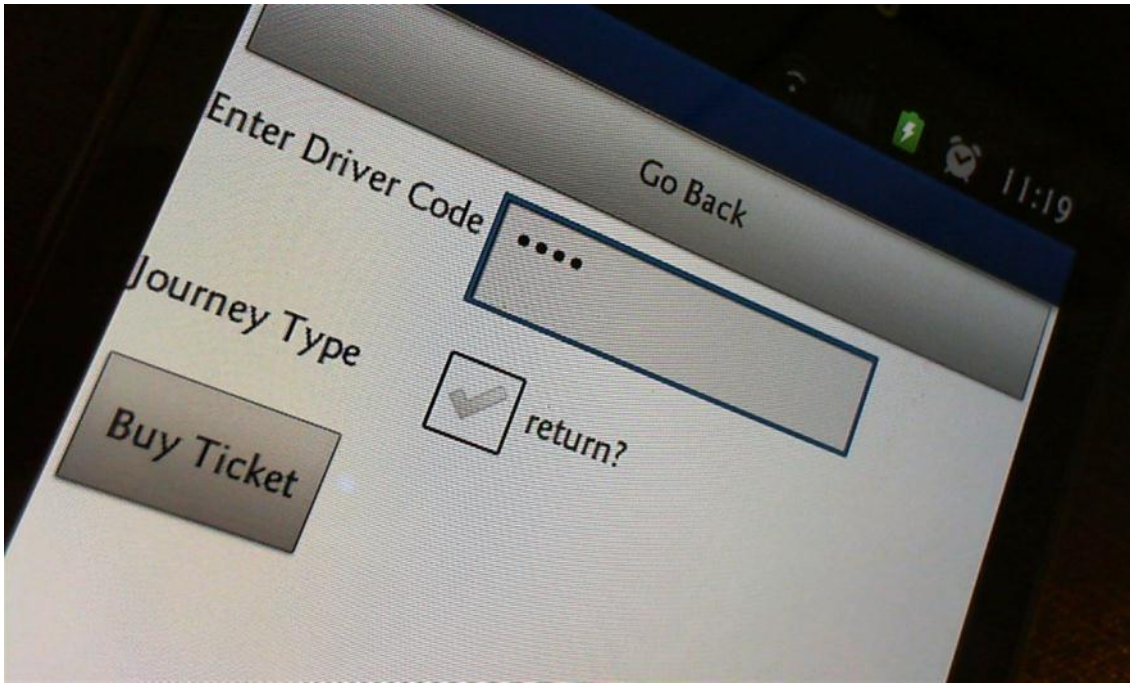


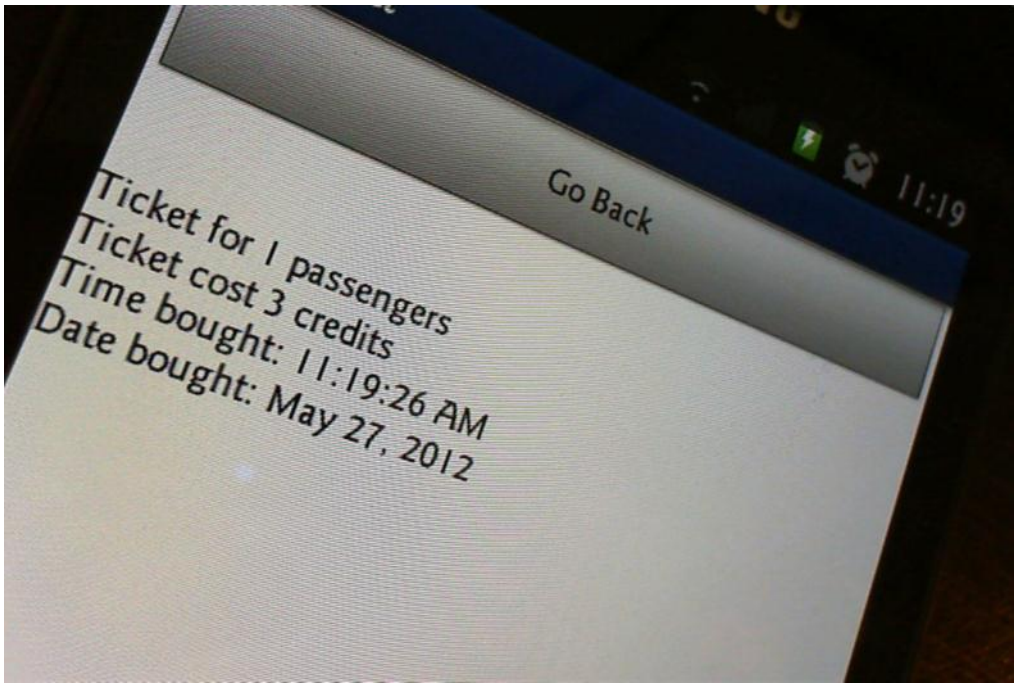
This is changed to this:

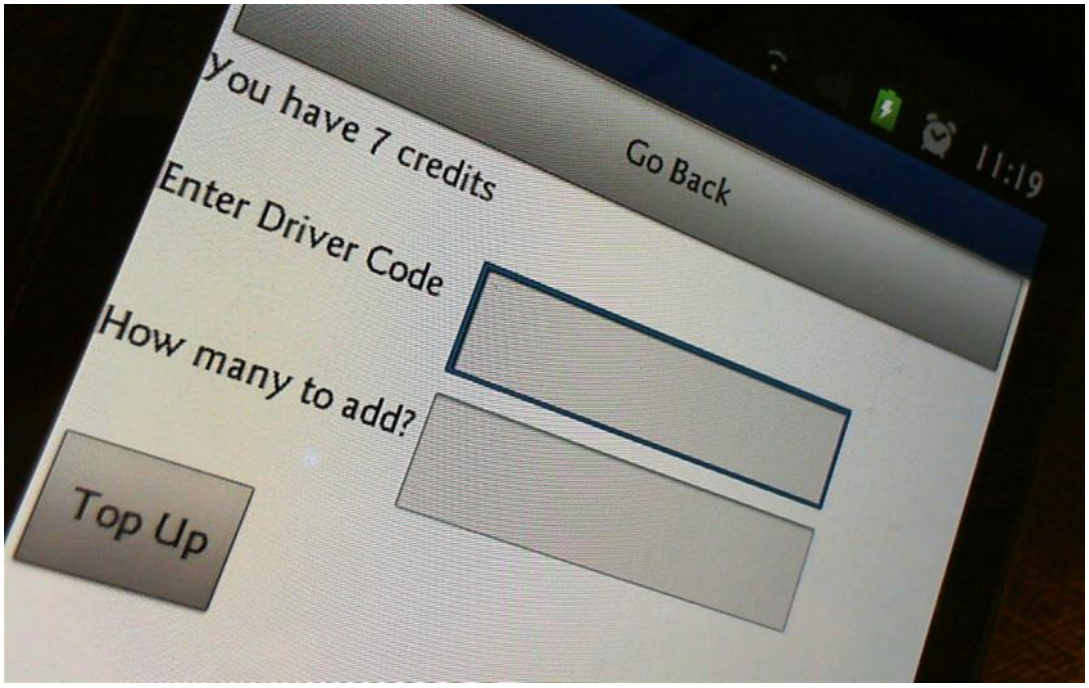


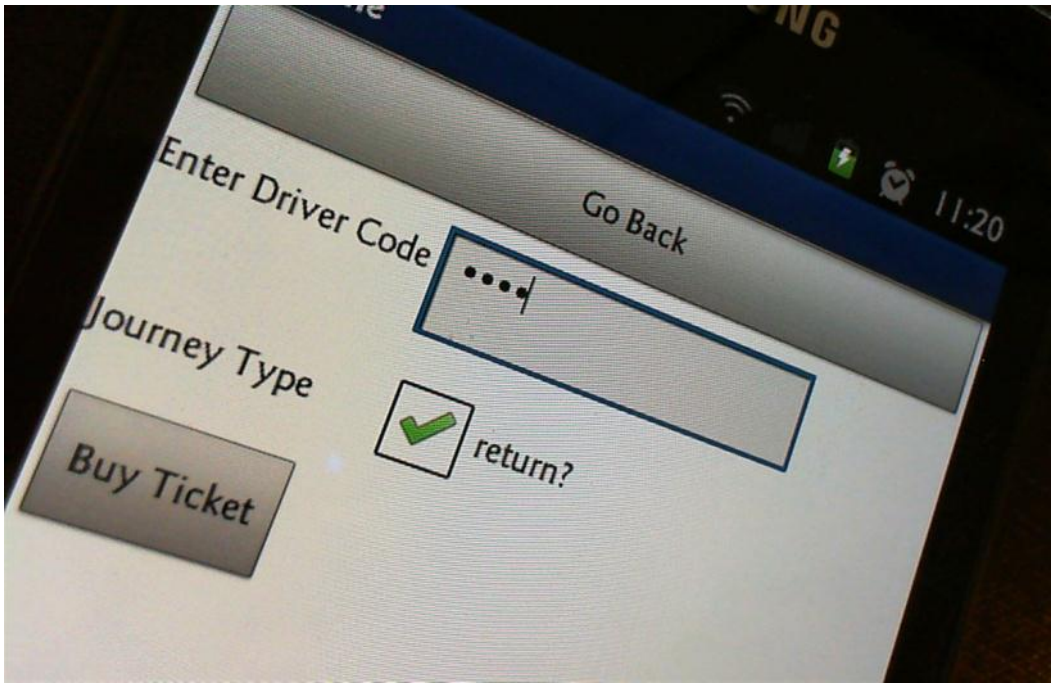
Re-testing for 3.4 and 3.5:

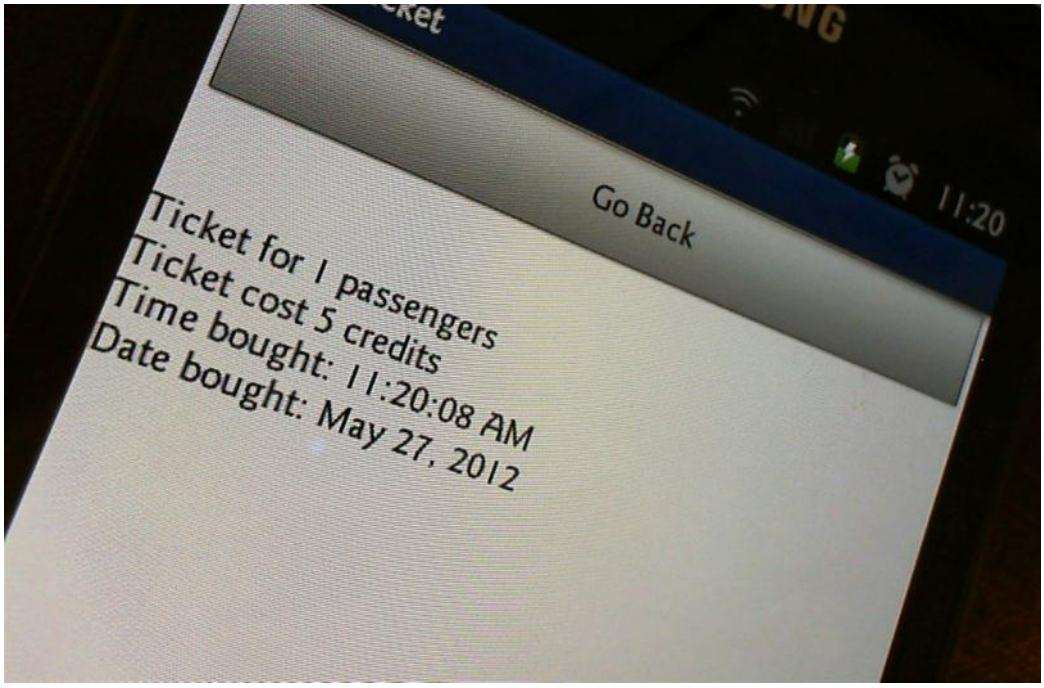


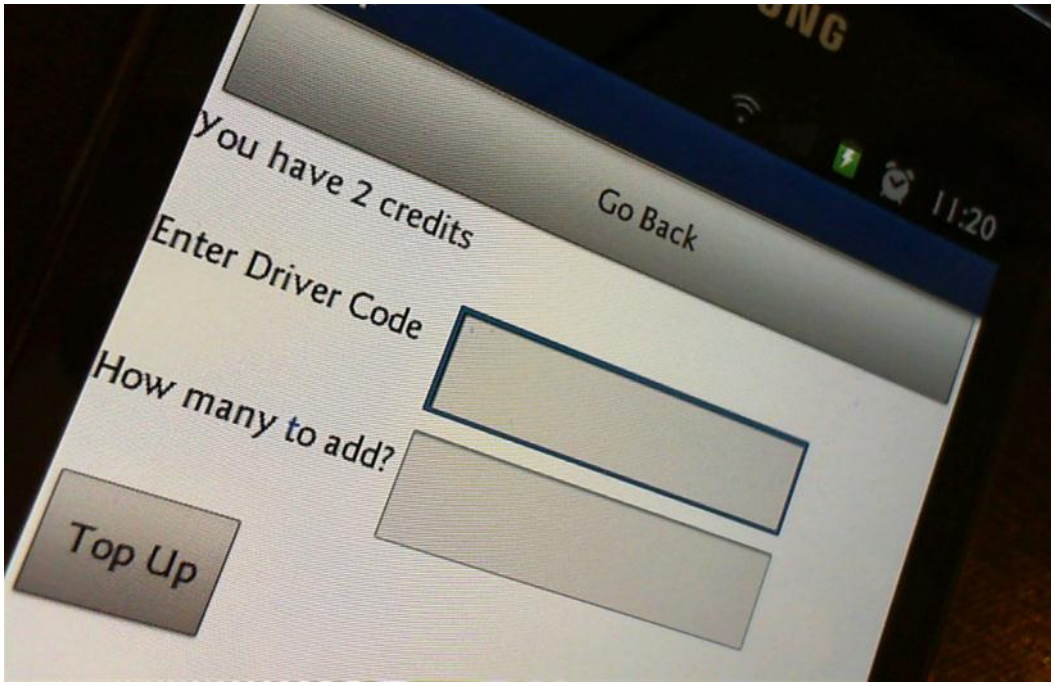

Go Back
11:19
Enter Driver Code
Journey Type
return?
Buy Ticket


Go Back
11:19
Ticket for 1 passengers
Ticket cost 3 credits
Time bought: 11:19:26 AM
Date bought: May 27, 2012

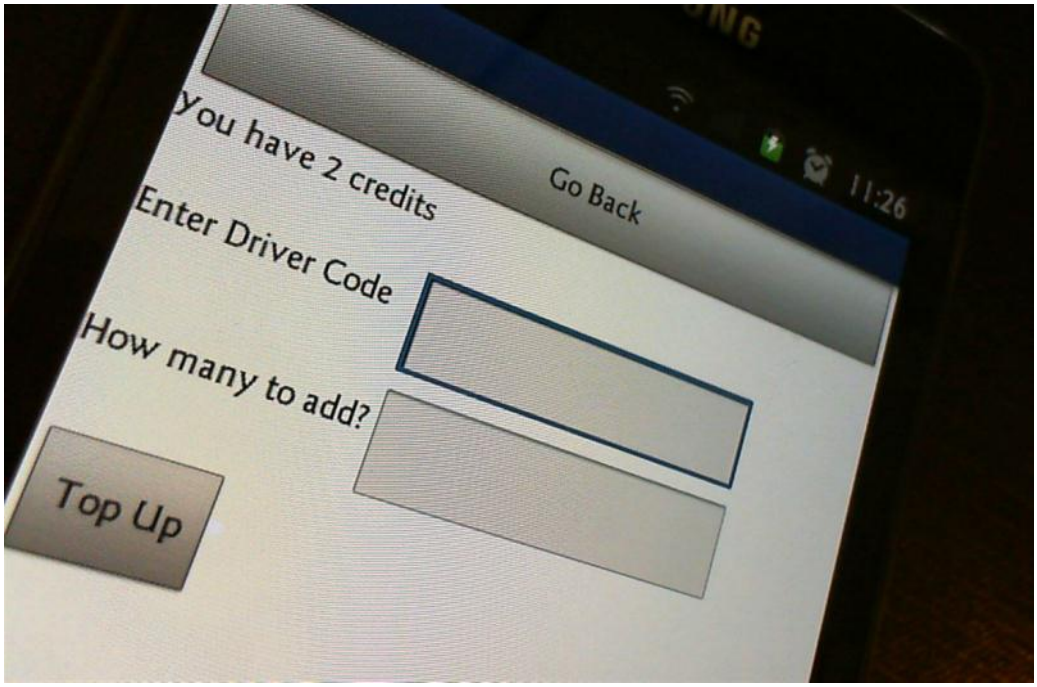

You have 7 credits
Go Back
11:19
Enter Driver Code
How many to add?
Top Up

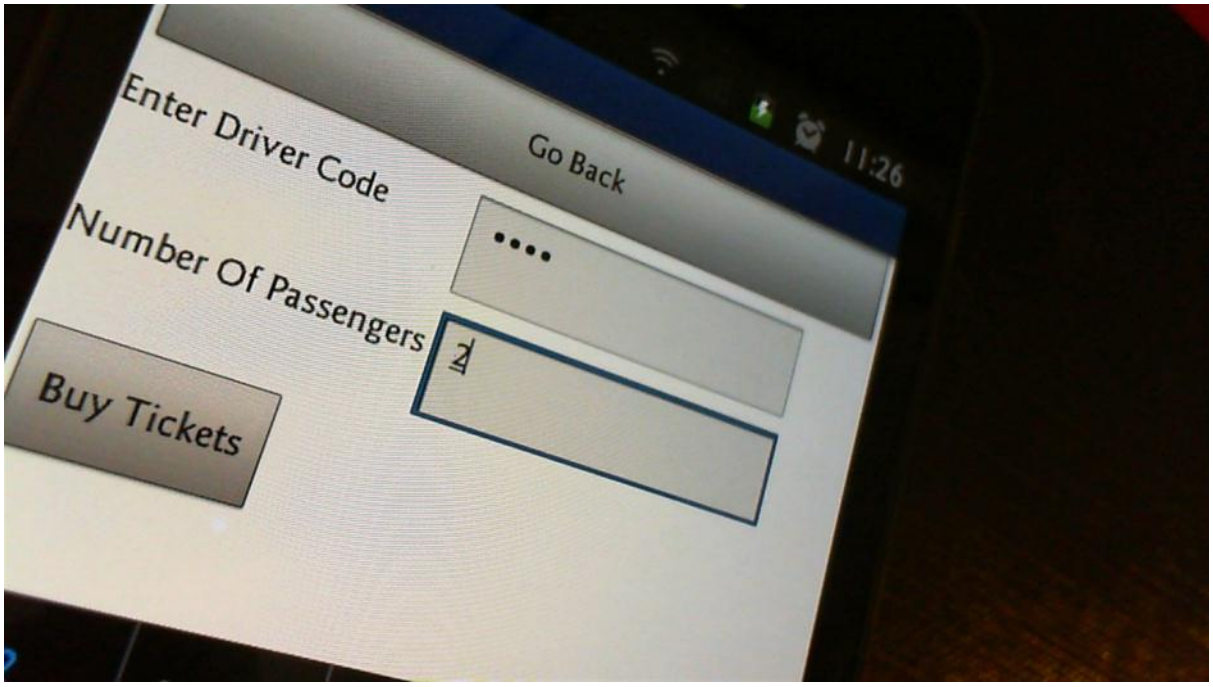

Go Back
11:20
Enter Driver Code
Journey Type
return?
Buy Ticket

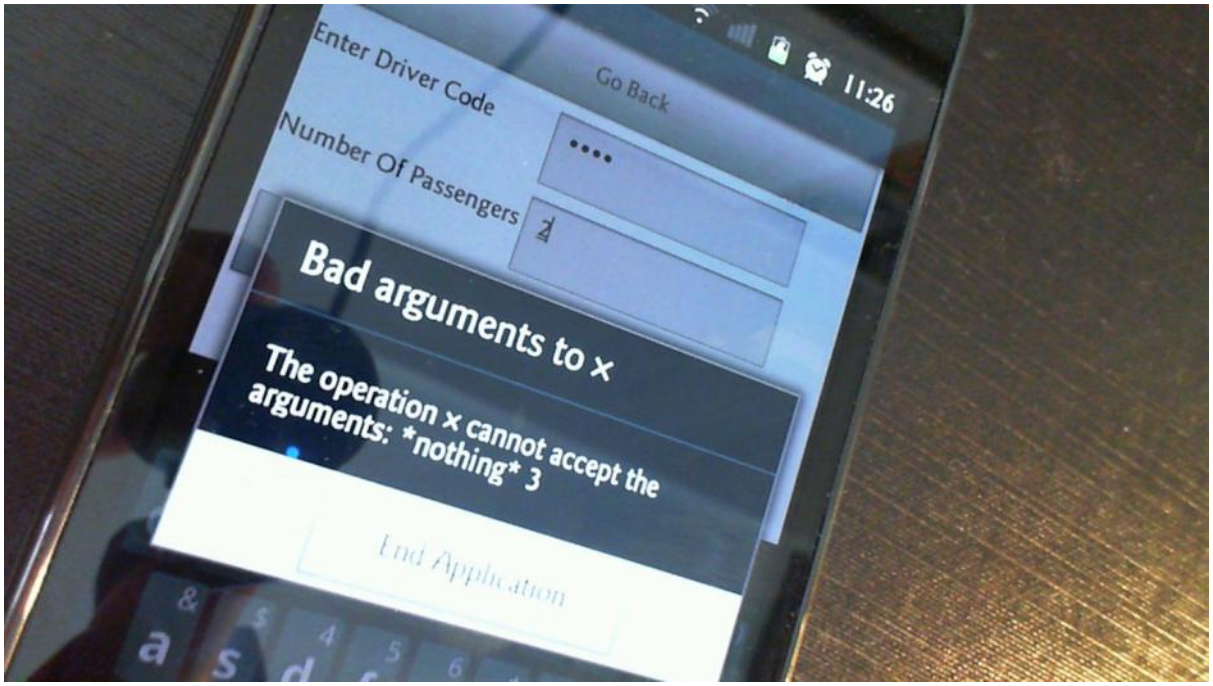

Go Back
11:20
Ticket for 1 passengers
Ticket cost 5 credits
Time bought: 11:20:08 AM
Date bought: May 27, 2012


Go Back
11:20
You have 2 credits
Enter Driver Code
How many to add?
Top Up

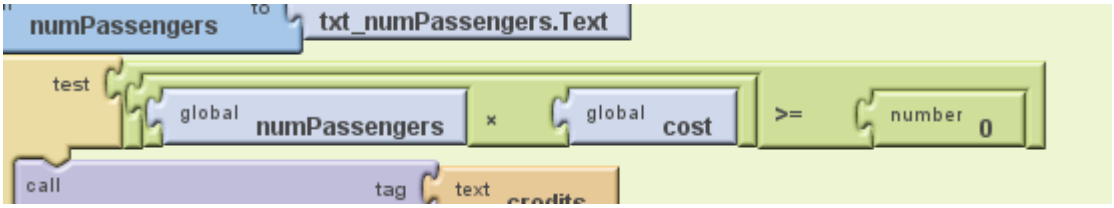
4.1

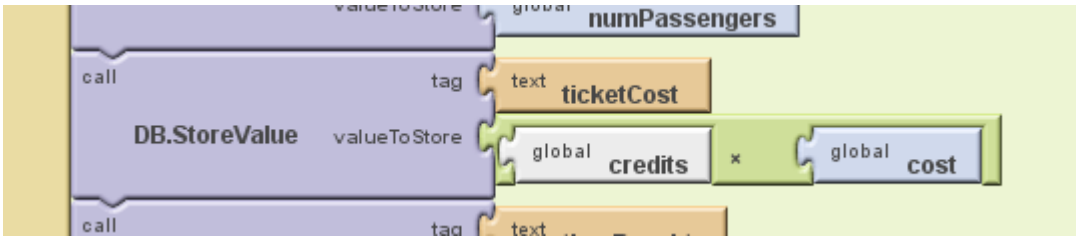

Go Back
11:26
You have 2 credits
Enter Driver Code
How many to add?
Top Up


Go Back
11:26
Enter Driver Code
Number Of Passengers
2
Buy Tickets

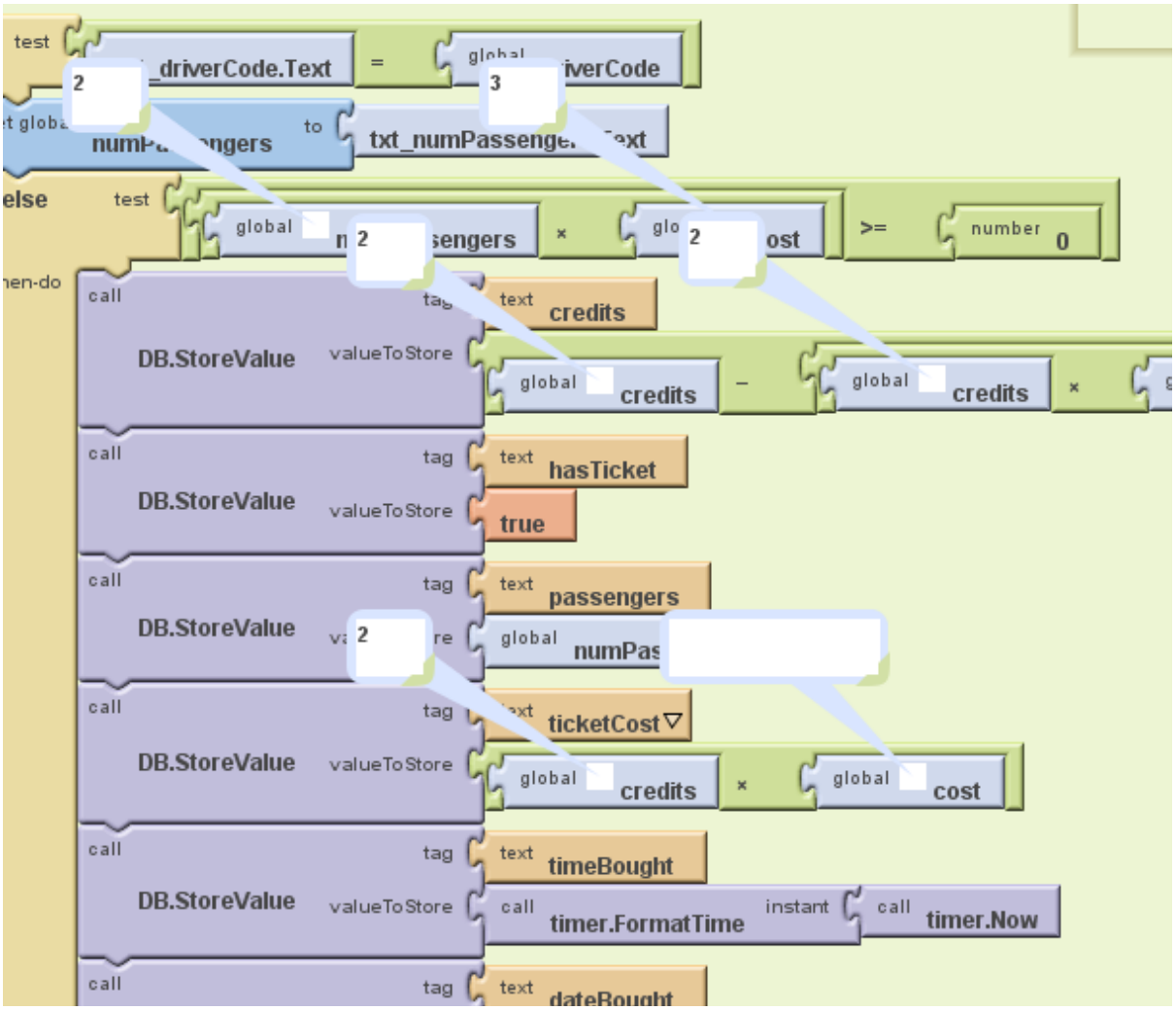


There are two places that could be causing this problem:

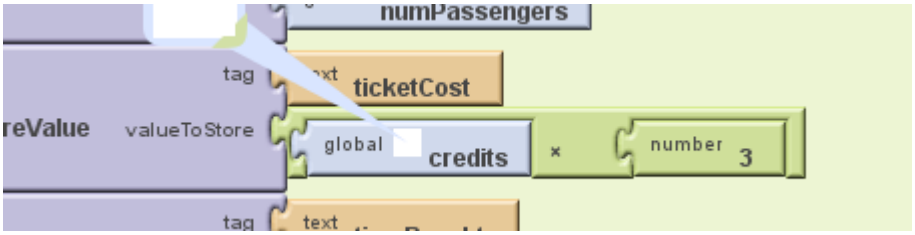




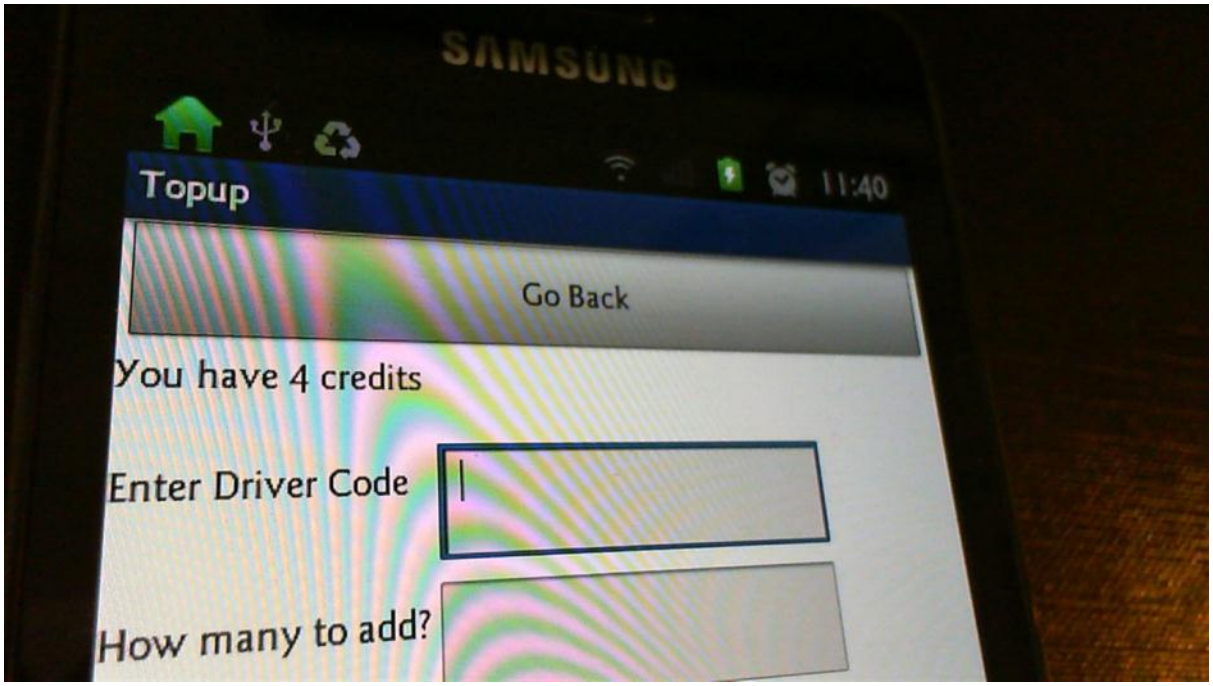
This is a shot with the variable watcher switched on and I can see that problem is the global cost because it isn't set:

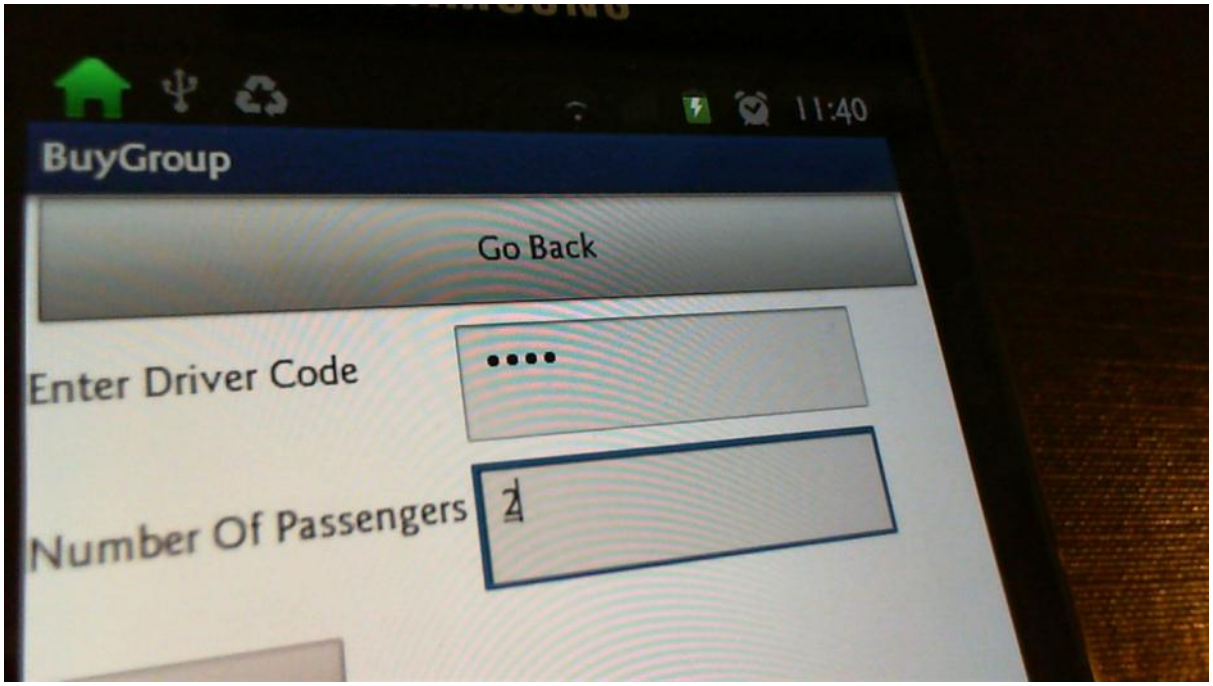


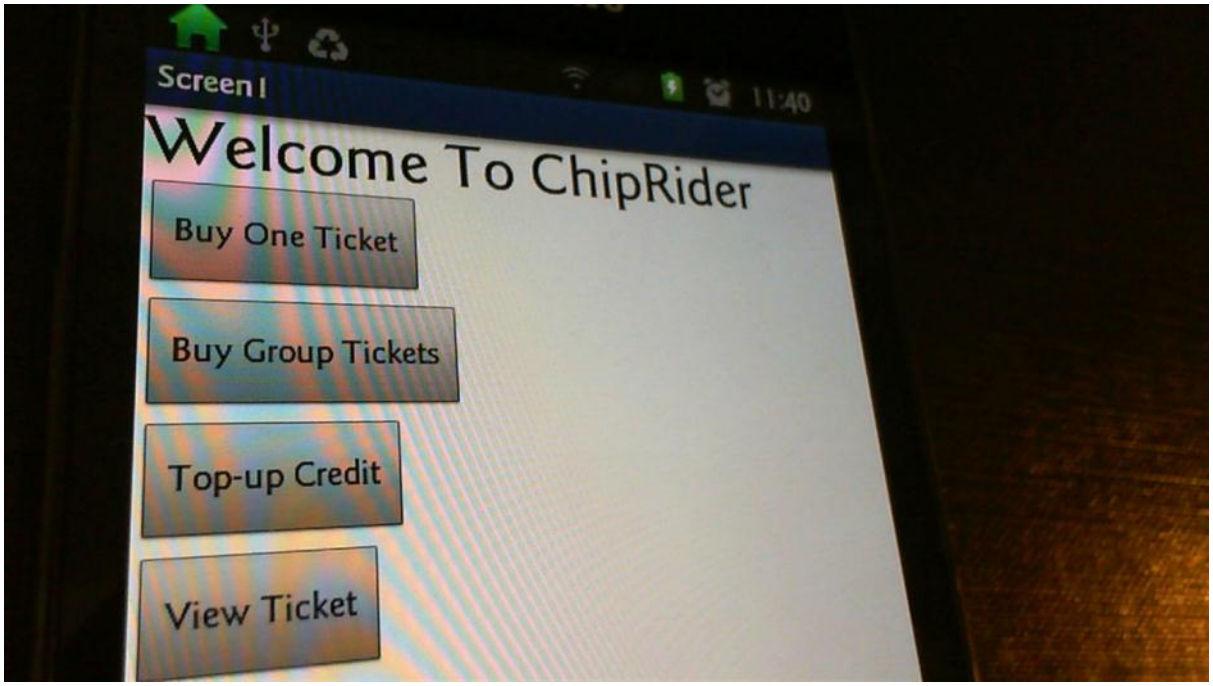
I don't know why this is the case because the other watcher on the global variable says it is 3, but I have put the value 3 in to make sure it doesn't happen:



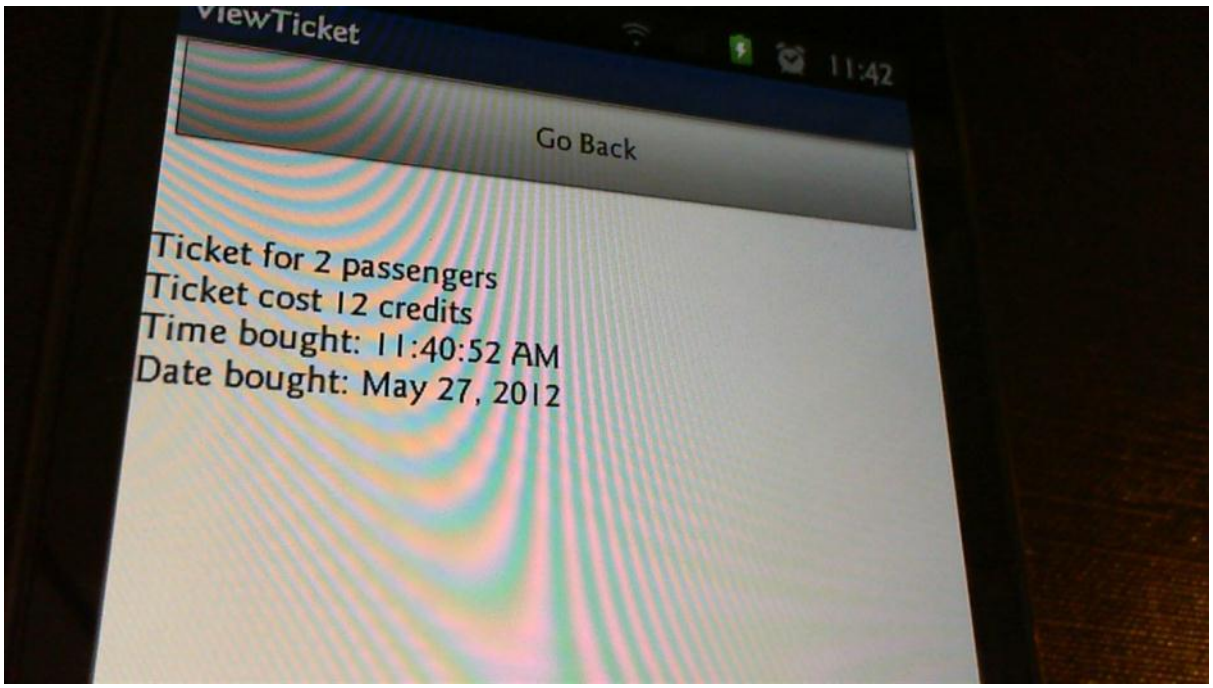
Re-testing for 4.1

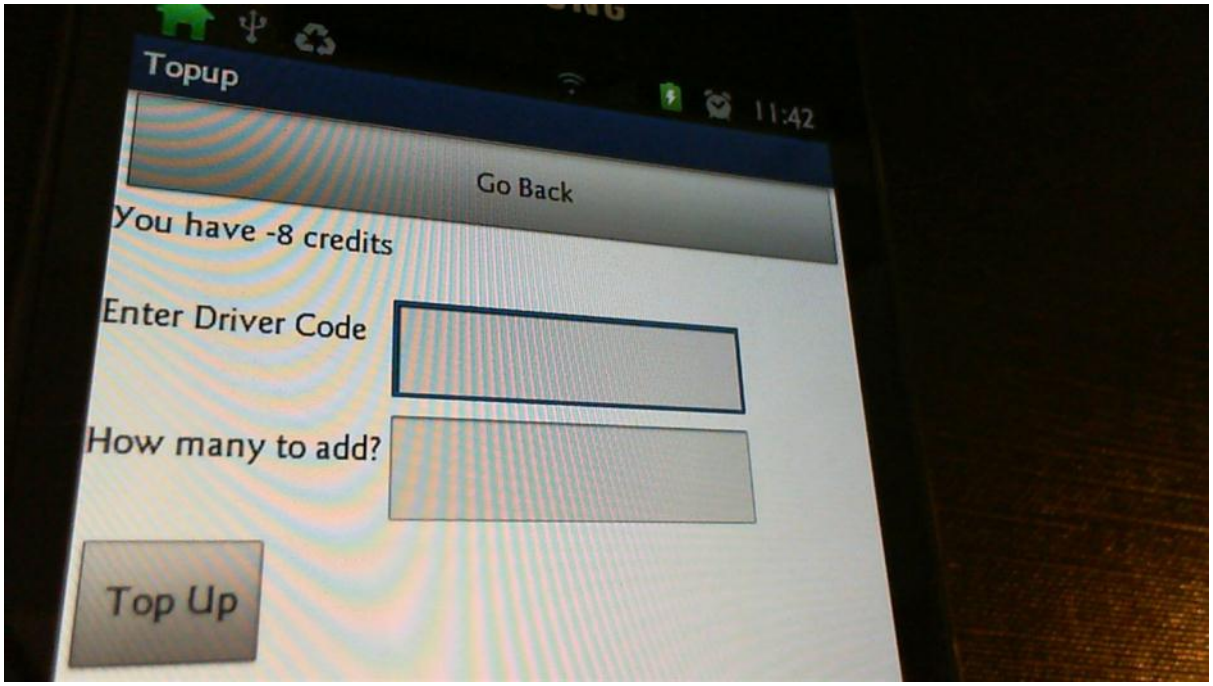




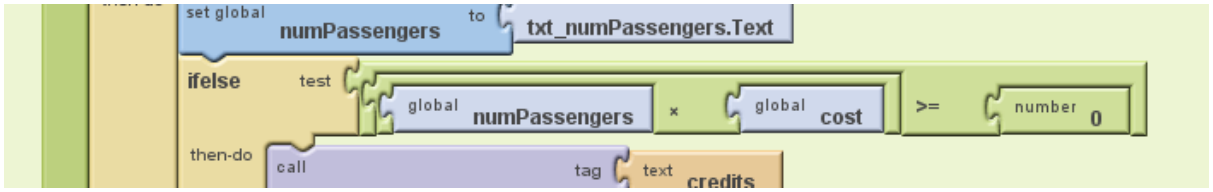


It doesn't crash but it now it doesn't go to the right screen and the credit has been taken anyway:

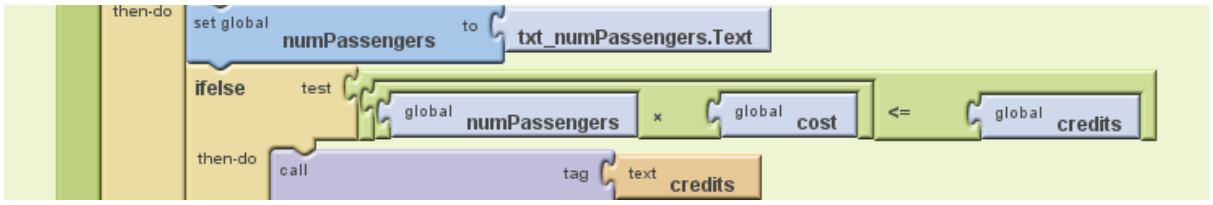




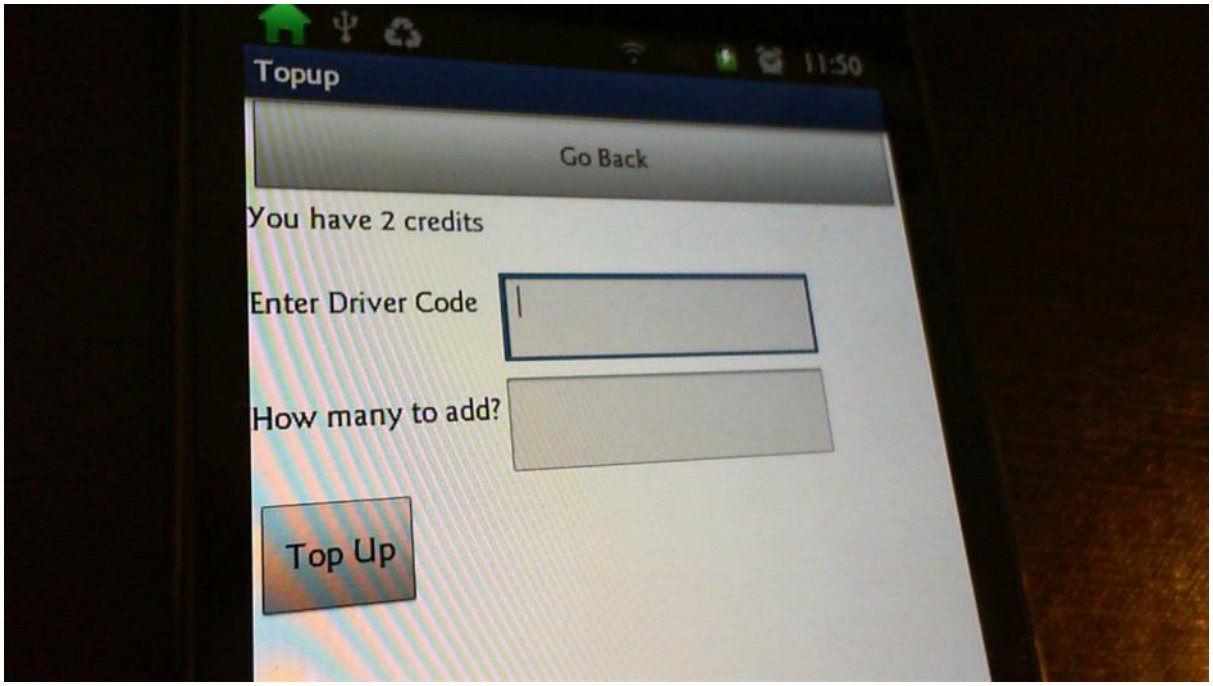
The two last images show that the credits have been taken off wrongly. I had a look at the code and saw this:

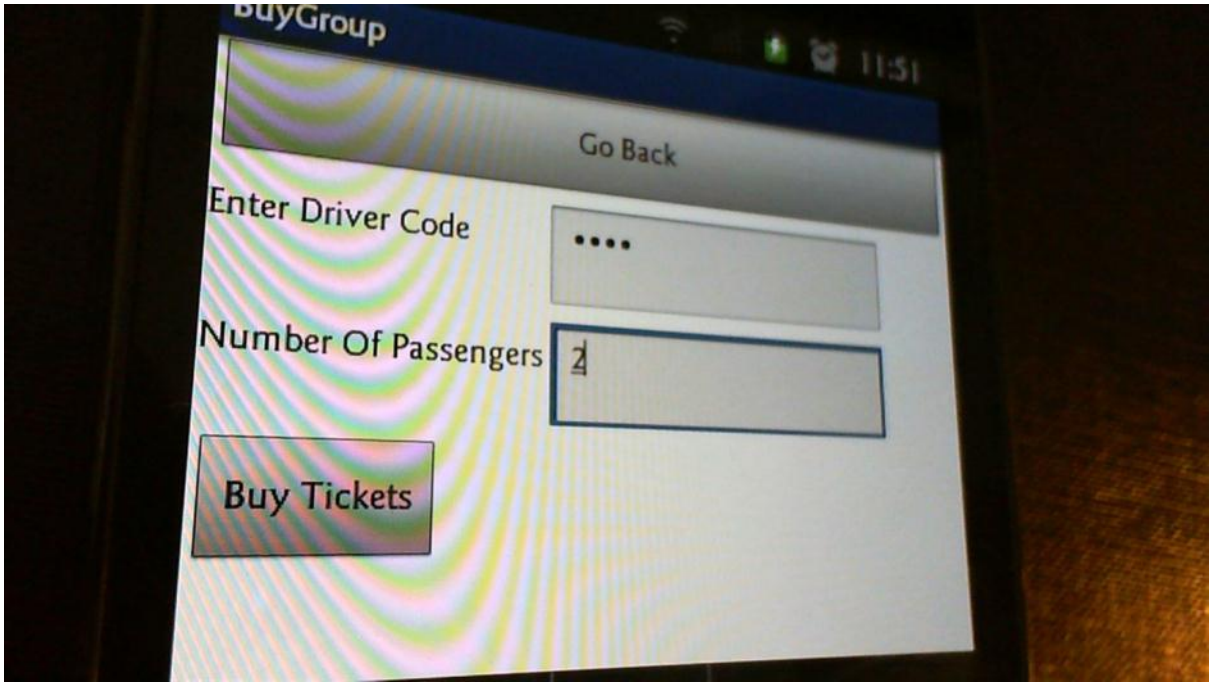


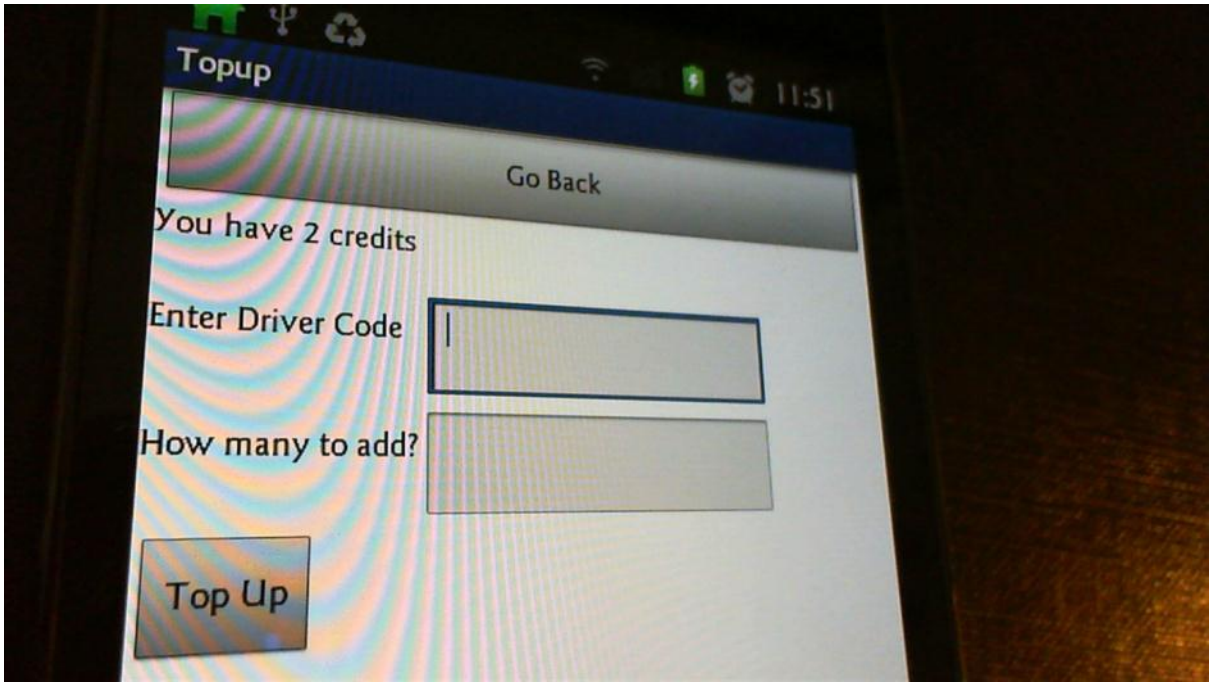
Changed to:



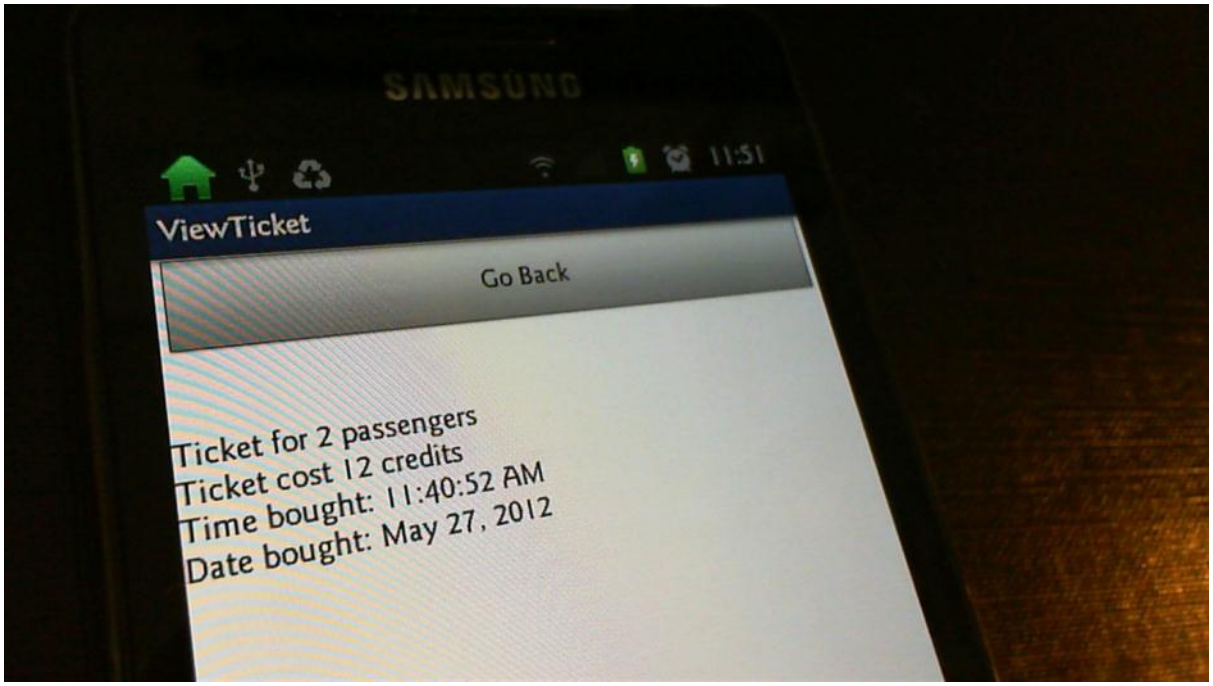
Re-testing for 4.1



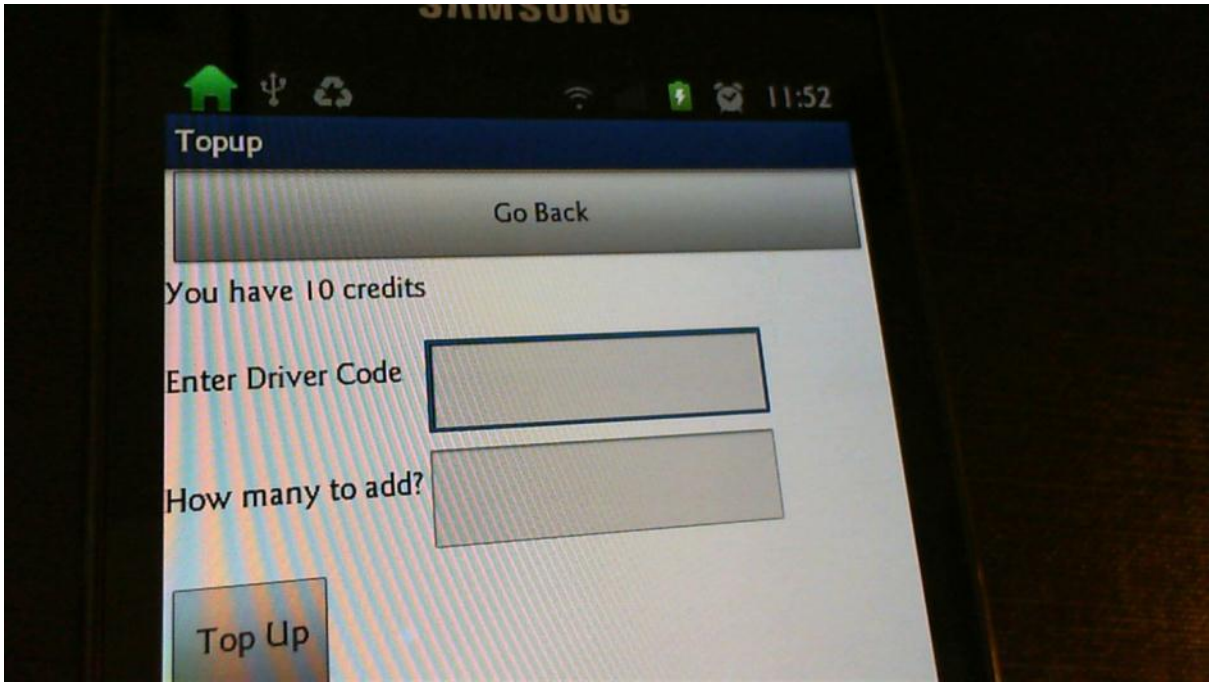


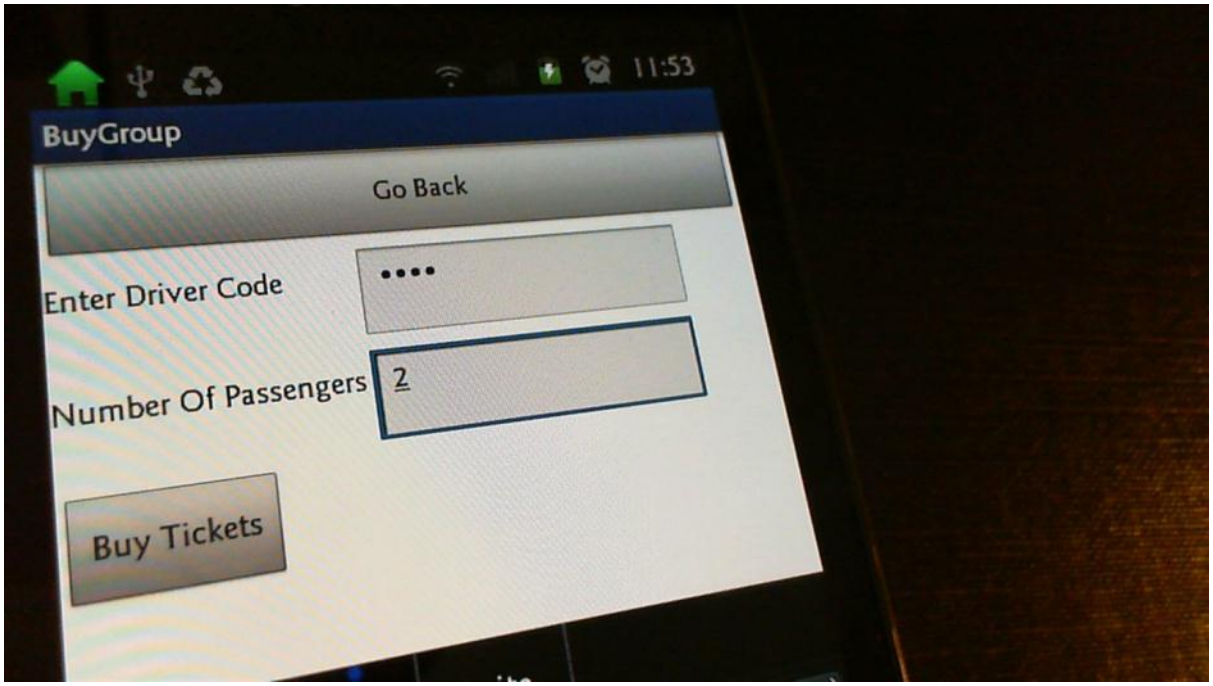


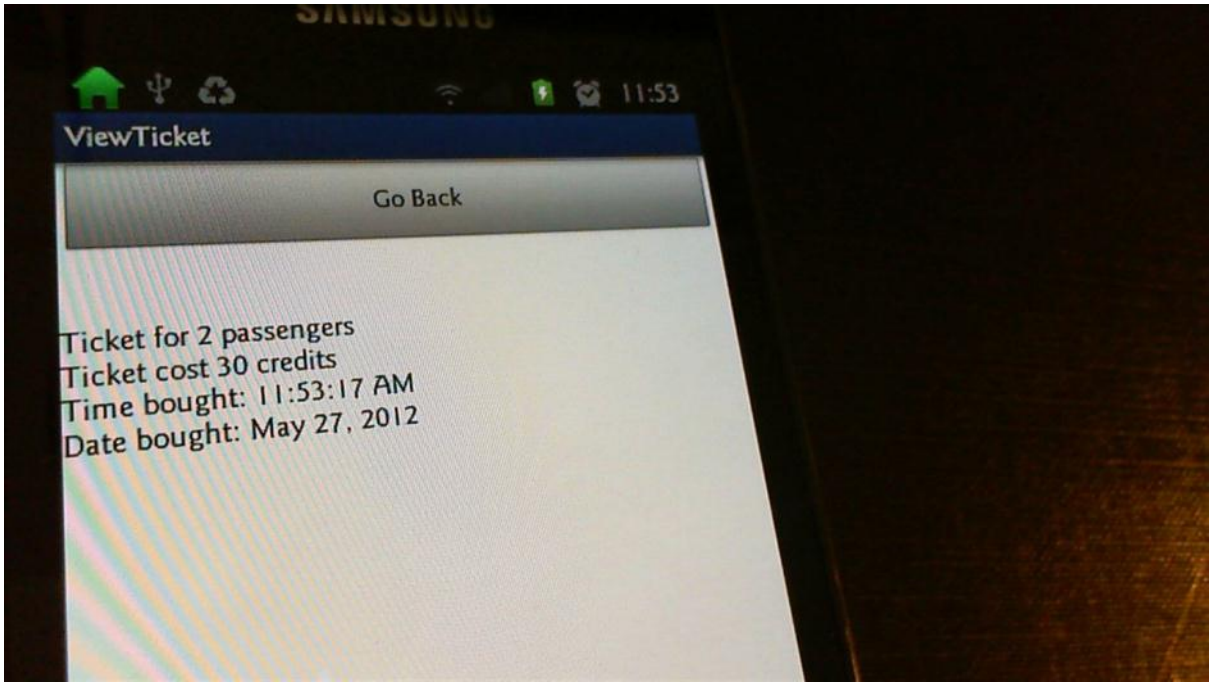
This screen shows that a new ticket hasn't been purchased:

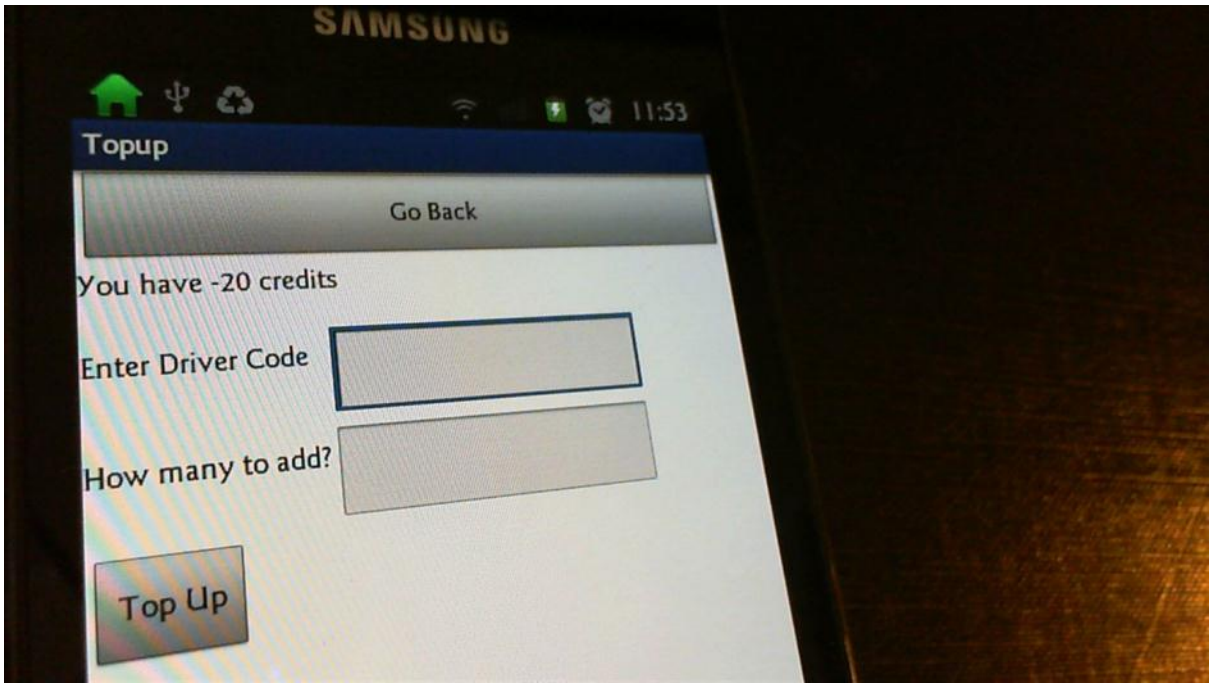


4.4

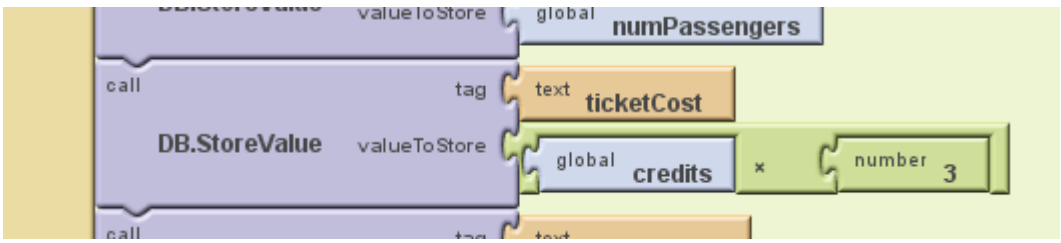

11:52
Topup
Go Back
You have 10 credits
Enter Driver Code
How many to add?
Top Up


11:53
BuyGroup
Go Back
Enter Driver Code
••••
Number Of Passengers
2
Buy Tickets

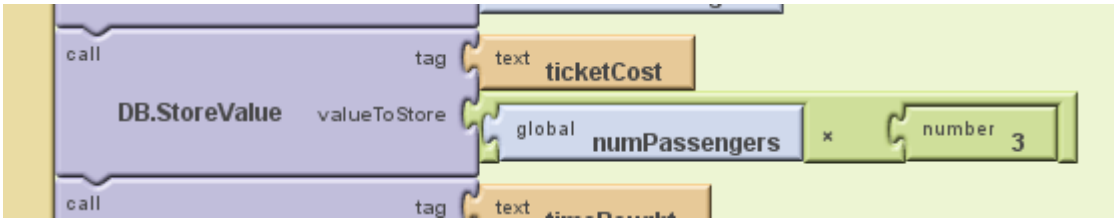




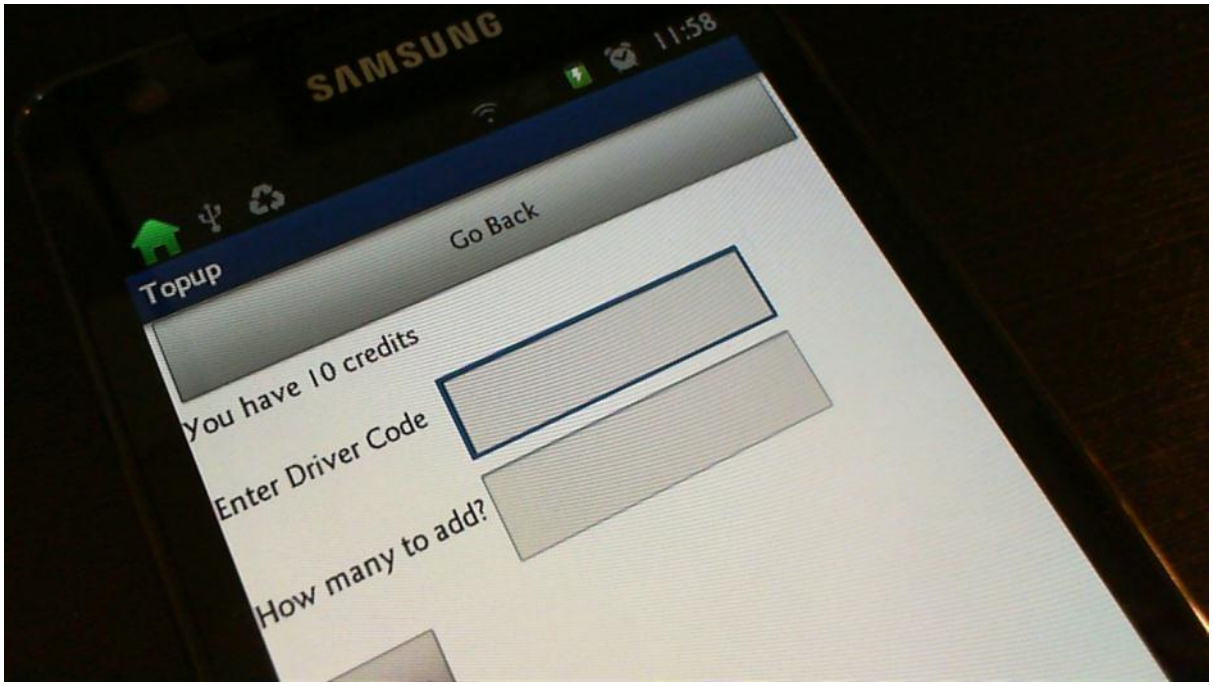
This part of the code is obviously wrong (it should store the cost of the ticket)

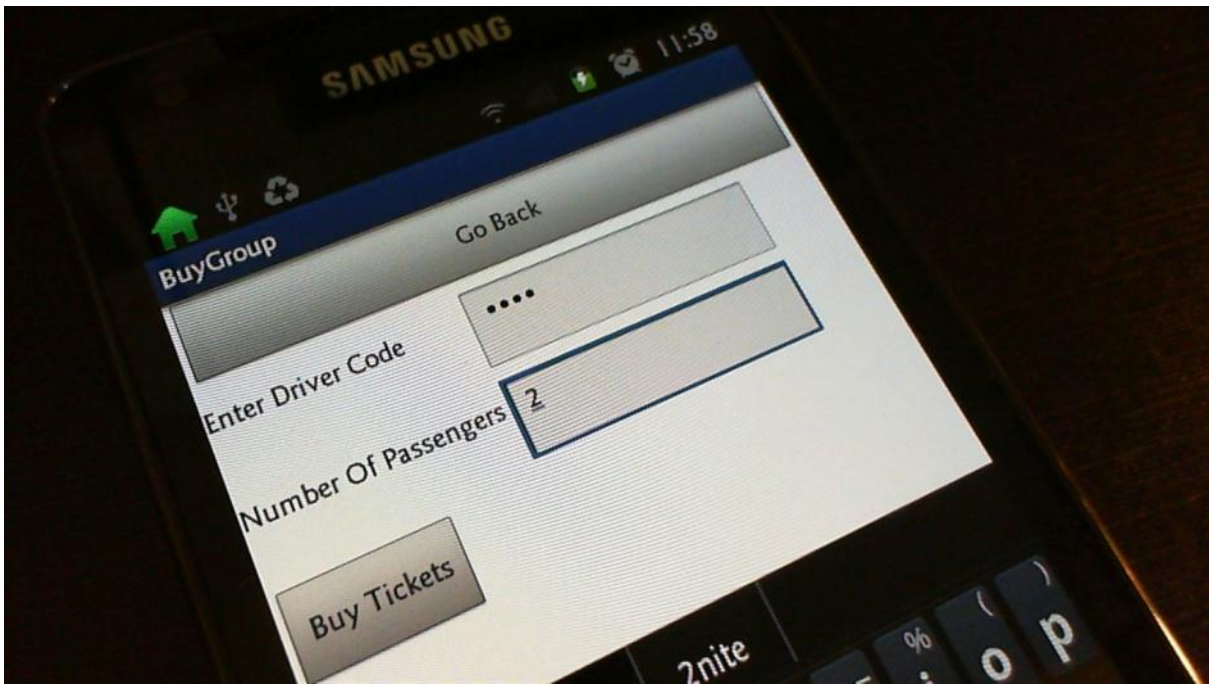


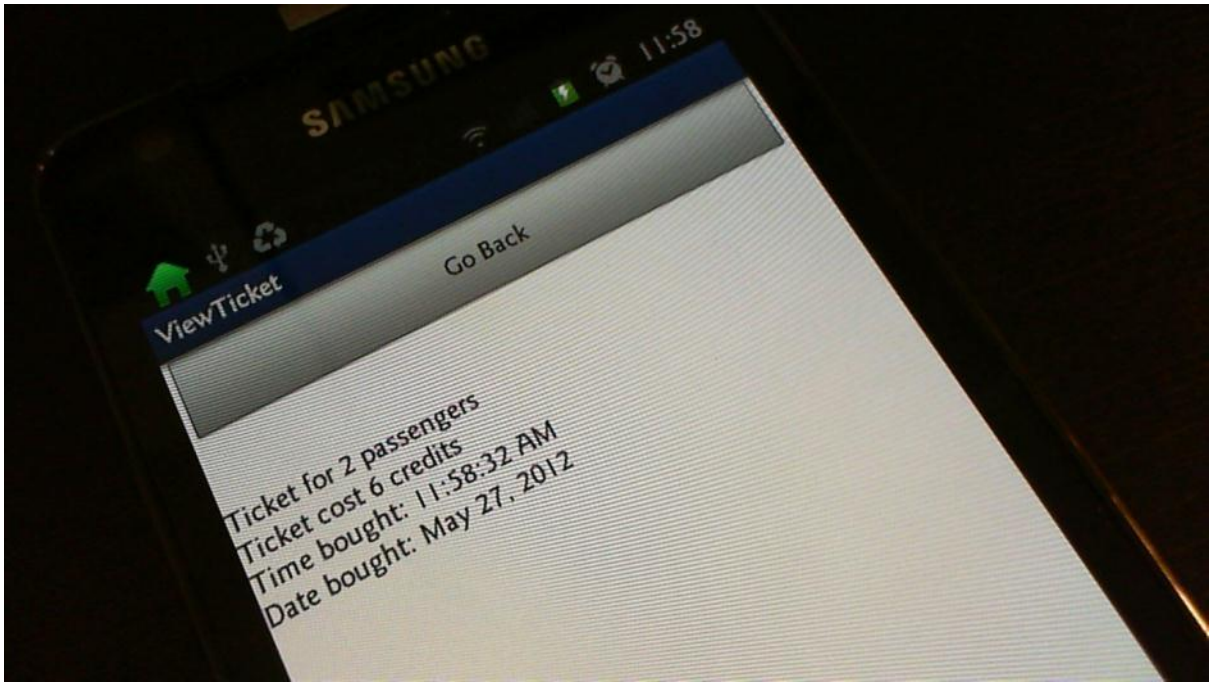
Changed to:

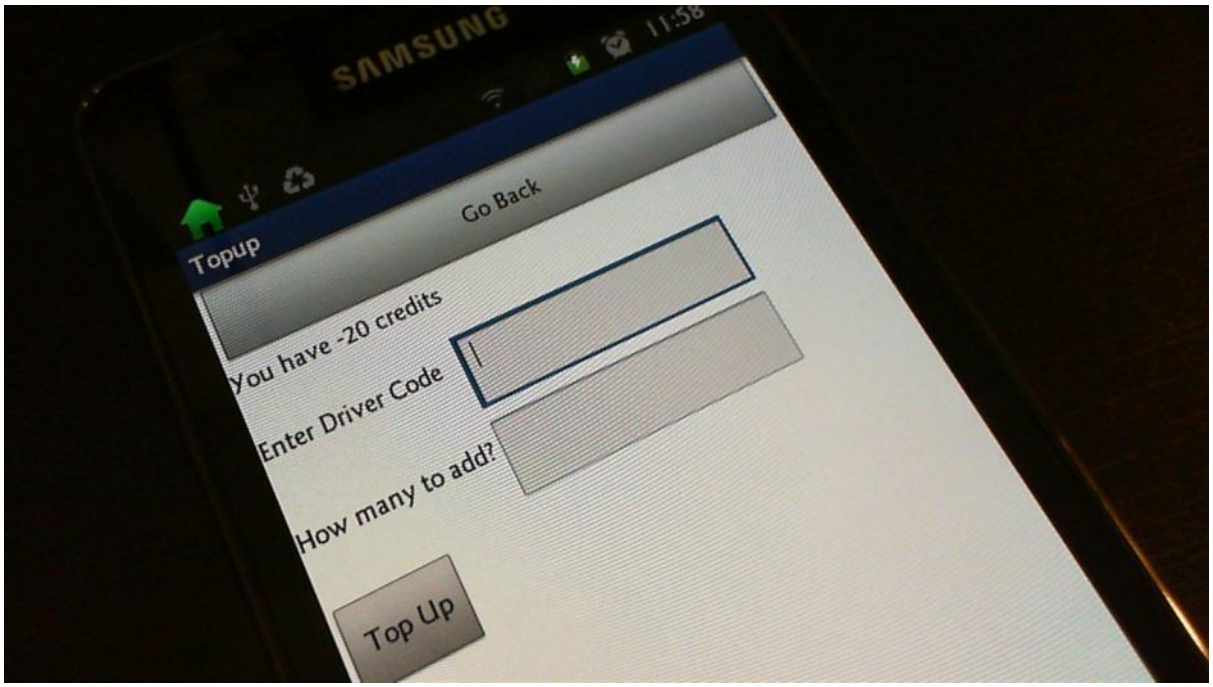


Re-testing for 4.4

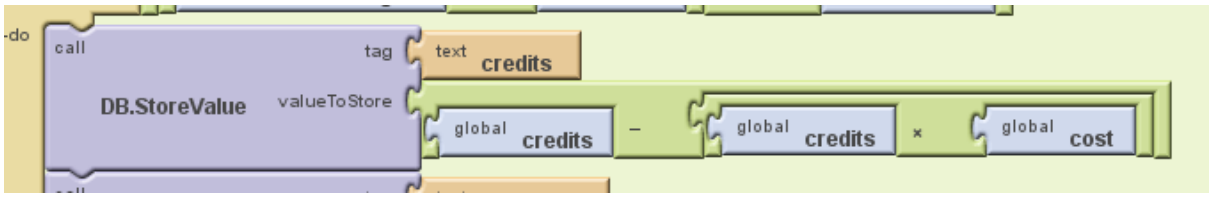




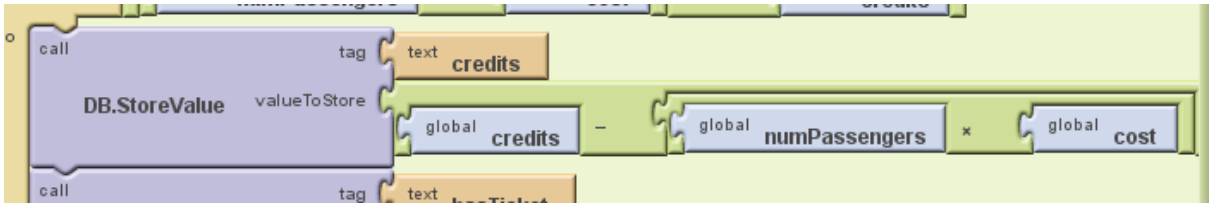




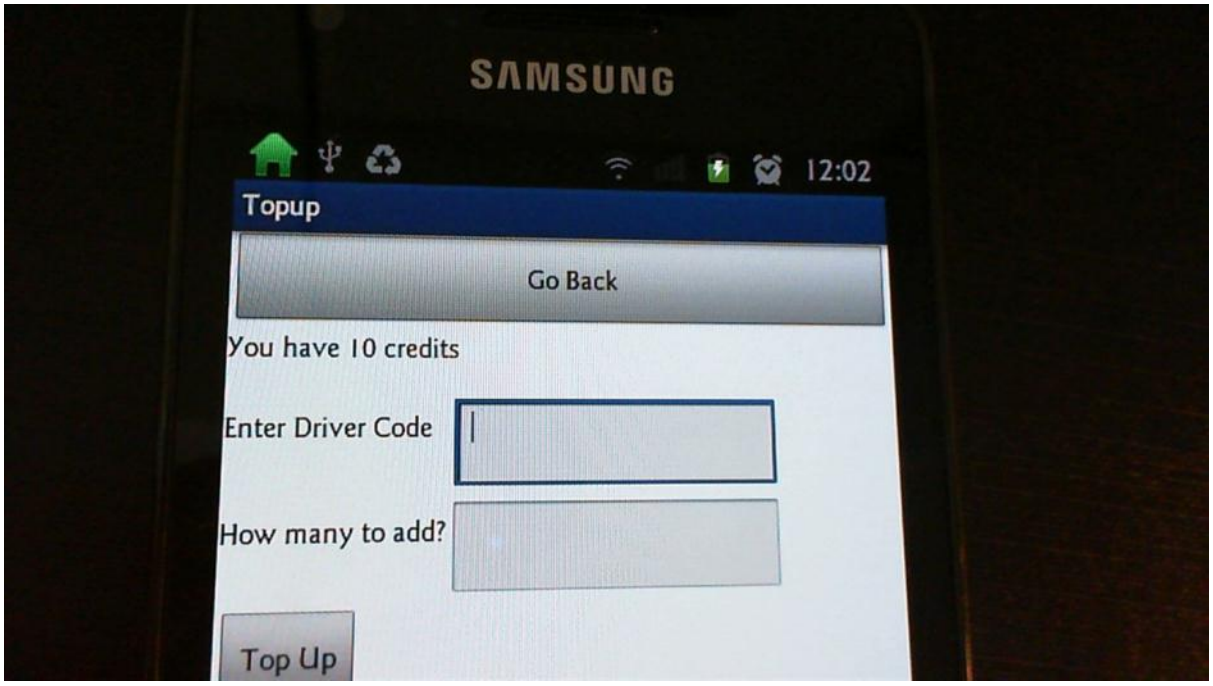
It is taking away far too many credits and this part of code shows why:

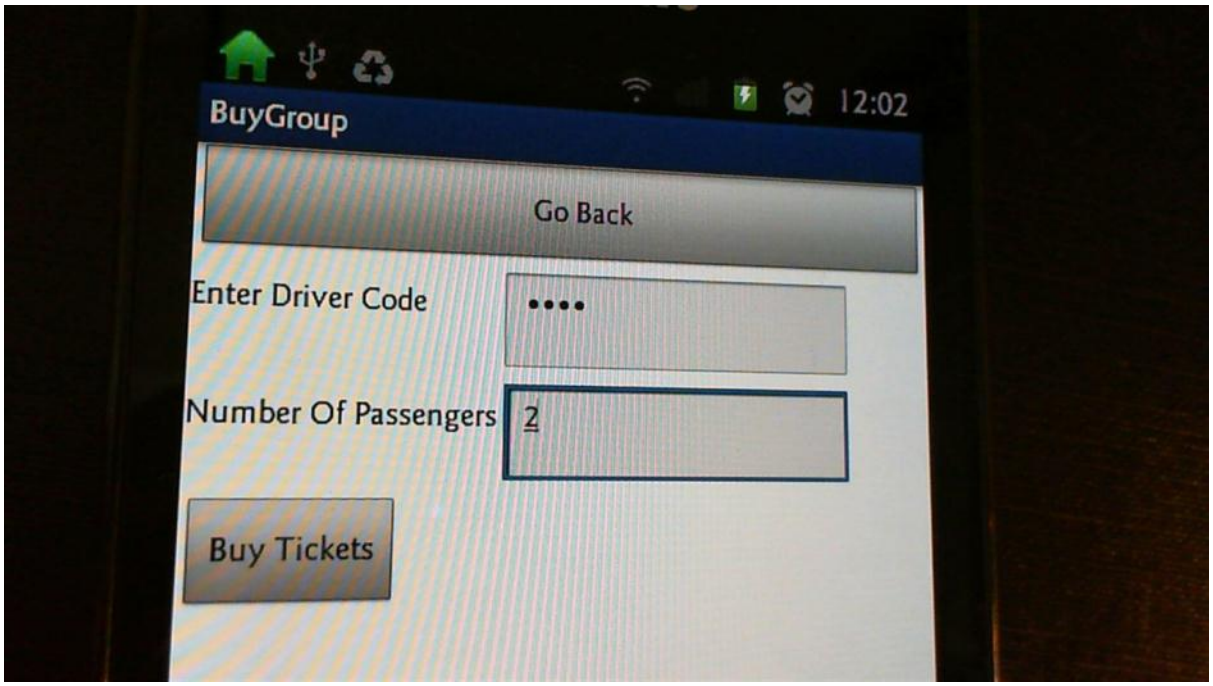


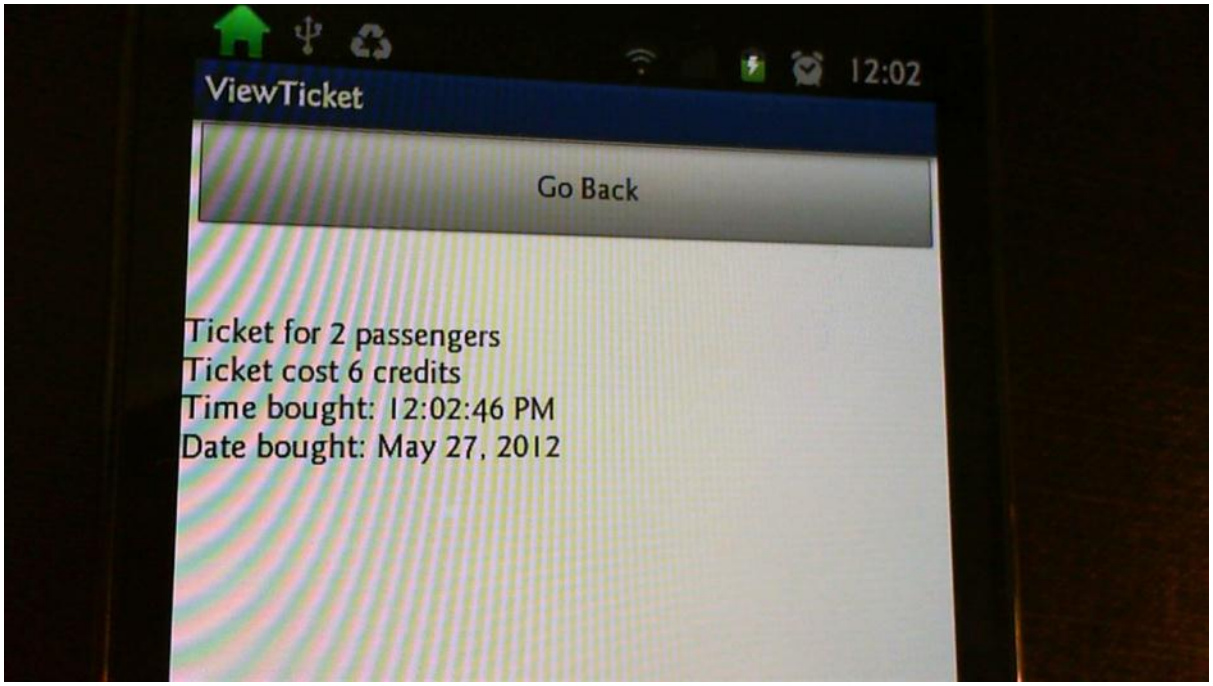
Changed to:

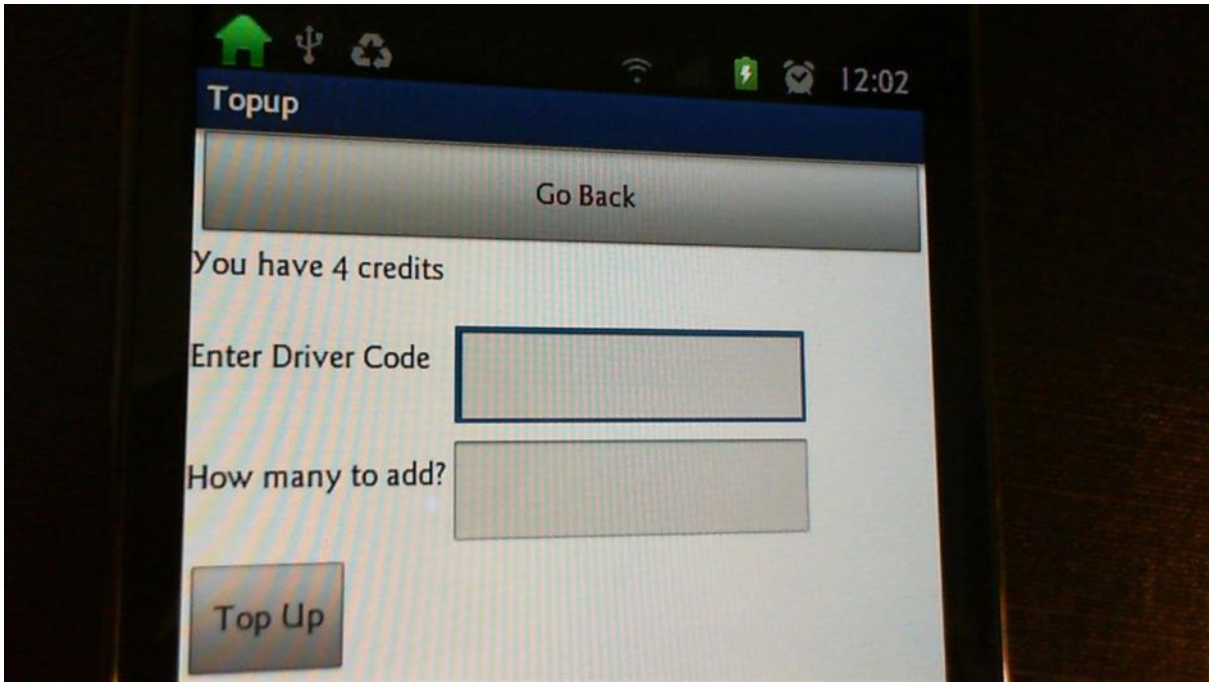


Re-testing for 4.4:

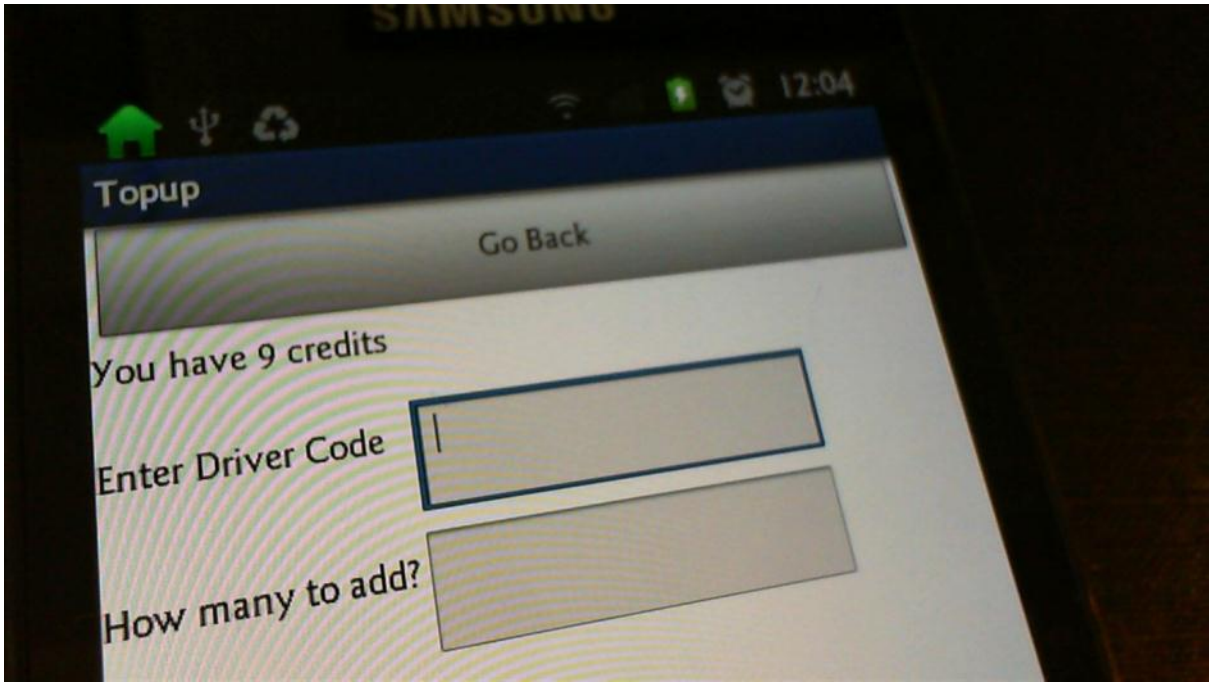


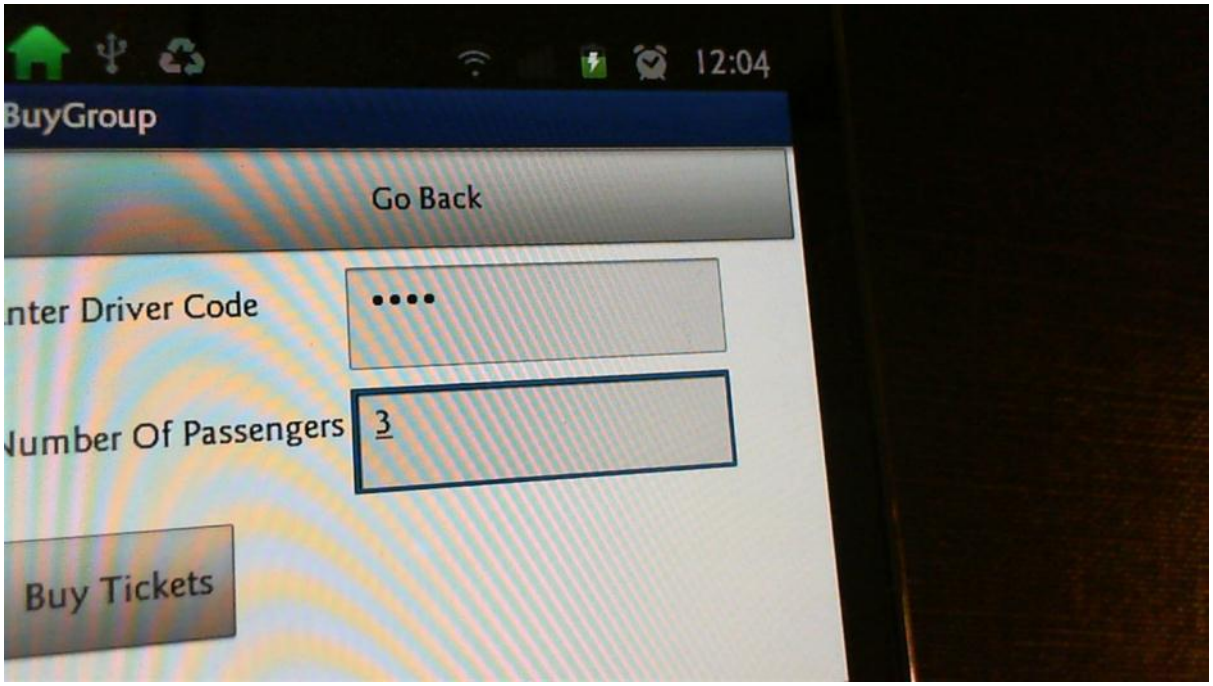


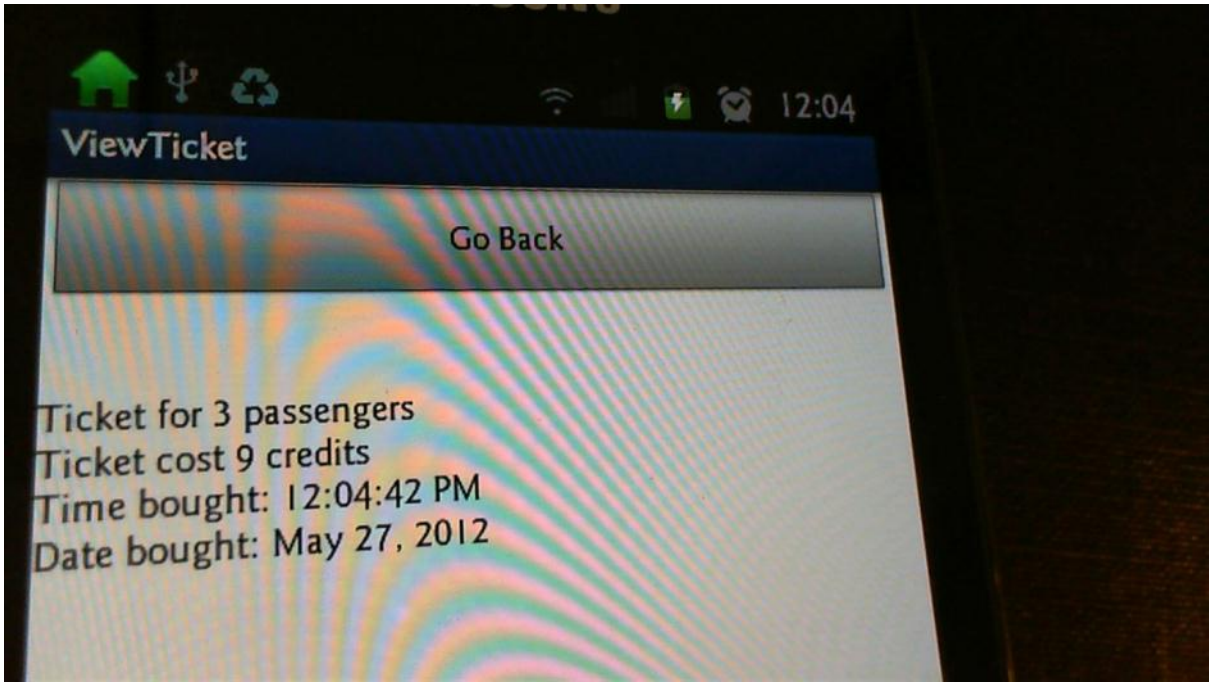

12:02
ViewTicket
Go Back
Ticket for 2 passengers
Ticket cost 6 credits
Time bought: 12:02:46 PM
Date bought: May 27, 2012

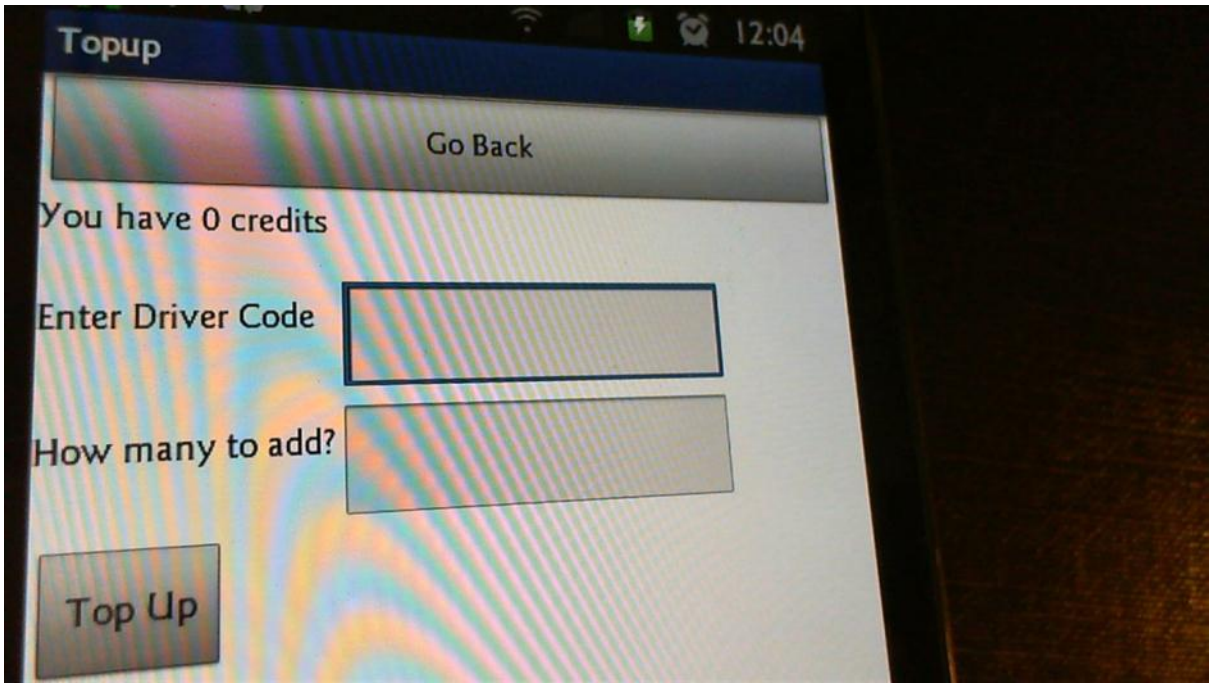

12:02
Topup
Go Back
You have 4 credits
Enter Driver Code
How many to add?
Top Up

4.5

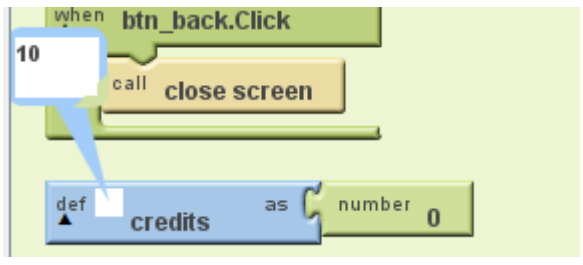

12:04
Topup
Go Back
You have 9 credits
Enter Driver Code
How many to add?

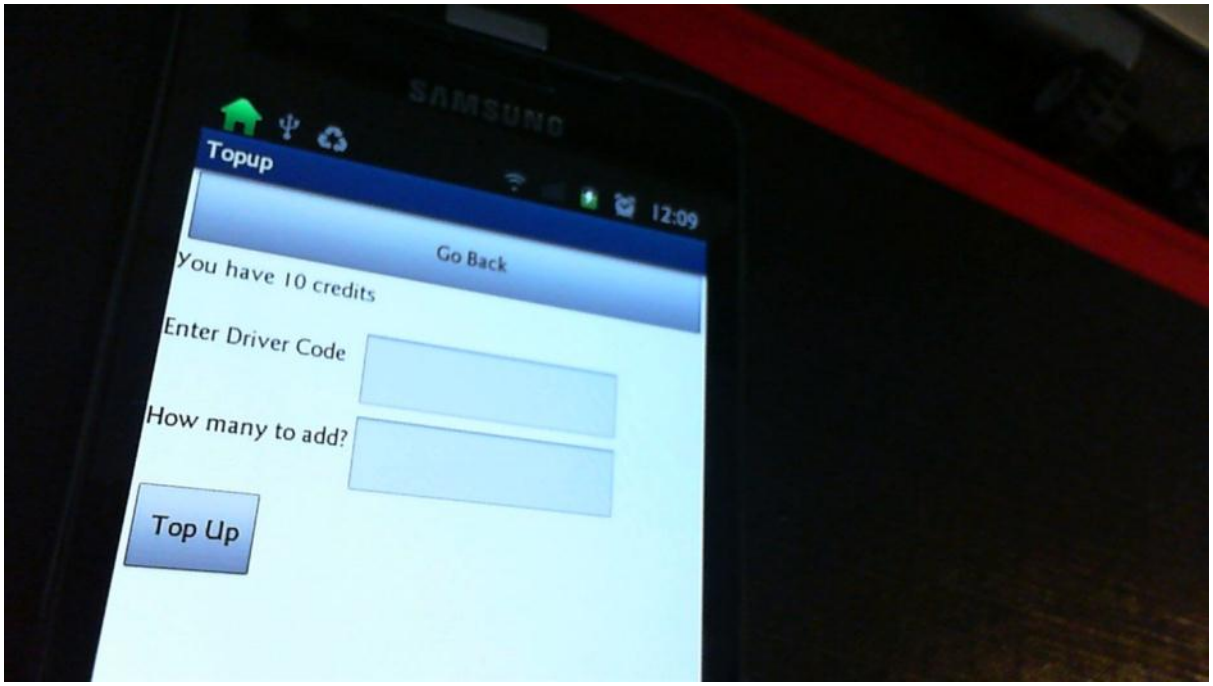

12:04
BuyGroup
Go Back
nter Driver Code
••••
Number Of Passengers
3
Buy Tickets


12:04
ViewTicket
Go Back
Ticket for 3 passengers
Ticket cost 9 credits
Time bought: 12:04:42 PM
Date bought: May 27, 2012

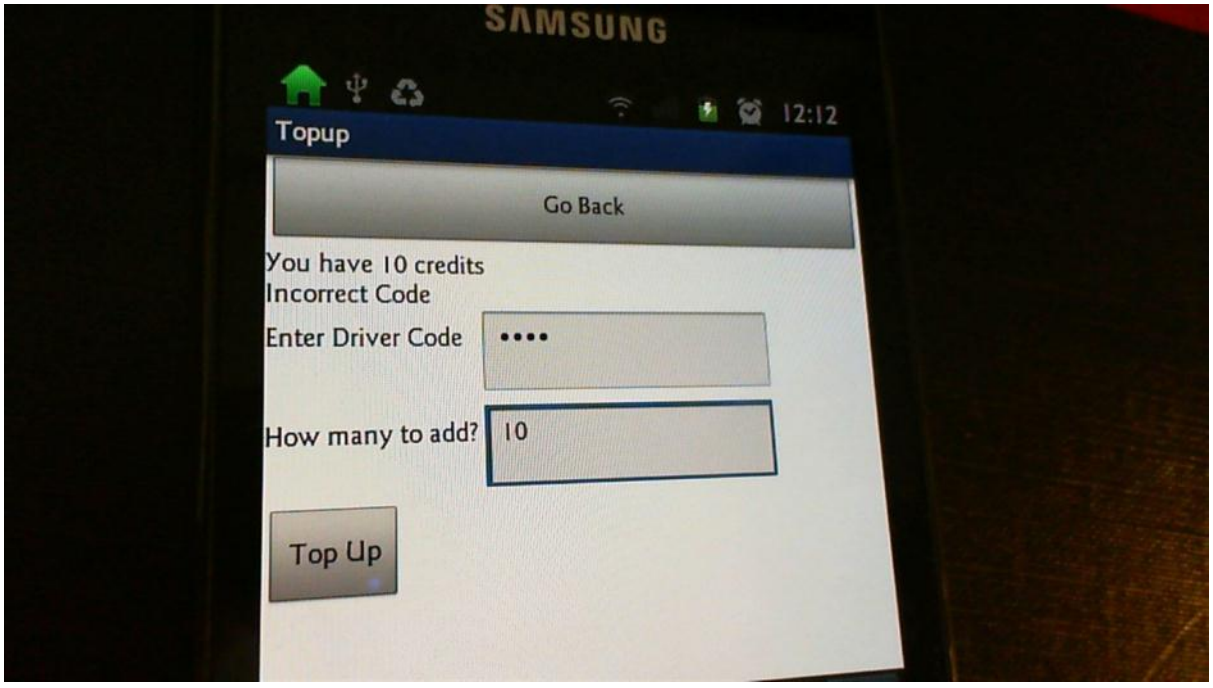

12:04
Topup
Go Back
You have 0 credits
Enter Driver Code
How many to add?
Top Up

5.2

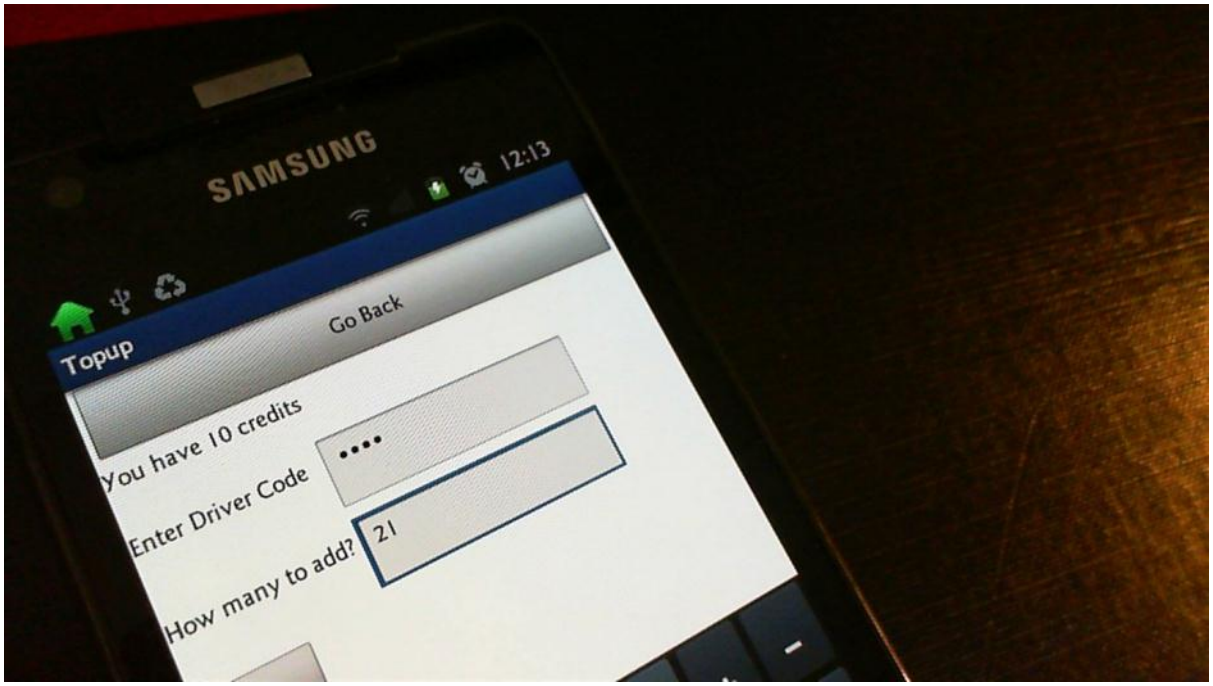

when btn_back.Click
10
call close screen
def credits as number 0

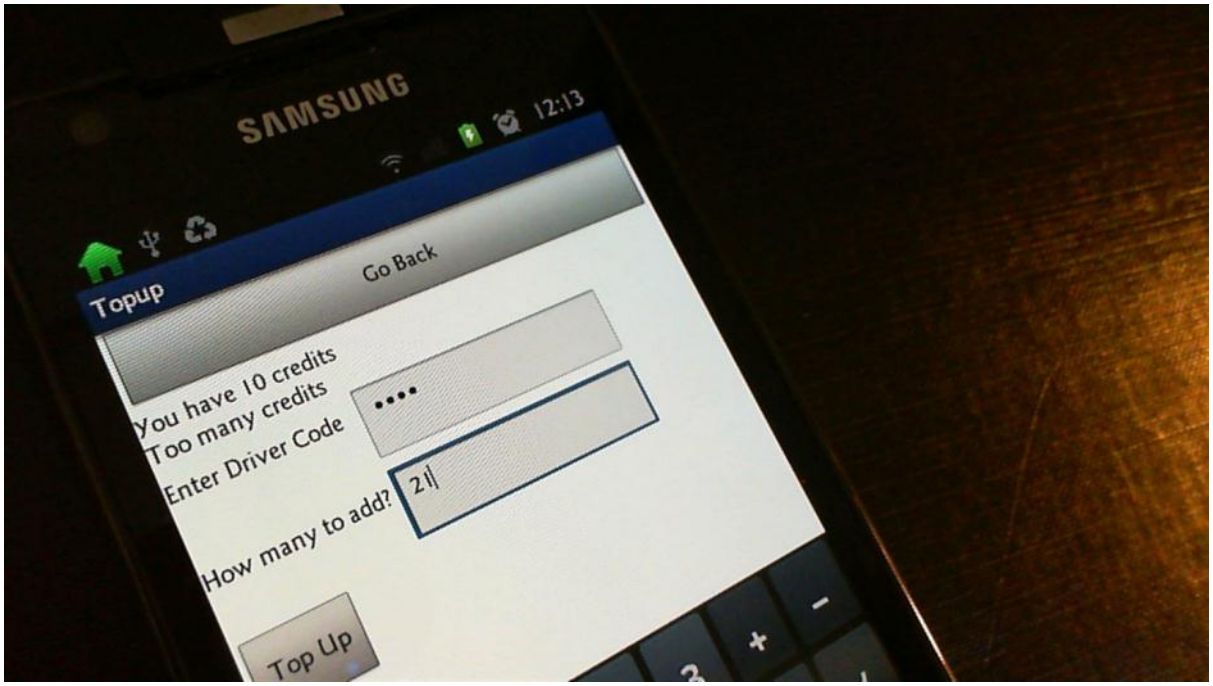

SAMSUNG
Topup
12:09
Go Back
You have 10 credits
Enter Driver Code
How many to add?
Top Up

5.3

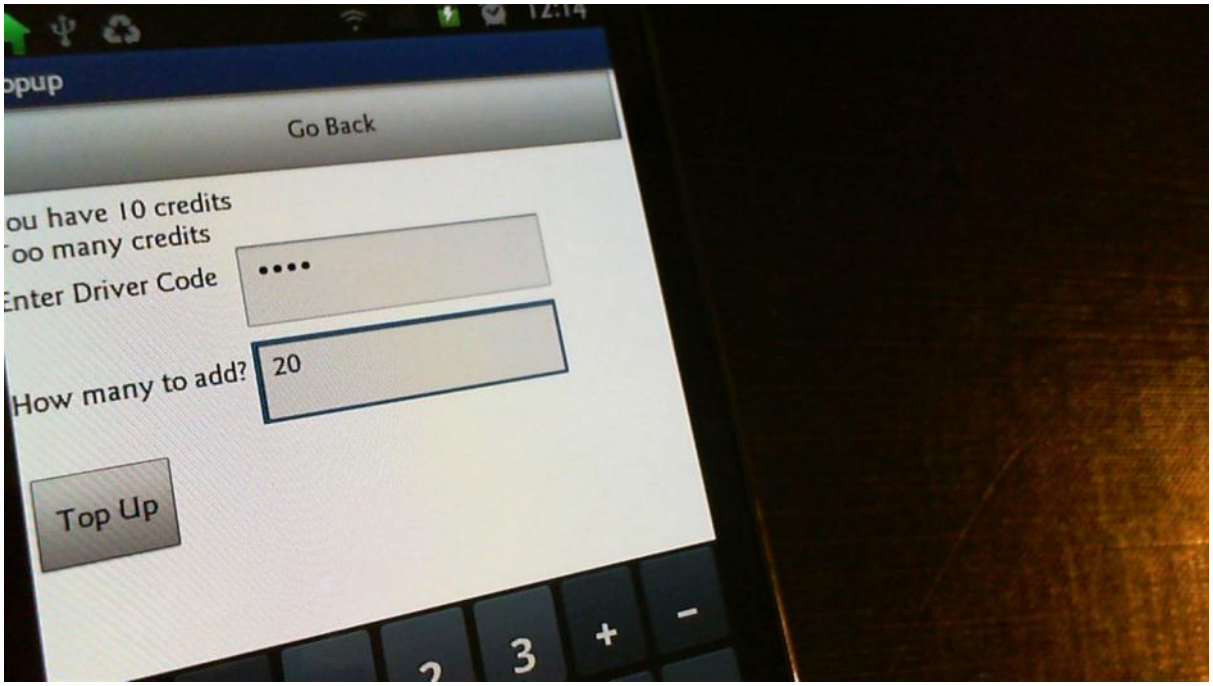

SAMSUNG
Topup
12:12
Go Back
You have 10 credits
Incorrect Code
Enter Driver Code
••••
How many to add?
10
Top Up

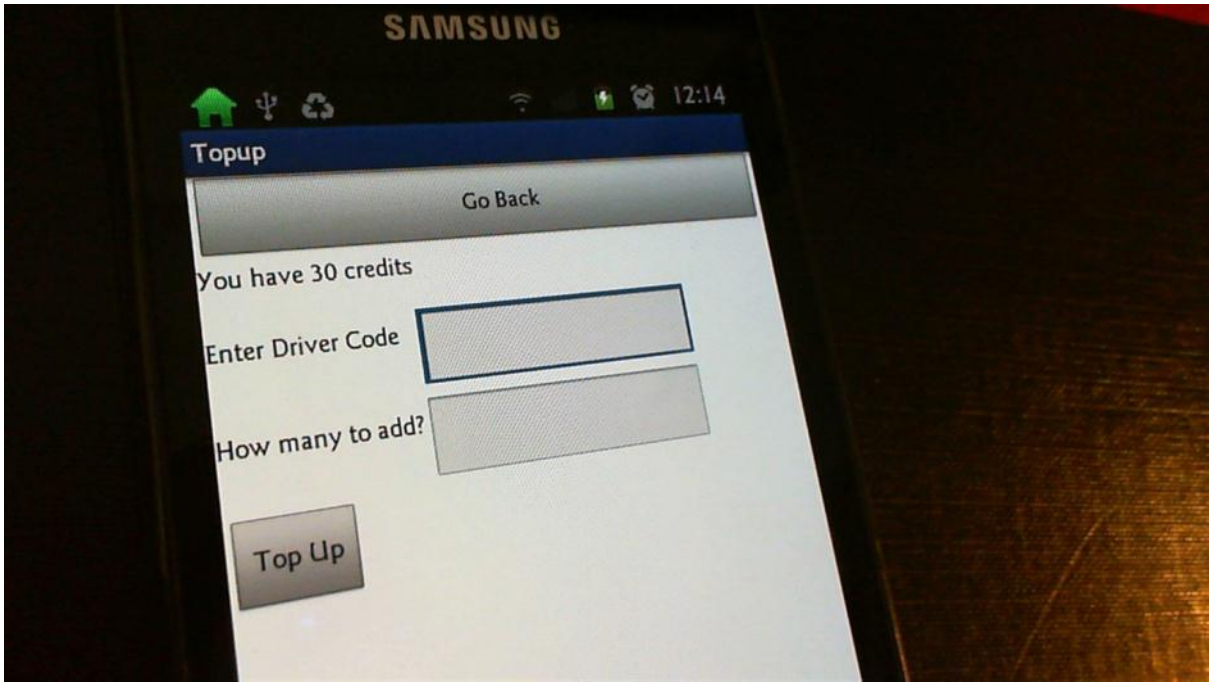
5.4


SAMSUNG
12:13
Topup
Go Back
You have 10 credits
Enter Driver Code
••••
How many to add?
21


SAMSUNG
12:13
Topup
Go Back
You have 10 credits
Too many credits
Enter Driver Code
••••
How many to add?
21
Top Up

5.5


opup
Go Back
ou have 10 credits
oo many credits
nter Driver Code
How many to add?
20
Top Up
3
+
-


SAMSUNG
12:14
Topup
Go Back
You have 30 credits
Enter Driver Code
How many to add?
Top Up

Evaluation

My solution contains all of the 4 main parts of this problem:

1. It allows the user to buy a single or return ticket
2. It allows the user to buy a single ticket for multiple passengers
3. It allows the user to buy extra credit
4. It can display the current ticket

In more detail:

The first screen the user sees has four buttons (one for each of the four points above). This screen doesn't have any other clutter because I didn't think this would add to the design of the app. The app could have details about the previous use (a data structure that contains previous tickets and top up history) but this wasn't in the brief. The last part of the brief hasn't been addressed directly but I think I could have done this by having another app that accesses the database and sets a flag to be 'off' or 'stolen' that is checked every time the app is used.

I have fully tested my app and am confident that the features work as they should. I had a series of problems that I have fixed and it now works fine. However, if this app was to be used as a proper app then I would need to make the database better. At the moment the driver code is set to '1234', this will be the same for every driver and isn't secure. Also, every phone would access the same information and I would have to have a better database that uses a primary key for the user so his or her credit and ticket information is unique to them. This is significantly more work but I would have to do it if my app was to go on sale.

I could improve my app by making it look more professional. At the moment I have just concentrated on it working well but I could also design a logo and make the overall design sleeker. The way people enter data could also be better – at the moment you have to use the normal typewriter input system to enter numbers, this would be much better if it just used a number pad. The input fields are mostly validated but I have found out that if you enter non-numeric data for the number of passengers then it causes an error, this would have to be fixed. My warning dialog could also be better.

The screens are all simple in design and they all have a 'back' button in the same place to make their use quick and easy. This is good because the app will be used in rushed situations. The ticket display is quite small and this could be changed but overall the screens work well.

If I had more time then I would do the fifth part of the brief and make the application look more professional but overall I am happy with the way it works.