Lesson 6 –
Mobile app development (part 1)

- S.P. Chong

Objectives

- In this lesson, you will learn to create a **mobile app**. for an **Android** phone.

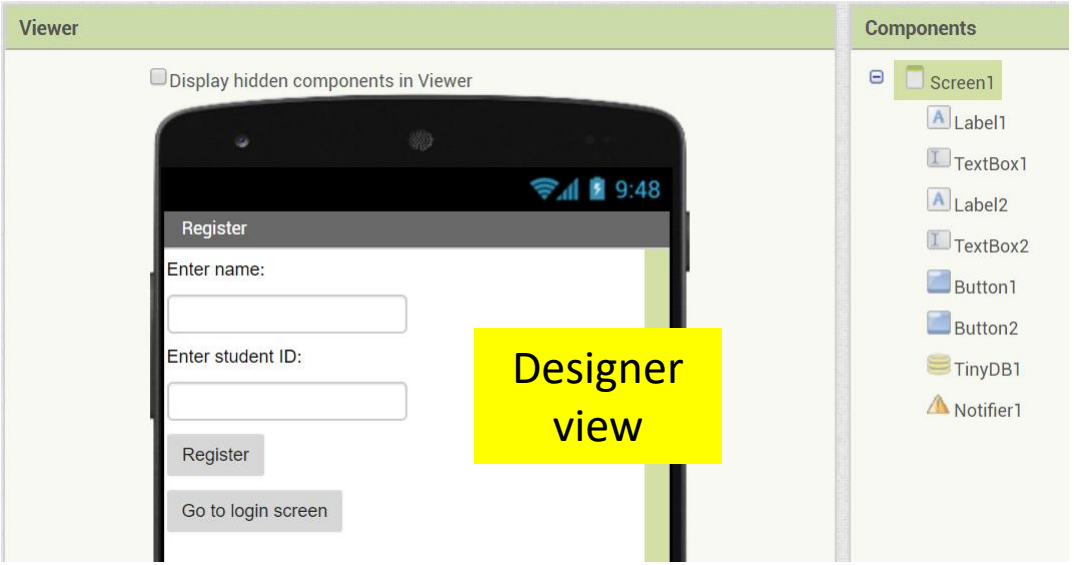
- The programming will be "**graphical**", which means you don't really need to learn a new programming language.

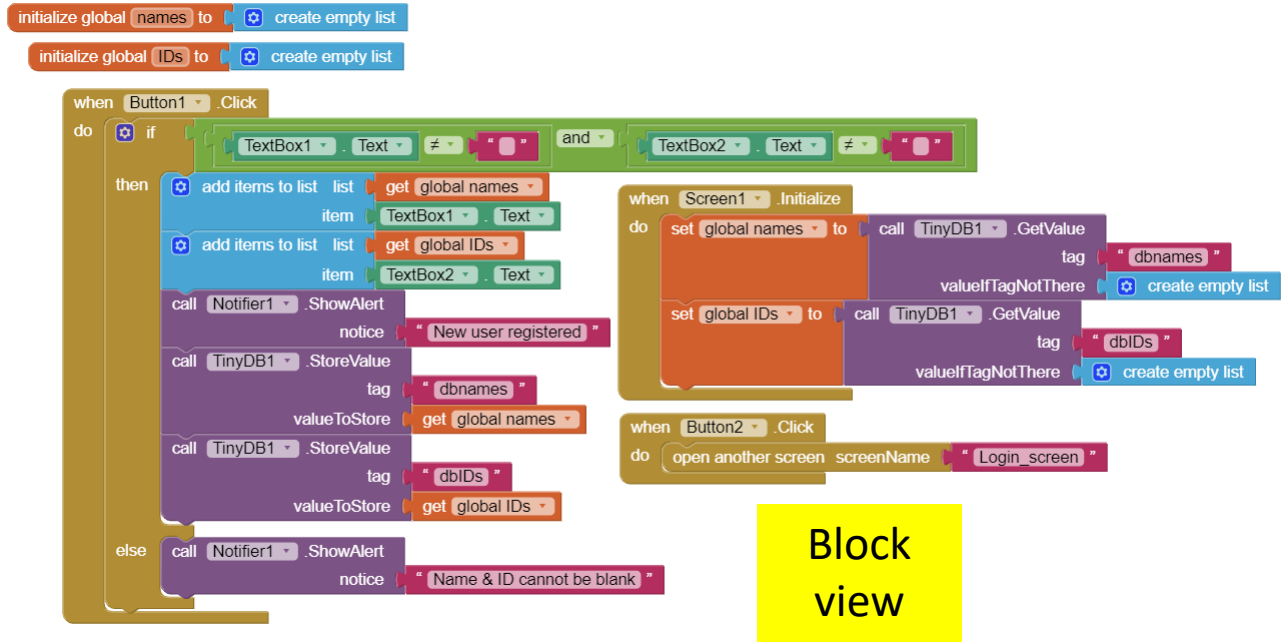
- You will learn to create **user interface**, handle **events** and use selected **phone features** such as location sensor.

- The example apps will allow a user to **register & login**, to do **remote monitoring & control**.

What is App Inventor?

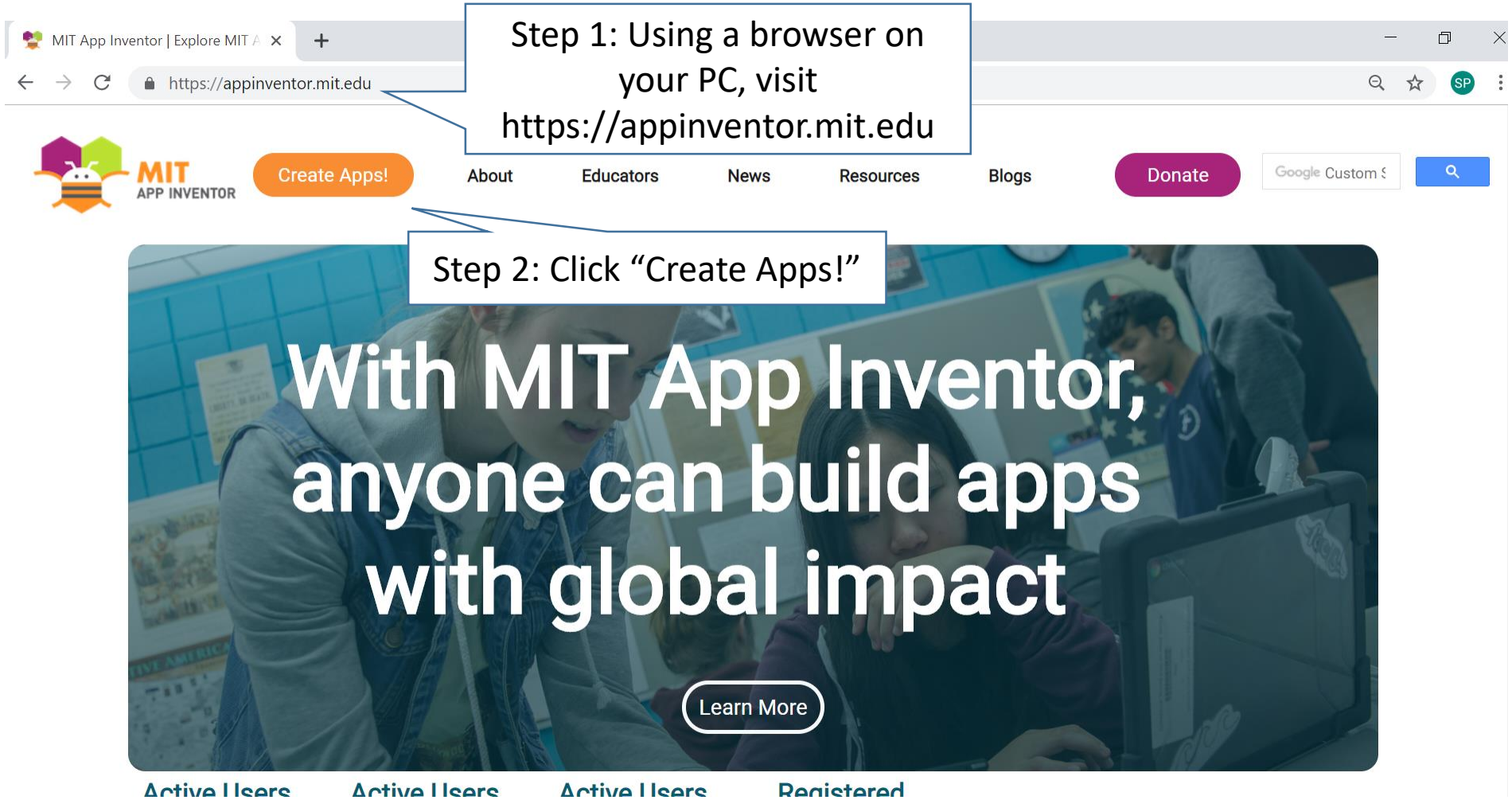
- App Inventor is a software package created by M.I.T., that allows you to develop an Android App without having to learn a new programming language.
- Its Designer view allows you create a user interface, by adding components (such as labels, text-boxes and buttons), laying them out on a “mobile phone” screen, and defining their properties.
- Its Block view allows you to define the app behaviour, by using a jigsaw puzzle / graphical method of programming.





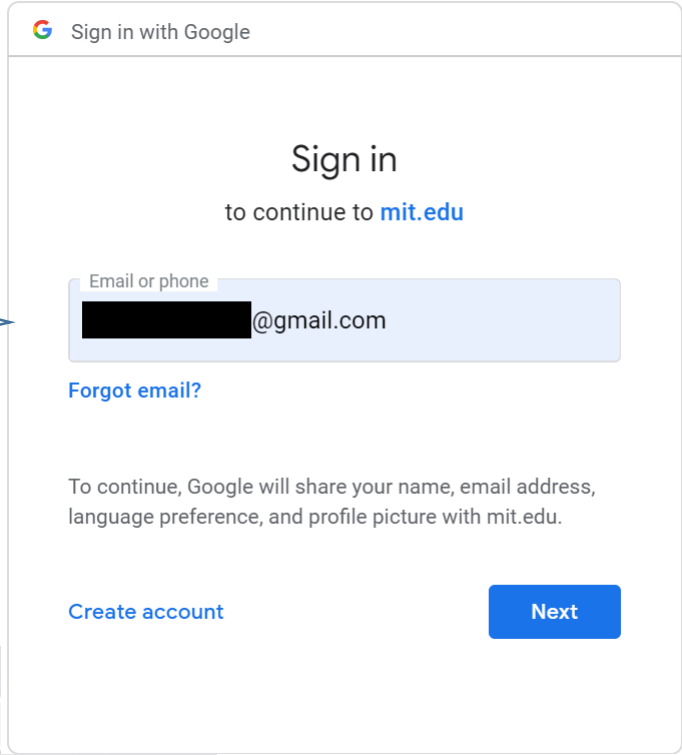
App Inventor – a quick intro

- To create an app using App Inventor, follow steps 1 to 14 below:

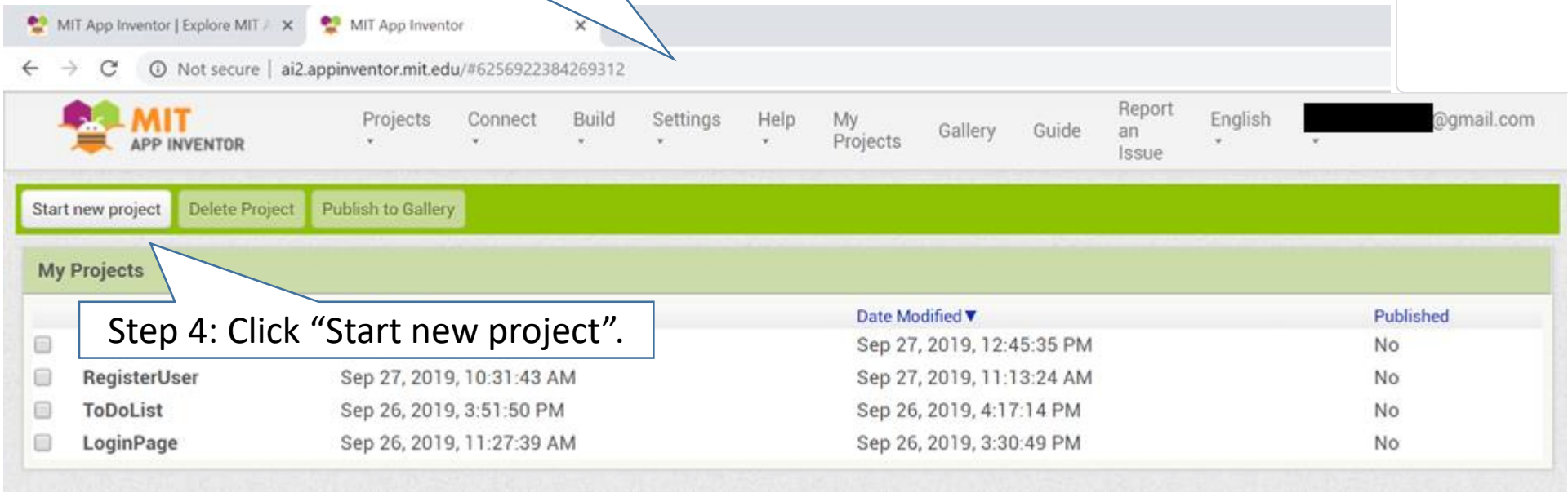


App Inventor – a quick intro (cont.)

Step 3: Sign in with a Google account.

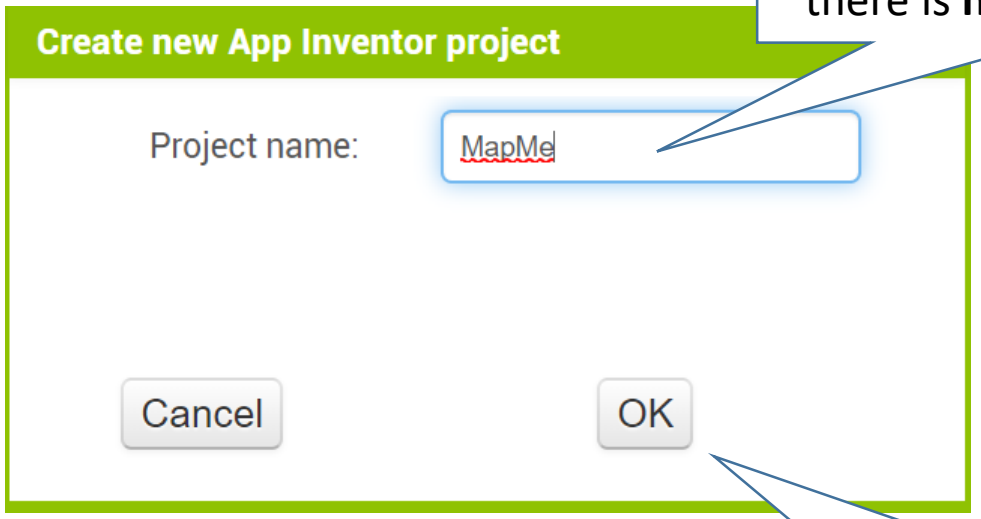


After signing in, you will be presented with a list of your **projects** (if you have used App Inventor before) or a welcome screen (if you are new to App Inventor):



Step 4: Click “Start new project”.

App Inventor – a quick intro (cont.)

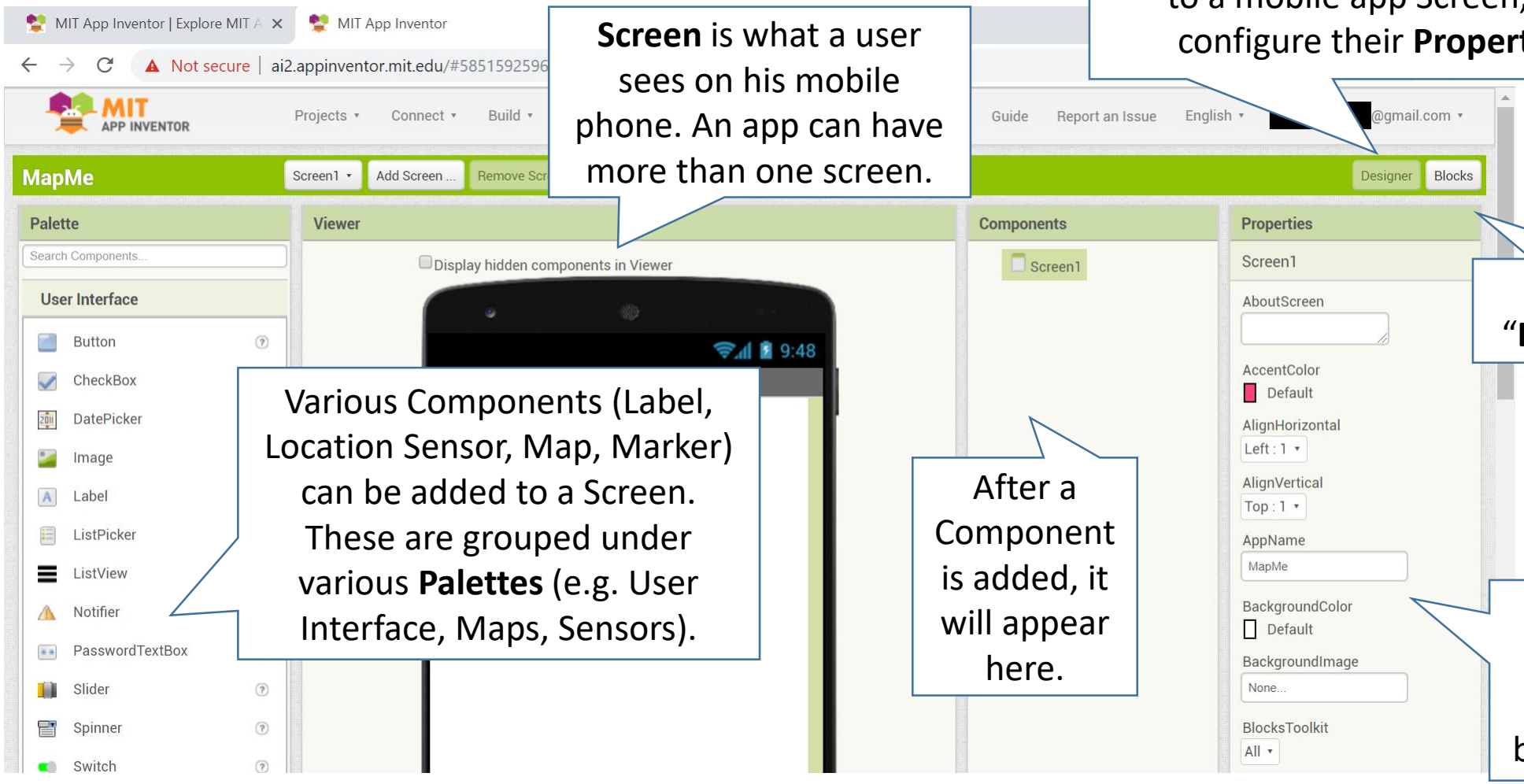


Step 5: Enter “MapMe” as the “Project name”, ensuring that there is **no space** in the name.

Step 5b: Click “OK” after that.

App Inventor – a quick intro (cont.)

You will be presented with the "**Designer**" view with a few panes. This is where you add the various **Components** (e.g. button, text box) to a mobile app Screen, and configure their **Properties**.



Screen is what a user sees on his mobile phone. An app can have more than one screen.

We will look at "**Blocks**" view later.

Various Components (Label, Location Sensor, Map, Marker) can be added to a Screen. These are grouped under various **Palettes** (e.g. User Interface, Maps, Sensors).

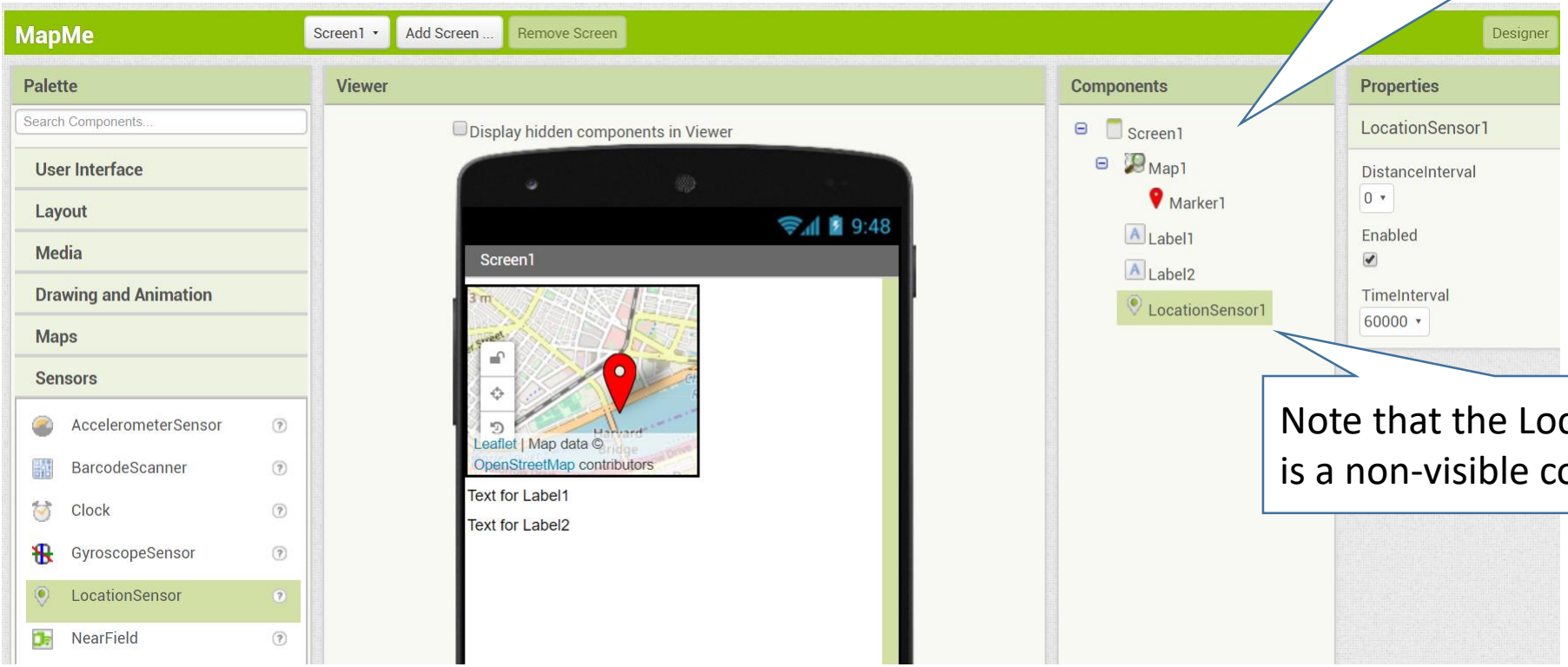
After a Component is added, it will appear here.

The default Properties of a Component can be modified here.

App Inventor – a quick intro (cont.)

Step 6: Add these Components to Screen 1:

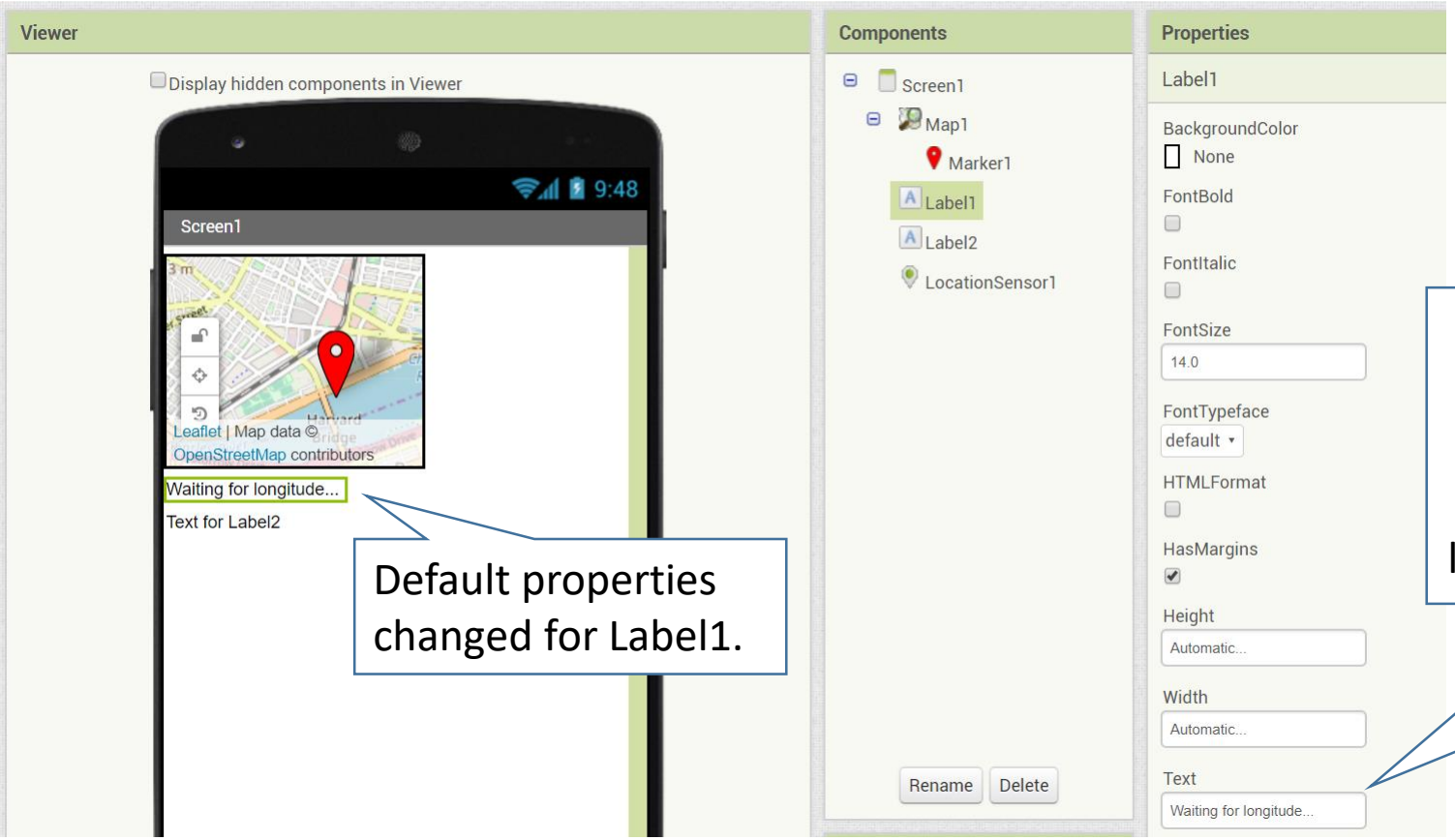
- ✓ Map (from Maps)
- ✓ Marker (from Maps)
- ✓ 2 x Label (from User Interface)
- ✓ LocationSensor (from Sensors)



Note that the LocationSensor is a non-visible component.

App Inventor – a quick intro (cont.)

It is a good practice to save your project every now and then.



Default properties changed for Label1.

Step 7: Change the Text for Label1 to become "Waiting for longitude…". Similarly, change the Text for Label2 to become "Waiting for latitude…". These 2 Labels will display the longitude and latitude from the LocationSensor.

Step 8: Once you are done with the "Designer" view, click "Blocks".

App Inventor – a quick intro (cont.)

Step 9: Complete the "**jigsaw puzzle**" program below:

MapMe | Screen1 | Add Screen ... | Remove Screen

Blocks

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1
  - Map1
    - Marker1
  - Label1
  - Label2
  - LocationSensor1

Viewer

```
when LocationSensor1 .LocationChanged
  latitude  longitude  altitude  speed
do
  call Map1 .PanTo
      latitude   get longitude
      longitude  get longitude
      zoom       5
  call Marker1 .SetLocation
      latitude   get latitude
      longitude  get longitude
  set Label1 . Text to get longitude
  set Label2 . Text to get latitude
```

```
initialize global latitude to 0
initialize global longitude to 0
initialize global Marker1 to " "
```

0 0

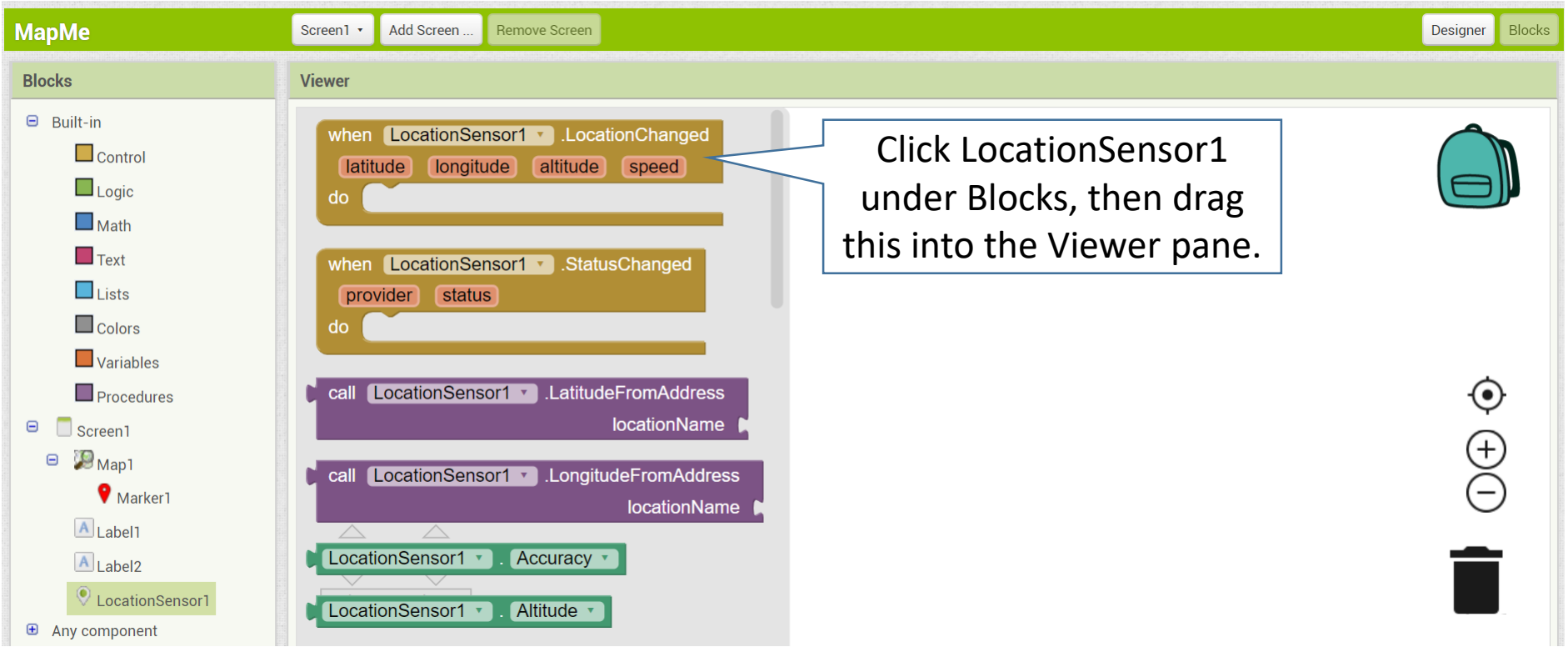
Show Warnings

All you need to do is to drag the code **Blocks** associated with the various Components to the **Viewer** pane and snapping them together.

The next slide shows you how the "when LocationSensor1.LocationChanged" piece is added.

App Inventor – a quick intro (cont.)

How the
"when LocationSensor1.LocationChanged"
piece is added.



Click LocationSensor1 under Blocks, then drag this into the Viewer pane.

App Inventor – a quick intro (cont.)

This slide shows you where the various blocks can be found.

These give “get latitude” & “get longitude” for the rest of the program.

These are from Variables

MapMe

Screen1 | Add Screen ... | Remove Screen

Blocks

- Built-in
  - Control
  - Logic
  - Math
  - Text
  - Lists
  - Colors
  - Variables
  - Procedures
- Screen1
  - Map1
    - Marker1
  - Label1
  - Label2
  - LocationSensor1

Viewer

```
when LocationSensor1 .LocationChanged
  latitude  longitude  altitude  speed
do
  call Map1 .PanTo
      latitude   get longitude
      longitude  get longitude
      zoom       5
  call Marker1 .SetLocation
      latitude   get latitude
      longitude  get longitude
  set Label1 . Text to get longitude
  set Label2 . Text to get latitude
```

```
initialize global latitude to 0
initialize global longitude to 0
initialize global Marker1 to “ ”
```

This is from Math.

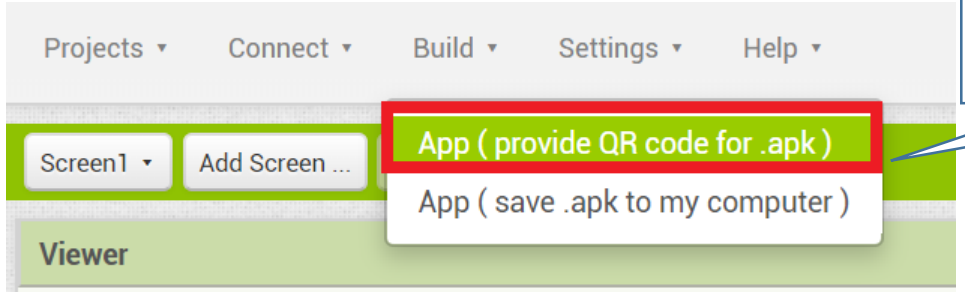
This is from Text.

These (4 purple & green blocks) are from Map1, Marker1, Label1 & Label2, obviously.

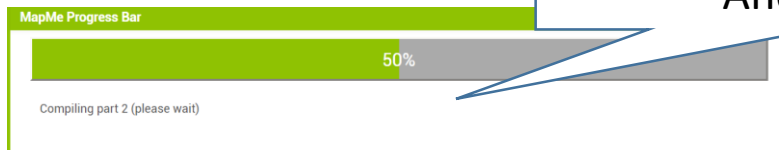
0 0

Show Warnings

App Inventor – a quick intro (cont.)

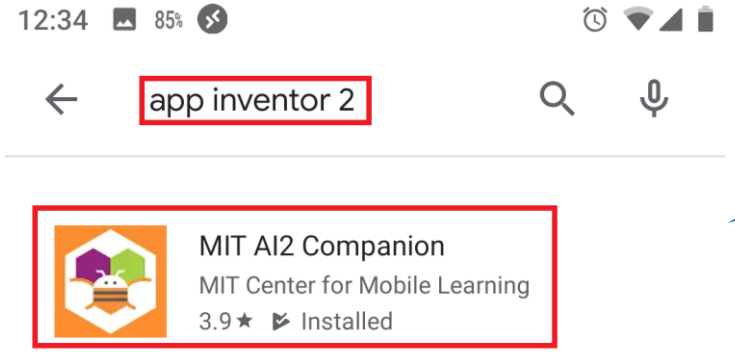


Step 10: Click Build > App (provide QR code for .apk).



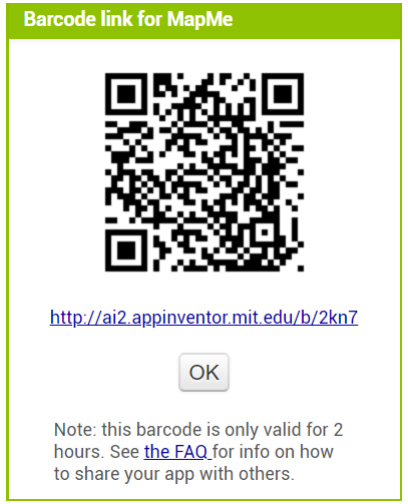
The app will be "compiled" into an **.apk** file for deployment in an Android phone.

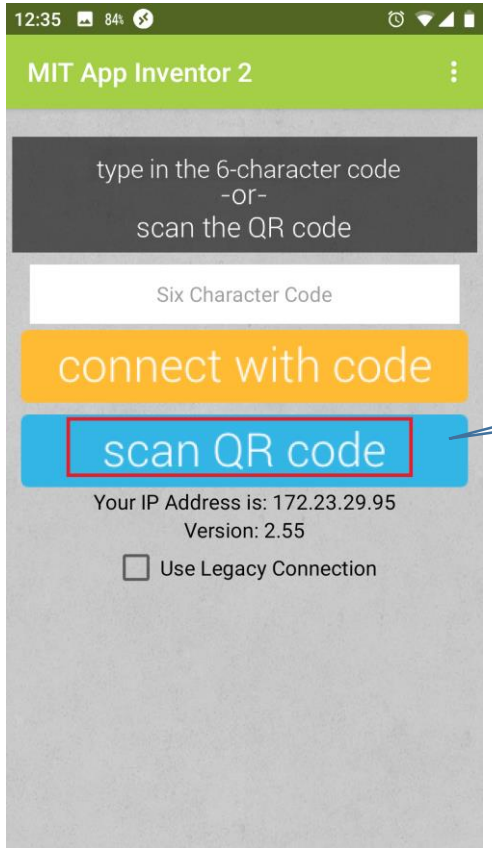
You will be given the QR code to download the .apk file – see next slide.



Step 11: Install the **MIT AI2 Companion** app on your Android phone.

App Inventor – a quick intro (cont.)



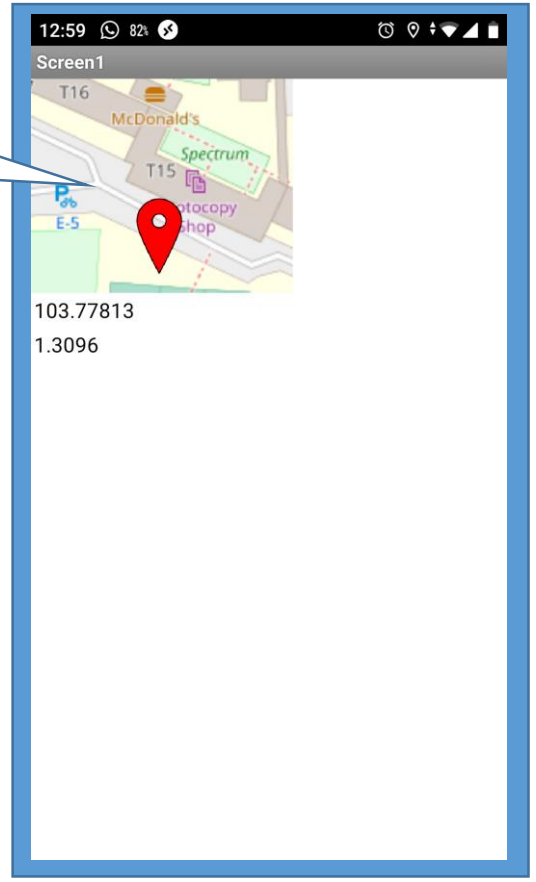


Step 12: Use the MIT AI2 Companion app to scan the **QR code** generated to download the .apk file.

Step 13: Install the MapMe app you have just created.

App Inventor – a quick intro (cont.)

Step 14: When the app is run, it will show your GPS location.



Congratulations! You have just created an Android app!

App Inventor – a quick intro (cont.)

To summarize, these are the key steps:

1. Browse to https://appinventor.mit.edu
2. Click “Create Apps!”
3. Sign in with a Google account.
4. Click “Start new project”.
5. Enter an appropriate “Project name”.
6. In the Designer view, add the required Components to the Screen(s).
7. Change the Properties of the Components, if necessary.
8. Change to Block view.
9. Program the app, the “jigsaw puzzle” way.
10. Build the app (i.e. “compile”).
11. Install the MIT AI2 Companion app on your (Android) phone.
12. Download the .apk file generated from your app.
13. Install your app on your phone.
14. Test your app on your phone.

The hardest parts are this…

…and this.

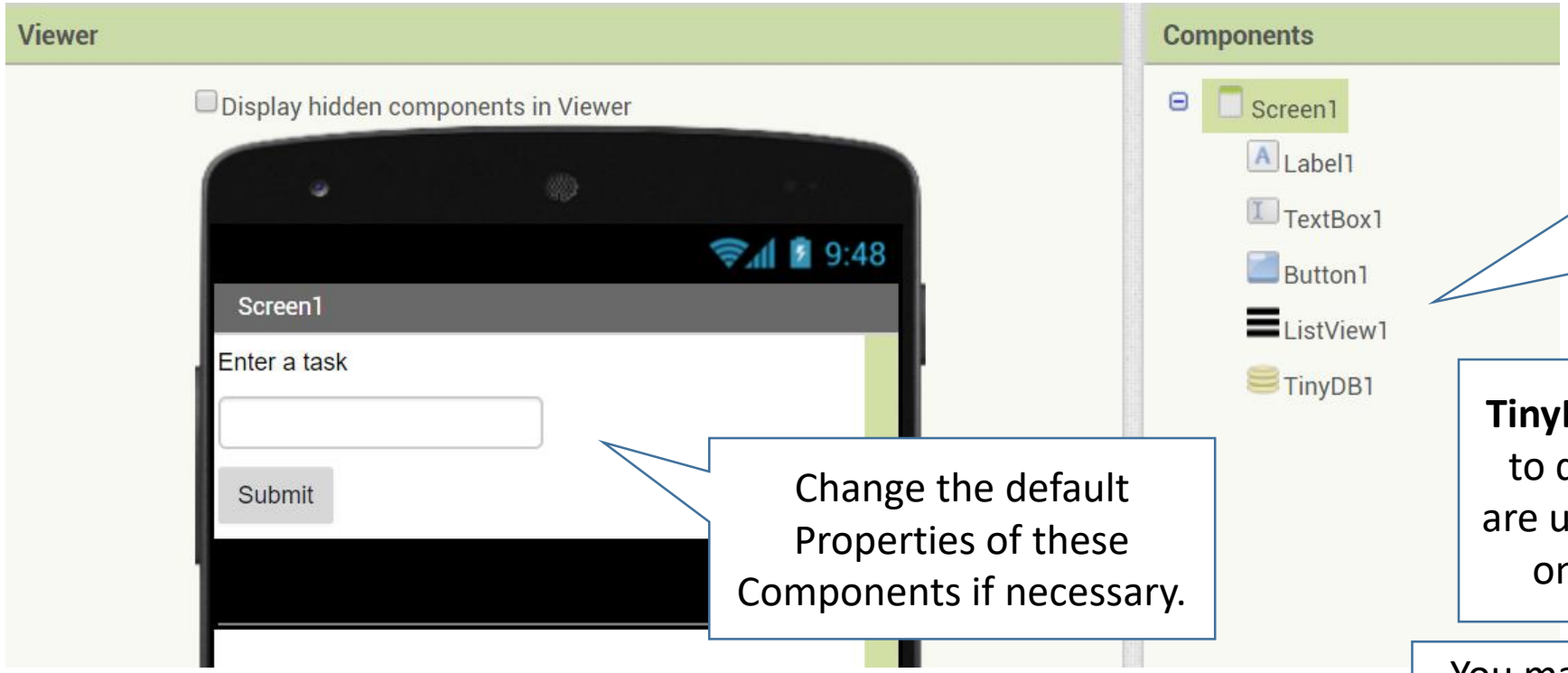
Lab Exercises

Instead of listening to lecture, we will learn by doing. We will create these few apps to learn App Inventor.

- Exercise 6.1 – “To Do List”
- Exercise 6.2 – “Registering/De-registering for an Event”
- Exercise 6.3 – “Register / login as a user” (homework)

Exercise 6.1 – To Do List

- Let's create a simple app to allow a user to **keep a list of "things to do"**.
- The Designer view / user interface looks like this:



The Label, TextBox, Button and ListView are from the "User Interface" Palette, while the TinyDB (non-visible) is from the "Storage" Palette.

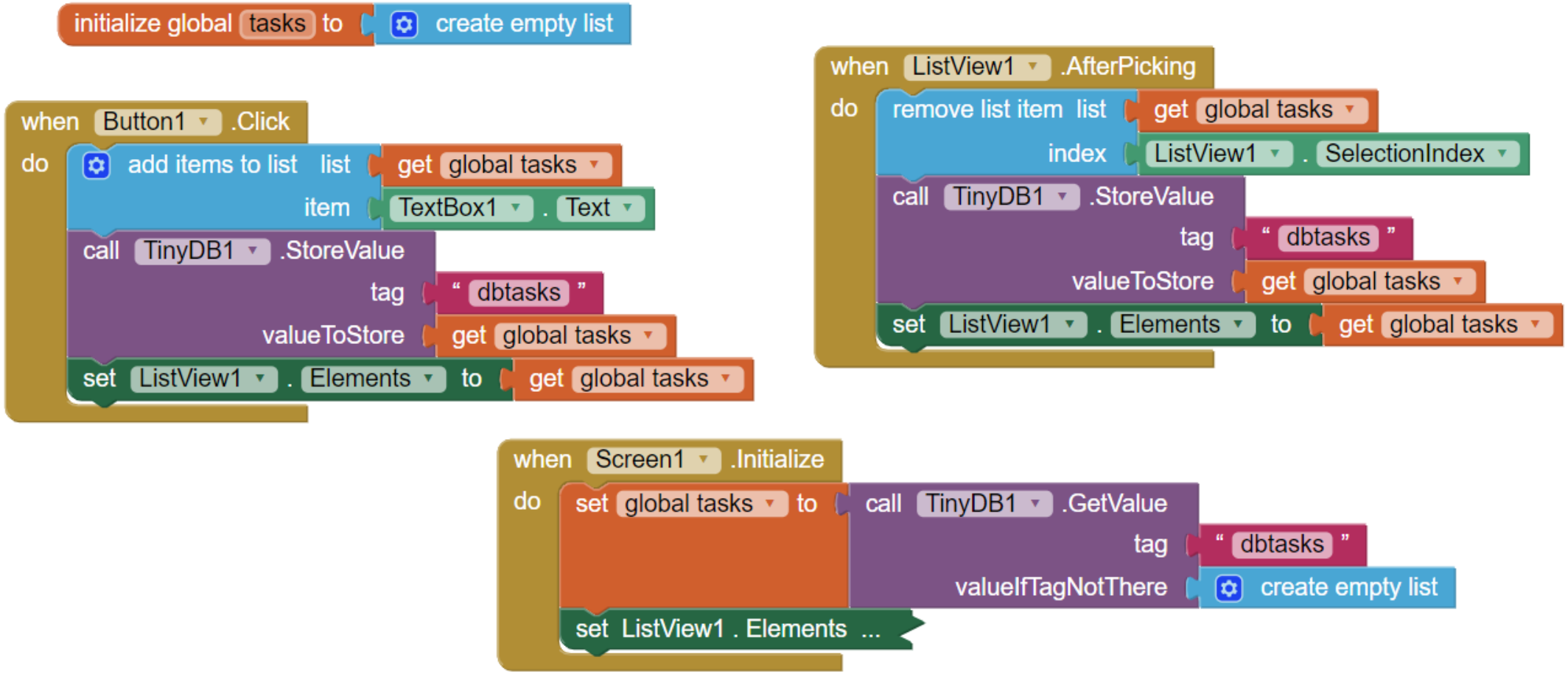
Change the default Properties of these Components if necessary.

TinyDB will keep the list of things to do in the phone. If variables are used, the contents will be lost once the app stops running.

You may want to learn about TinyWebDB & CloudDB **on your own**.

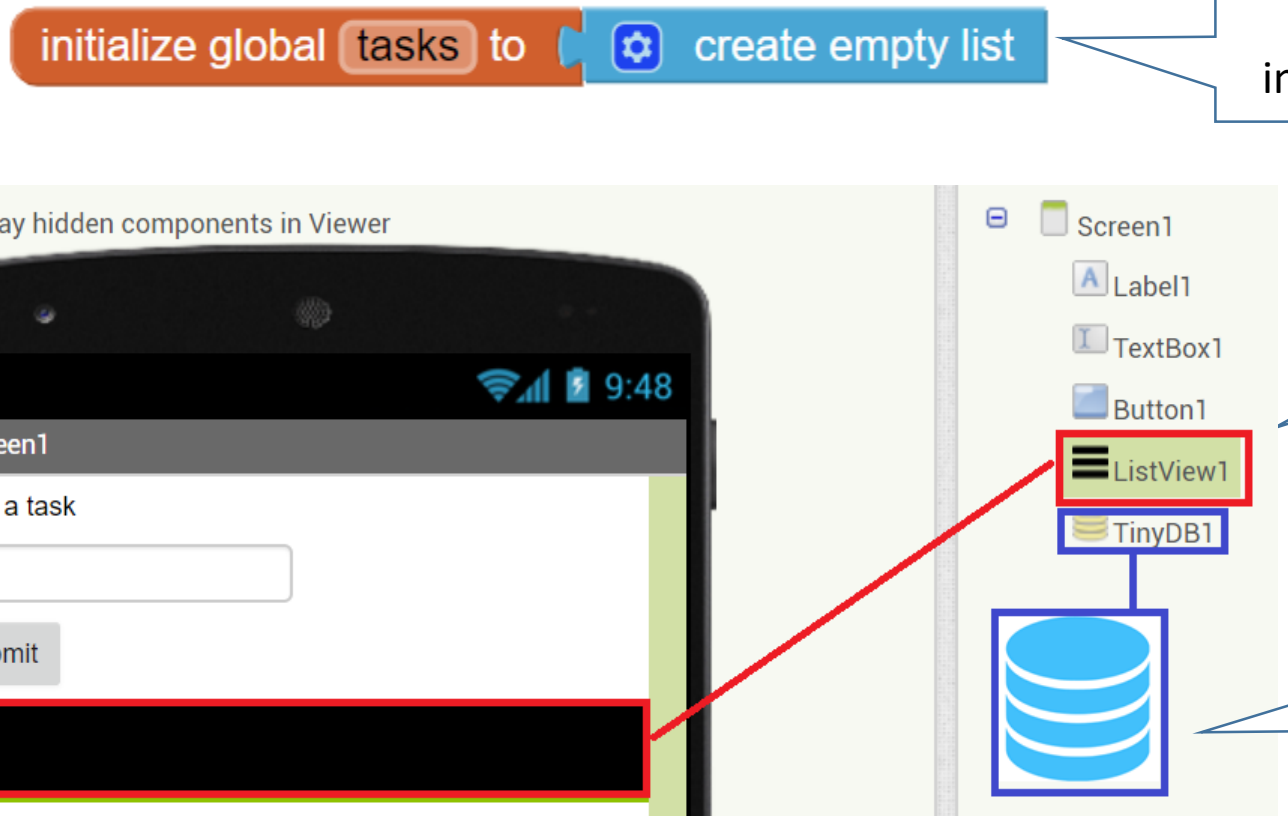
Exercise 6.1 – To Do List (cont.)

- The Block view looks like this. This will take some time to understand. The next few slides may help.



Exercise 6.1 – To Do List (cont.)

- To understand the Block view, first note that the “list of things to do” is “stored” in 3 places:



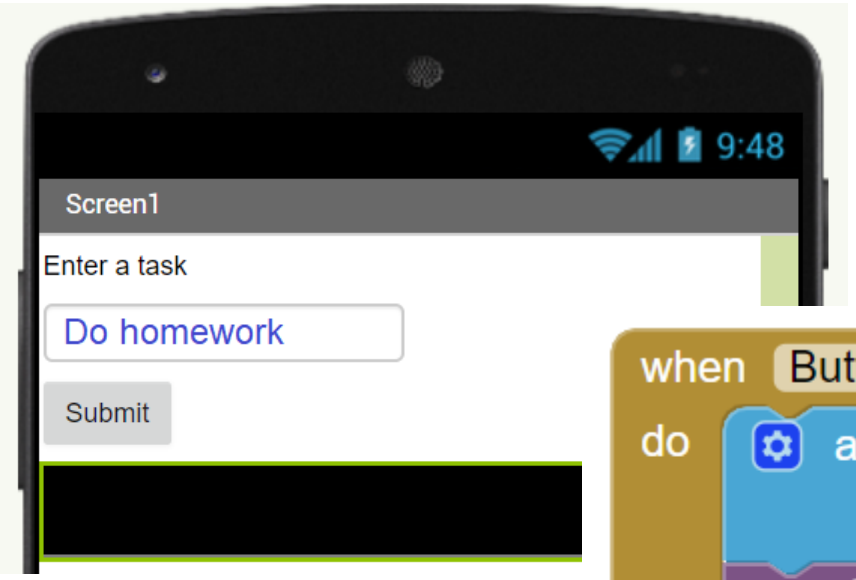
1. A global “list” **variable** called “**tasks**”, which is initialised to an empty list.

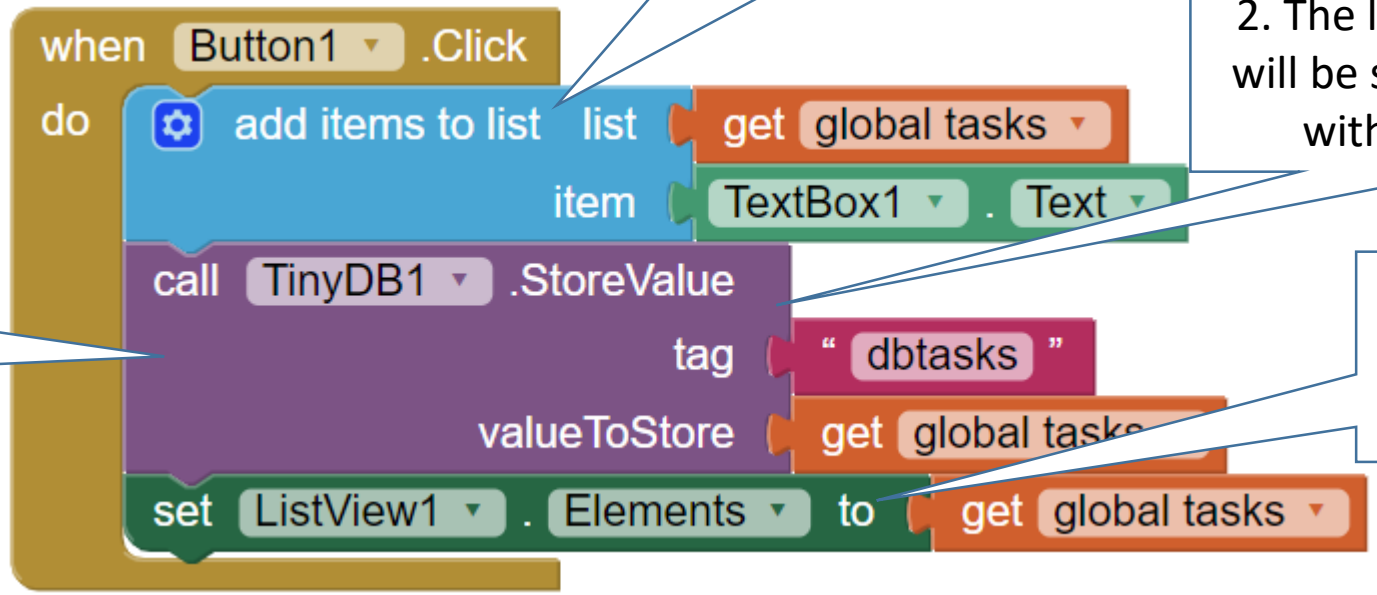
2. A **ListView** which displays the “list of things to do” for the user to see.

3. A **TinyDB** which stores the “list of things to do” in the mobile phone.

Exercise 6.1 – To Do List (cont.)

- What happens when the user **enters a task** e.g. Do homework and click Submit?





1. The item (i.e. the "Text" in the "TextBox1") will be added to the list (i.e. the global variable called "tasks")

2. The list (i.e. global "tasks") will be stored into the TinyDB with the tag "dbtasks"

3. The list (i.e. global "tasks") will be displayed in the ListView

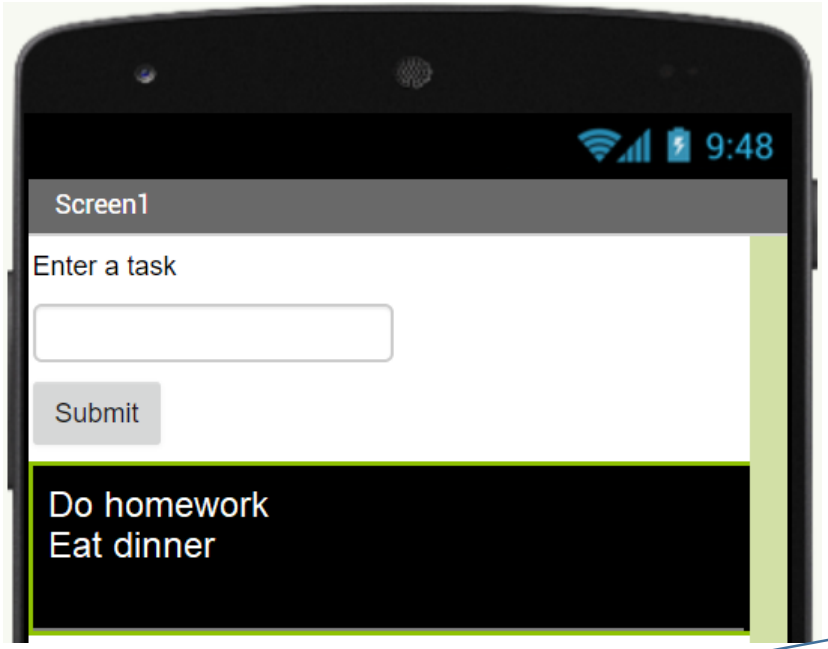
Note that TinyDB stores items as "tag, value" pairs, to allow items to be retrieved by their tags.

Exercise 6.1 – To Do List (cont.)

Index	Item
1	Do homework
2	Eat dinner

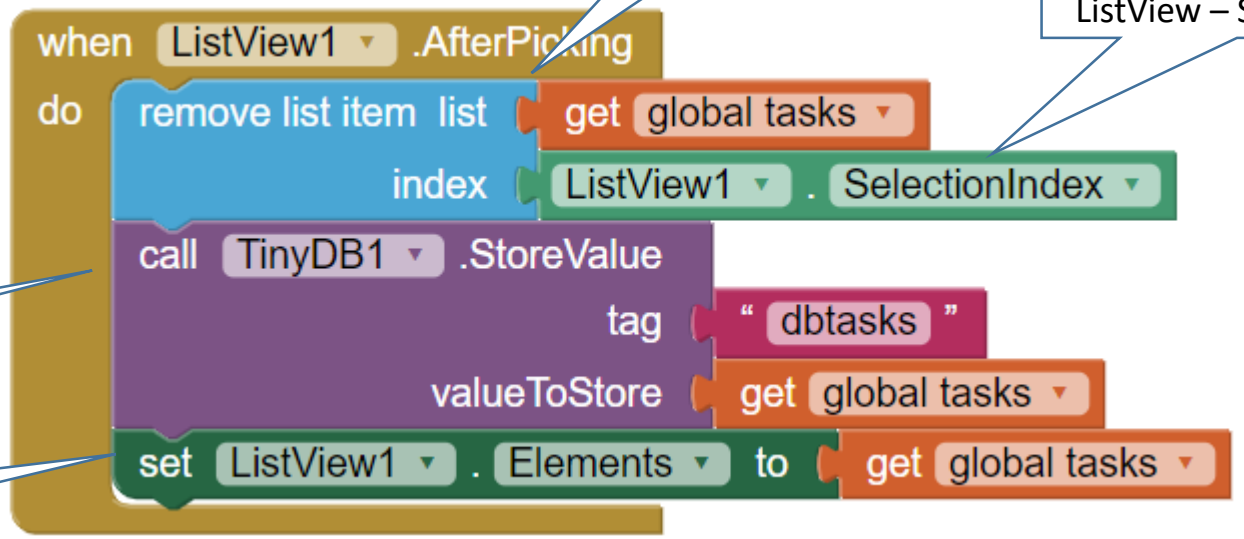
The “tasks” variable before the user clicks.

- What happens when the user **clicks on a completed task** e.g. Do homework?



1. The clicked item will be removed from the list.

The **index** of the item to be removed is provided by ListView – Selection Index.



2. TinyDB is updated.

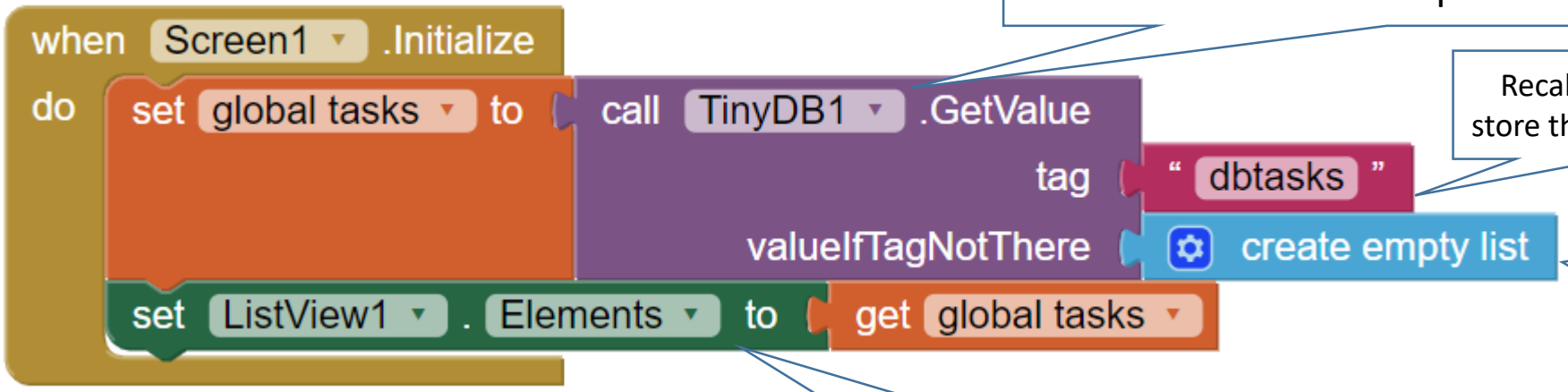
3. Likewise, ListView is updated.

Exercise 6.1 – To Do List (cont.)

- What does this block do (when **Screen1 initializes**…)?

"tasks" variable is initially an empty list.

1. If the user has added items to the list of things to do, TinyDB will have the list and the "tasks" variable will be updated with the list.



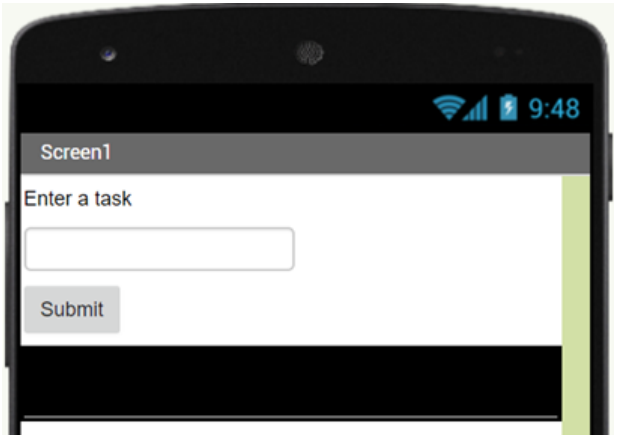
Recall that the tag to store the list is "dbtasks".

If the tag is not there (i.e. the user has not added any item), an empty list will be returned.

2. Likewise, ListView is updated.

Exercise 6.1 – To Do List (cont.)

- Once you have completed the app, try it out on your Android phone.

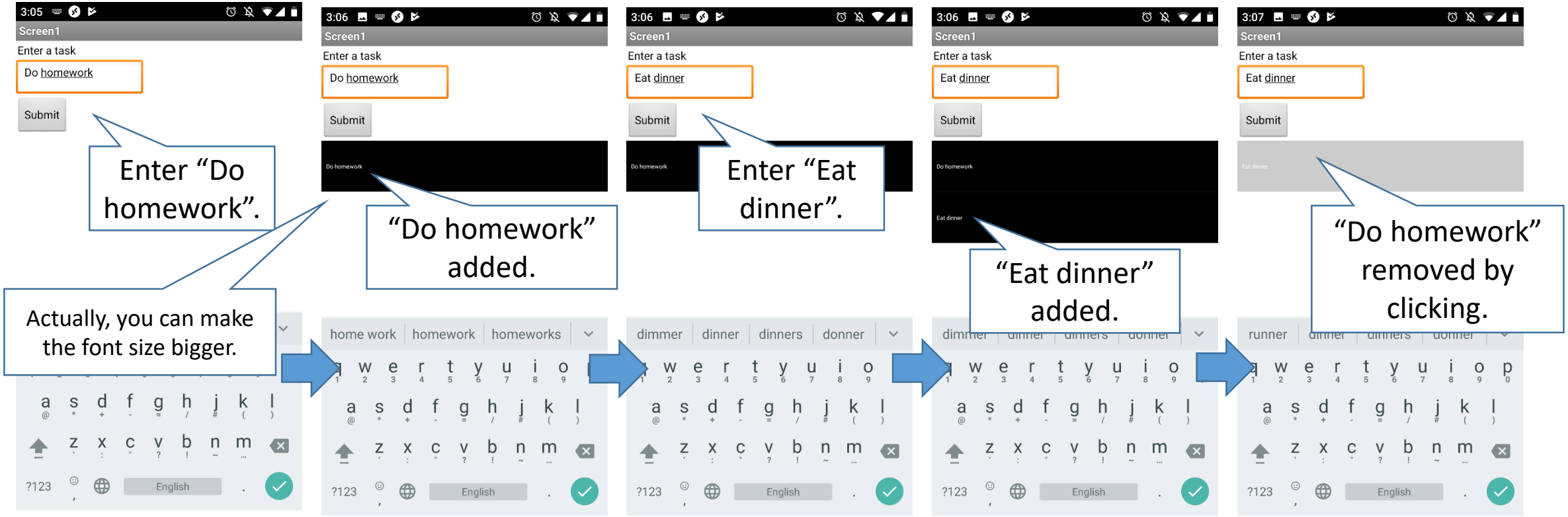



```
initialize global tasks to [create empty list]

when Button1 .Click
do  add items to list  list  get global tasks
                       item  TextBox1 . Text
    call TinyDB1 .StoreValue
                       tag   " dbtasks "
                       valueToStore  get global tasks
    set ListView1 . Elements to  get global tasks

when ListView1 .AfterPicking
do  remove list item  list  get global tasks
                      index  ListView1 . SelectionIndex
    call TinyDB1 .StoreValue
                       tag   " dbtasks "
                       valueToStore  get global tasks
    set ListView1 . Elements to  get global tasks

when Screen1 .Initialize
do  set global tasks to  call TinyDB1 .GetValue
                                 tag   " dbtasks "
                                 valueIfTagNotThere  [create empty list]
    set ListView1 . Elements ...
```

It is important to learn the concepts well, as we will be using these in the next app.

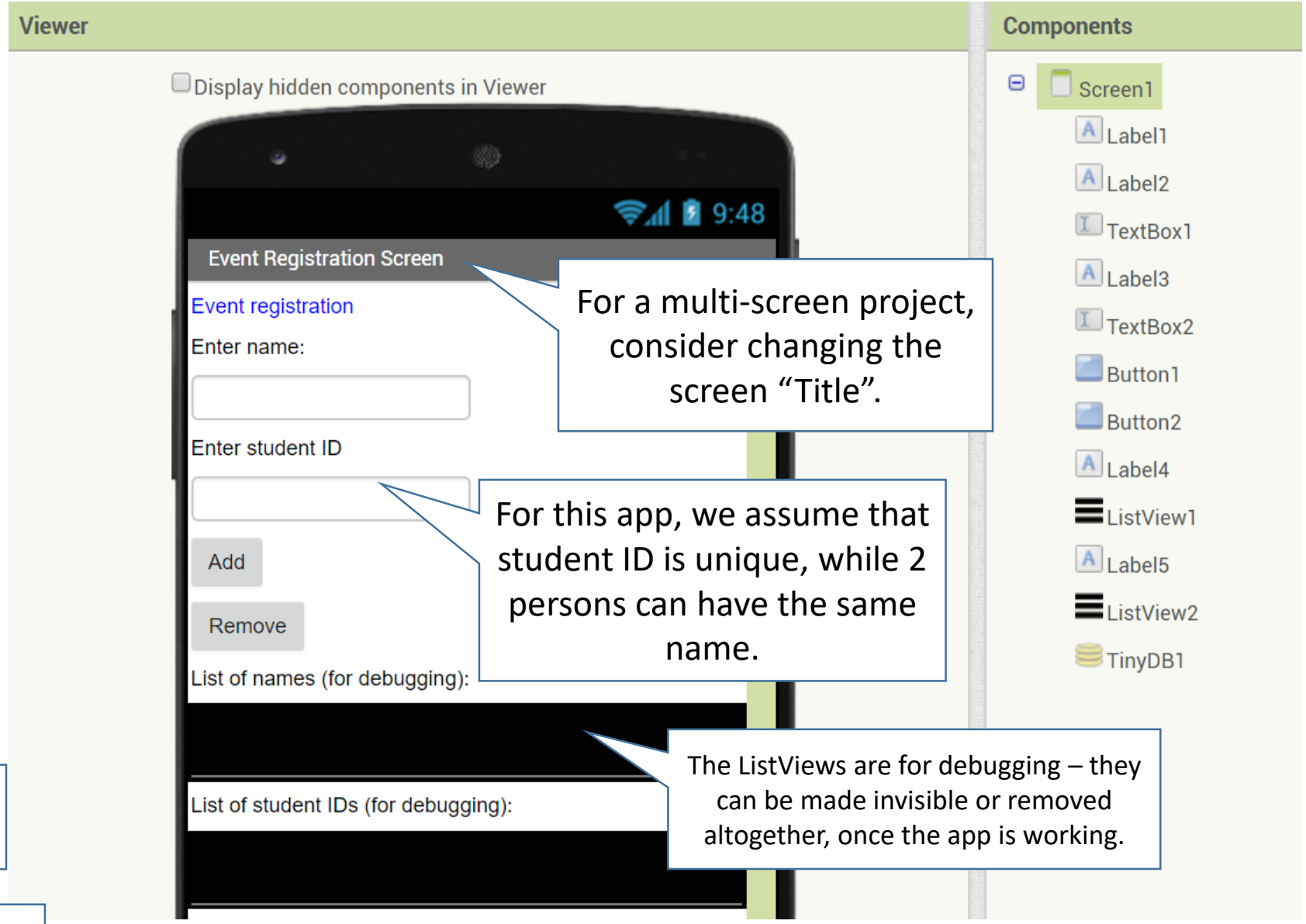
Exercise 6.1 – To Do List (cont.)

- **Sample run**:



Exercise 6.2 – Registering / De-registering for an Event

- Let's modify the "To Do List" app to allow a student to **register / de-register** for an **event**.
- The Designer view / user interface looks like this:



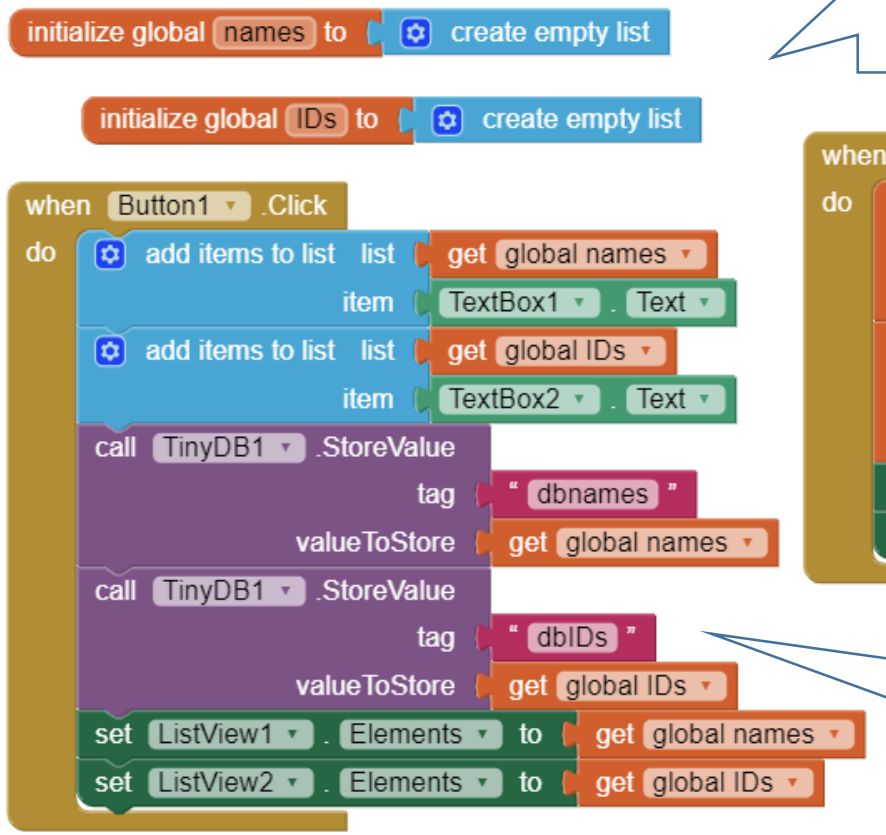
Can you see how a user can register / de-register for an event?

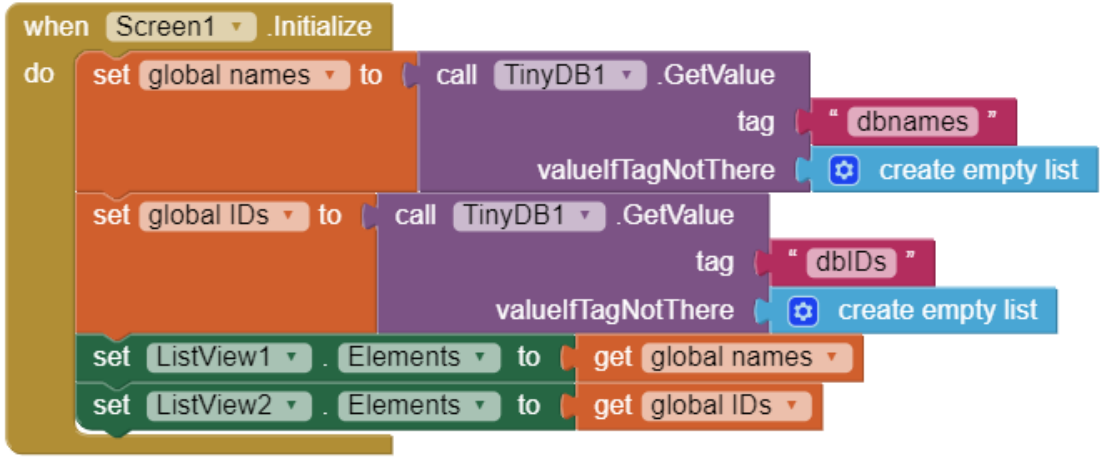
How can the "To Do List" app be modified to become this?

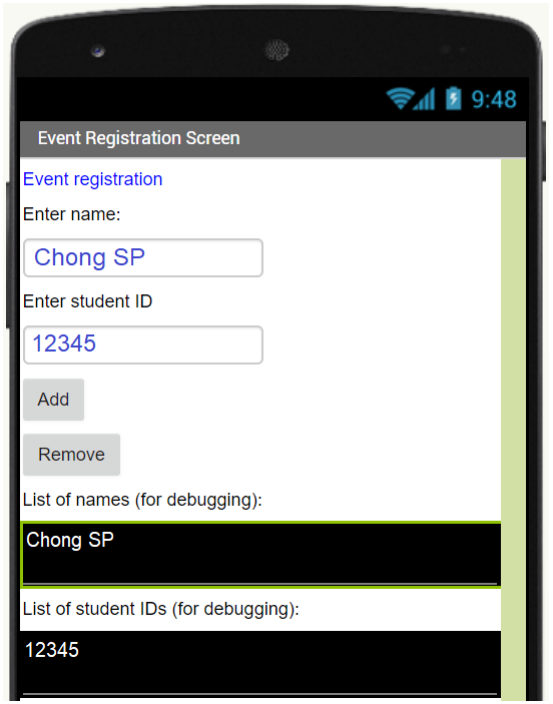
Exercise 6.2 – Registering / De-registering for an Event (cont.)

- First, the obvious. What happens when a user **add his name and student ID** to the event?

This is like adding an item to the list of things to do in the previous app. So you should be able to figure out the blocks of code below.







Note that the list of names is stored with the tag “dbnames”, while the list of student ID’s is stored with the tag “dbIDs”.

Exercise 6.2 – Registering / De-registering for an Event (cont.)

- Next, what happens when a user **remove his name and student ID** from the event?

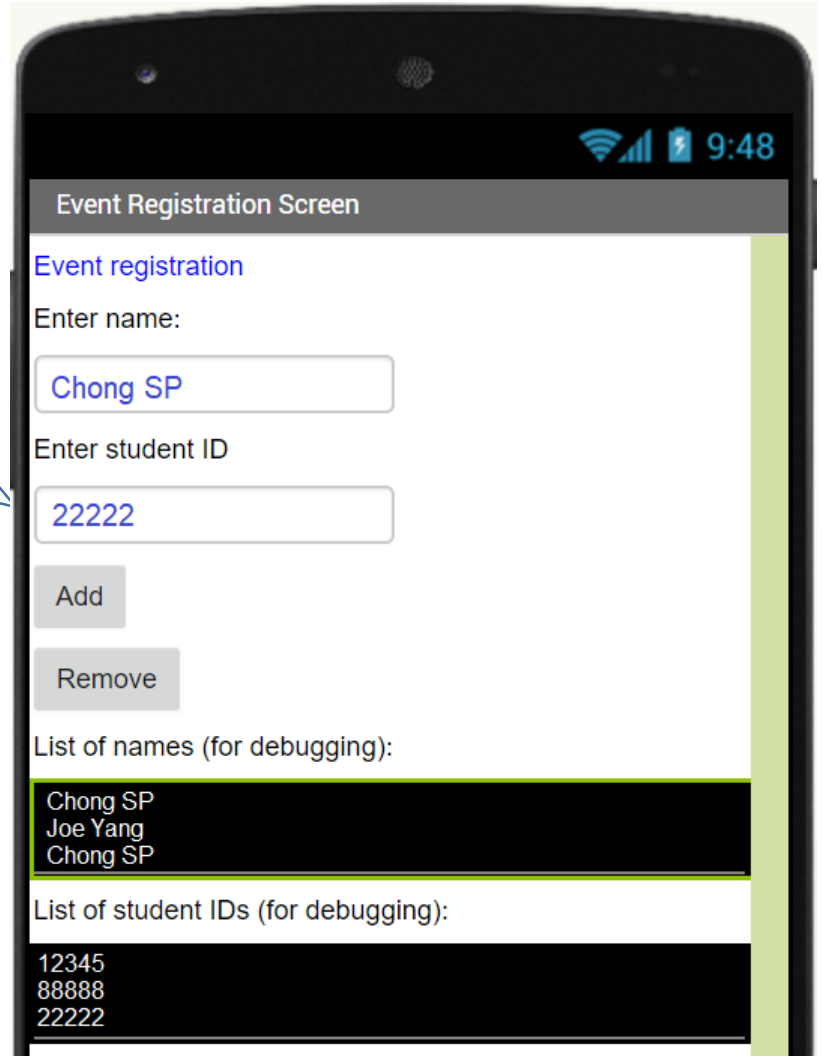
Let's look at a slightly tricky situation: There are 2 students with the same name. Of course, their student ID's will be different.

index	name	ID
1	Chong SP	12345
2	Joe Yang	88888
~~3~~	~~Chong SP~~	~~22222~~

So, when Chong SP with the ID 22222 de-registers for an event, it is the 3rd entry that should be removed.

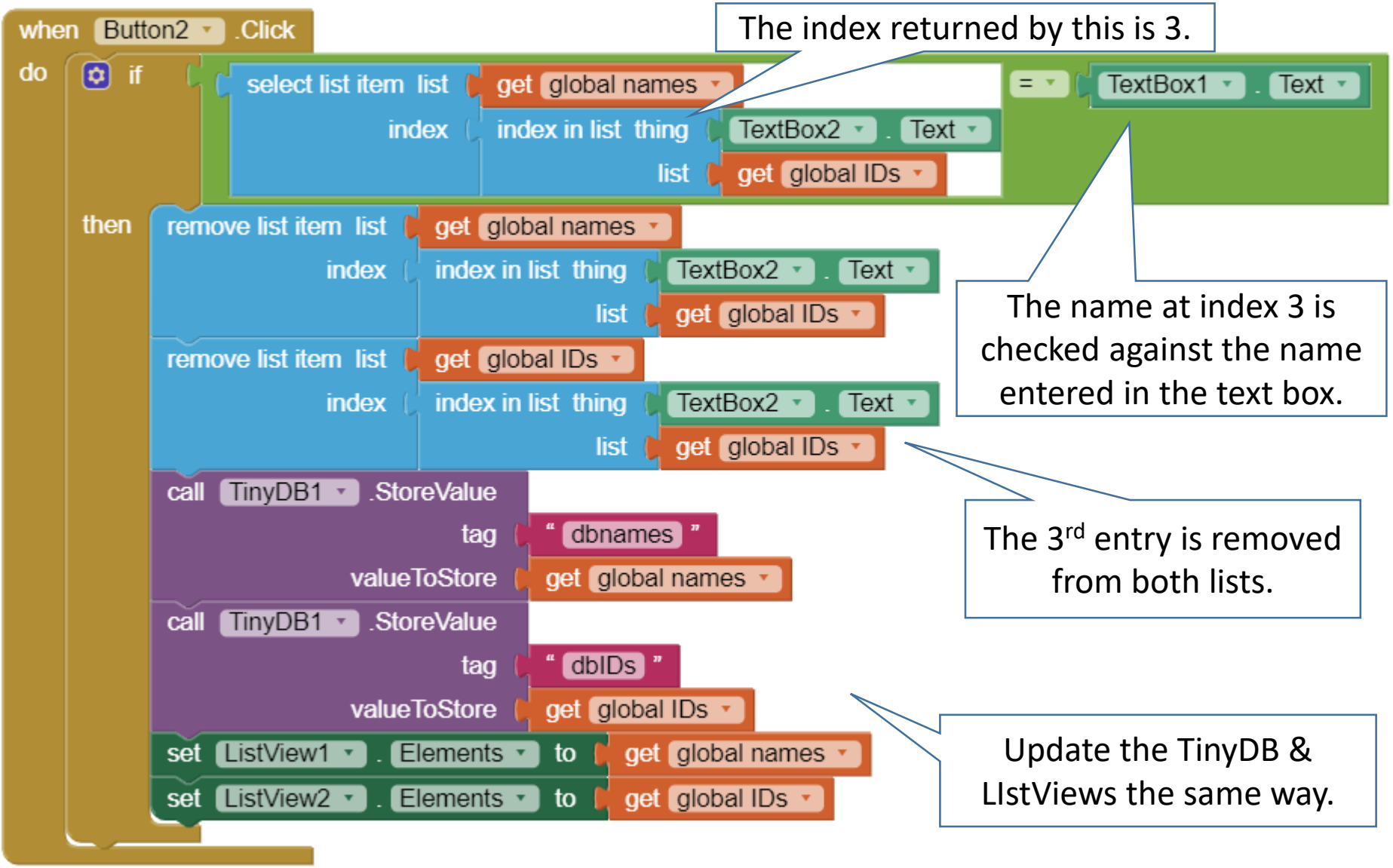
The app should first search for the entry whose ID matches that in the text box. In this case, a match is found at index = 3.

Once the entry is found, both the name & ID with the index 3 will be removed.



Exercise 6.2 – Registering / De-registering for an Event (cont.)

- The code block to search for an entry and to remove that entry is given below:



The index returned by this is 3.

The name at index 3 is checked against the name entered in the text box.

The 3rd entry is removed from both lists.

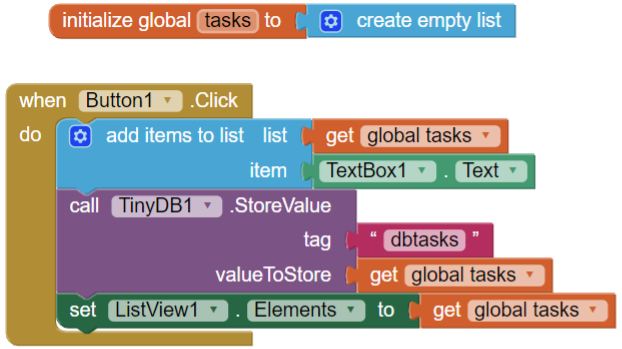
Update the TinyDB & LIstViews the same way.

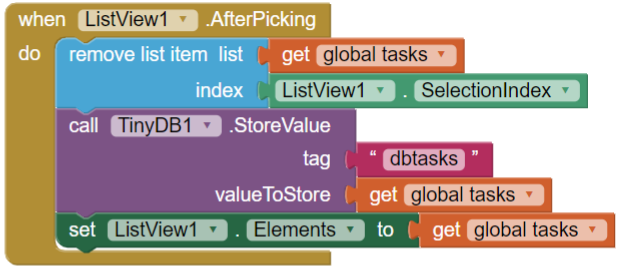
Do you know what's wrong with this?

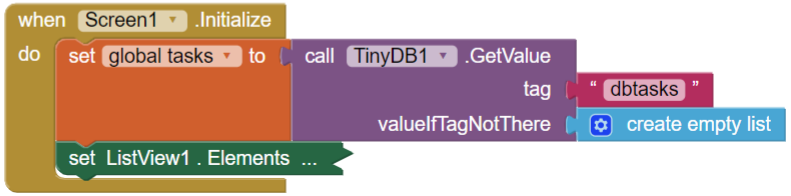
ndex	name	ID
	Chong SP	12345
	Joe Yang	88888
	Chong SP	22222

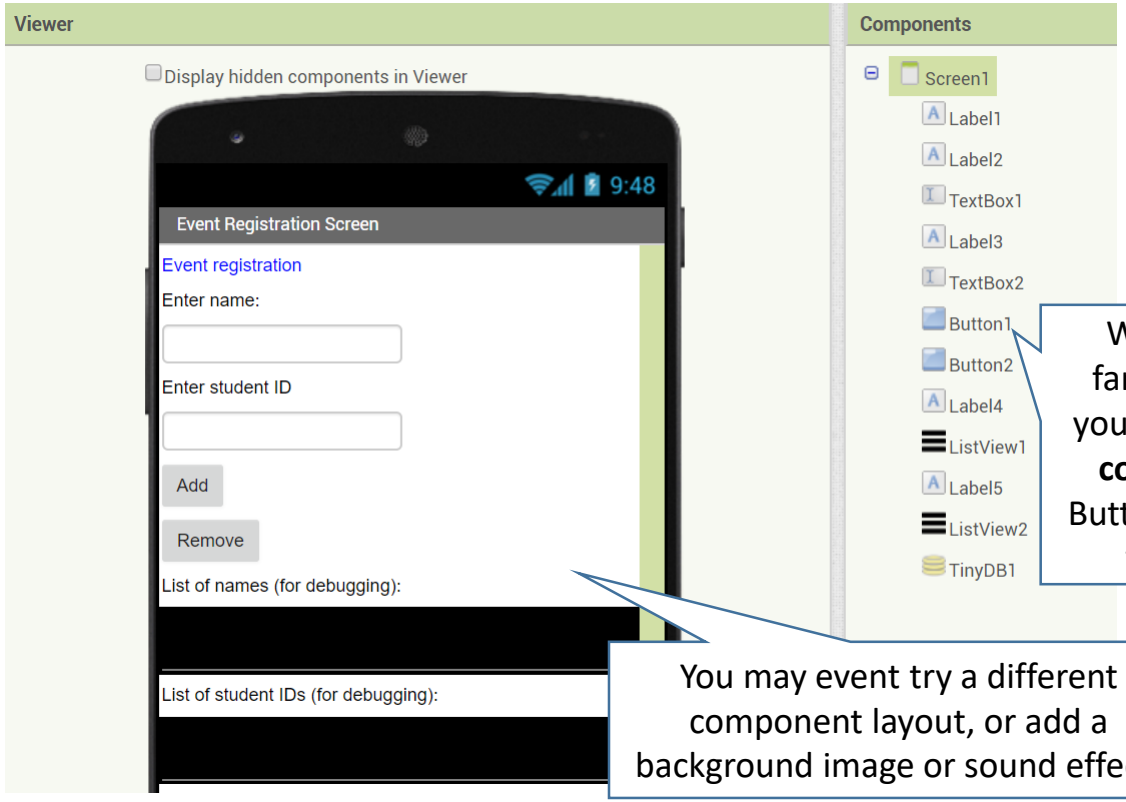
Exercise 6.2 – Registering / De-registering for an Event (cont.)

- Once your app is ready, try it out on your Android phone.



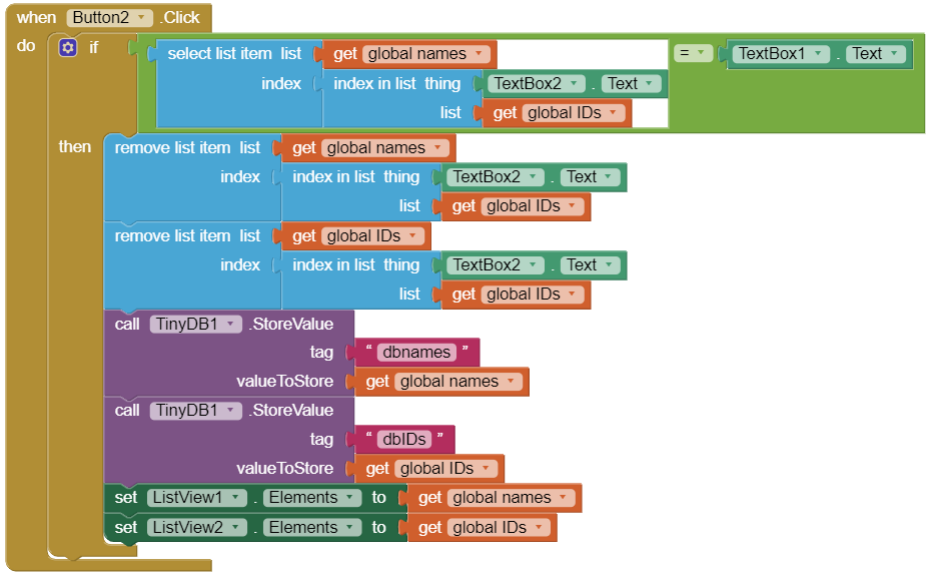






When you become more familiar with App Inventor, you may want to **rename the components** (for instance Button 1 as Add_Btn), to ease the app development.

You may event try a different component layout, or add a background image or sound effect.



Exercise 6.2 – Registering / De-registering for an Event (cont.)

- Once you have created the app, spend a few minutes to discuss (with another student) what problems users may encounter using this app.

It is always a good practice to reflect on what you have created – so that the user will have a good experience using your app..

There is NO checking to see if a student is ALLOWED to register for an event. To do that, a database will be needed.

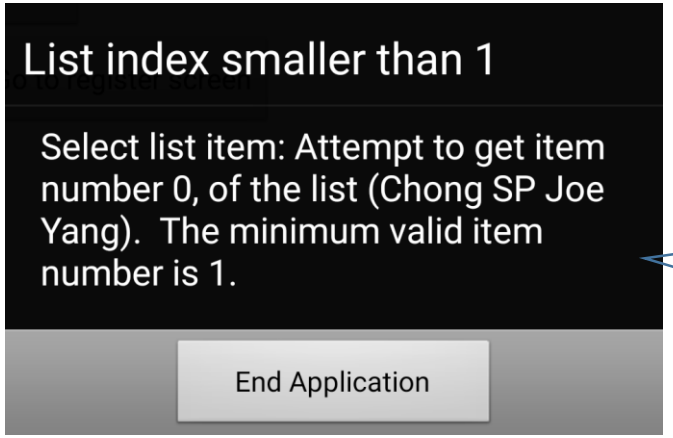
It is assumed that:

- a student enters his ID correctly,
- he only registers once, and that
- no one maliciously registers / de-registers for another person when he is not supposed to do so.

It is possible to add in code to prevent double entries.

The app uses a local database, so all students must go to the Android device (maybe a tablet) to register.

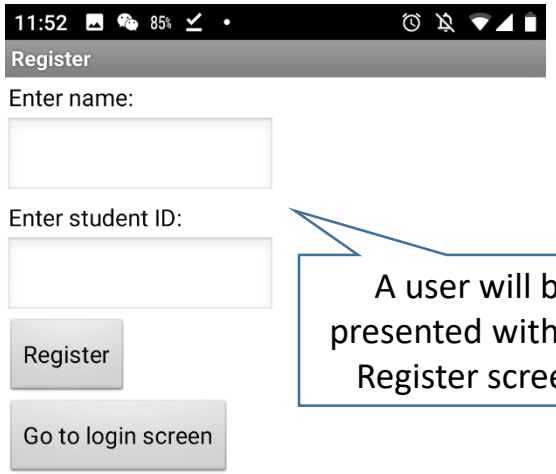
index	name	ID
1	Chong SP	12345
2	Joe Yang	88888
3	Chong SP	22222



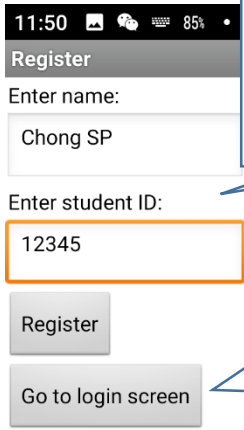
When a user de-registers using an "unregistered" student ID, it cannot be found in the list and "index in list" returns 0, and "select list item" gives this error message. We will learn to prevent this in Ex 6.3.

Exercise 6.3 – Register / login as a user (homework)

- As homework, modify the previous app to allow a user to register and login to use a system. Some screen-shots are provided as a guide.

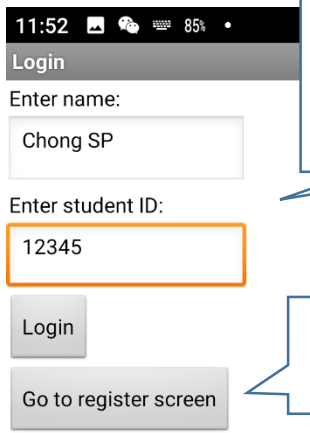


A user will be presented with this Register screen.



A user can register by entering his name and ID and clicking Register.

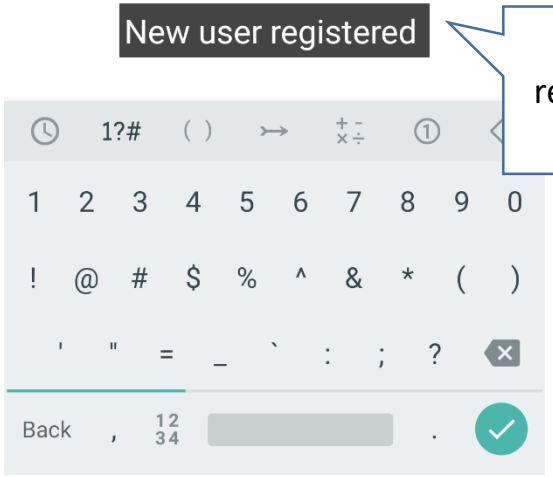
He can also switch to Login screen.



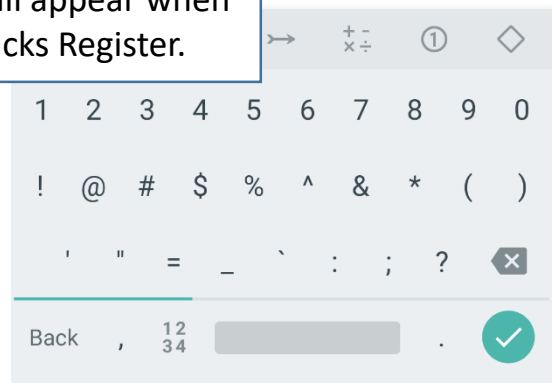
A user can login by entering his name and ID and clicking Login.

He can also switch to Register screen.

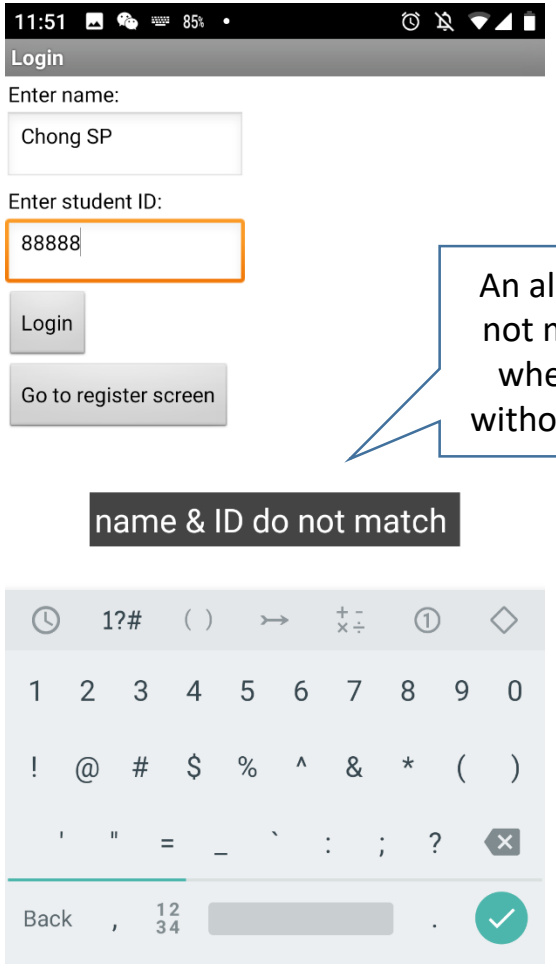
There are 3 screens in this app: Register (Screen1), Login and Login_success (not shown here).



An alert "New user registered" will appear when the user clicks Register.

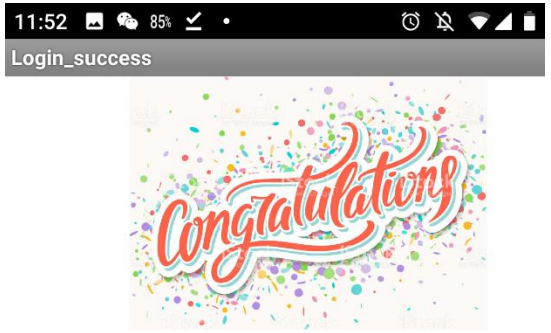


Exercise 6.3 – Register / login as a user (homework) (cont.)



An alert "name & ID do not match" will appear when the user logins without the correct info.

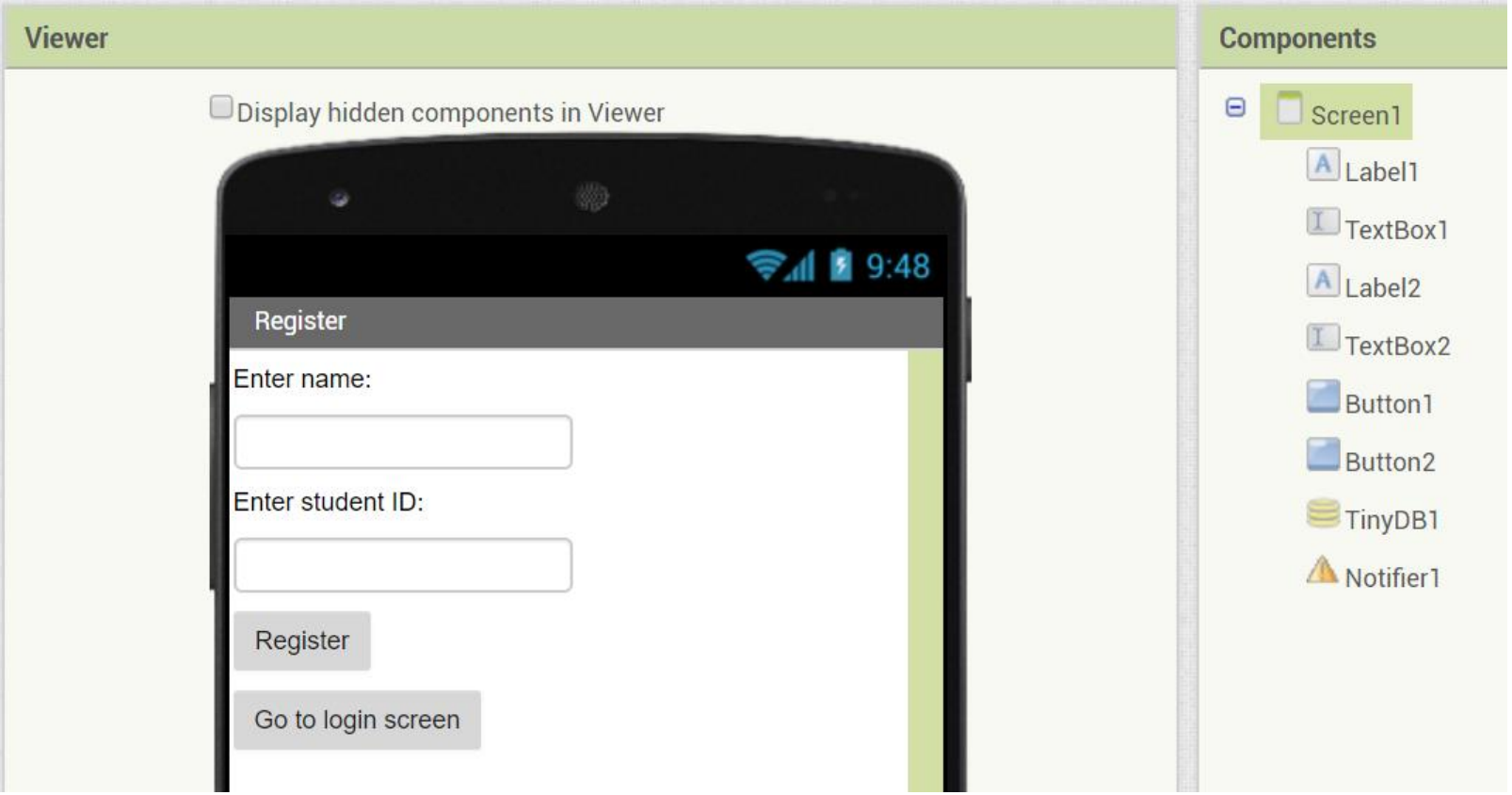
Successful login



If name & ID match a registered user in the database, there is a successful login, and the Login_success screen will be shown.

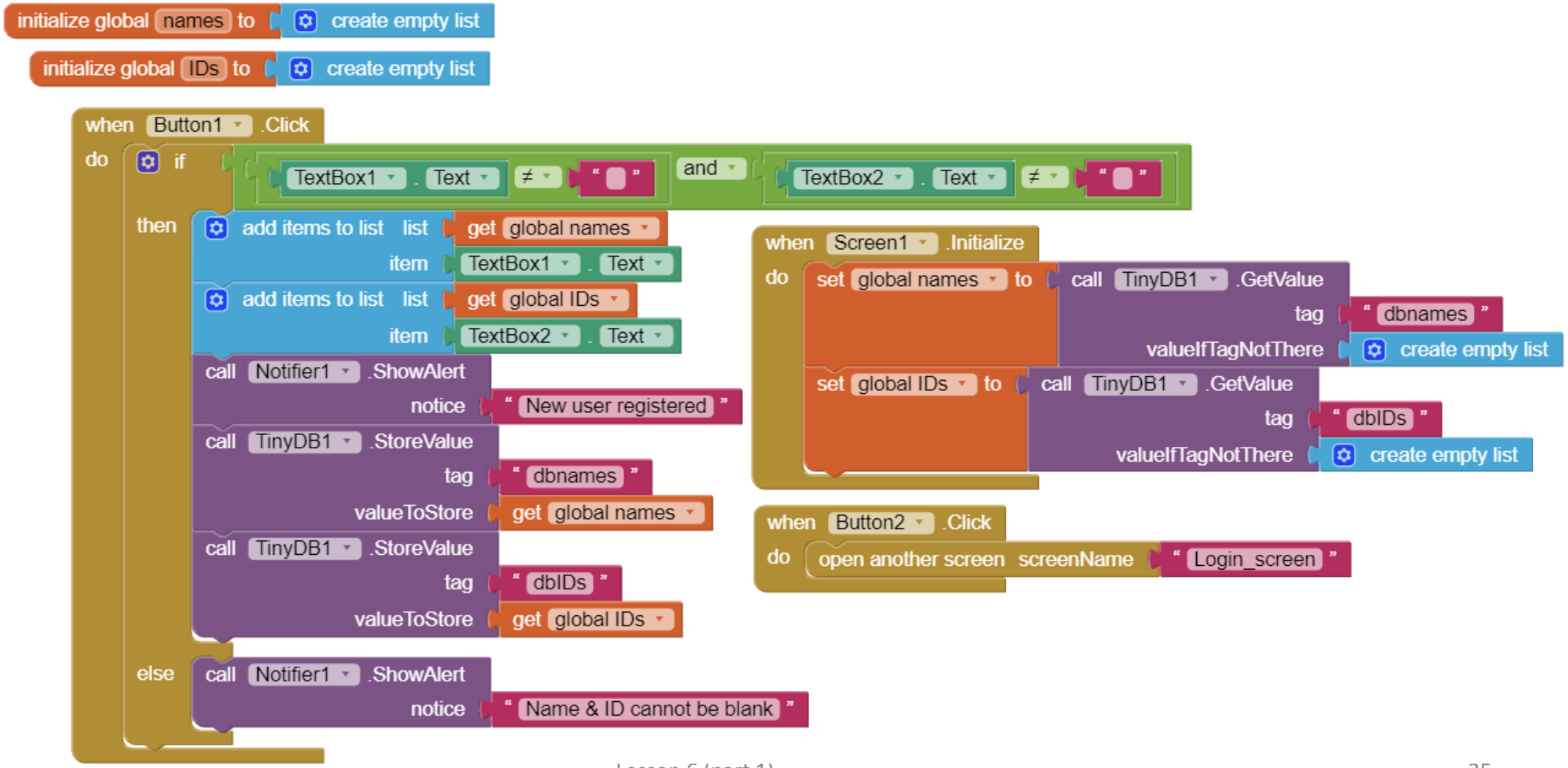
Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen1 (Register screen) – Designer view:



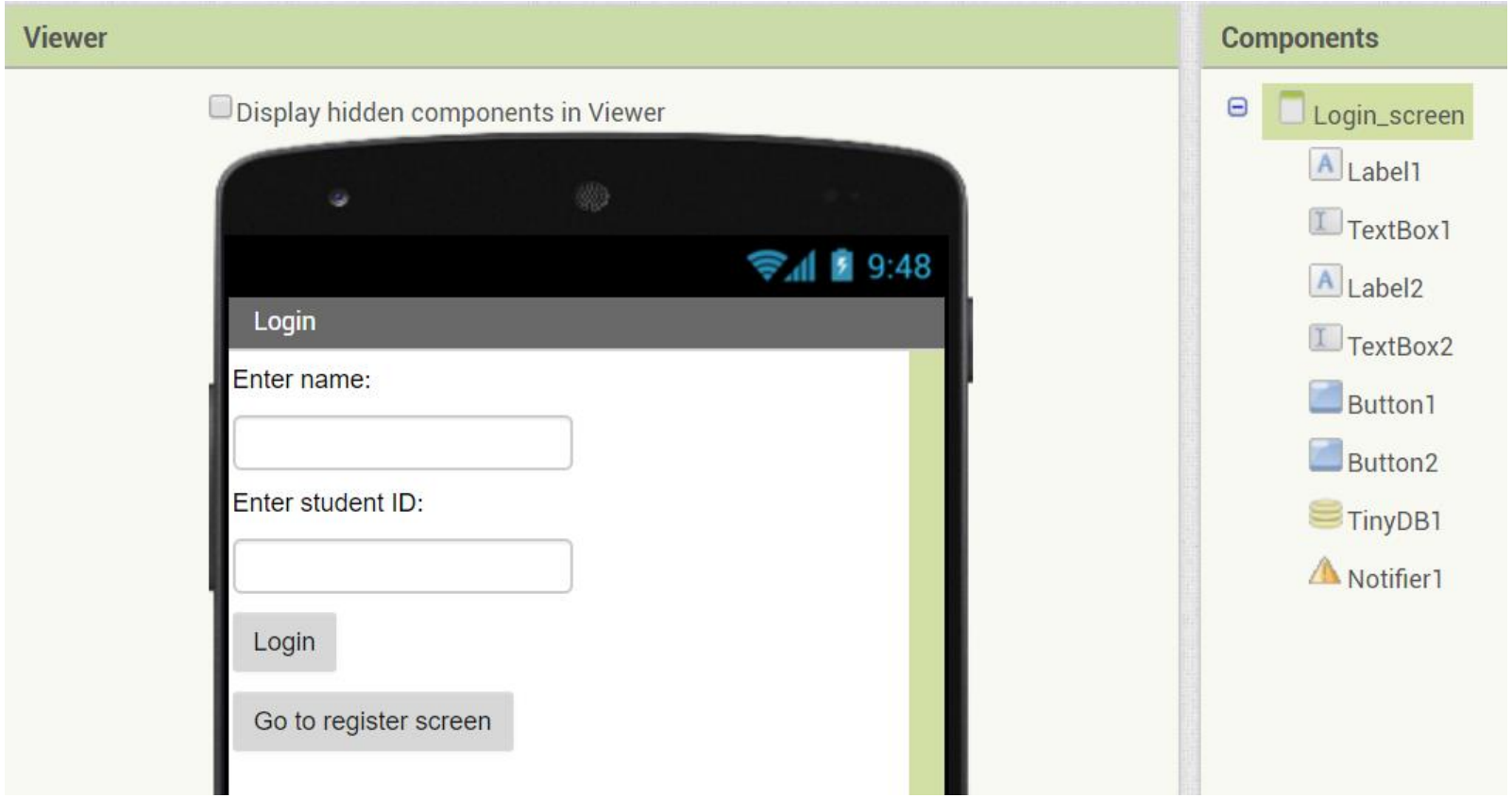
Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen1 (Register screen) – Blocks view:



Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen2 (Login screen) – Designer view:

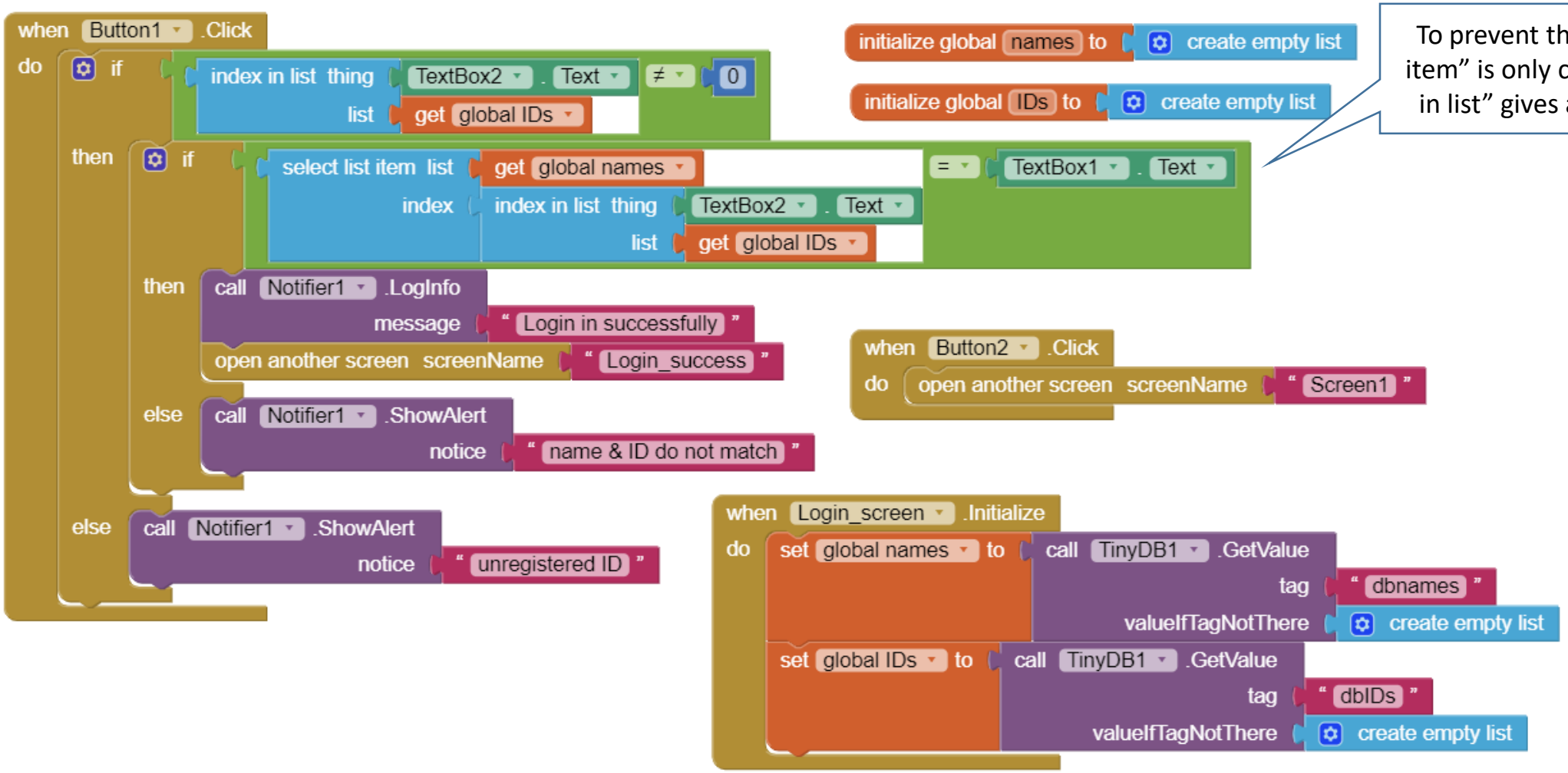


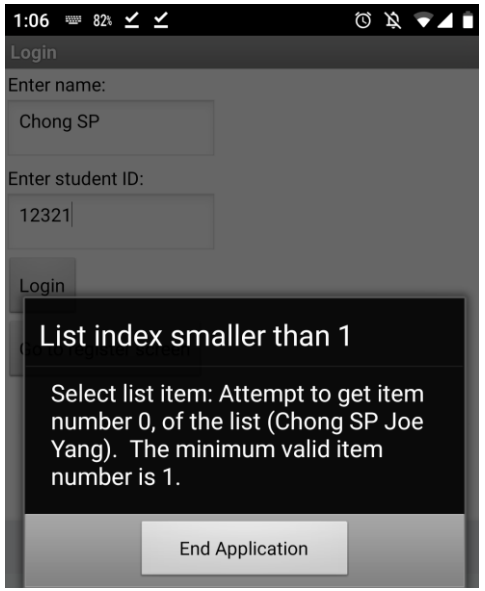
Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen2 (Login screen) – Blocks view:

When a user logins using an "unregistered" student ID, it cannot be found in the list and "index in list" returns 0, and "select list item" gives this error message.

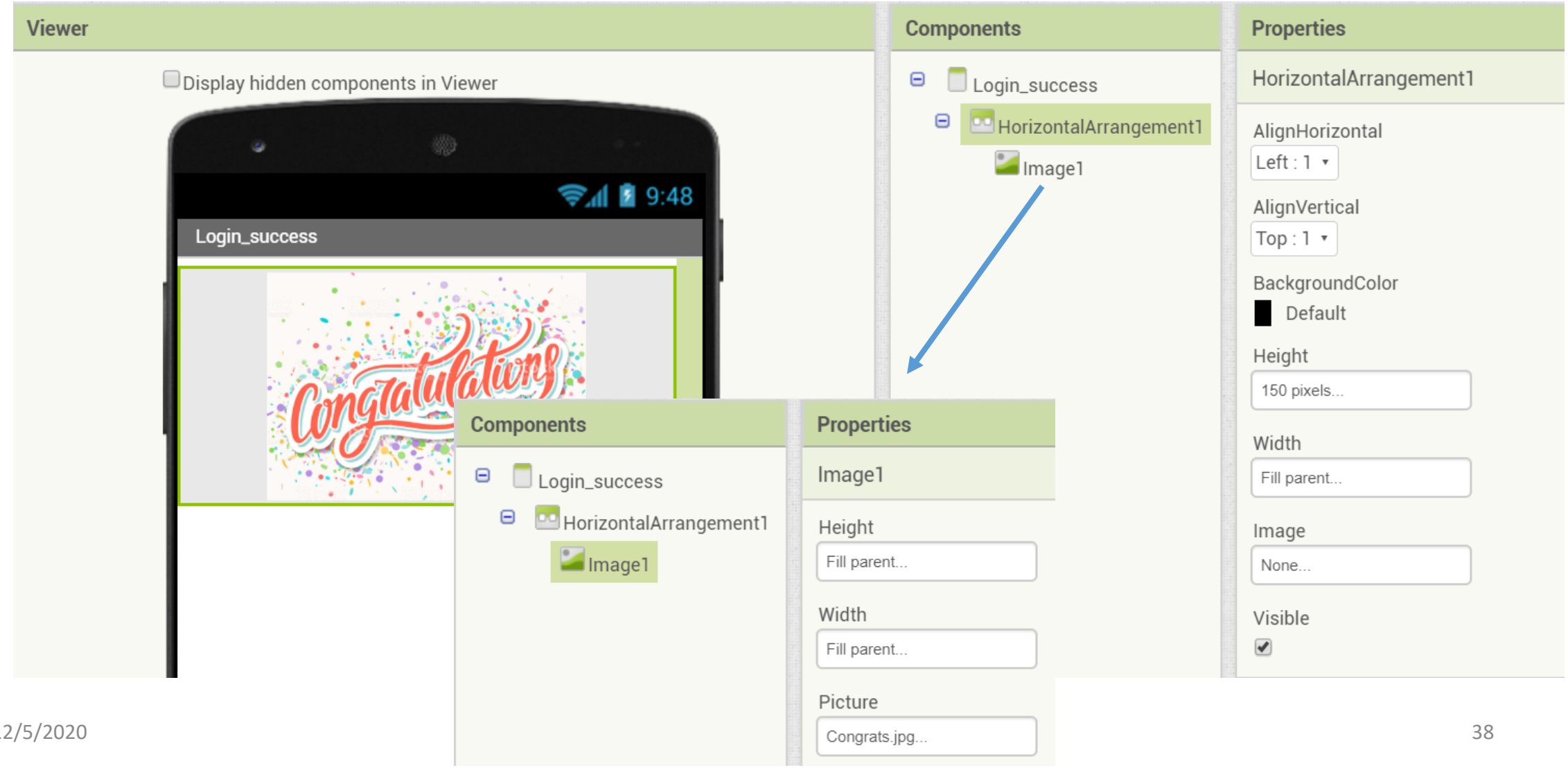
To prevent this, the "select list item" is only called when "index in list" gives a non-zero value.





Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen3 (Login_success screen) – Designer view (no Blocks view for this screen):




THANK
YOU