

Lesson 6 – Mobile app development (part 1)

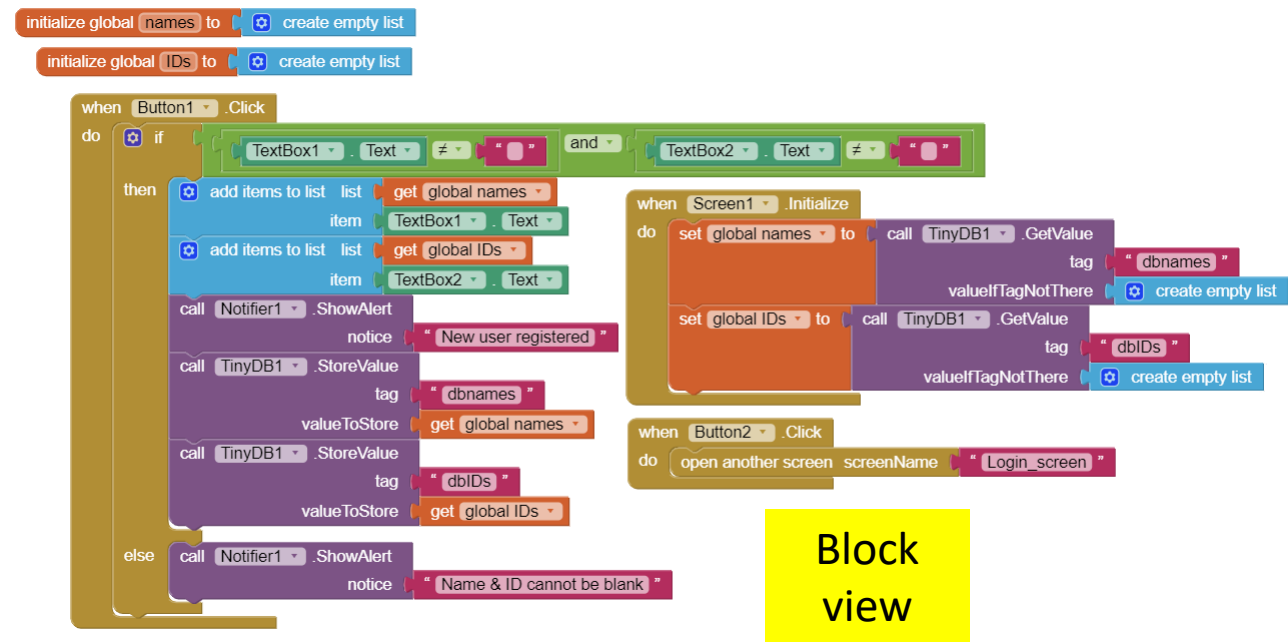
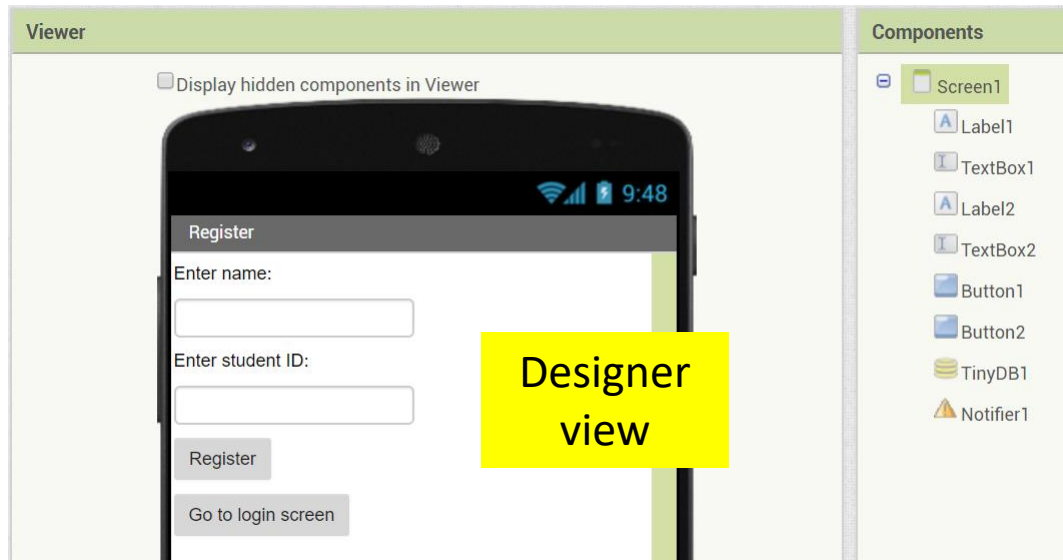
- S.P. Chong

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:

The image is a screenshot of the MIT App Inventor website. At the top, the browser address bar shows the URL <https://appinventor.mit.edu>. A callout box points to this URL with the text: "Step 1: Using a browser on your PC, visit <https://appinventor.mit.edu>". Below the browser, the website header features the MIT App Inventor logo, a "Create Apps!" button, and navigation links for "About", "Educators", "News", "Resources", "Blogs", "Donate", and a "Google Custom Search" bar. A large hero image shows two students working on a laptop. Overlaid on this image is the text "With MIT App Inventor, anyone can build apps with global impact" and a "Learn More" button. A second callout box points to the "Create Apps!" button with the text: "Step 2: Click 'Create Apps!'". At the bottom of the hero image, there are four labels: "Active Users", "Active Users", "Active Users", and "Registered".

App Inventor – a quick intro (cont.)

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 3: Sign in with a Google account.

Sign in with Google

Sign in
to continue to [mit.edu](#)

Email or phone
[redacted]@gmail.com

[Forgot email?](#)

To continue, Google will share your name, email address, language preference, and profile picture with mit.edu.

[Create account](#) [Next](#)

MIT App Inventor | Explore MIT / X MIT App Inventor X

← → ↺ Not secure | ai2.appinventor.mit.edu/#6256922384269312

MIT APP INVENTOR Projects Connect Build Settings Help My Projects Gallery Guide Report an Issue English [redacted]@gmail.com

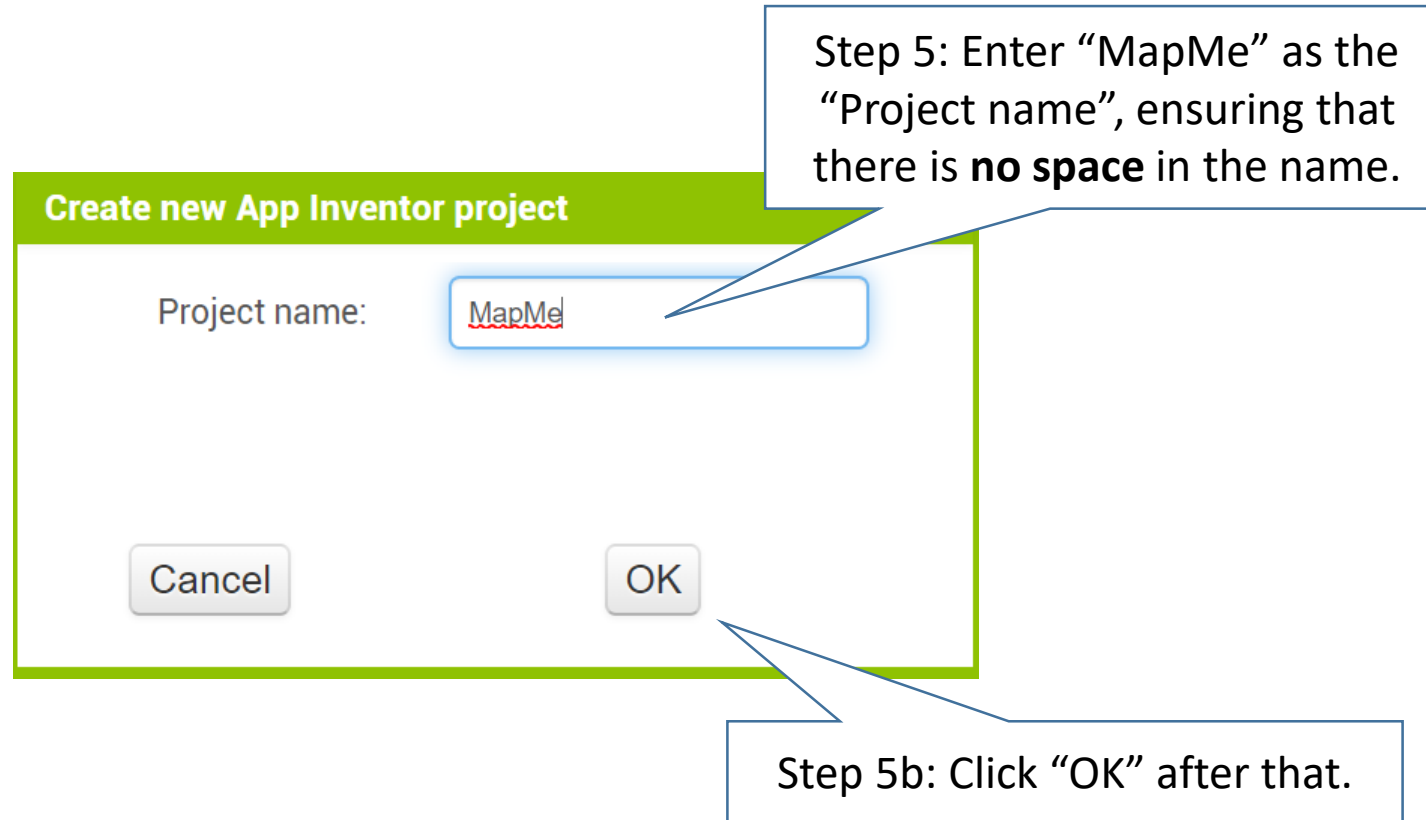
Start new project Delete Project Publish to Gallery

My Projects

Step 4: Click "Start new project".

		Date Modified ▼	Published
		Sep 27, 2019, 12:45:35 PM	No
<input type="checkbox"/>	RegisterUser	Sep 27, 2019, 10:31:43 AM	No
<input type="checkbox"/>	ToDoList	Sep 26, 2019, 3:51:50 PM	No
<input type="checkbox"/>	LoginPage	Sep 26, 2019, 11:27:39 AM	No

App Inventor – a quick intro (cont.)



The screenshot shows a dialog box titled "Create new App Inventor project" with a green header bar. Inside the dialog, there is a label "Project name:" followed by a text input field containing the text "MapMe". The input field is highlighted with a blue border. Below the input field are two buttons: "Cancel" on the left and "OK" on the right. Two callout boxes with blue borders and arrows provide instructions. The first callout box points to the input field and contains the text: "Step 5: Enter 'MapMe' as the 'Project name', ensuring that there is **no space** in the name." The second callout box points to the "OK" button and contains the text: "Step 5b: Click 'OK' after that."

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.)

The screenshot shows the MIT App Inventor web interface. At the top, there's a header with the MIT App Inventor logo and navigation links like 'Projects', 'Connect', and 'Build'. Below this is a green bar with the project name 'MapMe' and buttons for 'Screen1', 'Add Screen...', and 'Remove Screen...'. The main workspace is divided into four panes: 'Palette' on the left, 'Viewer' in the center, 'Components' on the right, and 'Properties' on the far right. The 'Palette' pane shows a list of components under the 'User Interface' category, including Button, CheckBox, DatePicker, Image, Label, ListPicker, ListView, Notifier, PasswordTextBox, Slider, Spinner, and Switch. The 'Viewer' pane shows a mobile phone screen with a map. The 'Components' pane shows a list of components, with 'Screen1' selected. The 'Properties' pane shows the default properties for 'Screen1', including AboutScreen, AccentColor, AlignHorizontal, AlignVertical, AppName, BackgroundColor, BackgroundImage, and BlocksToolkit.

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

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**.

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.

We will look at **“Blocks”** view later.

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)

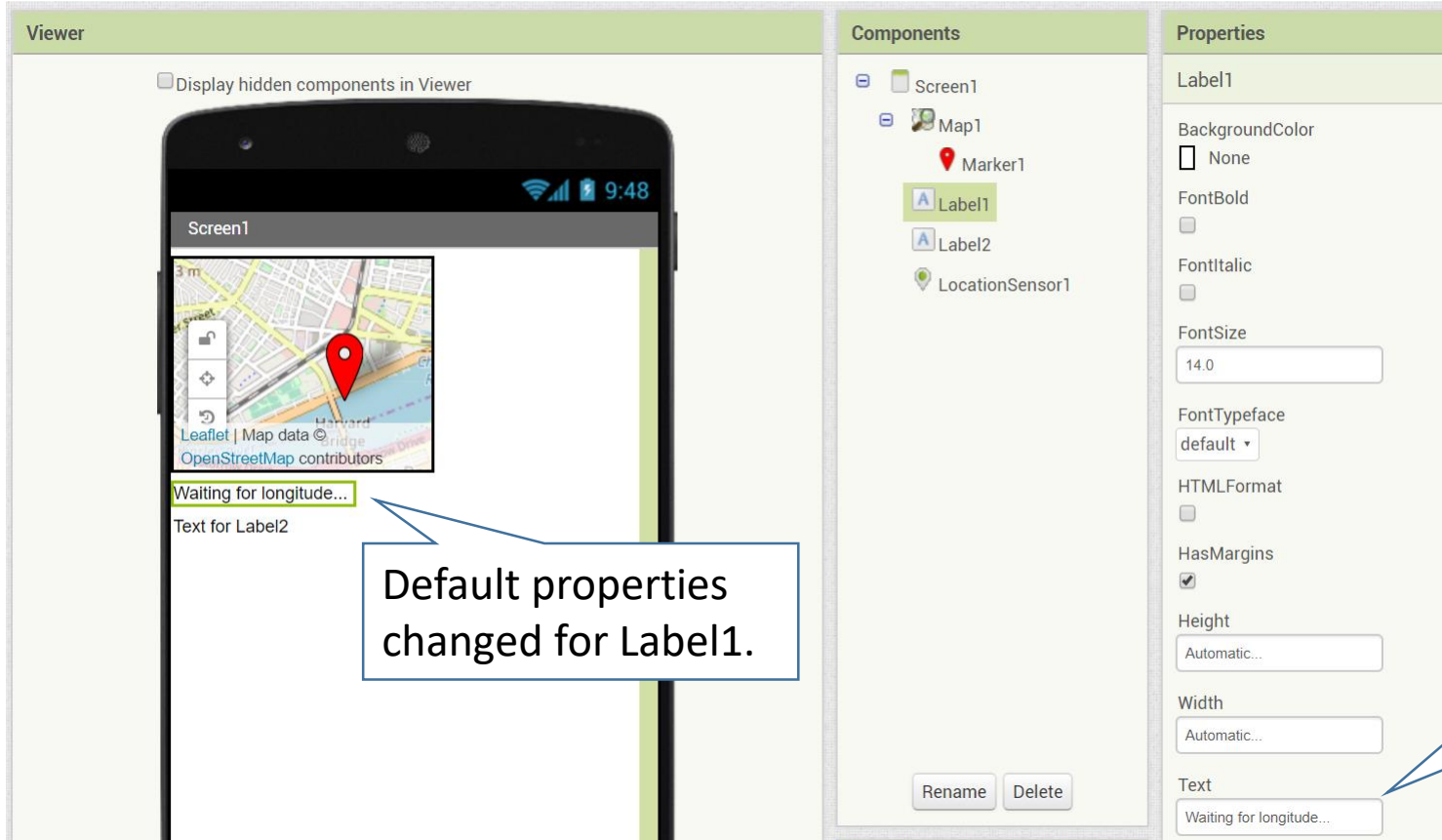
The screenshot displays the App Inventor workspace for an app named 'MapMe'. The interface is divided into several panels:

- Palette:** Located on the left, it contains categories like User Interface, Layout, Media, Drawing and Animation, Maps, and Sensors. The 'LocationSensor' component is highlighted in the Sensors section.
- Viewer:** The central area shows a mobile phone simulation. The screen displays a map with a red location pin and two text labels, 'Text for Label1' and 'Text for Label2'.
- Components:** On the right, this panel shows the component hierarchy for 'Screen1'. It lists 'Map1', 'Marker1', 'Label1', 'Label2', and 'LocationSensor1'. 'LocationSensor1' is highlighted.
- Properties:** Adjacent to the Components panel, it shows the configuration for 'LocationSensor1'. The 'DistanceInterval' is set to 0, 'Enabled' is checked, and 'TimeInterval' is set to 60000.

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.



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.)

MapMe

Screen1 Add Screen ... Remove Screen

Step 9: Complete the “jigsaw puzzle” program below:

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 “ ”

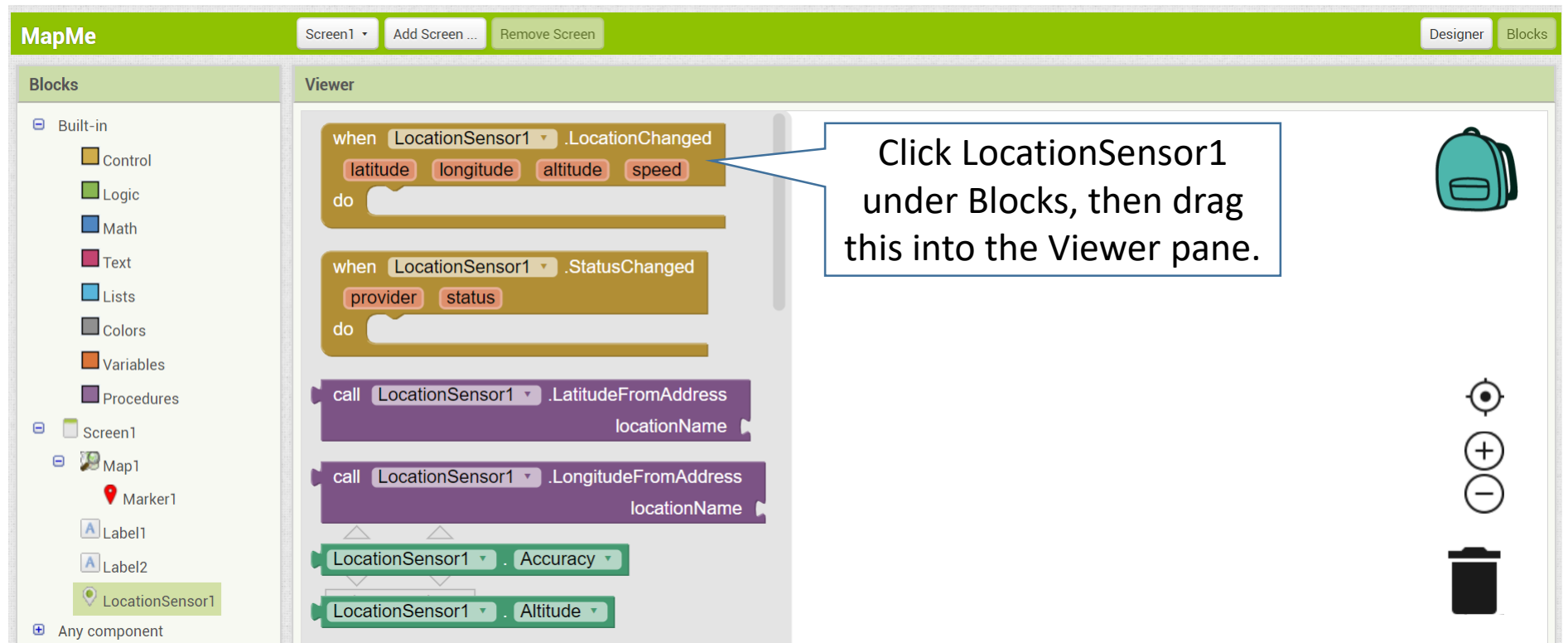
All you need to do is to drag the code **Blocks** associated with the various Components to the **Viewer** pane and snapping them together.

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

Show Warnings

App Inventor – a quick intro (cont.)

How the
“when LocationSensor1.LocationChanged”
piece is added.



The screenshot shows the App Inventor interface for an application named "MapMe". The top bar includes "Screen1", "Add Screen ...", "Remove Screen", "Designer", and "Blocks" tabs. The left pane, titled "Blocks", contains a "Built-in" category with sub-categories: Control, Logic, Math, Text, Lists, Colors, Variables, and Procedures. Under "Screen1", there is a "Map1" component with a "Marker1" sub-component, and a "LocationSensor1" component. The right pane, titled "Viewer", displays a sequence of blocks: a "when LocationSensor1 .LocationChanged" block with "latitude", "longitude", "altitude", and "speed" outputs; a "when LocationSensor1 .StatusChanged" block with "provider" and "status" outputs; two "call LocationSensor1 .LatitudeFromAddress" and "call LocationSensor1 .LongitudeFromAddress" blocks, both with "locationName" outputs; and two "LocationSensor1 . Accuracy" and "LocationSensor1 . Altitude" blocks. A callout box points to the "when LocationSensor1 .LocationChanged" block with the text: "Click LocationSensor1 under Blocks, then drag this into the Viewer pane." On the far right, there is a backpack icon and a vertical toolbar with a target icon, a plus icon, a minus icon, and a trash can icon.

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

The screenshot shows the App Inventor interface for a project named 'MapMe'. The interface is divided into three main sections: 'Blocks', 'Viewer', and a top navigation bar.

Top Navigation Bar: Contains 'Screen1', 'Add Screen ...', and 'Remove Screen' buttons.

Blocks Panel (Left): Lists various block categories under 'Built-in': Control, Logic, Math, Text, Lists, Colors, Variables, and Procedures. Below this, it shows components for 'Screen1' (Label1, Label2, LocationSensor1) and 'Map1' (Marker1).

Viewer Panel (Right): Displays a visual representation of the code blocks. The code is as follows:

```
when LocationSensor1.LocationChanged
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
```

Callouts and Block Sources:

- Variables:** Three orange blocks at the top right: 'initialize global latitude to 0', 'initialize global longitude to 0', and 'initialize global Marker1 to ""'.
- Math:** A blue block '5' used as the zoom value in the 'PanTo' block.
- Text:** A pink block '"' used in the 'initialize global Marker1' block.
- Map1, Marker1, Label1 & Label2:** Four purple and green blocks used in the 'do' section: 'call Map1.PanTo', 'call Marker1.SetLocation', 'set Label1.Text to', and 'set Label2.Text to'.

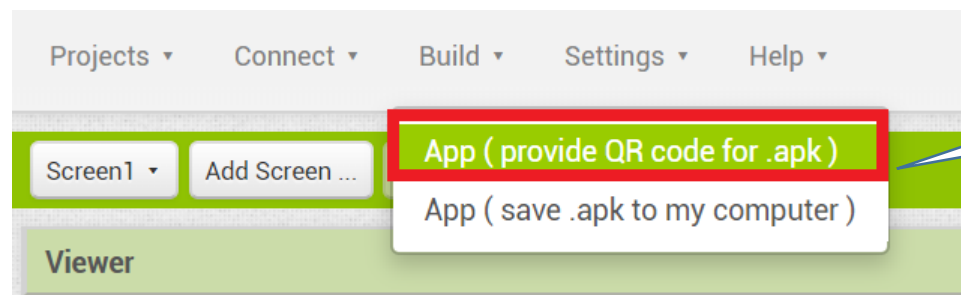
Bottom Bar: Contains a 'Show Warnings' button and two warning icons (a yellow triangle with an exclamation mark and a red circle with an X), both showing a count of 0.

This is from Math.

This is from Text.

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

App Inventor – a quick intro (cont.)



Projects ▾ Connect ▾ Build ▾ Settings ▾ Help ▾

Screen1 ▾ Add Screen ...

App (provide QR code for .apk)

App (save .apk to my computer)

Viewer

MapMe Progress Bar

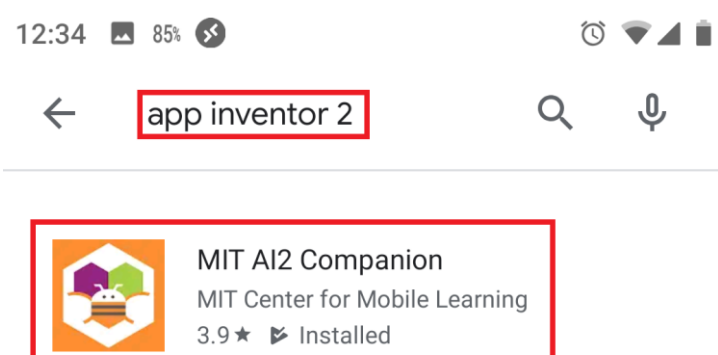
50%

Compiling part 2 (please wait)

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.



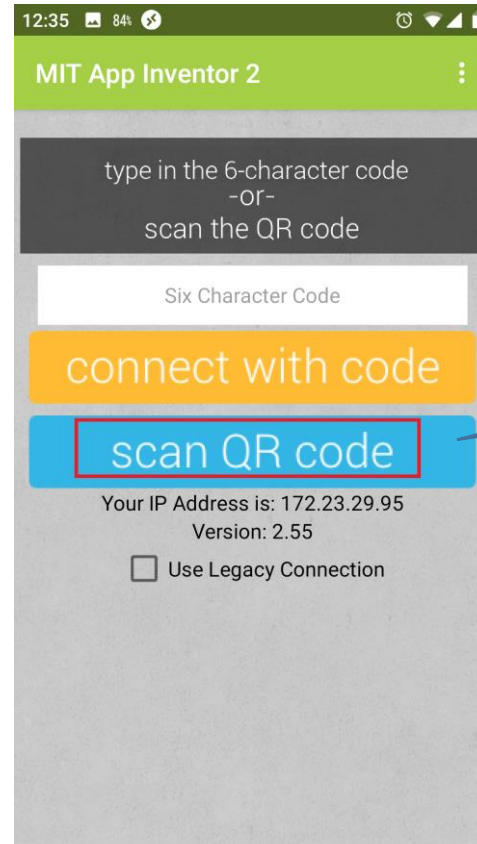
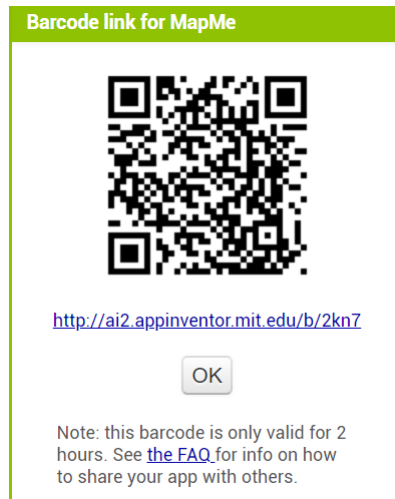
12:34 85%

← app inventor 2 🔍

 MIT AI2 Companion
MIT Center for Mobile Learning
3.9★ 📄 Installed

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

App Inventor – a quick intro (cont.)

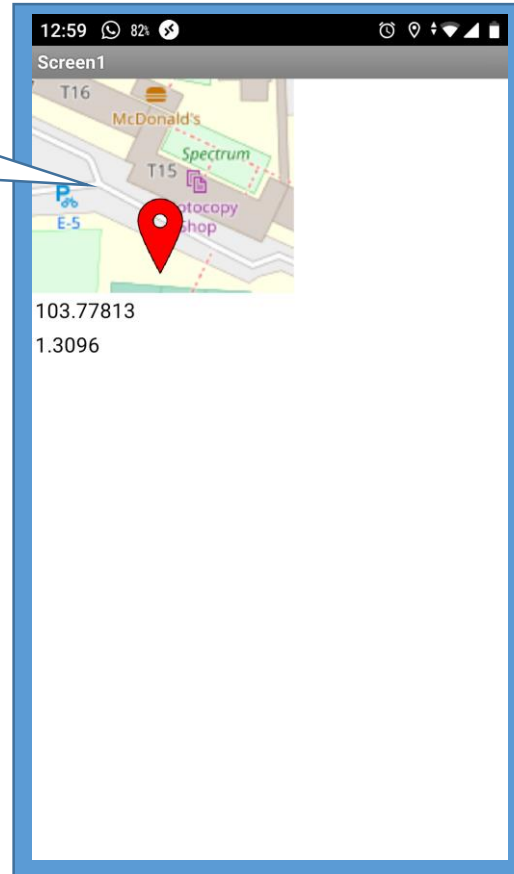


Step 12: Use the MIT AI2 Companion app to scan the **QR code** generated to download the .apk file.

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.

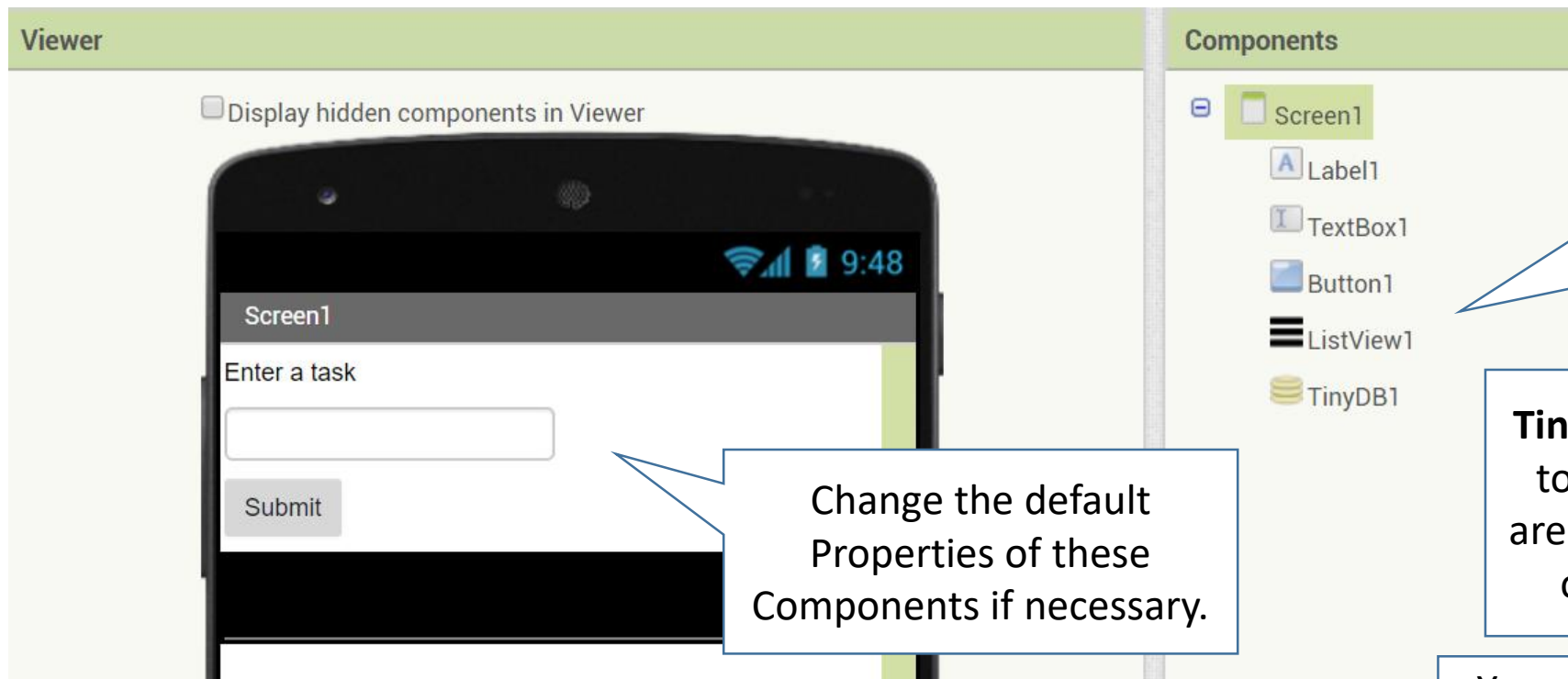
Lab Exercises

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.

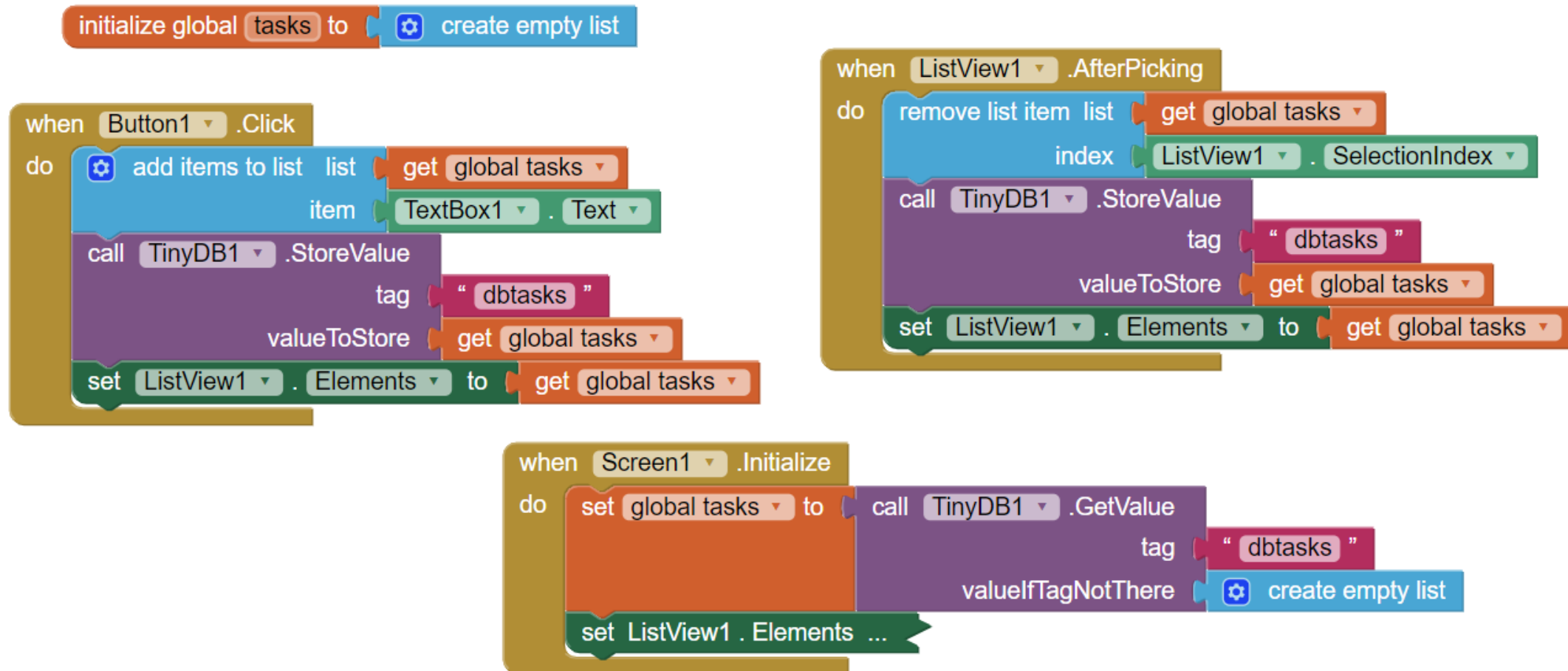
Change the default Properties of these Components if necessary.

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.

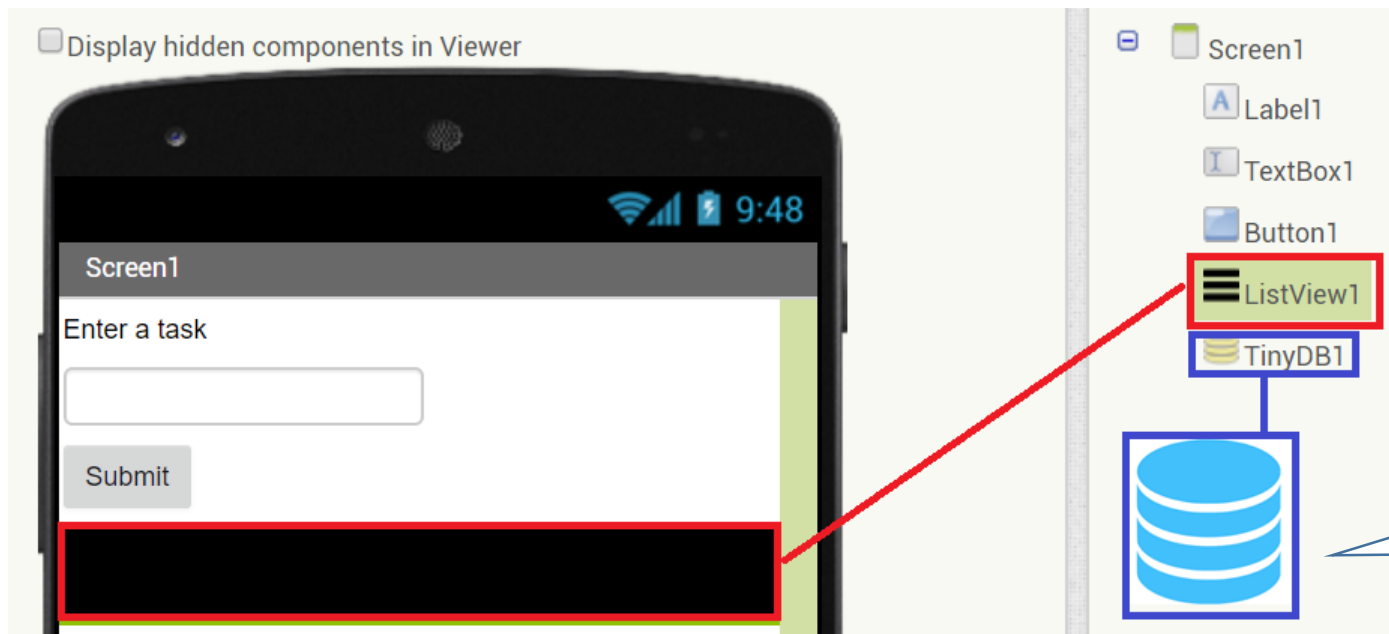


Exercise 6.1 – To Do List (cont.)

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

initialize global **tasks** to  create empty list

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?

The image shows a mobile app interface on the left and its Scratch code blocks on the right. The app interface has a status bar at the top with the time 9:48, signal strength, and battery level. Below the status bar is a header labeled "Screen1". The main content area has the text "Enter a task" followed by a text input field containing "Do homework" and a "Submit" button. Below the input field is a black rectangular area representing a list.

The Scratch code blocks are as follows:

- when Button1 .Click** (yellow block)
- do** (blue block)
- add items to list** (blue block) with **list** set to **global tasks** and **item** set to **TextBox1 . Text**.
- call TinyDB1 .StoreValue** (purple block) with **tag** set to **" dbtasks "** and **valueToStore** set to **global tasks**.
- set ListView1 . Elements** (green block) to **global tasks**.

Three callout boxes provide additional information:

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

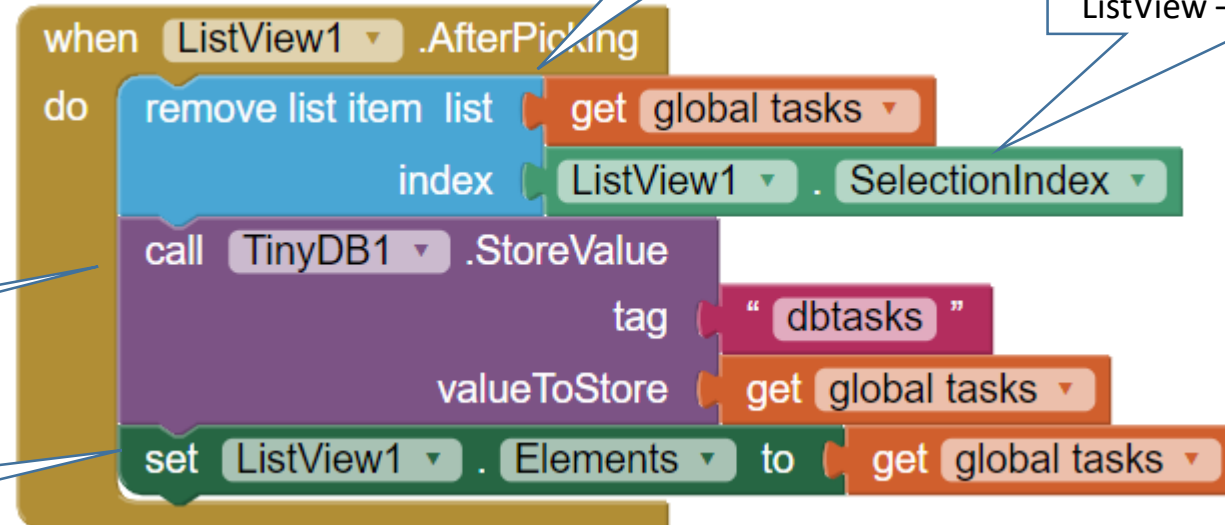
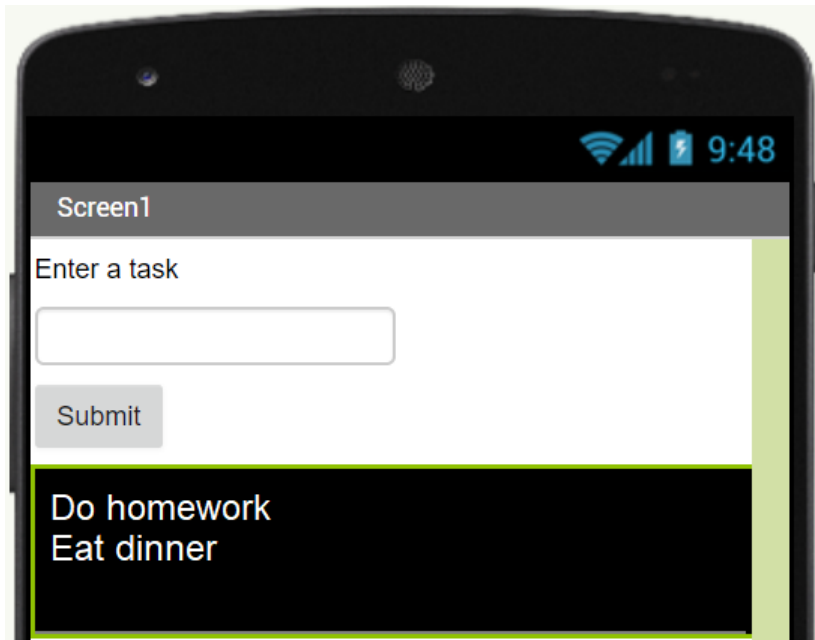
Note that TinyDB stores items as "tag, value" pairs, to allow items to be retrieved by their tags.

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?




2. TinyDB is updated.

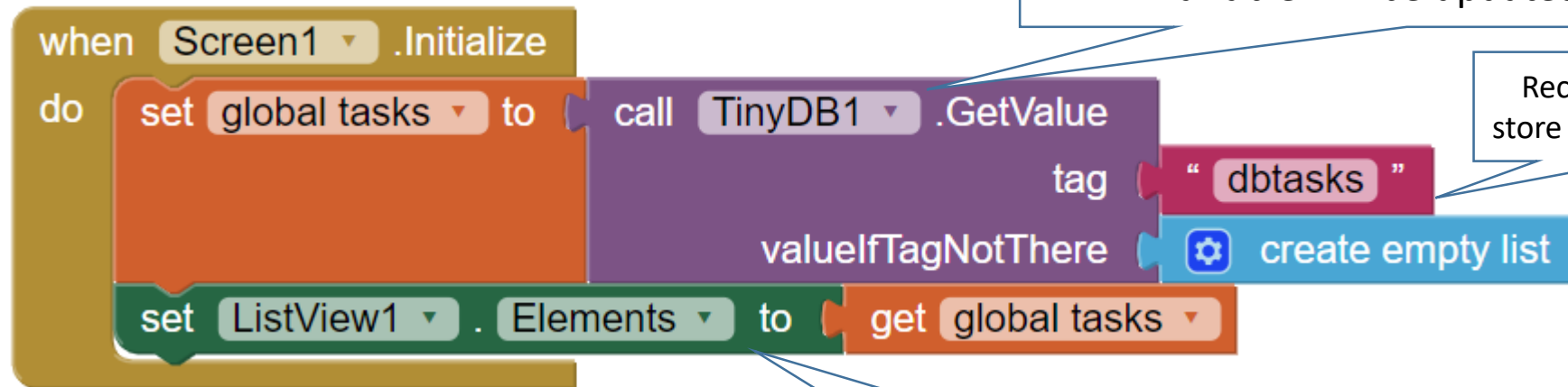
3. Likewise, ListView is updated.

Exercise 6.1 – To Do List (cont.)

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

initialize global **tasks** to  create empty list

“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.

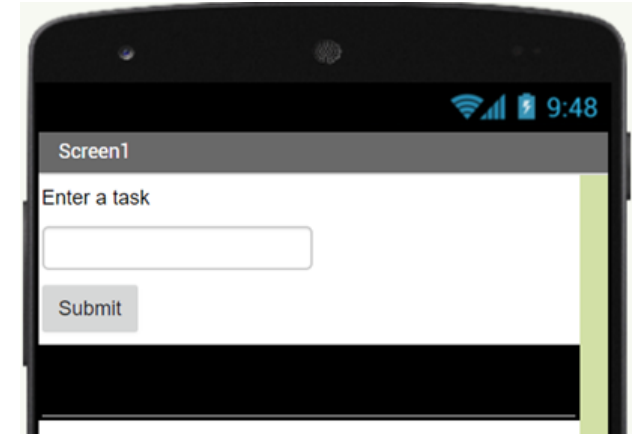
Recall that the tag to store the list is “dbtasks”.

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

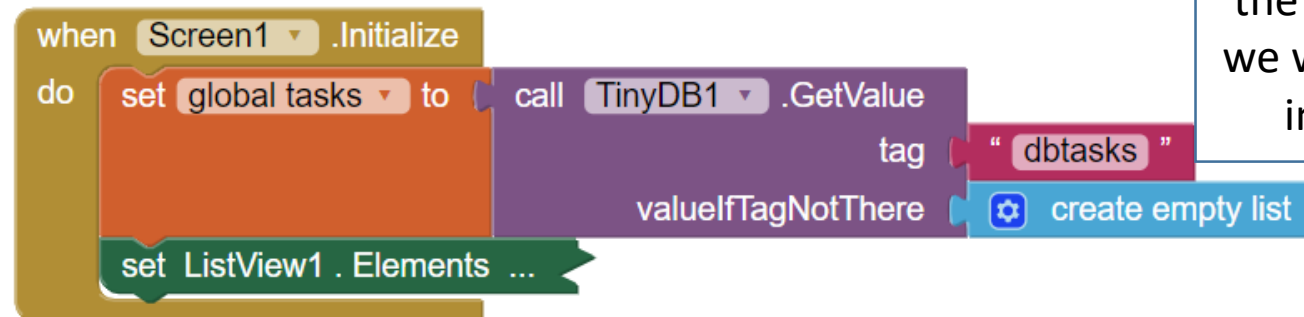
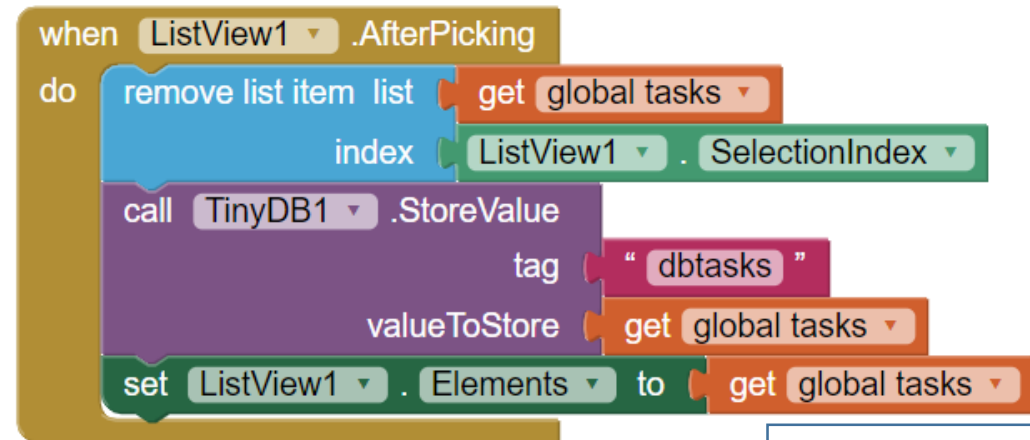
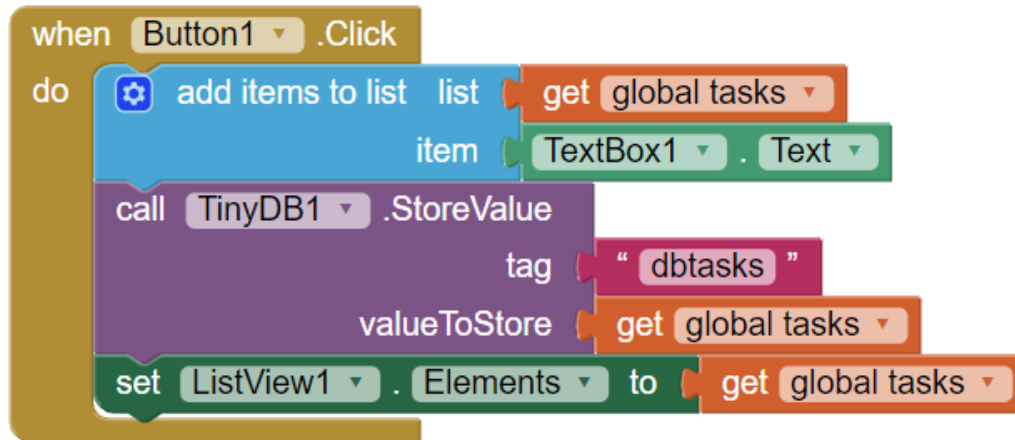
2. Likewise, ListView is updated.

Exercise 6.1 – To Do List (cont.)

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



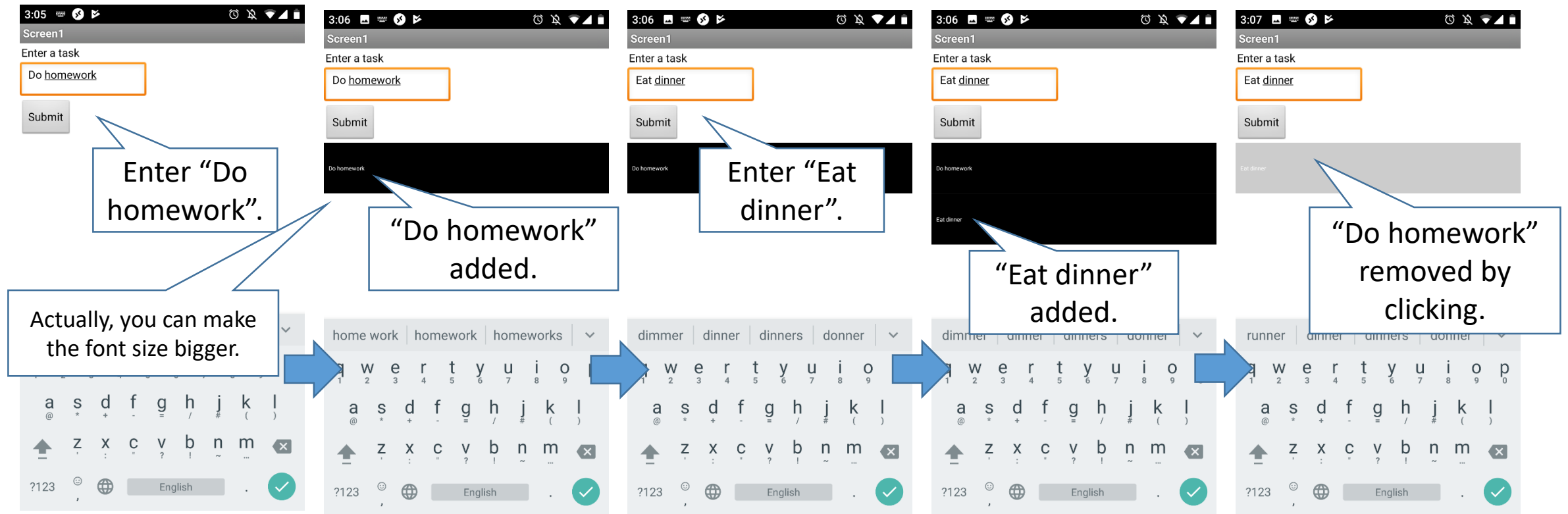
initialize global tasks to create empty list



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:

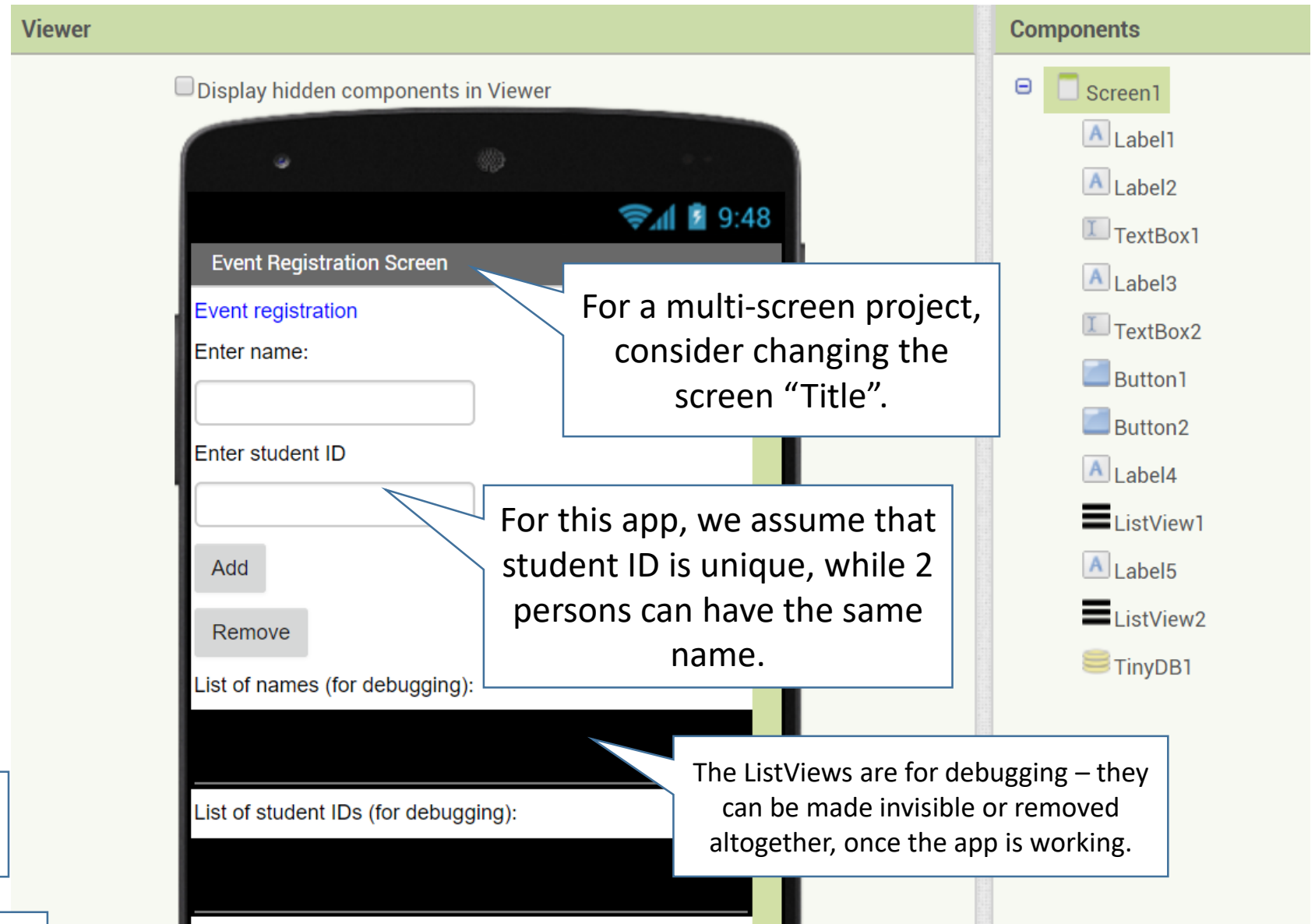


Exercise 6.2 – Registering / De-registering for an Event

- 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?

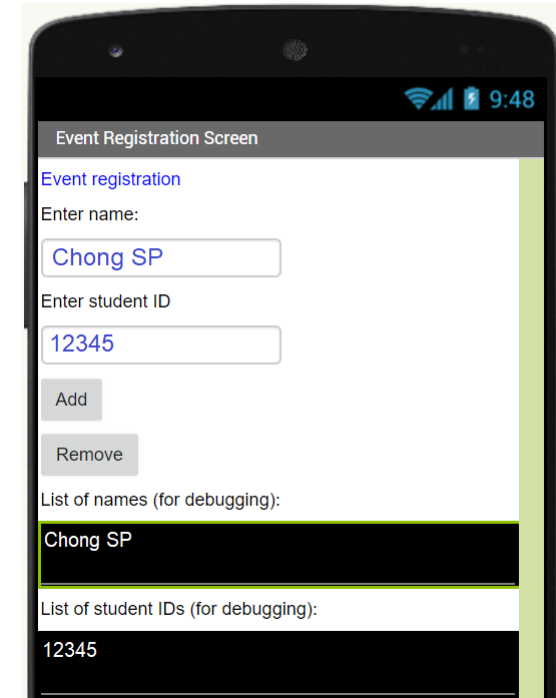
This is like adding an item to the list of things to do in the previous app. So you should be able to figure out the blocks of code below.

```
initialize global names to create empty list
initialize global IDs to create empty list

when Screen1.Initialize
do
  set global names to call TinyDB1.GetValue tag "dbnames" valueIfTagNotThere create empty list
  set global IDs to call TinyDB1.GetValue tag "dbIDs" valueIfTagNotThere create empty list
  set ListView1.Elements to get global names
  set ListView2.Elements to get global IDs

when Button1.Click
do
  add items to list list get global names item TextBox1.Text
  add items to list list get global IDs item TextBox2.Text
  call TinyDB1.StoreValue tag "dbnames" valueToStore get global names
  call TinyDB1.StoreValue tag "dbIDs" valueToStore get global IDs
  set ListView1.Elements to get global names
  set ListView2.Elements to get global IDs
```

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.

Event Registration Screen

Event registration

Enter name:

Chong SP

Enter student ID

22222

Add

Remove

List of names (for debugging):

Chong SP
Joe Yang
Chong SP

List of student IDs (for debugging):

12345
88888
22222

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:

when Button2 .Click

do

if

select list item list

index

get global names

=

TextBox1 . Text

index in list thing

list

get global IDs

remove list item list

index

get global names

index in list thing

list

get global IDs

remove list item list

index

get global IDs

index in list thing

list

get global IDs

call TinyDB1 .StoreValue

tag

"dbnames"

valueToStore

get global names

call TinyDB1 .StoreValue

tag

"dbIDs"

valueToStore

get global IDs

set ListView1 . Elements

to

get global names

set ListView2 . Elements

to

get global IDs

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 & ListView the same way.

Do you know what's wrong with this?

em list

index

get global names

index in list thing

list

get global names

em list

index

get global IDs

index in list thing

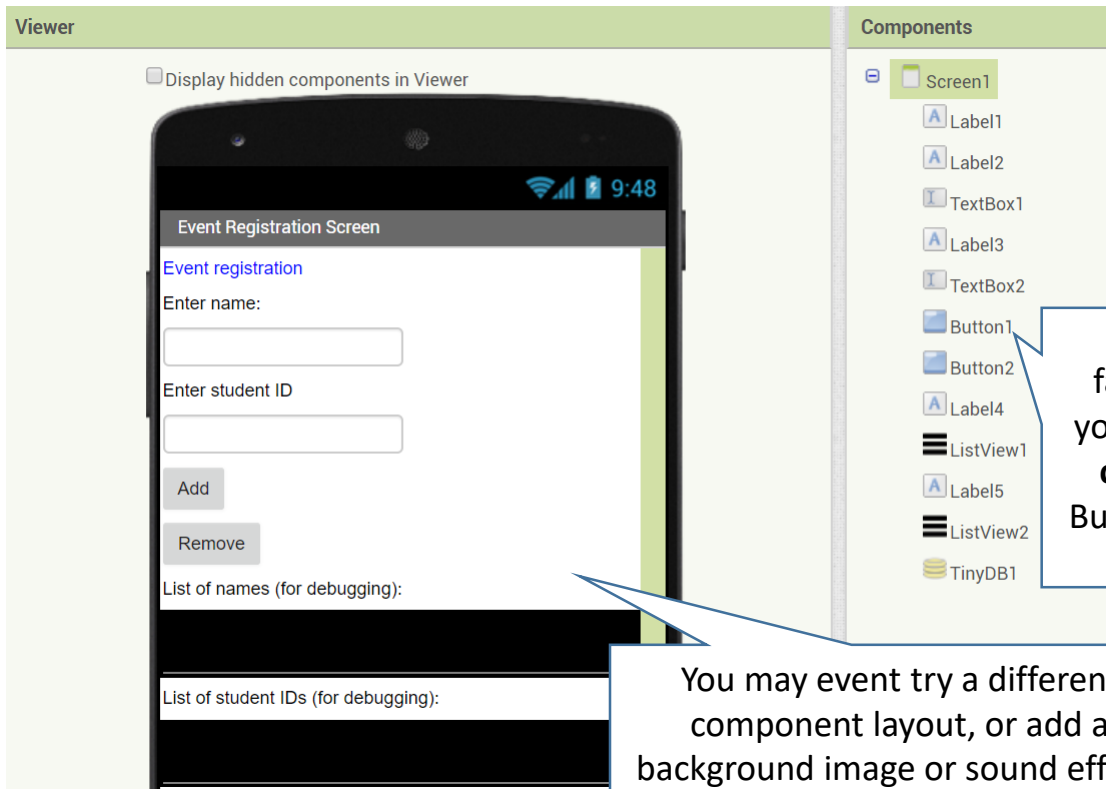
list

get global IDs

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

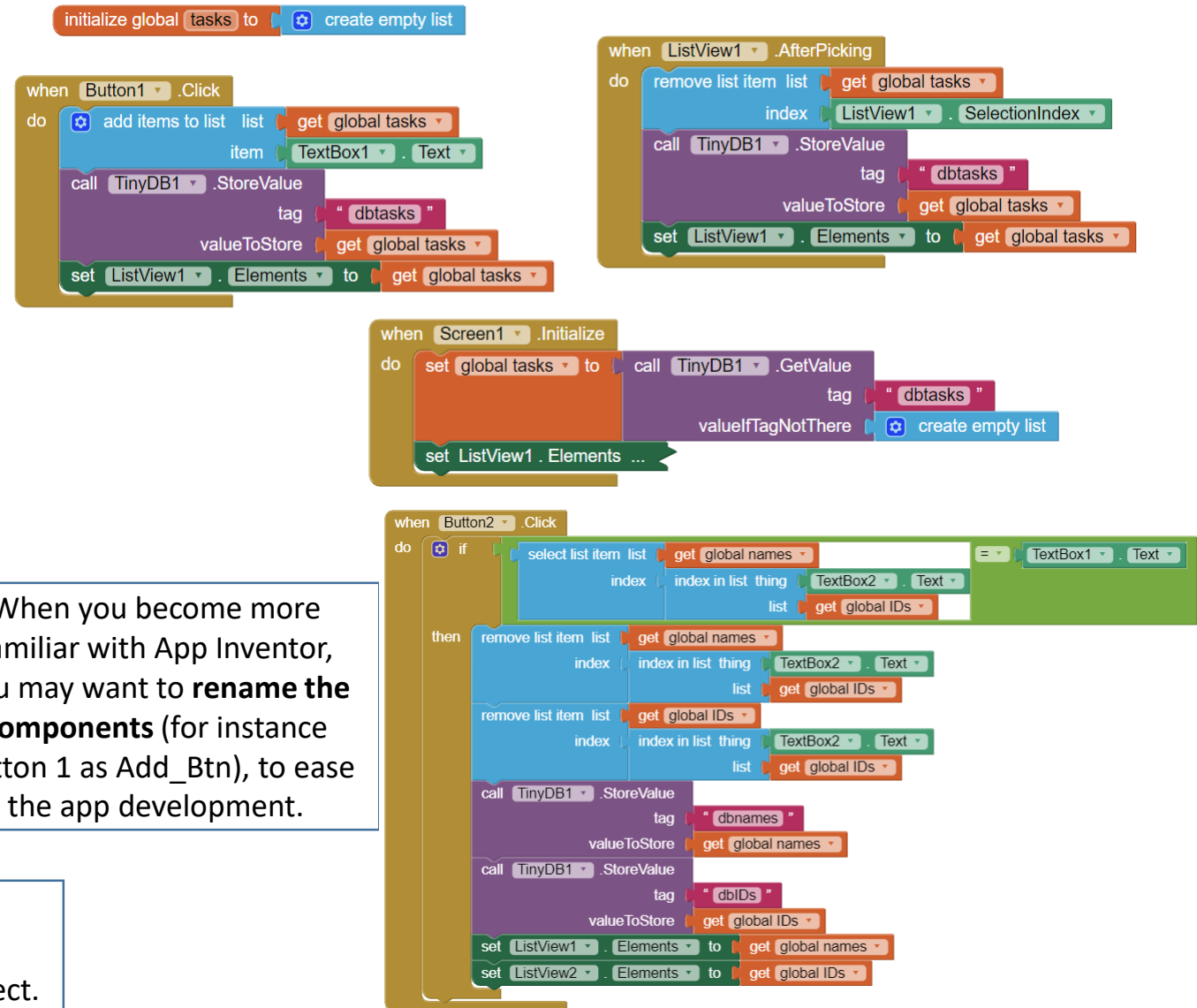
Exercise 6.2 – Registering / De-registering for an Event (cont.)

- 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 even 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.



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

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

List index smaller than 1

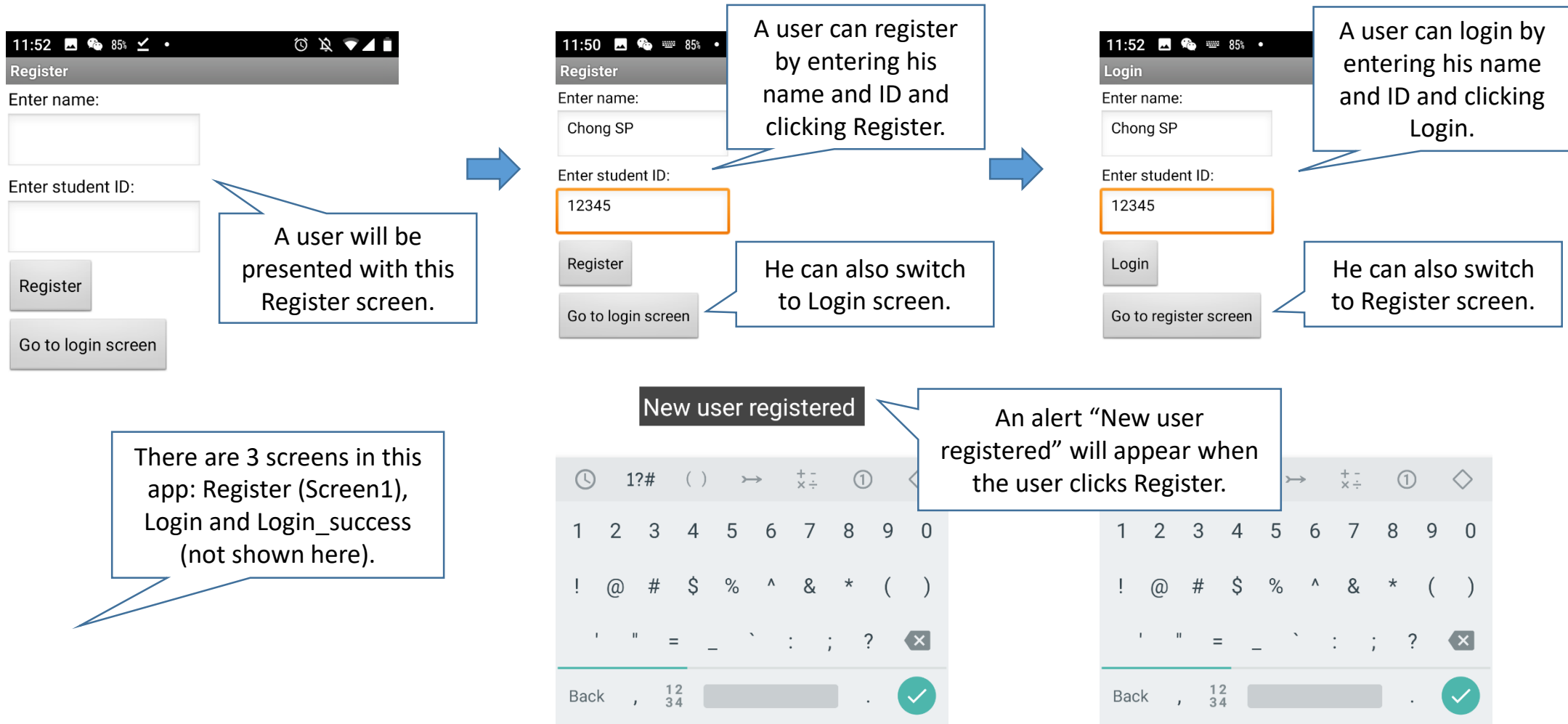
Select list item: Attempt to get item number 0, of the list (Chong SP Joe Yang). The minimum valid item number is 1.

End Application

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.



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

11:51 85%
Login

Enter name:
Chong SP

Enter student ID:
88888

Login

Go to register screen

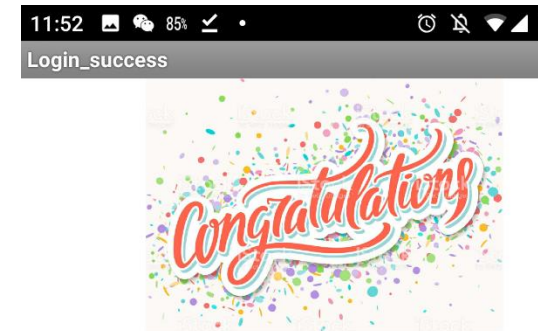
An alert "name & ID do not match" will appear when the user logs in without the correct info.

name & ID do not match

12/34

Back , . ✓

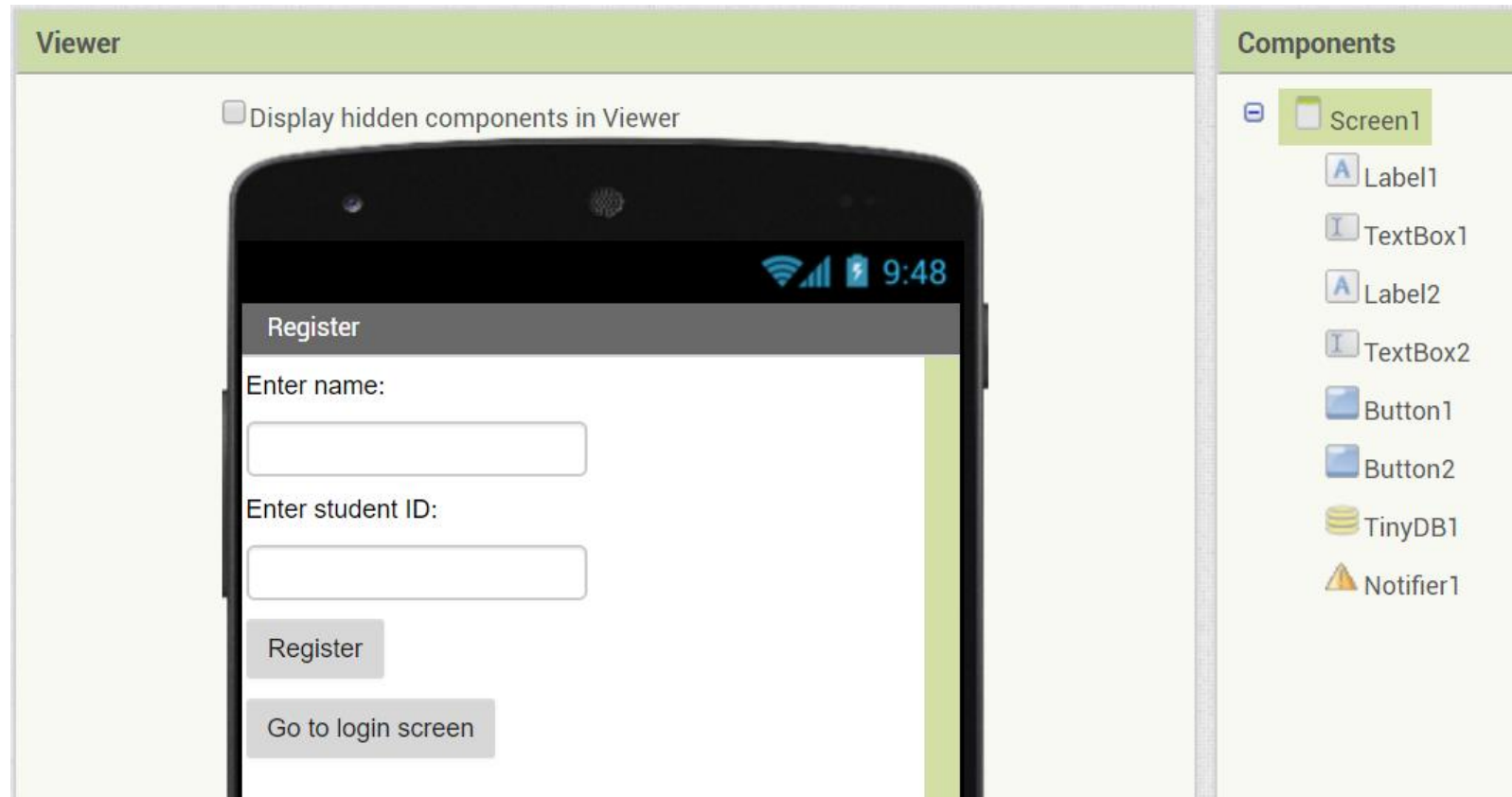
Successful login



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

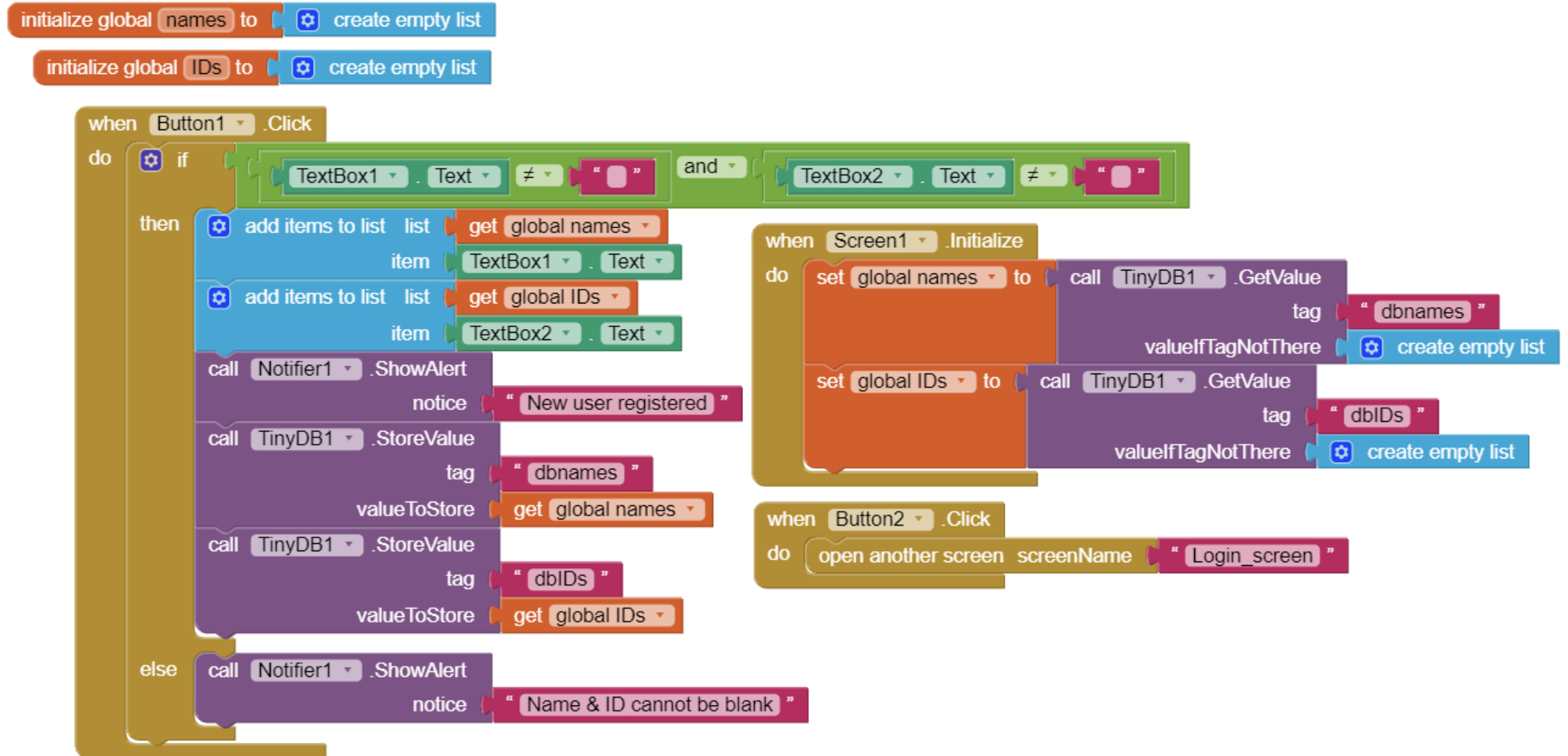
Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen1 (Register screen) – Designer view:



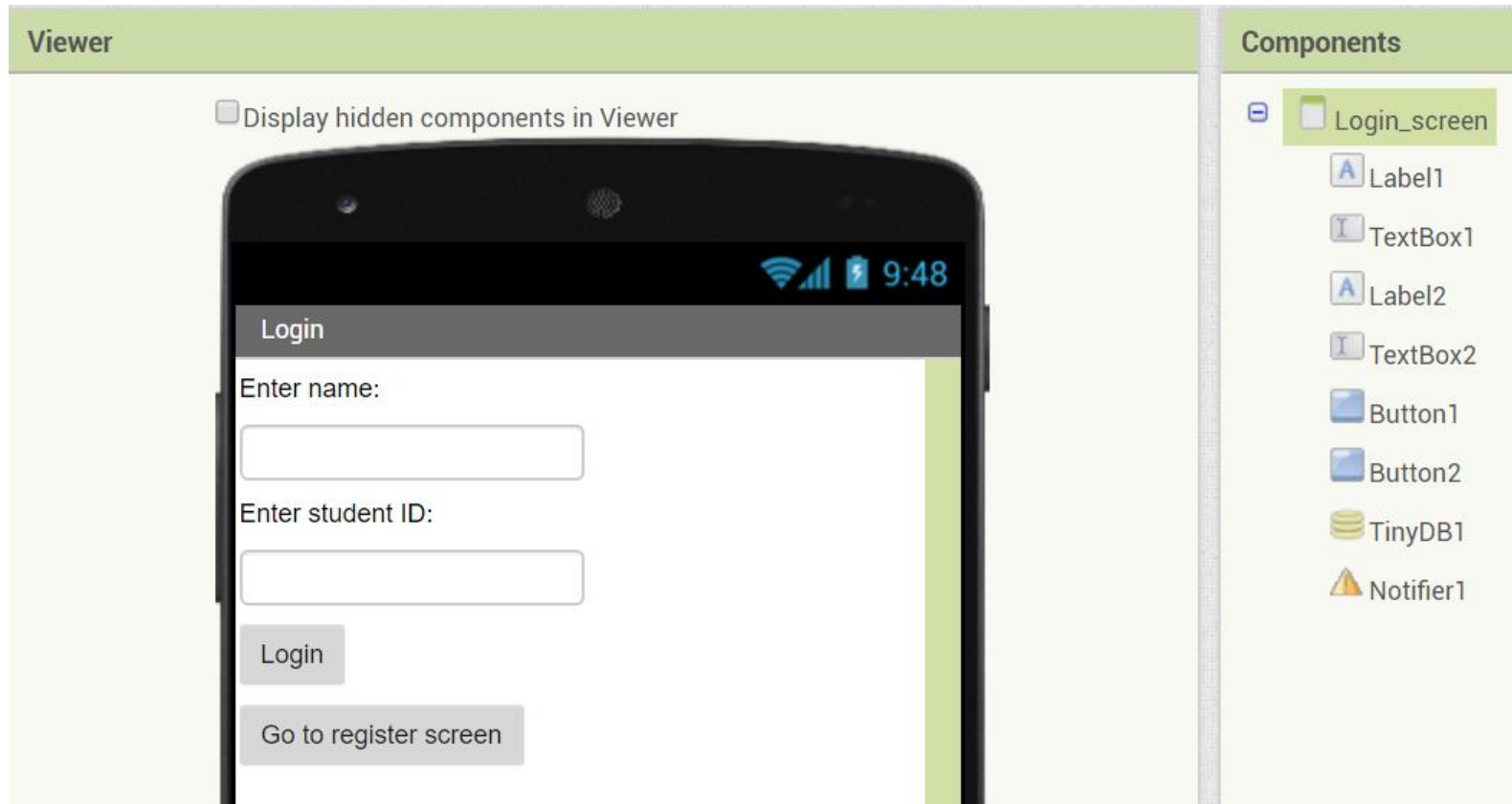
Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen1 (Register screen) – Blocks view:



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

- Screen2 (Login screen) – Designer view:

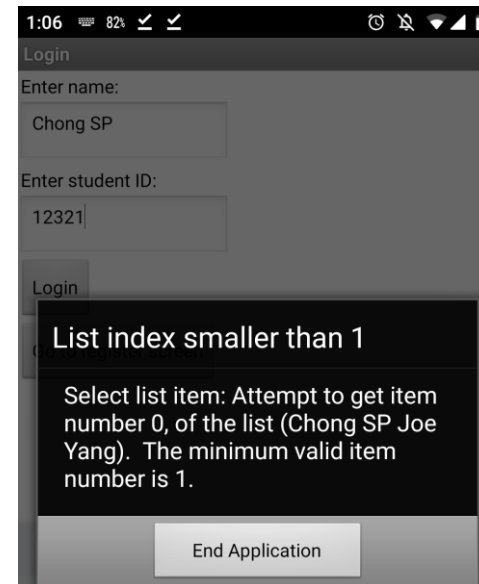
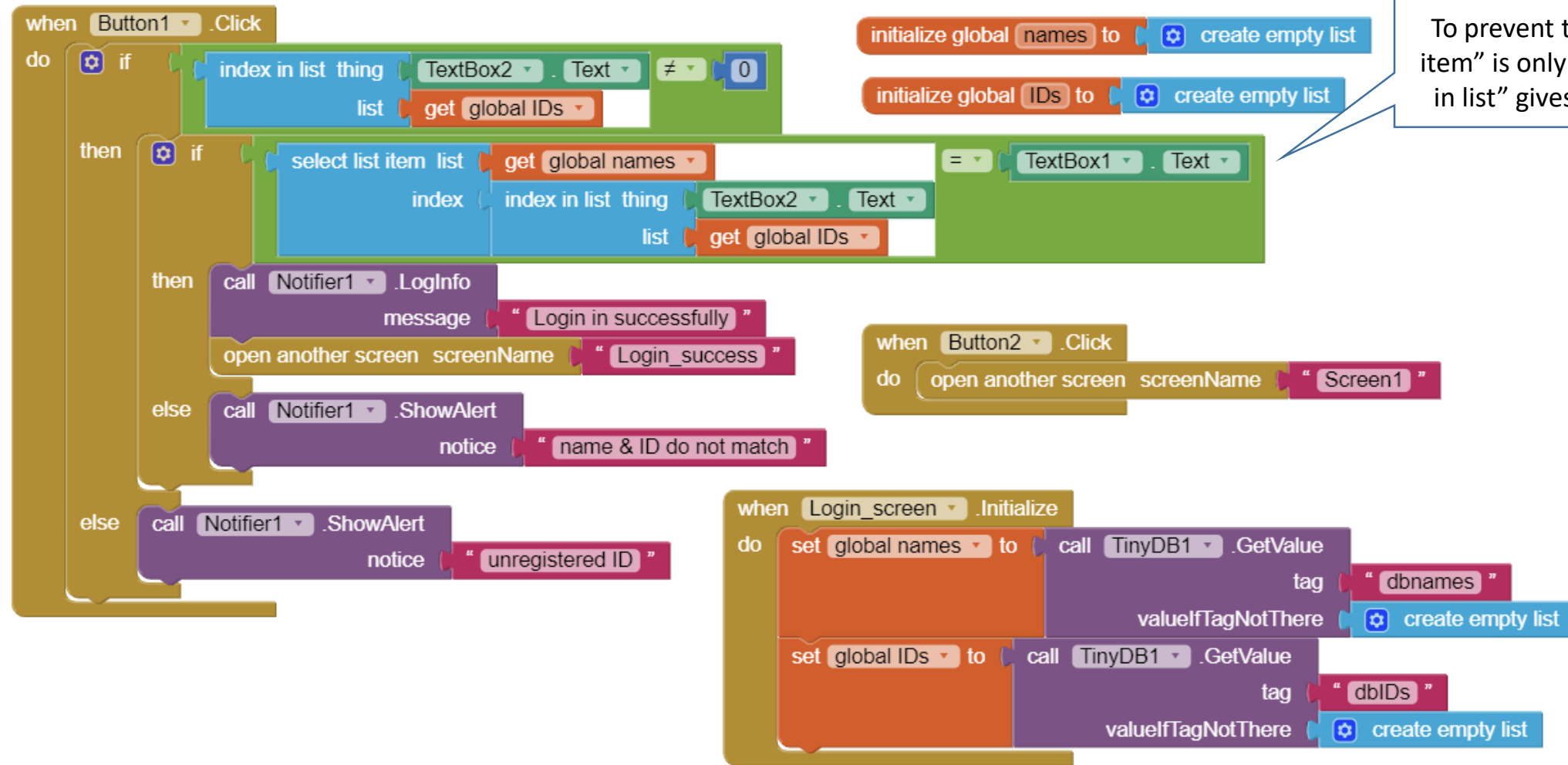


Exercise 6.3 – Register / login as a user (homework) (cont.)

- Screen2 (Login screen) – Blocks view:

When a user logs in 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):

The screenshot displays the Xcode Designer interface for the 'Login_success' screen. The main window is divided into three sections: 'Viewer', 'Components', and 'Properties'.

- Viewer:** Shows a mobile device screen with a 'Login_success' header and a 'Congratulations' message with confetti. A checkbox 'Display hidden components in Viewer' is visible at the top.
- Components:** Shows a hierarchy of components: 'Login_success' (root), 'HorizontalArrangement1' (child of Login_success), and 'Image1' (child of HorizontalArrangement1). A blue arrow points from 'Image1' in the Components panel to the Properties panel.
- Properties:** Shows the properties for the selected 'Image1' component. The properties include:
 - HorizontalArrangement1:** AlignHorizontal (Left : 1), AlignVertical (Top : 1), BackgroundColor (Default), Height (150 pixels...), Width (Fill parent...), Image (None...), and Visible (checked).
 - Image1:** Height (Fill parent...), Width (Fill parent...), and Picture (Congrats.jpg...).

A smaller inset window at the bottom shows the 'Components' and 'Properties' panels for the 'Image1' component, with the 'Picture' property set to 'Congrats.jpg'.

