



**Course 2 Capstone Peer Review 1 Tutorial**

This tutorial walks you through most of the steps involved in updating the starter code base to implement the Command pattern. The steps in this tutorial are:

[1. Clear the app memory](#_i9c8l9seyc2k) **3**

[2. Create and implement the abstract Command class](#_w002e735pl32) **5**

[3. Create and implement the AddItemCommand class](#_zh0escrp3e) **6**

[4. Update the ItemList class](#_gcloq76xrkv9) **7**

[5. Create and implement the DeleteItemCommand class](#_2kvxsti32at5) **7**

[6. Create and implement the EditItemCommand class](#_4mjjcppzhyzs) **8**

[7. Update AddItemActivity to use AddItemCommand](#_2y0p1zx2kimd) **9**

[8. Update EditItemActivity to use DeleteItemCommand and EditItemCommand](#_dxr7idmvobp7) **9**

[9. Create and implement the AddContactCommand class](#_4fyt60cbjo2k) **10**

[10. Update the ContactList class](#_3iwinerjqfma) **11**

[11. Create and implement the DeleteContactCommand class](#_jbfmbwg67wov) **11**

[12. Create and implement the EditContactCommand class](#_ncfj0fdf51ic) **11**

[13. Update AddContactActivity to use AddContactCommand](#_ouln21jb75y7) **11**

[14. Update EditContactActivity to use DeleteContactCommand and EditContactCommand](#_pagtp4ddak70) **12**

[15. Run the app](#_2f3biyp17bdy) **12**

**You do not necessarily have to go through all these steps manually, you could opt to start this assignment from the Peer Review 1 starter code base.**

**If you would like to opt to simply use the peer review 1 starter code base, you must still visit steps in the tutorial:**

[1. Clear the app memory](#_i9c8l9seyc2k) **3**

[9. Create and implement the AddContactCommand class](#_4fyt60cbjo2k) **10**

[10. Update the ContactList class](#_3iwinerjqfma) **11**

[11. Create and implement the DeleteContactCommand class](#_jbfmbwg67wov) **11**

[12. Create and implement the EditContactCommand class](#_ncfj0fdf51ic) **11**

[13. Update AddContactActivity to use AddContactCommand](#_ouln21jb75y7) **11**

[14. Update EditContactActivity to use DeleteContactCommand and EditContactCommand](#_pagtp4ddak70) **12**

[15. Run the app](#_2f3biyp17bdy) **12**

There are hints in these steps, so they are definitely worth checking out!

This application allows for items and contacts to be added, edited and deleted. Each of these actions can be thought of as a command and should have a Command class to execute it.

When you implement the Command Pattern for this assignment, the features and functionality of the app should not change. By implementing this design pattern you are simply organizing the code so that actions are conducted through command objects.

#### 1. (Optional) Clear the app memory (if you have SharingApp installed)

If you already have a previous version of SharingApp on your emulator it is a good idea to clear the app’s data. If we don’t clear the previously stored data then the app may crash due to changes made to the model.

After opening Android Studio click the **play button** to run the app.



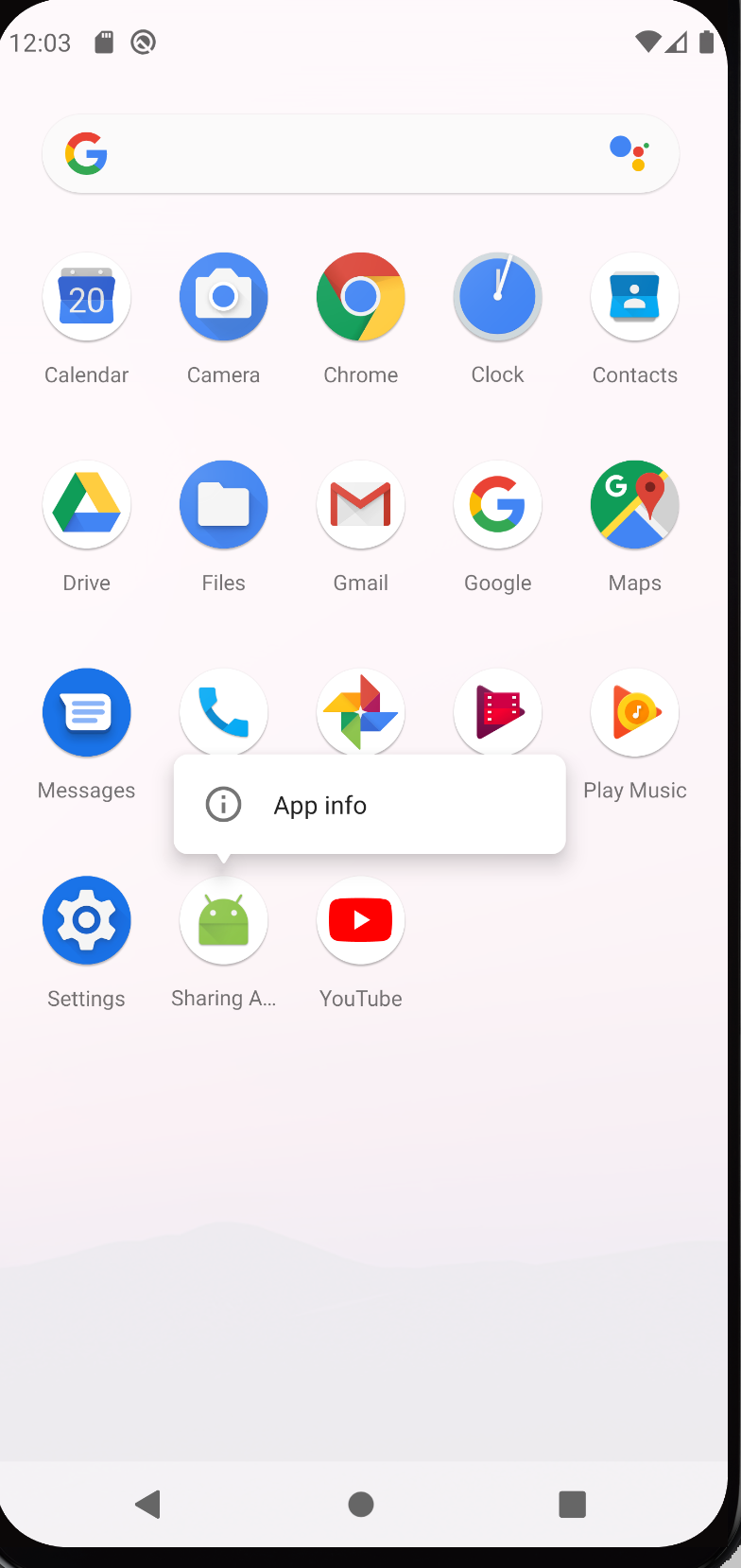
Be patient, the emulator may take a few minutes to load.

If the app launches and doesn’t crash -- great! You are done. Apparently the changes you made to the app did not have an effect on the data being stored. You can now move to **Step 2**.

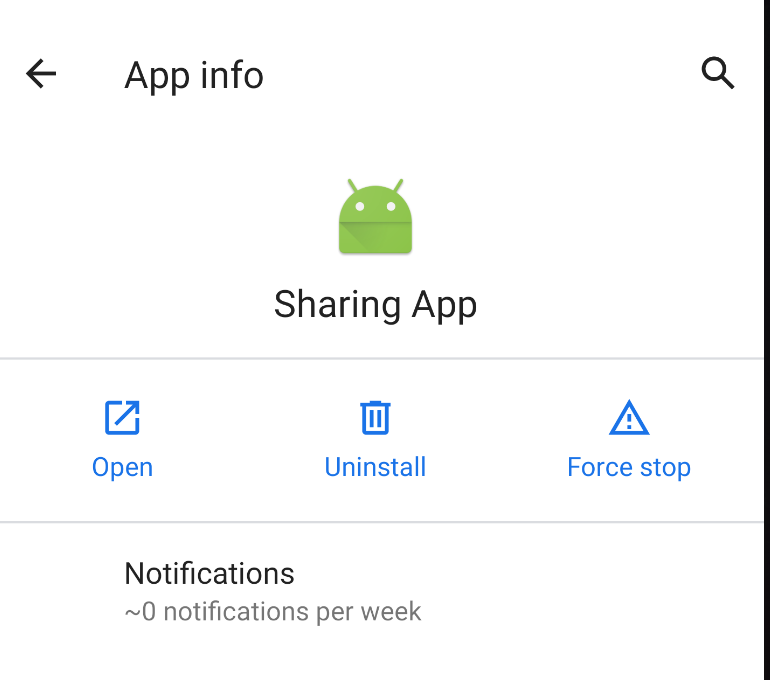
If it does crash -- don’t worry. A message will appear

to inform you that the app has crashed. Click **OK**.

Then, swipe up and find the Sharing **App. Long click** on the icon to see the **App info**



Click **App Info,** and then select **Uninstall**

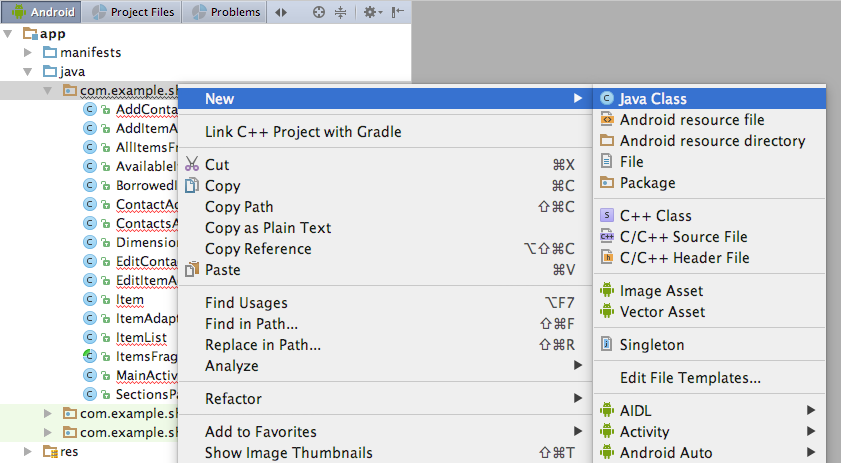


Now the app is uninstalled and all the previously stored data has been erased.

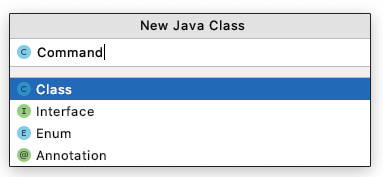
#### 2. Create and implement the abstract Command class

All concrete Command classes (**AddItemCommand**, **DeleteItemCommand**, **EditItemCommand**, **AddContactCommand**, **DeleteContactCommand**, and **EditContactCommand**) will inherit from an abstract **Command** class. We need to add this class to the project.

Create a new abstract class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**.



Name the class **Command** then hit **Enter**. This creates an empty **Command** class.



Replace the contents of the **Command** class with:

**package** com.example.sharingapp;

*/\*\**

*\* Superclass of AddContactCommand, EditContactCommand, DeleteContactCommand,*

*\* AddItemCommand, EditItemCommand, DeleteItemCommand*

*\*/*

**public abstract class** Command {

**private boolean is\_executed**;

**public** Command(){

**is\_executed** = **false**;

}

**public abstract void** execute();

**public boolean** isExecuted(){

**return is\_executed**;

}

**public void** setIsExecuted(**boolean** is\_executed) {

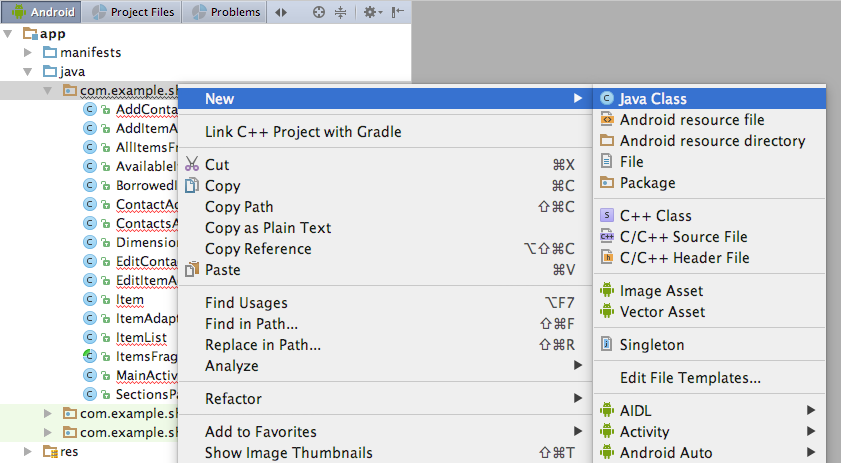
**this**.**is\_executed** = is\_executed;

}

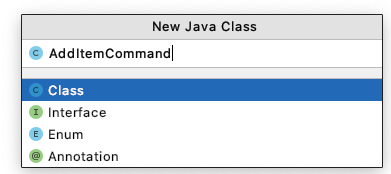
}

#### 3. Create and implement the AddItemCommand class

Create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**



Name the class **AddItemCommand**. Hit **Enter**. This creates an empty **AddItemCommand** class.



Replace the contents of the **AddItemCommand** class with:

**package** com.example.sharingapp;

**import** android.content.Context;

*/\*\**

*\* Command to add item*

*\*/*

**public class** AddItemCommand **extends** Command{

**private** ItemList **item\_list**;

**private** Item **item**;

**private** Context **context**;

**public** AddItemCommand(ItemList item\_list, Item item, Context context) {

**this**.**item\_list** = item\_list;

**this**.**item** = item;

**this**.**context** = context;

}

**public void** execute(){

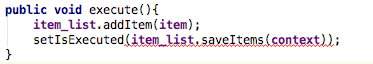
**item\_list**.addItem(**item**);

setIsExecuted(**item\_list**.saveItems(**context**));

}

}

Notice that item\_list.saveItems(context); is underlined in **red**.



This is because the **ItemList** class does not have a method called **saveItems()** that matches that signature, i.e, **setIsExecuted()** is expecting that **item\_list**.saveItems(**context**) will return a boolean, but instead it returns void.

Let’s take a look at **ItemList** to sort this out.

#### 4. Update the ItemList class

Double click on the **ItemList** class to open it. Navigate to the **saveItems()** method. Replace the current **saveItems()** method with:

**public boolean** saveItems(Context context) {

**try** {

FileOutputStream fos = context.openFileOutput(**FILENAME**, 0);

OutputStreamWriter osw = **new** OutputStreamWriter(fos);

Gson gson = **new** Gson();

gson.toJson(*items*, osw);

osw.flush();

fos.close();

} **catch** (FileNotFoundException e) {

e.printStackTrace();

**return false**;

} **catch** (IOException e) {

e.printStackTrace();

**return false**;

}

**return true**;

}

#### 5. Create and implement the DeleteItemCommand class

Create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**.

Name the class **DeleteItemCommand**. Hit **Enter**. This creates an empty **DeleteItemCommand** class.

Replace the contents of **DeleteItemCommand** with:

**package** com.example.sharingapp;

**import** android.content.Context;

*/\*\**

*\* Command to delete an item*

*\*/*

**public class** DeleteItemCommand **extends** Command {

**private** ItemList **item\_list**;

**private** Item **item**;

**private** Context **context**;

**public** DeleteItemCommand(ItemList item\_list, Item item, Context context) {

**this**.**item\_list** = item\_list;

**this**.**item** = item;

**this**.**context** = context;

}

**public void** execute() {

**item\_list**.deleteItem(**item**);

setIsExecuted(**item\_list**.saveItems(**context**));

}

}

#### 6. Create and implement the EditItemCommand class

Create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**

Name the class **EditItemCommand**. Click **OK**. This creates an empty **EditItemCommand** class. Replace the contents of **EditItemCommand** with:

**package** com.example.sharingapp;

**import** android.content.Context;

*/\*\**

*\* Command to edit a pre-existing item*

*\*/*

**public class** EditItemCommand **extends** Command {

**private** ItemList **item\_list**;

**private** Item **old\_item**;

**private** Item **new\_item**;

**private** Context **context**;

**public** EditItemCommand(ItemList item\_list, Item old\_item, Item new\_item, Context context) {

**this**.**item\_list** = item\_list;

**this**.**old\_item** = old\_item;

**this**.**new\_item** = new\_item;

**this**.**context** = context;

}

**public void** execute() {

**item\_list**.deleteItem(**old\_item**);

**item\_list**.addItem(**new\_item**);

setIsExecuted(**item\_list**.saveItems(**context**));

}

}

Now that we have created all the Commands related to the **ItemList** class (**AddItemCommand**, **DeleteItemCommand**, and **EditItemCommand**) it is time to update our Activities to use these Commands instead of directly interacting with the **ItemList** model.

#### 7. Update AddItemActivity to use AddItemCommand

Double click on **AddItemActivity** to edit it.

Locate the **saveItem()** method. Locate the following lines within the **saveItem()** method:

**item\_list**.addItem(item);

**item\_list**.saveItems(**context**);

**Replace** these (above) two lines with:

*// Add item*

AddItemCommand add\_item\_command = **new** AddItemCommand(**item\_list**, item, **context**);

add\_item\_command.execute();

**boolean** success = add\_item\_command.isExecuted();

**if** (!success){

**return**;

}

#### 8. Update EditItemActivity to use DeleteItemCommand and EditItemCommand

Double click on **EditItemActivity** to edit it.

Locate the **saveItem()** method. Locate the following lines within the **saveItem()** method:

**item\_list**.deleteItem(**item**);

**item\_list**.addItem(item);

**item\_list**.saveItems(**context**);

**Replace** these (above) three lines with:

*// Edit item*

EditItemCommand edit\_item\_command = **new** EditItemCommand(**item\_list**, **item**, updated\_item, **context**);

edit\_item\_command.execute();

**boolean** success = edit\_item\_command.isExecuted();

**if** (!success){

**return**;

}

Next, locate the **deleteItem()** method. Locate the following lines within the **deleteItem()** method:

**item\_list**.deleteItem(**item**);

**item\_list**.saveItems(**context**);

Replace these (above) two lines with:

*// Delete item*

DeleteItemCommand delete\_item\_command = **new** DeleteItemCommand(**item\_list**, **item**, **context**);

delete\_item\_command.execute();

**boolean** success = delete\_item\_command.isExecuted();

**if** (!success){

**return**;

}

Now that we have implemented all the **ItemList** related commands, it is your turn to implement the **ContactList** related commands: **AddContactCommand**, **DeleteContactCommand**, and **EditContactCommand**. You will find that these commands are essentially analogous to **AddItemCommand**, **DeleteItemCommand**, and **EditItemCommand**, respectfully. Additionally ,the process involved in updating the application to accommodate these new commands is also completely analogous.

#### 9. Create and implement the AddContactCommand class

Create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**.

Name the class **AddContactCommand**. Hit **Enter**. This creates an empty **AddContactCommand** class.

Implement the **AddContactCommand** class. **Hint**: this step is analogous to **Step 3**.

#### 10. Update the ContactList class

Double click on the **ContactList** class to open it. Navigate to the **saveContacts()** method.

Update the current **saveContacts()** method. **Hint**: this step is analogous to **Step 4**

#### 11. Create and implement the DeleteContactCommand class

Create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**.

Name the class **DeleteContactCommand**. Hit **Enter**. This creates an empty **DeleteContactCommand** class.

Implement the **DeleteContactCommand** class. **Hint**: this step is analogous to **Step 5**.

#### 12. Create and implement the EditContactCommand class

Create a new class by right-clicking on the **com.example.sharingapp** folder, then click **New** → **Java Class**.

Name the class **EditContactCommand**. Hit **Enter**. This creates an empty **EditContactCommand** class.

Implement the **EditContactCommand** class. **Hint**: this step is analogous to **Step 6**.

Now that we have created all the Commands related to the **ContactList** class (**AddContactCommand**, **DeleteContactCommand**, and **EditContactCommand**) it is time to update our Activities to use these Commands instead of directly interacting with the **ContactList** model.

#### 13. Update AddContactActivity to use AddContactCommand

Double click on **AddContactActivity** to edit it. Locate the **saveContact()** method.

Update the **saveContact()** method. **Hint**: this step is analogous to **Step 7**.

#### 14. Update EditContactActivity to use DeleteContactCommand and EditContactCommand

Double click on **EditContactActivity** to edit it.

Locate the **saveContact()** and method. Update the **saveContact()** method.

Locate the **deleteContact()** and method. Update the **deleteContact()** method.

**Hint**: this step is analogous to **Step 8**.

#### 15. Run the app

Assuming you have correctly implemented the Command Design Pattern then at this point you should be able to run the app by clicking the **play** button.



Be patient! It may take a few minutes to open and launch SharingApp.