Assignment 5
CS 3443 Application Programming
Due: Monday, April 8

Community Events Calendar

A community events calendar app lists upcoming local events. The main screen of the app displays brief descriptions of events using a RecyclerView (Figure 1). Each brief description consists of an event's date, category, and name. The events can be sorted by date, by category and by name. When sorting by event date, ties are broken by the event category and then the event name. When sorting by event category, ties are broken by the date and then by the name. When sorting by event name, ties are broken by date and then by category. The user can select the type of sorting by clicking on a button below the event descriptions. Next to each brief event description is a button that displays (as a second activity) the full description of the event.

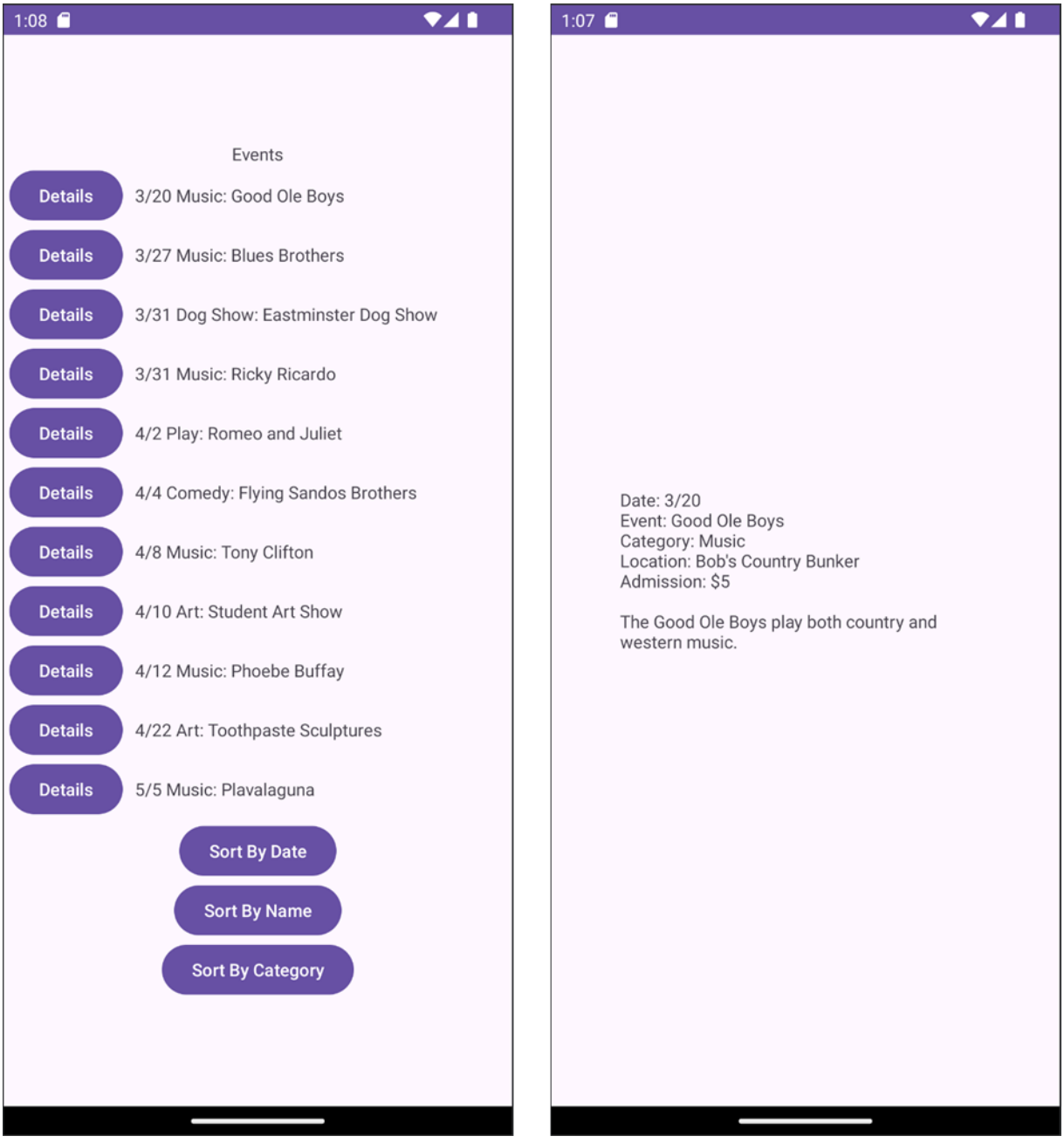


Figure 1. A community events app that can sort its events by date, category, and name (left). Clicking on the details button of an event displays detailed information about the event (right).

The event information for the app are in a comma-separated-values file containing one event per line. The order of the data on each line is: category, name, location, month, day of month, admission price, and a further description of the event.

Programming the Events Calendar

Create an Android Studio project named a5_abc123 where abc123 is your UTSA ID. For the minimum SDK, use API 33 Tiramisu.

This assignment requires three layouts: the main layout for the initial screen containing the RecyclerView, the layout for the details button and abbreviated description of each view within the RecyclerView, and the layout for a second activity that displays an event in detail. Included with this assignment are three XML files for these layouts. You may rename them.

Use the Model-View-Controller pattern and within your project package, create 3 subpackages named `model`, `view`, and `controller`.

Implement the packages and classes listed below. **Consult the UML class diagram at the end of this description for additional details to be included. You may implement additional methods in the classes.**

The `model` package is for code that reads events from an asset file and creates an `ArrayList` of the events

- The `CalendarEvent` class is for representing individual events. It has a getter for each of its instance variables. Its constructor takes values for each instance variable as arguments.
- The `EventListing` class reads a source of events and produces an `ArrayList` of `CalendarEvents`.

The `view` package is for code used to represent the View:

- The `ViewHolder` class extends `RecyclerView.ViewHolder` and is for accessing the views displayed by the `RecyclerView`. Each instance has access to the `TextView` and details `Button` for one event view.
    - The `ViewHolder` constructor inflates the layout for a view of an event and stores that view's `TextView` and details `Button`.
    - The `bind` method takes an instance of `CalendarEvent` and updates its view of the event. It sets its `TextView` to an abbreviated description of the event (see `EventView`) and sets the listener for its details `Button` (see `DetailsButtonListener`).
- The `Adapter` class extends `RecyclerView.Adapter<ViewHolder>` and keeps the `ArrayList` of `CalendarEvents` and acts as a mediator between the `RecyclerView` and the `ViewHolder`.
- The `EventView` class supports the different ways that event views are displayed.
    - The static `abbreviatedDescription` method takes a `CalendarEvent` as its argument and returns a string containing the abbreviated event description to be displayed in the `RecyclerView`. The abbreviated description consists of the event's date, category and name.
    - The static `fullDescription` method takes a `CalendarEvent` as its argument and returns a string containing the full event description to be displayed in the second activity as given in Figure 1.
    - The `sortByDate` method gets the `ArrayList` of `CalendarEvents` from the `Adapter` and sorts it by event date with ties broken by event category and then event name. After it is sorted, the zero-argument `notifyDataSetChanged` method of the `Adapter` must be called so that the `RecyclerView` can be updated. The `sortByDate` method should sort the `ArrayList` just once each time it is called.
    - The `sortByCategory` method gets the `ArrayList` of `CalendarEvents` from the `Adapter` and sorts it by event category with ties broken by event date and then event name. After it is sorted, the zero-argument `notifyDataSetChanged` method of the `Adapter` must be called so that the `RecyclerView` can be updated. The `sortByCategory` method should sort the `ArrayList` just once each time it is called.

    - The `sortByName` method gets the `ArrayList` of `CalendarEvents` from the `Adapter` and sorts it by event name with ties broken by event date and then event category. After it is sorted, the zero-argument `notifyDataSetChanged` method of the `Adapter` must be called so that the `RecyclerView` can be updated. The `sortByName` method should sort the `ArrayList` just once each time it is called.
- The `DateComparator` class implements `Comparator<CalendarEvent>` and is used for sorting `CalendarEvents` by event date with ties broken by event category and then event name.
- The `CategoryComparator` class implements `Comparator<CalendarEvent>` and is used for sorting `CalendarEvents` by event category with ties broken by event date and then event name.
- The `NameComparator` class implements `Comparator<CalendarEvent>` and is used for sorting `CalendarEvents` by event name with ties broken by event date and then event category.

The `controller` package has the Controller code.

- The `SortByDateListener` class is dedicated to listening to the sort-by-date button.
    - The `onClick` method just calls the `sortByDate` method of `mainController`.
- The `SortByCategoryListener` class is dedicated to listening to the sort-by-category button.
    - The `onClick` method just calls the `sortByCategory` method of `mainController`.
- The `SortByNameListener` class is dedicated to listening to the sort-by-name button.
    - The `onClick` method just calls the `sortByName` method of `mainController`.
- The `DetailsButtonListener` class is dedicated to listening to the details button.
    - The `DetailsButtonListener` constructor takes as its only argument a `CalendarEvent`.
    - The static `setMainController` method accepts a `MainController` instance.
    - The `onClick` method calls the `showDetails` method of `MainController` and passes to it its `CalendarEvent` instance.
- The `MainController` class has the following methods.
    - The `sortByDate`, `sortByCategory` and `sortByName` methods call their corresponding methods of the `EventView` class.
    - The `showDetails` method takes a `CalendarEvent` as its only argument and starts a second activity to show the full description of the event. The `EventView.fullDescription` method is called to create a string for the full description which is passed as an extra in the `Intent` instance used to start the second activity.

Note that the second activity that displays the details of the event is passive; it has no buttons and only a single TextView. Hence, there is no need for a controller for it. In addition, the second activity passes no information back to the initial activity.

Programming Style

Use good programming style. Your comments should include a description of each class and of each method in Javadoc. Your implementation should follow the DRY principle.

Testing

For testing, use the Medium Phone and the Tiramisu (API Level 33) system image.

Test your methods individually before testing the entire app. Use the events.csv file as part of your testing.

Deliverables

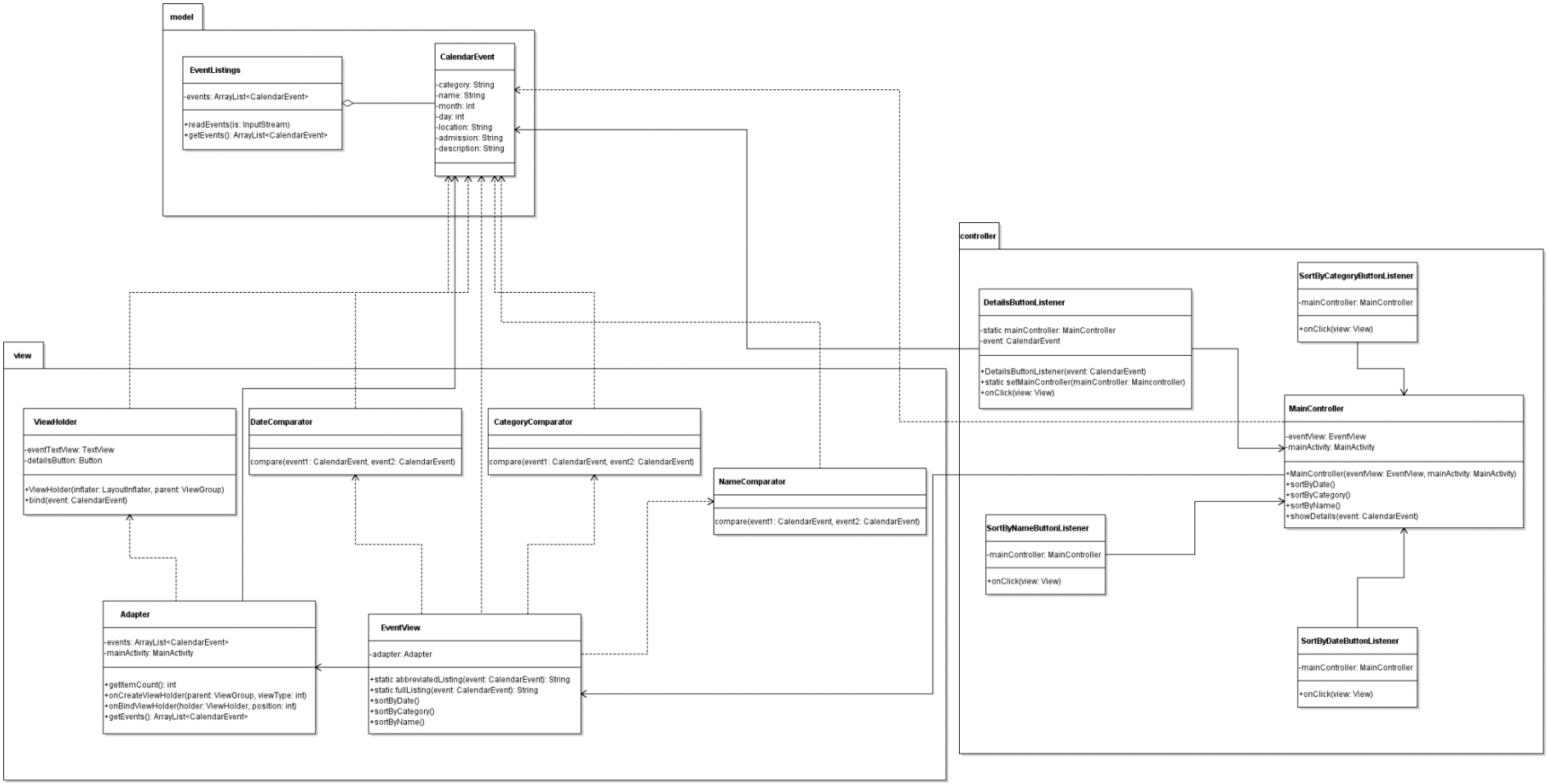
- Create an Android Studio project named eventcalendar.
- Implement the classes as described. You may define additional methods in the classes.
- Export your project to a zip file named a5_abc123.zip where abc123 is your UTSA student ID.
- Verify that the zip file contains your work.
- Submit your zip file on Canvas

- Verify that you have submitted your zip file correctly.

Rubric

1.	Good programming style: Javadoc comments, indentation, names, DRY	10%
2.	Correctly created Model	20%
3.	Correctly created View	40%
4.	Correctly created Controller	30%
	Total	100%

No credit is given for submissions that do not compile.


model
EventListings
-events: ArrayList<CalendarEvent>
+readEvents(is: InputStream)
+getEvents(): ArrayList<CalendarEvent>
CalendarEvent
-category: String
-name: String
-month: int
-day: int
-location: String
-admission: String
-description: String
controller
DetailsButtonListener
-static mainController: MainController
-event: CalendarEvent
+DetailsButtonListener(event: CalendarEvent)
+static setMainController(mainController: Maincontroller)
+onClick(view: View)
SortByCategoryButtonListener
-mainController: MainController
+onClick(view: View)
MainController
-eventView: EventView
-mainActivity: MainActivity
+MainController(eventView: EventView, mainActivity: MainActivity)
+sortByDate()
+sortByCategory()
+sortByName()
+showDetails(event: CalendarEvent)
SortByNameButtonListener
-mainController: MainController
+onClick(view: View)
SortByDateButtonListener
-mainController: MainController
+onClick(view: View)
view
ViewHolder
-eventTextView: TextView
-detailsButton: Button
+ViewHolder(inflater: LayoutInflater, parent: ViewGroup)
+bind(event: CalendarEvent)
DateComparator
compare(event1: CalendarEvent, event2: CalendarEvent)
CategoryComparator
compare(event1: CalendarEvent, event2: CalendarEvent)
NameComparator
compare(event1: CalendarEvent, event2: CalendarEvent)
Adapter
-events: ArrayList<CalendarEvent>
-mainActivity: MainActivity
+getItemCount(): int
+onCreateViewHolder(parent: ViewGroup, viewType: int)
+onBindViewHolder(holder: ViewHolder, position: int)
+getEvents(): ArrayList<CalendarEvent>
EventView
-adapter: Adapter
+static abbreviatedListing(event: CalendarEvent): String
+static fullListing(event: CalendarEvent): String
+sortByDate()
+sortByCategory()
+sortByName()