**Project 4**

**Due Date: Sunday, April 28 - no due date extension for this Project!!**

**Larger Group Project**

**Objectives**

Familiarity with

* event-driven program execution
* state dependence in application (the **State** design pattern)
* **Model-View-Adapter** architecture
* Android framework

**Description**

In this project, you will work in your large Groups (from the **UML** class diagram in-class exercise) to implement a very simple timer as an Android application. There is no explicit Project link for this project, but you should **start from this example** [**stopwatch**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fstopwatch-android-java&sa=D&sntz=1&usg=AOvVaw0yv3125ZUCZGJR85Y8u7tz) **IntelliJ IDEA project:** [**https://github.com/LoyolaChicagoCode/stopwatch-android-java**](https://github.com/LoyolaChicagoCode/stopwatch-android-java).

You may also find parts of this [**clickcounter**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fclickcounter-android-java&sa=D&sntz=1&usg=AOvVaw0wruIrirOurBhmqdBL6R1F) IntelliJ IDEA project helpful for managing the Activity life cycle, especially playing a sound as part of the Activity (the "beep" needed in Project 4), and possibly saving and restoring state (eg, when the device is rotated): [**https://github.com/LoyolaChicagoCode/clickcounter-android-java**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fclickcounter-android-java&sa=D&sntz=1&usg=AOvVaw0wruIrirOurBhmqdBL6R1F).

**Go on to the next pages for Functional and Non-Functional Requirements and other information.**

**Functional Requirements (55%)**

The timer has the following controls:

* (0.5) One two-digit display of the form 88.
* (0.5) One multi-function button.

The timer behaves as follows:

* (0.5) The timer always displays the remaining time in seconds.
* (0.5) Initially, the timer is stopped and the (remaining) time is zero.
* (0.5) If the button is pressed when the timer is stopped, the time is incremented by one up to a preset maximum of 99. (The button acts as an **increment** button.)
* (0.5) If the time is greater than zero and three seconds elapse from the most recent time the button was pressed, then the timer beeps once and starts running.
  + Note: if the time reaches the preset maximum of 99 the timer acts the same way as if three seconds had elapsed - it beeps and starts running.
* (0.5) While running, the timer subtracts one from the time for every second that elapses.
  + In particular, the display only changes 1 second after the 3-second timeout occurs or the timer value reaches 99.
* (0.5) If the timer is running and the button is pressed, the timer stops and the time is reset to zero. (The button acts as a **cancel** button.)
* (0.5) If the timer is running and the time reaches zero by itself (without the button being pressed), then the timer stops and the alarm starts beeping continually and indefinitely.
* (0.5) If the alarm is sounding and the button is pressed, the alarm stops sounding; the timer is now stopped and the (remaining) time is zero. (The button acts as a **stop** button.)
* (0.5) The timer handles rotation by continuing in its current state. **Note: if you use** [**stopwatch**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fstopwatch-android-java&sa=D&sntz=1&usg=AOvVaw0yv3125ZUCZGJR85Y8u7tz) **as the base for your timer project you will get these 0.5 points automatically.**

**Extra Credit (worth 20%)**

* **(2.0 extra credit)** Add a 2-digit editable text area, editable only when the timer is stopped, where the user can type in the time and then press enter or click the button to start the timer.
  + Note that this will require the emulator or device to have a physical or on-screen keyboard to enter the digits, and it is a non-trivial addition to the project!!

**Nonfunctional Requirements (25%)**

* (2.0) Follow the design principles discussed so far. Maintain a clear, responsibility-based separation among the different building blocks. It is recommended that you start with the [stopwatch](https://www.google.com/url?q=https%3A%2F%2Fbitbucket.org%2Floyolachicagocs_comp313%2Fstopwatch-android-java&sa=D&sntz=1&usg=AOvVaw24nsomh2ayO7MM6i78vFJV) example.
  + The adapter should be lean (as in the stopwatch example).
  + Most of the complexity should be buried in the model (also as in the stopwatch example).
  + Unlike the stopwatch example, the dynamic model (state machine) has guards involving the current time. Think carefully how to inject the dependency on the time model into the state machine. Do not use static members!
  + **(0.5 extra credit)** Use of the state pattern (APPP chapter 36) is recommended. **If you reuse and update the** [**stopwatch**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fstopwatch-android-java&sa=D&sntz=1&usg=AOvVaw0yv3125ZUCZGJR85Y8u7tz) **example, you will likely get this extra credit.**
  + Generally follow good Android development practice: [**http://developer.android.com**](http://developer.android.com/)
  + Use this Timer class: [**http://developer.android.com/reference/java/util/Timer.html**](http://developer.android.com/reference/java/util/Timer.html)
* (0.5) For beeping, use media playback to play a notification sound (or some other suitable mechanism):
  + [**http://stackoverflow.com/questions/10335057/play-notification-default-sound-only-android**](http://stackoverflow.com/questions/10335057/play-notification-default-sound-only-android)
  + use getApplicationContext() to obtain the required context reference
  + note that [**clickcounter**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fclickcounter-android-java&sa=D&sntz=1&usg=AOvVaw0wruIrirOurBhmqdBL6R1F) plays a sound, so you can review how it does that
* **(1.0 extra credit)** Ensure your application is testable.
  + Ensure your application includes comprehensive unit, integration, and functional tests using techniques from the clickcounter and stopwatch examples where appropriate. (See also [**http://developer.android.com/tools/testing/testing\_android.html**](http://developer.android.com/tools/testing/testing_android.html))

**Written Part/Documentation (20%)**

Please include these deliverables in the **doc** folder of your project 4.

* (0.5) Include the extended state machine model from a future in-class group activity. (Minimally, a cell phone scan of your drawing is acceptable.)
* (0.25) Use inline comments to document design details in the code.
* (0.25) Add javadoc comments to document how to use the abstractions (interfaces, classes) you designed. You will not have to create javadoc HTML.
* Include a brief (300-500 words) report on
  + (0.5) Your Group development journey during this project. Focus on aspects you find noteworthy, e.g., process, pairing, testing, design decisions, refactoring, use of the repository.
  + (0.5) The relationship between your extended state machine model from this project and your actual code. Possible talking points are:
  + What are the similarities and differences between model and code?
  + Is it more effective to code or model first?
  + Now that you have code, what, if any, changes would you make to your model?

**Grading Criteria**

* Stated percentages above for the major categories
* Stated points above for the specific items
* Total 10 points with opportunities for up to 2.5 points of extra credit (0.5 for using the State Design Pattern, and 2.0 if you make the 2-digit input area an editable text area)
* Deductions of up to 1 point for
  + deviation from the required project folder structure (see [**stopwatch**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fstopwatch-android-java&sa=D&sntz=1&usg=AOvVaw0yv3125ZUCZGJR85Y8u7tz)) - if you start from stopwatch, as you should, do not change its overall structure (you may change some class names, in particular the names of the timer states!)
  + inability to run and/or test if you have implemented timer tests

**How to submit**

As the first step in working on this project, one of your larger Group members will import the [**stopwatch**](https://www.google.com/url?q=https%3A%2F%2Fgithub.com%2FLoyolaChicagoCode%2Fstopwatch-android-java&sa=D&sntz=1&usg=AOvVaw0yv3125ZUCZGJR85Y8u7tz) repository to a private Bitbucket repository shared among all of you and your instructor. The name of the repository shall be ***cs313413sp24groupNp4***, where N is your group number found in the online class slides from Week 9.

When your work is ready to be graded, please notify your instructor by submitting a comment in the Sakai Project 4 assignment or a Piazza note, including your Bitbucket repo’s HTTPS URL.