Lab 04 Git Merge and SurfaceView

In this lab, we will continue to work with the Birthday Cake app. Between you and your partners, decide who will be Person A, B, and C.

This lab has two disparate topics:

1. The most common type of event on mobile computers is the “touch” or “tap.” In this lab you’ll practice responding to such events.
2. In software projects where multiple programmers are accessing the same source code, they occasionally enact simultaneous changes to the same files. Source control handles this with conflict resolution and merging. You will practice doing that today.

# Merging without Conflicts

To begin, all of you should be collaborators on the same GitHub repository that contains a working version of this app with the features that were implemented in the previous lab. Take the time now for one person to add the other partners as collaborators. Remember, that when you invite someone as a collaborator, they must accept the invitation (sent via email) before they can actually collaborate.

Each person should checkout this repository and verify that it is a working Android project. Do not checkout the project by downloading a .zip file. Instead, import it via the Android Studio Welcome screen. (You may have to close your current project to get to this screen.) Follow these steps:

1. Find the main GitHub page for the project. Select Clone or Download on the right-hand side. Do not select Download ZIP. Instead, copy the URL.
2. On the Android Studio welcome screen, select: Check out project from Version Control→Git. A Clone Repository dialog will appear:
   1. URL: <paste the URL you copied>
   2. Directory: select a new folder on your system
3. Hit the Clone button. You will be asked if you want to open the project. Say, yes.

Now verify that all of you have the same code checked out and can run the app.

Each person will now each make a different minor change to the birthday cake app.

**Person A:** Change the name of the Blow Out button to be Extinguish instead. Then wait until your other teammates are done with their changes.

**Person B:** Change the candles so they are noticeably wider. Then wait until your other teammates are done with their changes.

**Person C:** Change the color of the cake (the magenta part). Then wait until your other teammates are done with their changes.

Important: Once you are all finished, person A should check their files into GitHub *first* while the other teammates wait. Remember to put a clear, descriptive comment on your commit and don’t forget to push. Before proceeding, verify that the change is on GitHub.

Now, person B should commit and push their changes to GitHub. Upon doing the push, you should receive an error message:

*Push of current branch master was rejected. Remote changes need to be merged before pushing.*

This has occurred because the repository was changed since the last time person B checked it out. Select the Merge button to merge person B’s changes with person A’s changes. (A “Push Rejected” message bubble may be in the bottom right; you can ignore it.) Before proceeding, verify that both changes are now on GitHub.

|  |
| --- |
| **Oops!**  If you do not receive the push rejected message above, then you’ve likely failed to follow the instructions correctly. To help you troubleshoot:   * Verify that all members of the team are collaborators (not just invited to be) * Did you check out the repository using its URL rather than a .zip file? * Did you use the same URL? * Did person A remember to push their changes (not just commit them)? * Did person B do a “pull” at any point? They should not have.   Once you know what happened. Repeat the process with similar changes to the code until a merge occurs as it should. |

Then person C should repeat person B’s steps.

Finally, person A and B should retrieve person C’s change by doing a Pull. Select VCS→Git→Pull from the menu.

**Checkpoint 1 (30 points):** Show the instructor or lab assistant that the merged app is running for everyone on their respective tablets. Then, show the completed merge as it is logged on GitHub.com.

# Touch Events

Everyone will modify the birthday cake app to respond to user touches on the CakeView, but they will each have somewhat different tasks. Remember, responding to events in an Android app requires you to follow three steps:

1. Identify the specific type of event you want to handle and the Java interface you must implement to handle it. **Tip:** For touch events, this is the OnTouchListener interface.
2. Implement the interface in a particular class within your program and implement the required methods in that class.
3. Register a listener object with the source of the events.

Regardless of which person you are:

* your implementation must make proper use of the model, view and controller
* If you need to create any new Paint objects, do not create them in your onDraw() method. (Why is this bad practice?)

Do not check your change into GitHub when you finish! Wait for your teammates to finish this checkpoint.

**For all of the items below, the element should not appear on the screen when the user starts the app. It should appear for the first time when the user touches the screen.**

**Person A:** Modify the birthday cake app so that it reports the x,y location where a touch occurs. You should use the Canvas.drawText() method to “draw” this information in the lower right-hand corner of the CakeView canvas in large red text. (Don’t add a new View to your layout.) It is ok to hardcode this coordinate for this current purpose.

*Hint for person A:* You will need to use the Paint.setTextSize() method.

**Person B:** Modify the birthday cake app so that when the user touches the drawing, a blue balloon with a string is drawn in that location. Your balloon cannot be perfectly round but, instead, have a more “balloon-like” shape. Each subsequent touch causes the balloon to move to a new location. Thus, there should never be more than one balloon on the screen at the same time. The **center** of the balloon should be where the user touched the screen.

**Person** **C:** Modify the birthday cake app so that when the user touches the drawing, a green and red checkerboard pattern (as shown below) is drawn in that location. Each subsequent touch causes the pattern to move to a new location. Thus, there should never be more than one checkerboard pattern on the screen at the same time. The **center** of the pattern should be where the user touched the screen.

|  |  |
| --- | --- |
|  |  |
|  |  |

**Checkpoint 2 (40 points):** Show your instructor or lab assistant the change you and your partner made to the app. Then explain how you complied with the model-view-controller pattern in your implementation.

# Git Merge

Do not begin this section until everyone has finished with the previous checkpoint. Then work together to do this section.

Person A should begin by checking their changes into GitHub (add, commit and push). Verify on GitHub that person A’s changes are there.

Next:

1. Person B should commit their changes and then attempt to Push them. This push should fail with the same message as before.
2. Press the Merge button on the dialog as you did before. This time, the computer is unable to do the merge for you because some changes occurred in the same place in the same file. A Conflicts dialog will appear with a list of files that have conflicts. It’s likely all four of your .java files will need attention but perhaps only two or three.
3. For each file:
   1. select the file and hit Merge… A Merge Revisions window will appear. The left pane will show what person A checked in. The right pane will show what person B wants to check in. The center pane contains the merged version of the two.
   2. Use the provided controls to create a merged version of the file. You will likely need to discuss as a team what you want to do here. When you are finished, click the Apply button.
4. Once all files have been merged, you will be returned to the main screen.
5. Run the app. Both partners’ new functionality should now be present together in the same app. If not, you will need to fix it and commit again.
6. Repeat the Push operation. If you’ve done your task correctly, it should succeed.

Then repeat the same for Person C.

Now persons A and B should be able to perform a Pull from their computer and get the latest merged code as well.

**Checkpoint 3 (30 points):** Show the instructor or lab assistant that the merged app is running for all teammates on their respective computers/tablets. Then, show the completed merge as it is logged on GitHub.com.

That’s all folks! Congrats on completing lab 4! 