Phase 2 Code Review

Sonia Sun (u1428723)
partner: Beatrice Dande

1 Code to Improve

Each person picks a piece of code from your own project (class, method, etc) that you think can be improved. Discuss that code in detail, improve it, and commit those improvements (ie, refactor something with a helper that has fresh eyes)

1.1 Sonia's Project

The following code snippet

```
fun undoLastAction() {
    if (paths.isNotEmpty()) {
        val lastItem = paths.last()
        val lastPath = lastItem.first
        val lastPathProperty = lastItem.second
        paths.remove(lastItem)
        pathsUndone.add(Pair(lastPath, lastPathProperty))
    }
}
```

can be simplified to:

```
fun undoLastAction() {
    if (paths.isNotEmpty()) {
        val lastItem = paths.removeAt(paths.size - 1)
        pathsUndone.add(lastItem)
    }
}
```

You can find the commit here:
https://github.com/YuanRuQian/CS6018-Group-Project/commit/11b87db2f38b1a13c9dc43190e6ff19879d86173

1.2 Beatrice's Project

The logic of the following function is more complicated than it should be.

```
   fun getBoardData(): Board {
        if(!this::board.isInitialized) {
            return if(currentBoardName == DynamicConfig.currentEditBoard) {
                board
            } else {
                board = reloadBoard()
                currentBoardName = DynamicConfig.currentEditBoard
                board
            }
        }
        else {
            board = reloadBoard()

            currentBoardName = DynamicConfig.currentEditBoard
            return board
        }

    }
```

We improve it to:

```
fun getBoardData(): Board {
    if (!this::board.isInitialized || currentBoardName !=
DynamicConfig.currentEditBoard) {
        board = reloadBoard()
        currentBoardName = DynamicConfig.currentEditBoard
    }
    return board
}
```

You can find the commit here:
https://github.com/xxsyang/cs6018/commit/15e707db07df9df0a4f9382f844dbe91545a3c3f

2 Similar Functionality

Pick a piece of functionality that both of your projects include. Discuss the similarities/ differences between your implementations.

Functionality:
Both of our drawing applications display a list of historical drawings on the entry page. This list serves as an overview of previously created artworks.

Similarities:

- We both use `LazyColumn` composable to display the list of drawings
- Each item in the list is clickable. Clicking on an item will lead the user to the associated drawing.

Differencies:

- We present the drawings in descending order based on their last modified date, their application displays drawings in ascending order, ordered by creation time.
- For each item in the `LazyColumn`, we use `ElevatedCard` to display the drawing title, last modified date, and thumbnail. In contrast, they use `Text` to display each drawing's title.
- We store each item in a separate `DrawingListItem` class. They store each item in a list of strings, specifically filenames.
- Our application supports the removal of items from the `LazyColumn` through a left-swipe action.

3 New Test

Pick part of your code that you don't think is well tested. Work with your partner to add more tests to exercise that part of your project and commit them to your repo

3.1 Sonia's Project

We added a test for updating the drawing title. See the commit here:
https://github.com/YuanRuQian/CS6018-Group-Project/commit/235e6426d764faf17bb087afc822578fc659887f

```
@Test
    fun testAddAndDeleteAndUpdateADrawing() {
        runBlocking {
            val lifecycleOwner = TestLifecycleOwner()
            val info = DrawingInfo(Date(), Date(), "TestImage", null, null)
            var count = 0
            lifecycleOwner.run {
                withContext(Dispatchers.Main) {
                    val allDrawing = dao.allDrawingInfo().asLiveData()
                    allDrawing
```

```
                        .observe(lifecycleOwner, object :
Observer<List<DrawingInfo>> {
                            override fun onChanged(value:
List<DrawingInfo>) {
                                when (count) {
                                    1 -> {
                                        Log.d("DBTest", "1-add test")
                                        Assert.assertEquals(1, value.size)
                                        Assert.assertEquals("TestImage",
value[0].drawingTitle)
                                    }
                                    2 -> {
                                        Log.d("DBTest", "2-update title")
                                        Assert.assertEquals(1, value.size)
                                        Assert.assertEquals("New Title",
value[0].drawingTitle)
                                    }

                                    3 -> {
                                        Log.d("DBTest", "3-delete test")
                                        Assert.assertEquals(0, value.size)
                                        allDrawing.removeObserver(this)
                                    }
                                }
                            }
                        })

                    dao.addDrawingInfo(info)
                    count += 1
                    dao.updateDrawingInfoTitle("New Title", 0)
                    count += 1
                    dao.deleteDrawingInfoWithId(0)
                    count += 1
                }
            }
        }
    }
```

3.2 Beatrice's Project

We wrote a database test for the function `InsertAll()`. See the commit here: https://github.com/xxsyang/cs6018/commit/1d36c1bdc5585e66184ff2e1e03122b3d1c1ac6c

```
@Test
    fun testInsertAll() {
        runBlocking {
            val newDrawing = Item(name="Drawing1", image = ByteArray(10))
            val lifecycleOwner = TestLifecycleOwner()
            var count = 0
            lifecycleOwner.run {
                withContext(Dispatchers.Main) {
                    val _allDrawing =
MutableLiveData<List<Item>>(dao.getAll())
                    val allDrawing: LiveData<List<Item>> = _allDrawing
                    allDrawing.observe(lifecycleOwner) {
                        if (count == 1) {
                            Assert.assertTrue(it.contains(newDrawing))
                        }
                    }
                    dao.insertAll(newDrawing)
                    count += 1
                }
            }
        }
    }
```