# A1: Fish Tank

**Due** Oct 14, 2019 by 11:59am **Points** 10

In Assignment 1, you will clone a mostly-working but badly-designed and buggy Android program that shows a fish tank containing items such as fish, seaweed, and bubbles. You will refactor it thoroughly to remove bugs and design flaws, then add one more kind of fish tank item (you choose!), perhaps crabs or snails or eels or rocks or bubblers.

**Have an Android phone? You can run apps!** Now you can have a beautiful fishtank on it and show it to your friends and family. Here is the **Run apps on a hardware device (https://developer.android.com/studio/run/device)** page from Google. You need to enable USB debugging and connect it to your laptop, and then you can select your phone as the target device instead of the Pixel 3 emulator. (Note that it may crash on your particular device; it really is not a good program. If that's the case, it's sometimes because of an invalid index, and that happens because the array size calculations aren't forgiving. If that happens, it's because a method called `createTankItems` is creating fish at indexes outside the bounds of the array on your screen. To fix it, simply change the indexes. This won't affect the grading at all. You can switch to choosing random valid starting locations if you like. To make debugging easier, you might only create a single fish (but please create a nice number in the final version that you push), or you might create hundreds randomly-placed items to see what happens.

**You will make many changes to this program.** There are no autotests that you need to satisfy. Look for things that you don't like and fix them. We give you a lot of guidance and suggestions in this handout.

We expect this to take you 8 to 10 hours (very roughly — there is wide variation). As a result, we are extending the due date to Mon 7 Oct at noon.

## Getting ready for the project

The main goal of this assignment is for you to get ready for the project so that you can meaningfully contribute. This means you need to work on three things: understanding and using Git, getting better at Java, and using Android Studio efficiently. Use this assignment to practice all three!

For Android Studio, learn the fancier features — **please do this on a different project, or do it after you've completed Steps 1 and 2, below**.

- Click on a method, class, or variable in your code and select menu item Navigate —> Declaration to jump to where the name is declared. Right-clicking and selecting Go To —> Declaration works as well. Check out the other Navigate menu items.
- Right-click on a name and try Find Usages.

- Right-click and Analyze —> Inspect Code. Expand the results and right-click on any of them. Note that you can have Android Studio fix the issue for you! Make sure you understand the suggestion before you accept it.
- For many more tips, watch the animations on the **Android Studio Tips and Tricks (https://www.raywenderlich.com/2807578-android-studio-tips-and-tricks)** website.

## Learning objectives

By the end of this assignment, you will have done the following:

- Worked with professional tools that require some setup before you start to work.
- Installed Google's style guide in Android Studio
- Worked with Android Studio's code formatter
- Used Git to regularly commit changes
- Used Git to push your project to MarkUs — **do this regularly!** Commits that are not pushed before the deadline will not be marked.
- Learned about, identified, and fixed Code Smells (see below)
- identified design flaws and refactored the code to fix them
- added features to the refactored code

## The first things you need to do

You must do the following steps before you make other changes to the code. **More detailed instructions are below in the "Getting set up" section.**

Clone the FishTank project from your MarkUs repository:

https://markus.teach.cs.toronto.edu/git/csc207-2019-09/YOUR_UTORID_HERE

1. Start with all other projects closed.
2. Select Check out project from Version Control —> Git. Copy and paste the MarkUs URL and replace YOUR_UTORID_HERE with … well, you can probably guess.
3. You'll be asked whether you want to Checkout from Version Control. Yes.
4. Create project from existing sources and click Next.
5. For Project location, append FishTank/FishTank and click Next.
6. Click Next to add the Java source files.
7. For the Gradle-wrapper library, click Next.
8. To accept the module structure, click Next.
9. You'll be asked to select the project SDK. Choose Android API 29 Platform and click Next.
10. Click Finish.

That should do it. You should now be able to build and run. Try it! You should be greeted by a lovely fish tank that uses rather advanced graphics.

**Don't edit any code yet! We have a few more things that we want you to do first.**

## Getting and running the code

Inside your repository is a directory called `FishTank`. In that directory is a Java program.

**Don't change the code yet.**

Run the program. You should see a window containing fish, bubbles, and seaweed.

Click the Logcat tab at the bottom of the window. You'll see lines like these:

```
        2019-09-25 21:39:16.869 1532-1575/uoft.csc207.fishtank I/System.out: Turned around>
        2019-09-25 21:39:16.870 1532-1575/uoft.csc207.fishtank I/System.out: 17 15
```

Every now and then, you'll see an exception like this one:

```
2019-09-25 21:40:03.640 1651-1688/uoft.csc207.fishtank W/System.err: java.lang.ArrayIndexOutOfBound
sException: length=33; index=36
2019-09-25 21:40:03.640 1651-1688/uoft.csc207.fishtank W/System.err: at uoft.csc207.fishtank.Fish.b
lowBubble(Fish.java:59)
2019-09-25 21:40:03.640 1651-1688/uoft.csc207.fishtank W/System.err: at uoft.csc207.fishtank.Fish.m
ove(Fish.java:147)
2019-09-25 21:40:03.641 1651-1688/uoft.csc207.fishtank W/System.err: at uoft.csc207.fishtank.FishTa
nkManager.update(FishTankManager.java:71)
2019-09-25 21:40:03.641 1651-1688/uoft.csc207.fishtank W/System.err: at uoft.csc207.fishtank.FishTa
nkView.update(FishTankView.java:93)
2019-09-25 21:40:03.641 1651-1688/uoft.csc207.fishtank W/System.err: at uoft.csc207.fishtank.MainTh
read.run(MainThread.java:47)
```

You'll fix this bug as you work through the assignment. **Don't debug it yet, though.** It's likely that it'll disappear because of other refactorings that you're doing to the code.

## Installing a Google style plugin in Android Studio

Remember how you can reformat code using Android Studio using menu `Code → Reformat Code`? That operation can be customized using a plugin.

Google, like most software companies, has **a style guide (https://google.github.io/styleguide/javaguide.html)**. There is a plugin for Android Studio that implements this style guide.

1. Open up the Android Studio settings/preferences, click Plugins, then click the Marketplace tab.
2. Search for `google-java-format`, install the plugin, and enable it for the project.
3. You may have to restart Android Studio.

If you need help, search the course discussion boards and the web, and ask if you can't find an answer, ask!

## What to do and taking notes

You must do this assignment one step a time, **committing and pushing your changes whenever you finish a step**.

### Step 1: Create notes.txt

In Android Studio, locate the `notes.txt file in the app directory of the project.`

Copy and paste this into your file:

```
#########
# Step 1
Code affected: none
Created this file.
```

Add and commit it with this commit message: `"Added notes.txt for keeping track of changes during a1."`. Copy and paste, don't type it yourself! Then push your changes.

**Sanity check:** Clone your repo again, somewhere else on your hard drive, and make sure that `notes.txt` exists and has the expected contents. If it doesn't, keep trying until it does. Ask for help if you need it.

### Step 2: Reformat the starter code

Open each .java file in turn and select `Code->Reformat Code`, or use the keyboard shortcut. Save all the files as you go. Don't fix bugs, don't manually add spaces, don't move anything around. You'll get a chance to do that soon.

Run the program to make sure that it still works.

Make a note in `notes.txt` that you reformatted all the source code. Copy and paste this as your note underneath your Step 1 note:

```
#########
# Step 2
Code affected: all .java files
Reformatted using the Google Java style formatter.
```

Git add each `.java` file, and also `notes.txt`. Commit and push.

It's up to you whether you want to do a sanity check, but we recommend it.

Notice that, so far, everyone's repositories should have exactly the same changes in them, and your `notes.txt` file should look like this:

```
#########
# Step 1
Code affected: none
Created this file.

#########
# Step 2
Code affected: all .java files
Reformatted using the Google Java style formatter.
```

## The starter code

Here are all the Java files in the project. All are in package uoft.csc207.fishtank.

Here are Classes that describe and manage items in the fish tank. **These are the only Java files you will modify.**

- **Fish**: a fish that swims around and occasionally blows a bubble.
- **Bubble**: a bubble in the tank. Bubbles occasionally grow bigger as they rise in the tank.
- **Seaweed**: a piece of seaweed that waves gently in the currents.
- **HungryFish**: a larger fish that otherwise behaves like a Fish.
- **FishTankManager**: keeps track of the items in the fish tank.

Here are the classes that are responsible for displaying the fish tank. **You do not need to modify these.**

- **MainActivity**: the main program. It's quite short.
- **FishTankView**: The Android view that displays the fish tank.
- **MainThread**: Manages threading and updates.

## Step 3: fix a simple flaw in class Fish

There are many design flaws in this code. Here is a simple one: method `turnAround` in class `Fish` has duplicate code and an unnecessary `if` statement. Fix it: that method body only needs to be 2 lines long, and it doesn't need an if statement.

Add this note to `notes.txt`:

```
#########
# Step 3
Code affected: Fish.turnAround
Flaw: unnecessary if, leading to duplicate code.
Fix: replaced if statement with a single assignment statement.
```

That is the format that your notes should follow: the list of classes and methods that are affected, the flaw that you addressed, and a note about how you fixed it. The flaw and the fix should be concise: no more than a couple of sentences each. **The TAs will be using these notes as the primary guide to marking your assignment, so be clear in your communication.** If we can't understand your commit messages, we can't give you an accurate mark. Feel free to use your college's writing centre if you are unsure of your writing skills!

Add and commit `notes.txt` and `Fish.java` (with a clear commit message), and push.

## Step 4 to Step N-2

This is where you will do the bulk of the work. Continue identifying flaws and their fixes and **fix one flaw per push**. After each fix, make a clear note in `notes.txt`, then add, commit (with a clear commit message), and push your changes.

For each flaw, **figure out what the code is doing, then decide on a way to improve it**. Don't skip the understanding part — step through the code in the debugger to get a feel for what it's doing, then ask yourself whether there's a better way. Most of the time, there is!

There are quite a few flaws in the code, and these steps will contain the bulk of your work.

### A word about commits

You get to decide the order in which you will fix things, and you get to decide what a reasonable size commit is. The one rule is that your program should still compile and run. If it doesn't compile or it doesn't run, don't commit it!

Sometimes a commit will only be for a single method, perhaps to eliminate a convoluted if statement or while loop. Sometimes it will affect many files, perhaps if you introduce inheritance to greatly improve the code.

Don't stress about making a mistake when you commit. You won't lose any marks for a few silly mistakes, especially if you demonstrate by the end of the process that you've learned the git workflow.

Please feel free to refactor liberally. For example, if code is hard to read because it's convoluted, figure out what it does and then figure out a simpler way to write it. If you need to add parameters to a `move` method in order for a fish tank item to know when it's reached the edge of the screen, add them.

Android Studio finds many such problems. Look for the short dashes down the right of the code screen. Hover over them.

- chances to use inheritance to eliminate redundant code — introduce a superclass (or more than one!)
- a weird and inconsistent coordinate system — clean it up!
  - you should standardize on a single coordinate system and make sure all the fish tank items use the same one. Note that introducing inheritance can make this much easier.

- bad names — change them! (fix them as you find them; you don't even need to describe this fix in a commit message, unless it was the main reason for a commit)
- nearly-duplicate code — refactor!
- ugly code — clean it up!
- bad visibility modifiers — change them!
- recalculating a value over an over rather than using a temporary variable — clean it up!
- a **huge** waste of processing time looping over a *sparse* (nearly-empty) 2D array — switch to using a `java.util.List` of fish tank items instead.

## Step N-1: fix the bug/memory leak

By now, you should know the code quite well. The bug described at the top of the handout happens when a fish swam outside the window and blows a bubble, attempting to create it with a negative index in the 2D array. When you replaced the 2D array, this stopped happening.

You might still have a memory leak related to this: `Bubble`s float above the window forever, as long as something points to them. Fix this by either preventing creatures from moving outside the window (fish, for example), or removing them from your data structures when they float too high (bubbles) so that they get garbage collected. Use common sense to decide which creatures fall into each category.

Describe your fixes in `notes.txt`, and add, commit, and push your changes. If you fixed it in an ealier step, say which one.

## Step N: add a new fish tank item

Now that your code is beautiful, design and add at least one new item to your fish tank. It doesn't have to be particularly complicated, but your item **must** interact with other items in the fish tank. For example, you might make a snail that moves around the bottom of the tank and eats pieces of seaweed, a shark that chases nearby fish and eats them, or a new kind of fish that swim around together in a school. There are many possibilities. We are running **MOSS** (http://theory.stanford.edu/~aiken/moss/) to detect similarities, so please don't copy code from each other.

Make a final note in `notes.txt` describing your new item: what is it, and what behaviour does it have? There isn't a flaw, so just describe your change in a concise paragraph or bullet-point form after the `# Step N` note header.

## Step N+1: Code Smells

Many of the flaws in the starter code are "code smells". Consider what happens when food starts to go bad in your fridge: the fridge often starts to smell before you find any bad food. Likewise, in code, there are certain features that will continue to make your coding more and more difficult to work with until you fix them. We call these code smells.

There are many websites that describe code smells. Our current favourite is called **Code Smells, published by Source Making** **(https://sourcemaking.com/refactoring/smells)**. For your last step, find a code smell that is still present in your code, fix it, and describe it in `notes.txt`.