Nathan Brathwaite, Zachary Aird, Khareen Proverbs

[Email address]

Abstract

Analyze a piece of code given and then improve upon it and create a better code that follows greater coding standards

Software Architecture

Project Report

Contents

[Code Smells of Runtime Project 3](#_Toc60063235)

[Improvements 6](#_Toc60063236)

[CODE SMELL SEASIDE DEVELOPMENT 10](#_Toc60063237)

[FUTURE PROGRESS 11](#_Toc60063238)

[CONCLUSION 11](#_Toc60063239)

# Code Smells of Runtime Project

# **GITHUB LINK:**

[Swen-Third-Year/ZOO-MANAGEMENT-SYSTEM (github.com)](https://github.com/Swen-Third-Year/ZOO-MANAGEMENT-SYSTEM)

**Class Issues**

* The program contains a superclass known as “Animal” and has 4 subclasses that are based on it. Unfortunately, these classes are freeloader classes and offer no additional functionality. These 4 subclasses only have one method each and that is to produce a single print statement, acting as a constant class. With that, it also fits the description of a refused bequest since there was no need to create these subclasses.
* The Animal class has examples of Parameter Creep. An example being the method “animal name(String animal name)”. This function takes in the string parameter “animalName” but immediately requests the user to enter a new value for animalName then makes its result based on that input. This use of parameter then immediate replacement makes the parameter unnecessary and this indicates parameter creep.
* It was also discovered that multiple of the functions used in the program were not used inside of their classes nor outside.
* There is a variable called animalType which is meant to describe the species of the animal. This is not an appropriate name because can easily trick a user or fellow programmer think it is something else.
* Within the Zoo class, there is a method called addAnimal which is overly complex and long, taking up most of that class’s code and can be easily simplified.

**Bloaters**

* The code isn’t bloated and is nicely compact. Such that is not expansive and untidy that one cannot follow and pick up what that code is trying to be. It is well laid out and thoroughly order and uses the object-oriented design very effectively to separate objects and make classes and their names cater to the code contained inside the class
* Within the Animal.java class, we see that they used primitives like int for the sicknessStatus then determining the output which is a string.
* Suggestion
* It would be good to use small objects to determine the sickness because if the code needs to be upgraded, the use of the int isn’t the best identifier. I.e. users want to enter the words animal is sick instead of selecting just 1 or 2

**Object-orientation Abusers**

* The first thing is that code is broken into classes which make it cleaner. Using the object-oriented style, the code is easier to understand how it is representing the real-life structure of the zoo. By using the classes I can see how the code is separated for better fluidity and to adjust later. However, the code is missing a lot of comments making the individual actions of a class harder to understand
* Some of the classes seem to inherit unnecessarily. Such that they acquire code that isn’t employed. Because the animal classes are only to change one single method in the animal class
* Methods are well named and easy to comprehend +
* Solution/suggestion
* Remove unnecessary classes and use if statements inside the animal class
* Change Preventers
* During the review of the code, there may be issues regarding the change. The classes (tiger, giraffe etc) are unnecessary because they dedicate all other animals to be added must be made as a class and rather not added to list, array or database of sorts.
* Solutions/Suggestion
* It would be best to remove these classes and then proceed to make animals be put into a list or database such that the zoo can adjust animals freely

**Dispensable**

* The comments in the code are far and few
* There is a fragmented code that looks almost identical to each other in the zoo class use cases. (duplicated code) lines 27 - 67
* The classes that inherit from the animal can be deemed as lazy classes

Suggestions

* The code should be better commented for easier understanding; therefore, the next programmer can see how the coder went about getting the code to work in the manner it does
* We found this identical to add unnecessary lines. This could be a remedy with an object that could seamlessly switch between the different animals. As the hard-coded animals hinder further alterations to code

**Couplers**

- From the way, it views the zookeeper class simply exist to tell another class what to do without being delegated any sort of work

Suggestion

- Remove this class and put it together with another class that has more functionality like the Zoo class

**Nathan’s suggestions**

I suggest that the unnecessary code be removed and made into objects so that the zoo program can grow to demands of the zoo itself. These are hardcoded classes for the different animals cause the program to be stuck to these animals. By using the objects and databases we can effectively allow for whatever modifications to be made to the type of animals to the zoo

**Zachary’s suggestions**

1. Remove the 4 Animal subclasses and incorporate their print statements into a function.
2. Remove the Animal variable, “animalType”. Replace with a string variable called “species”
3. Remove the redundant Animal methods that do not have an impact on the system. They are as follows: animalAge(int animalAge), animalName(String animalName), TypeAnimal(int typeanimal)
4. Restructure addAnimal() method.
   1. Since the switch cases of the addAnimal method are so similar a single function can be written with differences in the cases being the parameters.

**Khareen’s suggestions**

1. The 4 animal subclasses only have 1 print statement each of these is duplicated code. These can be put into a single function with Boolean/ if statements to decide the animal and print according to message.
2. There should be more comments explaining the functionality or purpose of some functions so that even the most novice programmer can easily follow.
3. The program is limited to the coded animal subclasses. What could happen is the add animal method “animalType” could have been set to a larger animal group such as “mammals, fish, birds, reptiles, and amphibians.” These groups would replace the animal subclasses as they would all have specific traits. After one is selected then you can proceed to add a name, age etc. This would allow the code to be more dynamic and encourage growth. A programmer would not be required every time a new type of animal is added. That or this zoo will only hold the preprogrammed animals.

# Improvements

We made drastic improvements to the code for Software Architecture for initial Runtime Zoo program.

Some of these improvements but not limited to.

* Reduce duplicate code
* Put comments

Firstly, we built a whole new application from the ground up but with the initial premise of Runtime’s code. To help manage a new and improve the Zoo Management system.

We create a new highly improved App. The app was coded in Kotlin, a language-based off Java that provides far more flexibility than standard Java.

**Example of Our Code**

MapsFragment.kt

package com.swen3.zooman.ui.map  
import androidx.fragment.app.Fragment  
import android.os.Bundle  
import android.view.LayoutInflater  
import android.view.View  
import android.view.ViewGroup  
import com.google.android.gms.maps.CameraUpdateFactory  
import com.google.android.gms.maps.GoogleMap  
import com.google.android.gms.maps.OnMapReadyCallback  
import com.google.android.gms.maps.SupportMapFragment  
import com.google.android.gms.maps.model.LatLng  
import com.google.android.gms.maps.model.MarkerOptions  
import com.swen3.zooman.R  
  
class MapsFragment : Fragment() {  
  
 private val callback = *OnMapReadyCallback* **{** googleMap **->** */\*\*  
 \* Manipulates the map once available.  
 \* This callback is triggered when the map is ready to be used.  
 \* This is where we can add markers or lines, add listeners or move the camera.  
 \* In this case, we just add a marker near Sydney, Australia.  
 \* If Google Play services is not installed on the device, the user will be prompted to  
 \*, install it inside the SupportMapFragment. This method will only be triggered once the  
 \* the user has installed Google Play services and returned to the app.  
 \*/*

As we can see here there is detailed description detailing how the program operates. As seen below the code is seamless and easier to follow and understand. Classes were not too big or too small and everything performed an action providing reasoning to each class and file

val sydney = LatLng(-34.0, 151.0)  
 googleMap.addMarker(MarkerOptions().position(sydney).title("Marker in Sydney"))  
 googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney))  
 **}** override fun onCreateView(  
 inflater: LayoutInflater,  
 container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 return inflater.inflate(R.layout.*fragment\_maps*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
 val mapFragment = *childFragmentManager*.findFragmentById(R.id.*map*) as SupportMapFragment?  
 mapFragment?.getMapAsync(callback)  
 }

***As you can see several comments abound and each method does something different and are not just duplication of others.***

***The main activity in comparison to the original offers more functionality. No method is unnecessarily long, and methods are appropriate for what they do.***

MainActivity.kt

package com.swen3.zooman.ui  
  
import android.os.Bundle  
import android.view.Menu  
import com.google.android.material.navigation.NavigationView  
import androidx.navigation.findNavController  
import androidx.navigation.ui.AppBarConfiguration  
import androidx.navigation.ui.navigateUp  
import androidx.navigation.ui.setupActionBarWithNavController  
import androidx.navigation.ui.setupWithNavController  
import androidx.drawerlayout.widget.DrawerLayout  
import androidx.appcompat.app.AppCompatActivity  
import androidx.appcompat.widget.Toolbar  
import com.swen3.zooman.R  
  
class MainActivity : AppCompatActivity() {  
  
 private lateinit var appBarConfiguration: AppBarConfiguration  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
 val toolbar: Toolbar = findViewById(R.id.*toolbar*)  
 setSupportActionBar(toolbar)  
  
 val drawerLayout: DrawerLayout = findViewById(R.id.*drawer\_layout*)  
 val navView: NavigationView = findViewById(R.id.*nav\_view*)  
 val navController = *findNavController*(R.id.*nav\_host\_fragment*)  
**// Passing each menu ID as a set of Ids because each  
// menu should be considered as top-level destinations.**  
 appBarConfiguration = *AppBarConfiguration*(  
 *setOf*(  
 R.id.*nav\_home*, R.id.*nav\_animals*, R.id.*nav\_tickets*, R.id.*nav\_trainers*, R.id.*nav\_map* ), drawerLayout  
 )  
 *setupActionBarWithNavController*(navController, appBarConfiguration)  
 navView.*setupWithNavController*(navController)  
  
 /\*val scrollingFrag = ScrollingFragmentDashboard()  
 supportFragmentManager.beginTransaction()  
 .add(R.id.nav\_host\_fragment, scrollingFrag)  
 .commitAllowingStateLoss()\*/  
 }  
  
 override fun onCreateOptionsMenu(menu: Menu): Boolean {  
 // Inflate the menu; this adds items to the action bar if it is present.  
 *menuInflater*.inflate(R.menu.*main*, menu)  
 return true  
 }  
  
 override fun onSupportNavigateUp(): Boolean {  
 val navController = *findNavController*(R.id.*nav\_host\_fragment*)  
 return navController.*navigateUp*(appBarConfiguration) || super.onSupportNavigateUp()  
 }  
}

# CODE SMELL SEASIDE DEVELOPMENT

The Code is clean and was well built. We tried to correct any issues that may have been discovered and to extend the functionality of the system to make a more useful application that would be more desirable to potential clients.

We removed the deep-rooted issues of the previous code and realized the need to expand the basic needs of the application

The program was well commented to allow for easier future development on it.

# FUTURE PROGRESS

* We hope to add the further feature to the code and allow for more robust security
* Greater communication between the Web App and Mobile applications
* Develop an IOS version of the app
* Further, test the application
* Expand the Capabilities of the application

# CONCLUSION

The application was redeveloped to a greater degree than what we were initially given. We are satisfied with the end product and do wish to further the capabilities of the application beyond that of its initial capabilities. The program can be further developed and using the code and design smells will help make sure the app is best made as possible.