Quentin Dye’s Portfolio

Central New Mexico Community College

Advisor—Robert Garner

Fall 2021

Table of Contents

1.0 Class Construction…………………………………………………………………………………………………………………….3

1.1 Java……………………………………………………………………………………………………………………………….3

1.2 C++………………………………………………………………………………………………………………………………..4

1.3 C#………………………………………………………………………………………………………………………………….5

2.0 Inheritance and Ploymorphism…………………………………………………………………………………………………6

3.0 GUI Construction………………………………………………………………………………………………………………………9

4.0 Database Manipulation…………………………………………………………………………………………………………..13

5.0 Web Research…………………………………………………………………………………………………………………………17

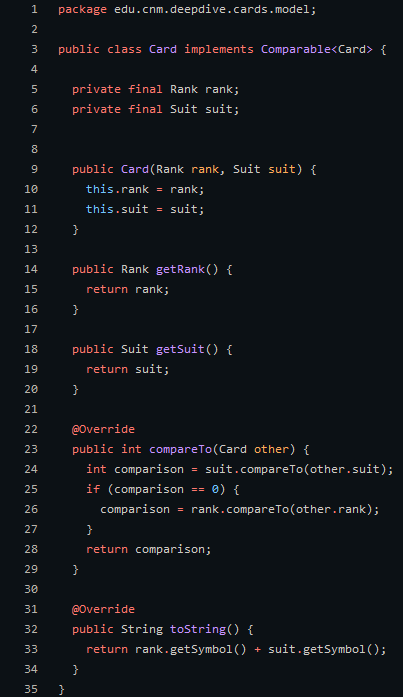
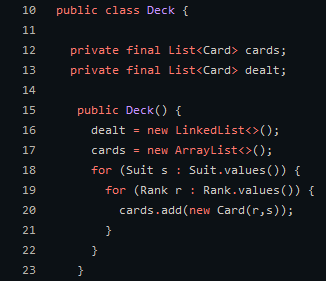
5.1 Ski America App…………………………………………………………………………………………………………..17

5.2 Stock Roller Android Client………………………………………………………………………………………….19

5.3 Depreciation Calculator……………………………………………………………………………………………...21

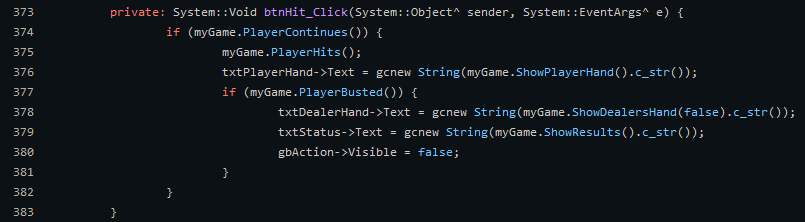
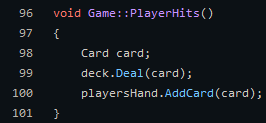
1.0 Class Construction

1.1 Java:

 This example can be found on [github](http://github.com/Quentin-D-NM/cards-model). The class structure in java is fairly simple once you learn it. The only thing that is required is the class keyword and a valid class name. Almost anything you could imagine is a valid class name but you should stick to the uppercase noun convention. While those are the only things required in class declaration it has a lot more going on. First the implements keyword followed by an interface. This gives us the ability to override the compareTo() method. We also override the toString() method though that is not apart of the Comparable<T> interface. Finally we have a couple final variables called rank and suit which are set in our overloaded constructor. This example uses the card class to create a deck of cards. Each have a suit and rank, which is set automatically by the card constructor. Each unique card is added to the cards Array-List.

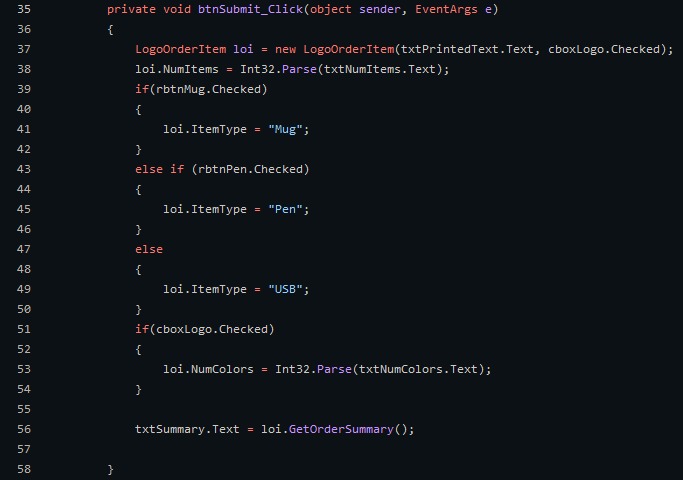
1.2 C++:

 In C++ the way classes are created are actually done in the header files. This is a game class for a card game called craps. As you can see the class structure here in c++ is far different than that of java. For example in java you need to set each member variable as public or private but it c++ you simply have a block of private variables/methods and a block of public variables/methods. In the actual c++ files Game.cpp we find that there is no class, but methods that define they are apart of the game class as shown below.



In the code above we actually see our game class in action as we have a button click event the triggers some methods like ‘PlayerContinues()’ and ‘PlayerHits()’ which continues the game, and alters the state of the actual game object.

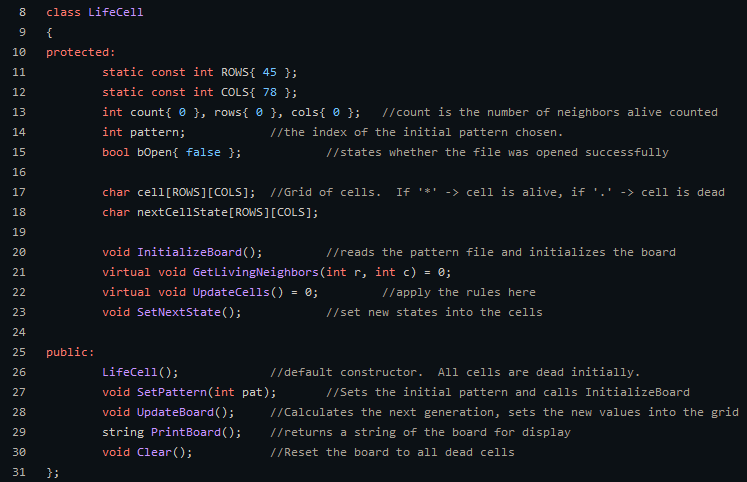
1.3 C#:

 In C# the actual class declaration is similar to java in that the only thing required is the class keyword and the valid Class name. The big difference in C# is that you can have these things called properties that eliminate the need to create setter and getter methods for each private variable for proper encapsulation. C# also gives us the ability to do what is called constructor chaining. While you can do this in java I did not have a good example. Constructor chaining gives us the ability to reduce code redundancy and set default values fairly easily.

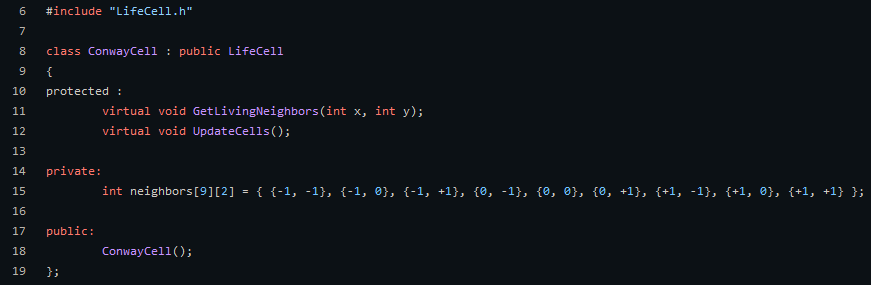
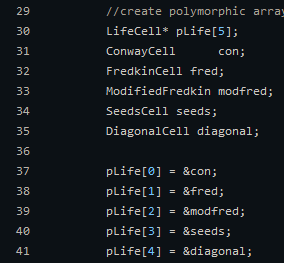
In the code we have a button click event that creates a LogoOrderItem grabs all of the relevant data and calculates the final price. It then calls its GetOrderSummary method to display the info to the user.

2.0 Inheritance and Polymorphism

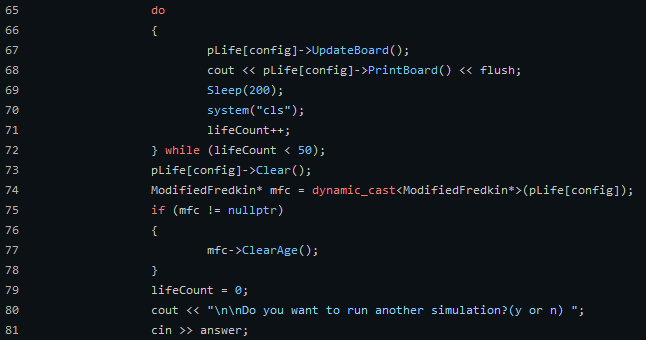
In programming inheritance is the idea that a base class can pass on properties and behaviors to another derived class. In this example I have programmed Conway's game of life. LifeCell is the base cell type for all other cell types. We create this base class as there are a set number of methods and variables that we want all Cell types to share, such as initializeBoard() or setNextState(). These methods do not need to be unique as they will be the same for every derived class. Although GetLivingNeighbors and UpdateCells are unique to each cell type because they each differ in behavior. For example the DiagonalCell type looks for neighbors that are diagonal of that cell, hence the name. Although the default ConwayCell looks for neighbors right next to itself and above/below itself. Thus these are marked virtual and up the children to implement.

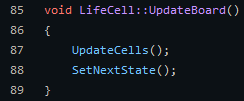


The ConwayCell class, as seen below, inherits from the LifeCell class. As you can see this only implements the virtual classes from LifeCell, all other methods are defined by the LifeCell class. The virtual methods are the only ones that need to be unique for each cell type.

In the code to the right we see the creation of an array of LifeCell pointers. We then add 5 unique Cell Type objects to the array. This works because all of the Cell types seen below are children of the base class LifeCell.

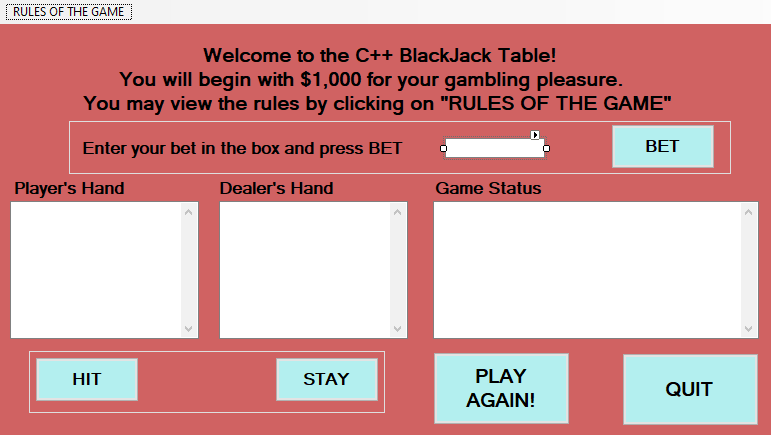
In the code below we see that our array of LifeCell pointers is being used to update our UI which we refer to as UpdateBoard(). How does this happen if each pointer has a unique type with a unique behavior?



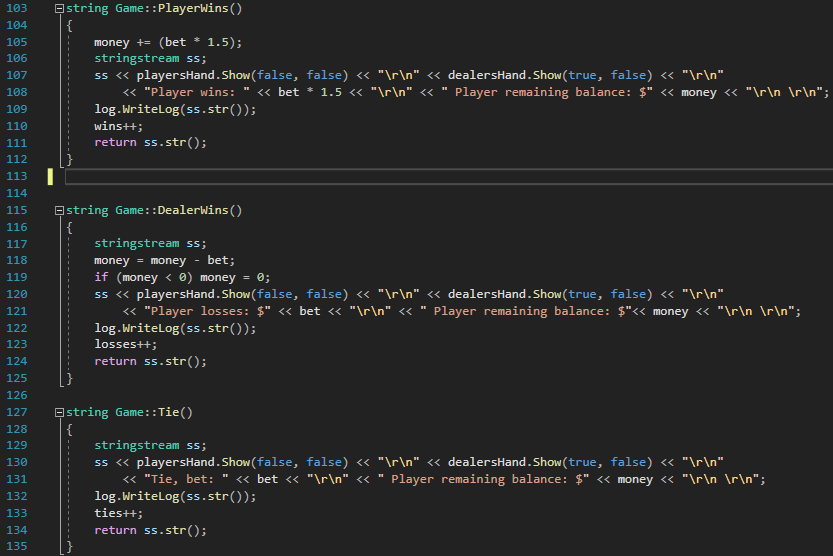
 While UpdateBoard is a base class method only it calls UpdateCells() which is unique to each type. This gives us the ability to quickly switch between types and utilize unique behavior in each.

3.0 Graphical User Interface Construction

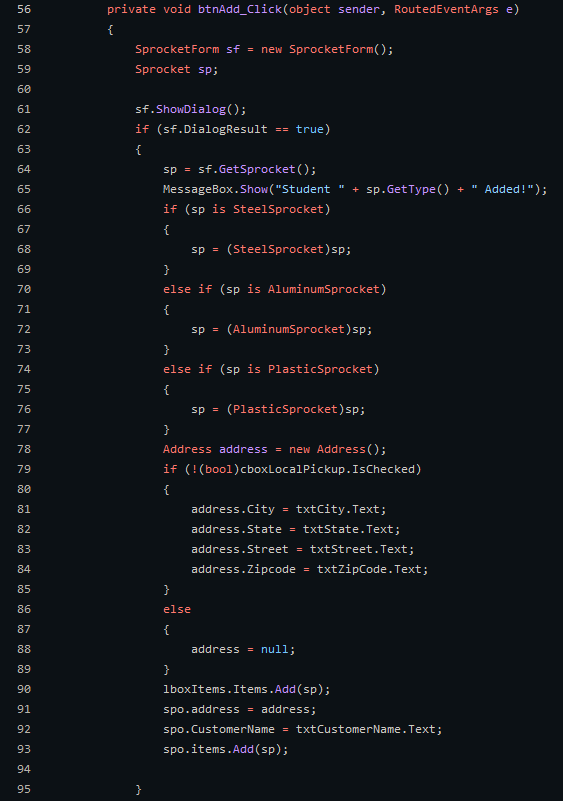
While at CNM I created both windows form apps in both C# and C++. I also created some java GUI Applications using JavaFx. Although I believe in general C# and C++ are better when developing Desktop applications. The only down side is neither are very easy to port over to Mac or Linux. Below you see a group project I worked on in my first year that lets you play Blackjack.

 Creating these forms is fairly simple with Visual Studios design mode, though there is also the option to write these forms using code. Although this difficult to visualize without actually looking at the form, so the designer mode is easiest. If you name all of you elements appropriately writing code to utilizing this GUI is fairly simple. The only difficult part is converting the type returned by the element to a useful type like Integer or String.

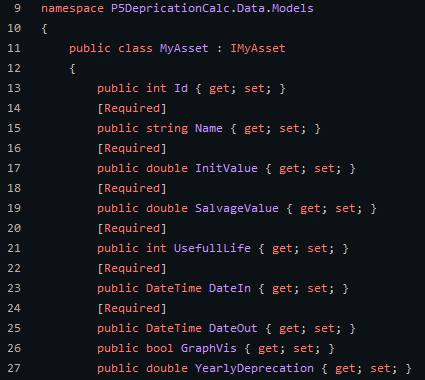
 As you can see below this is the on click method written for the ‘bet button’ name ‘btnBet.’ To write code to use this UI we use the elements name and the element propertie we want. For example ‘txtBet→Text’ will return the text property associated with the txtBet element.

This program also keeps track of all of the wins and losses via a file it writes to every time the player wins, losses or ties with the dealer. It also performs a close action to summarize the games played while the logger was open.

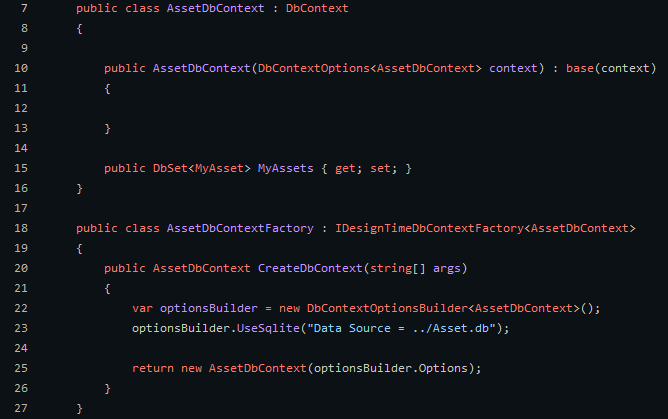
These forms also give you the ability to open dialogs to get more information. For example in sprocket order form I made for C# 1 I have the main window open a separate window to enter detailed info about the sprocket that needs to be ordered as seen below.

 All you have to do to work with data between two forms is to create an instance of the dialog in the main form and use it to gather the data needed. Here you can see we create the Sprocket Form and show it to the user. Once the user confirms his submission in the form its pushed back to the main form and we can add the data we just got from the Sprocket form to the Sprocket order.

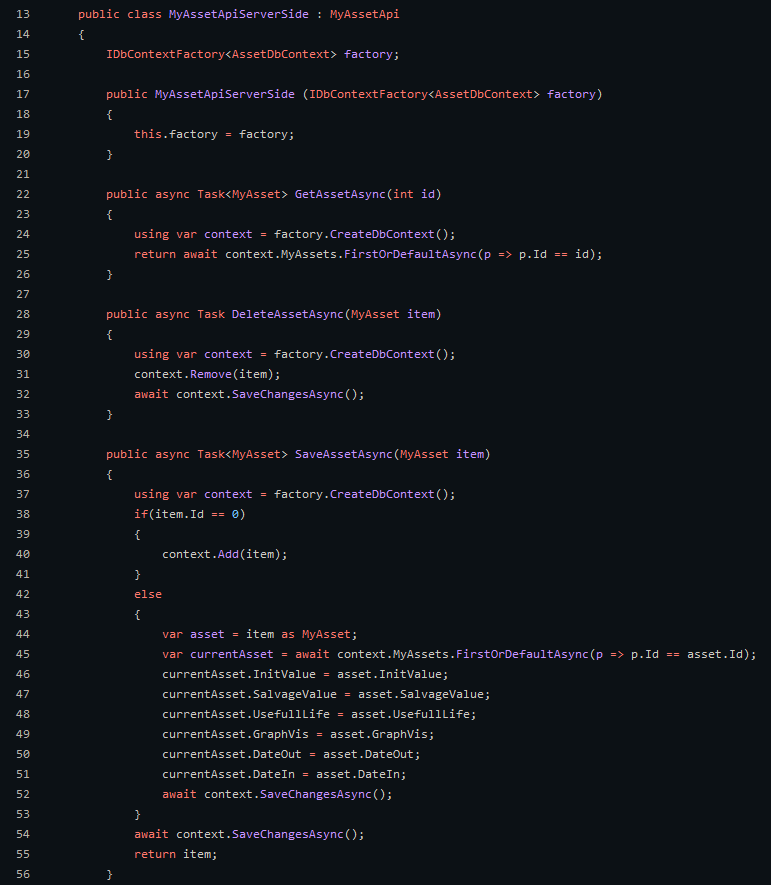
4.0 Database Manipulation:

Connecting to a database was one of the more challenging things but I was able to accomplish this with both java spring and c#. This example is written in C#. To create a database in a Blazor application you have to add a new class library by doing the command ‘dotnet new classlib -o P5DeprecationCalc.Data’ you also need to add this to your sln with the command ‘dotnet sln add P5DeprecationCalc.Data.’ Finally you need to add the EntityFrameworkCore.Tools package and the EntityFrameworkCore.Sqlite package to your project. After you complete those steps you simply start creating your data classes as seen below. This class holds a couple different values including asset name, Initial value and Date added and Date removed variables. This is kind of like a blue print as this will be generated into a SQL table.

We also need to create a database context class which is how we will actually interact with the database. This file also contains a second class that is used to configure our database when running migrations. Each time we change something about the data classes we must run a migration to update the tables in our database. To actually run our migrations we simply run the command ‘dotnet-ef migrations add migrationName.’ This example also shows us the location of the database. Its located in the root folder of the project and it is called Asset.db.

 While we have created a database we have no way of accessing it until we create a database API. The best way to do this in Blazor is to create an interface which will outline our Get, Save, and Delete operations. Then write the actual methods in a API class called MyAssetApiServerSide which uses our interface MyAssetApi.

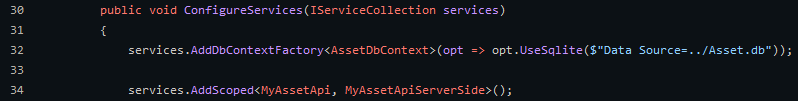
There are a couple of different operations that are setup in my Api. One of them being GetAssetAsync by Id, which will match the id passed to a unique id in the database. The other two operations use a MyAsset object to get the id and match them to an item in the database. All of this is done using our IDbContextFactory to create a database context and using that context to make changes to the database.

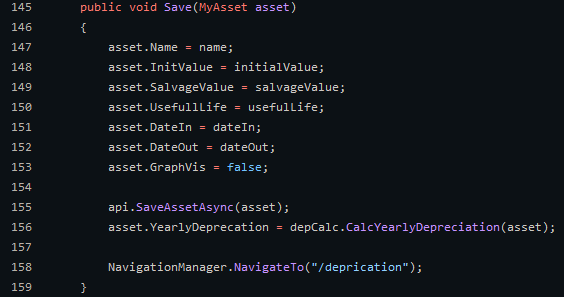


Finally for the database to work you need to include the following packages.



You also need to add the following lines to the ConfigureServices method in Startup.cs to add the database context factory as a service, as well as your Api.

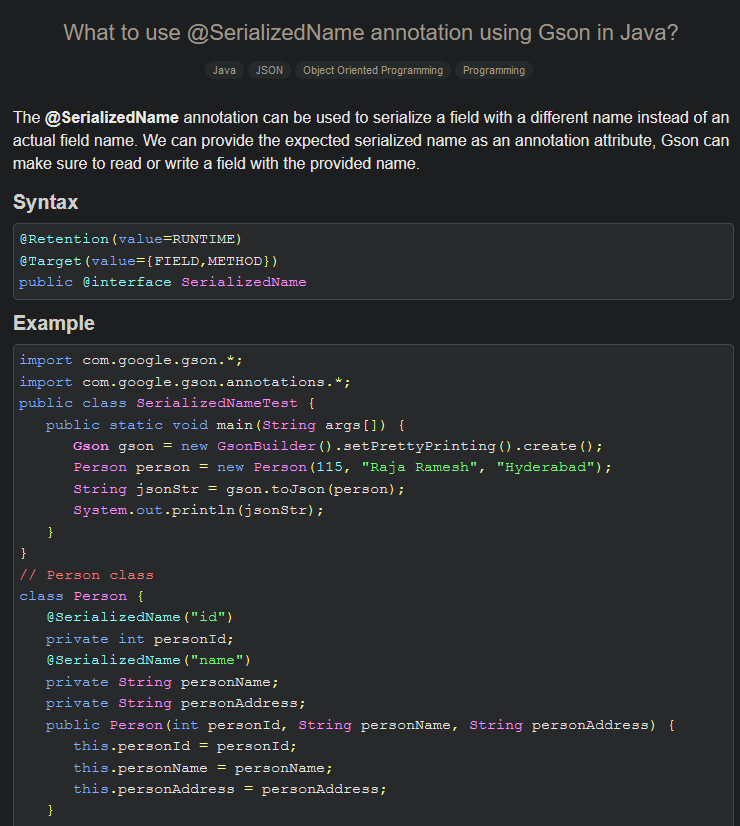
 With that setup you can now interact and save items to the database as shown in the code below. This code uses the existing MyAsset data class to build an object to be saved to the database. Now all you have to do is use the api variable, injected as a service with the line ‘@inject MyAssetApi api’, and call SaveAssetAsync.



5.0 Web Research:

5.1 Ski America App:

One of the many skills acquired through my journey at CNM includes the ability to research existing code and implementing those solutions in my own code. One of the first times I found how useful the internet can be was when I began building an Android app that utilizes Weather data from ski resorts to give skiers a forecast on the mountain. This app uses googles gson which can automatically serialize java objects to json and serialize json to java objects.

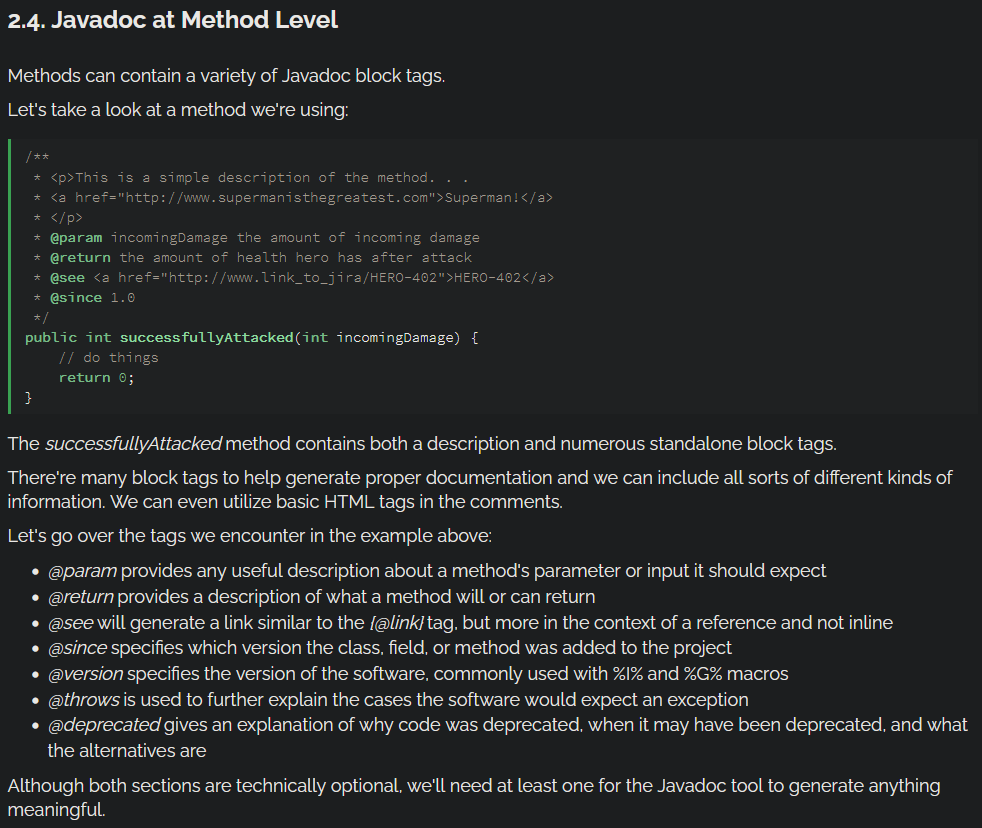
 [Link](https://www.tutorialspoint.com/what-to-use-serializedname-annotation-using-gson-in-java)

I used my research to automatically serialize json data coming from my weather forecast service to java objects to be displayed to the user. This is useful to ensure your object is being serialized correctly, its also helpful when a variable name does not match your own naming conventions like “totalSnowfall\_cm” being encoded to totalSnowFall.

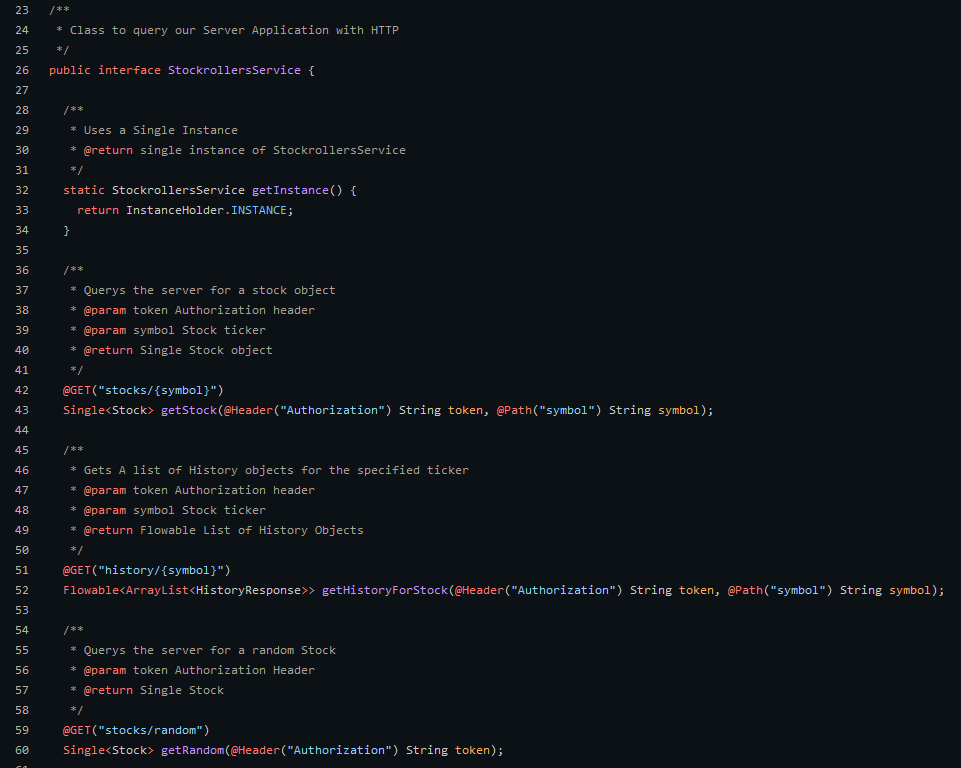


5.2 Stock Rollers Android Client:

Documentation is important in any project and can be key to remembering what your own code does. It will also help other programmers work on your code as they know exactly what is supposed to be happening. Javadoc is a widely used application that will turn all of your code into clear documentation by simply including what are know as javadoc comments. These comments include specific syntax that helps auto generate documentation.

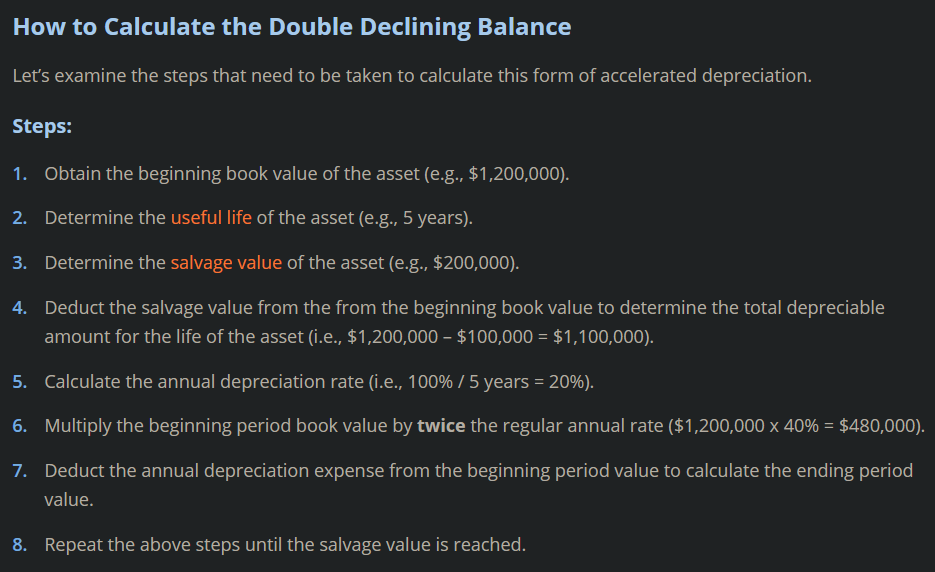
 [Link](https://www.baeldung.com/javadoc)

I use these javadoc comments in my own code as shown below. You can include @param and describe what the variable is supposed to be. In this case it shows the type Stock and the name ticker, indicating that you need to pass a fully qualified Stock object to this method. These comments also give you the ability to describe what they are supposed to be doing. Useful for not only yourself but other programmers as well.



5.3 Depreciation Calculator

While a lot of the time my research is about programming, occasionally I am asked to perform a calculation that I am not familiar with. Internet research is a perfect approach to this problem. For example in Program 4 of my C# 1 class I was asked to make a program that calculates both straight line deprecation and double declining deprecation. So I did some research on how to calculate both of those things.

 [Link](https://corporatefinanceinstitute.com/resources/knowledge/accounting/double-declining-balance-depreciation/)

I used this explanation to write the correct code to calculate the double declining depreciation.

