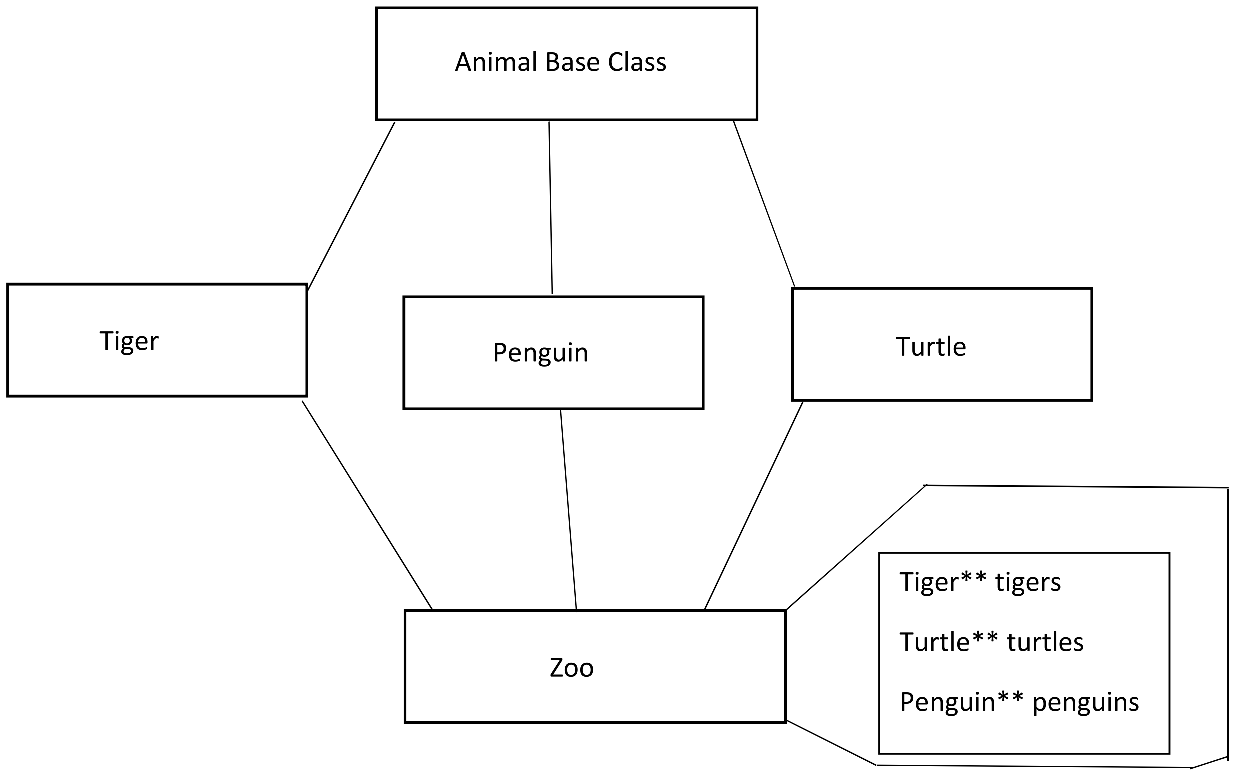
Project 2 Reflection

This assignment had us dive deeper into class inheritance to create a simple Zoo simulation. The structure was straightforward. I created an Animal base class with member variables/functions that were derived into Tiger, Penguin, and Turtle classes.



The zoo class creates three dynamically allocated arrays which each hold one of the 3 derived class types (Tiger, Penguin, Turtle). This is shown in the above diagram. The Zoo class holds the member variable balance, which is the amount of money the user has to spend. It contains member functions which add animals to their respective arrays, runs a random event generator, calculates daily profit, decrements the balance after transactions and operating costs, and functions that return the arrays which are privately declared in the Zoo class. Much of my time in this project was spent making generic functions for adding animals, purchasing animals, removing animals, and resizing animal arrays that could be reused for the 3 different animal types. I didn't want to hard code a function for each type. The main function contains the game loop and creates a Zoo object. The game loop continues until the user runs out of money or decided to exit the game. There were struggles I had with this project. When I tried to account for animal ages in the birth scenario, I kept running into segmentation faults. I ended up commenting out my code and using a simpler formula for this process. I also ran into memory leak issues when trying to delete/free the memory that was used for the tigers, penguins, turtles arrays. I ended up commenting out this code as well. I commented out many of my trial and errors just as a point of reference. It helps me work through a problem.

Test Cases

Main function: Enter yes or no to purchase an adult animal in the game loop	Expected: cin.get() would correctly extract the first character needed for my conditional statement.	The program ended up hanging due to a bad character being stored in the input buffer. This required me to add functions to clear my input buffer.
Main function: Entered a character when being prompted for a int. This occurred when the program asked the user to enter starting amount of tigers, turtles, and penguins.	Expected: My getInput() function would correctly identify the wrong type due to its use of cin.fail().	Entering h when being prompted for an int led to the program correctly asking you again to enter an integer type between 1 and 2. Proved my function works.
Zoo.cpp: removeAnimal() I tested removeAnimal() to ensure that this function worked. This function would be used in the case that an animal died.	Expected: I expected the last animal in the array would be removed.	The last animal in the array was correctly removed. This was tested by testing a function call on the index that was removed. This led to a compilation error.
Main function: Tested integer values of 5 and 15 when being prompted for either 1 or 2 quantity for starting amount.	Expected: I expected my getInput() function to correctly determine that these values were out of range.	The function correctly asked the user to renter an int value between 1 or 2. Proved that function works.
Main function: Entered "rhgkg" when being prompted for yes or no on whether or not to continue to next day.	Expected: The program would ouput "Invalid Input" and while loop would reiterate.	The program successfully outputted "Invalid Input to the user" and asked them again whether or not they wanted to continue.
Zoo.cpp: Test my addAnimal() function to ensure that the correct animal object was being added to the correct animal array. I added a tiger to my tiger array using addAnimal and used get functions to ensure the correct animal was stored.	Expected: getAnimalType() would return "tiger" and getCost() would return 10000.	"tiger" was successfully returned for getAnimalType() and 10000 was successfully returned for getCost(). This ensured that a tiger object was correctly being stored in my tiger array.
Zoo.cpp: I tried to delete the dynamically allocated memory by using a for loop to delete every index in each array using the delete[] operator. I then used delete[]tigers, delete[]penguins, delete[]turtles to completely free the memory.	Expected: Program would successfully compile and execute with no memory leaks.	Program did compile and run but generated a "core dump" issue when choosing to exit the game. This proved that my method for freeing the memory was wrong and that I would have to do it differently.