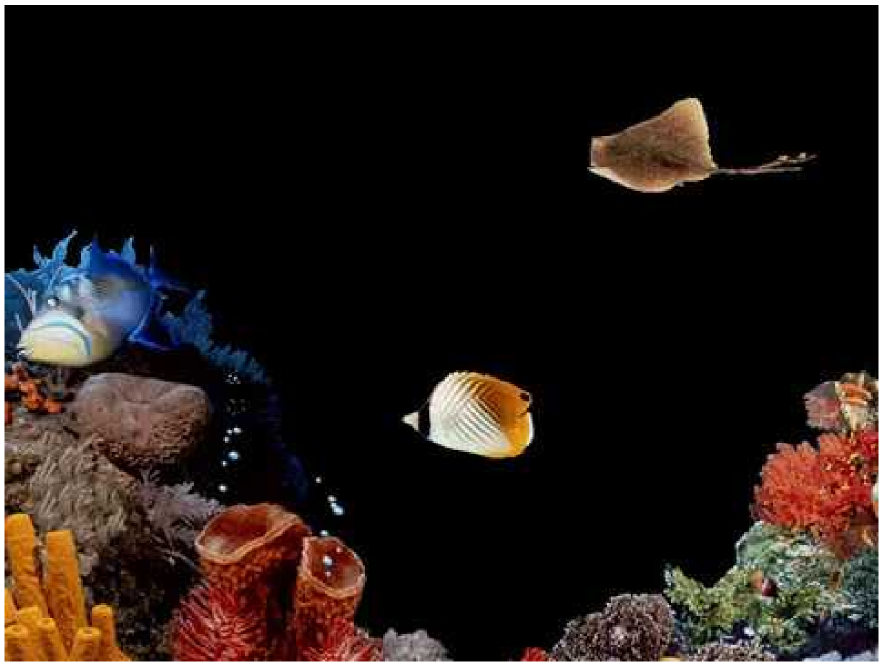
CS 202 Assignment 5
Virtual and Abstract Functions



Overview

In this assignment we will be making a virtual aquarium. We'll need two things to do this: a fish tank and fish to put in that tank. Much like real life, there will be some diversity in the types of fish. All of the fish will be capable of swimming, but in different ways. Three kinds of fish will live in the tank: **HorizontaFish**, which can swim left and right, **VertiFish**, which can swim up and down, and **WallFish**, which just float in place blocking other fish from swimming.

In order to make this work, you will need to finish programming all of the types of fish. Each will need to swim in their own way, meaning we'll need to implement a swim function virtually. There also can't be any fish that aren't a specific species, so the fish class will need to be abstract to avoid this.

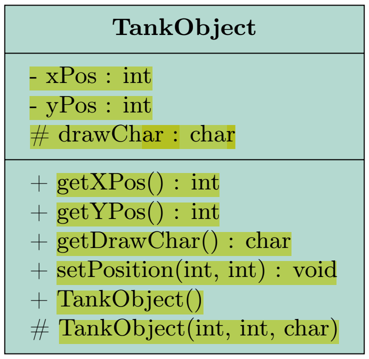
A virtual function is a function that's behavior can be redefined in a derived class. In this example, a generic **Fish** may not be able to swim, but all of the types of fish that inherit from it will be able to swim and do so in different ways. The base **Fish** class will also be abstract for our program, that is, it will contain virtual functions that do not have defined behavior and thus MUST be redefined in a derived class. In C++, any class that contains one or more abstract functions is itself abstract, meaning that it cannot be instantiated. That being said, we can still use variables of abstract types, as we will do with the Fish in our program.

Classes

- **Tank -** The class that will hold all of the fish. This will have a 2D array of **TankObject** pointers and some associated variables like the width and the height of the Tank.
- **TankObject -** Base class for all objects that may go into the tank. Sets up a basic blueprint with x and y coordinates for objects that will go into the tank and a char to use to draw the object in the tank. While not abstract, this is not a class you will want to instantiate. That being said, if you'd like to represent blanks spaces/water with this class, that is okay.
  Note that because arrays are generally row first then column, the x and y coordinates are flipped from the traditional math representation.
- **Fish -** The abstract base class for all types of fish in the tank. Inherits from **TankObject**. This class adds the abstract swim() and onWallCollision() functions that all fish should have.
- **VertiFish -** A very single minded fish. This **Fish** swims only in vertical straight lines. Every frame that this fish swims it should go up or down, depending on which way it is facing. When it hits a wall, it will turn around.
- **HorizontaFish -** A close cousin of the **VertiFish**, this fish behaves almost identically, but only swims horizontally.
- **WallFish -** Incredibly lazy. This species of fish sits in place and stops other fish from swimming around it. When its swim() function is called, it just wastes time.

UMLs

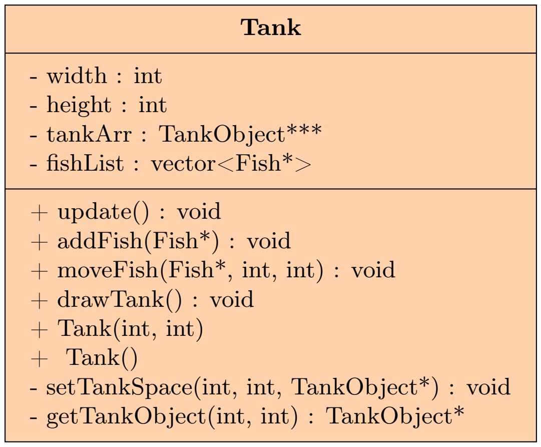
Below are UML diagrams for all of the classes



TankObject

- xPos : int
- yPos : int
# drawChar : char

+ getXPos() : int
+ getYPos() : int
+ getDrawChar() : char
+ setPosition(int, int) : void
+ TankObject()
# TankObject(int, int, char)



Tank

- width : int
- height : int
- tankArr : TankObject***
- fishList : vector<Fish*>

+ update() : void
+ addFish(Fish*) : void
+ moveFish(Fish*, int, int) : void
+ drawTank() : void
+ Tank(int, int)
+ Tank()
- setTankSpace(int, int, TankObject*) : void
- getTankObject(int, int) : TankObject*

Fish : public TankObject
static count : int
+ virtual swim(Tank* tank) = 0 : void # virtual onWallCollision() = 0 : void # Fish(int, int, char)

VertiFish : public Fish
- facingUp : bool
virtual onWallCollision() : void + virtual swim(Tank* tank) : void + VertiFish(int, int, bool = true)

HorizontaFish : public Fish
- facingRight : bool
virtual onWallCollision() : void + virtual swim(Tank* tank) : void + HorizontaFish(int, int, bool = true)

WallFish : public Fish
virtual onWallCollision( ) : void + virtual swim(Tank* tank) : void + HorizontaFish(int, int, bool = true)

Important Functions

Below are explanations of important functions for each class. For convenience, all functions that are implemented for your are in blue, and all functions that you will need to implement are in red.

TankObject

- **TankObject(int x, int y, char drawChar)** - Constructor for TankObject that sets its position to the given x and y and also sets the char to draw the object with in the Tank.
- **TankObject()** - Default constructor which is here mostly for convenince. Sets the draw char to an empty space

Tank

- **void update()** - This function updates all of the Fish currently in the Tank via the ***fishList*** by calling their swim() function. Used by the main loop.
- **void drawTank()** - Draws the contents of the Tank to the screen and clears any other text that was previously on the screen. Used by the main loop.
- **void setTankSpace(int x, int y, TankObject* obj)** - Optional function that sets the given (x, y) position in the ***tankArr*** array to obj. This exists to help if the fact that y comes before x in our array bothers you.
- **TankObject* getTankObject(int x, int y)** - Returns the TankObject currently at the given (x, y) position. Exists for the same reason as above.

- **void addFish(Fish* fish)** - Adds a new Fish to the tank. This should place the fish in the ***tankArr*** based on its (x, y) position and then add the fish to the fishList vector.

- **void moveFish(Fish* fish, int xMove, int yMove)** - Attempts to move the given fish by the given x and y distances. If the fish hits a wall (i.e. goes out of bounds of the Tank or the position it lands on is a WallFish), then call the fish's onWallCollision() function to let it know it is going to hit a wall and do not move the fish. If the position that the fish will land at is occupied by another fish already, do not move the fish. If the space the fish wants to move to is free, move the fish there by setting the array space it wants to move to to that Fish, and then set its old position to NULL to represent it being empty. This should also update the Fish's position.

- **Tank(int width, int height)** - Allocates the ***tankArr*** as a 2D array of Fish*. This should first allocate ***height*** many rows of Fish**s, then allocate ***width*** many columns (Fish* elements) within those rows. Finally, this should go through the allocated 2D array and initialize all of the Fish*s to NULL. We will use NULL to represent an empty space with no Fish.

- **~Tank()** - The Tank destructor will need to deallocate the ***tankArr*** and all of the Fish in the Tank. The 2D array will need to be deallocated in the opposite order of its allocation. First deallocate all of the Fish (the ***fishList*** is an easy way to do this). Next, deallocate all of the rows, then finally deallocate the row pointers, being the tankArr pointer itself. ***HINT: delete[] is needed to de-allocate arrays***

Fish

- **virtual void onWallCollision()** - The abstract function for when a Fish runs into a wall.

- **virtual void swim(Tank* tank)** - The abstract function that makes a Fish swim every frame. Will be different for all types of Fish.

VertiFish

- **void onWallCollision()** - This is a callback function, which will be called by the Tank if the Fish tries to swim past a wall. Flips the direction of the Fish, represented by the ***facingUp*** boolean, then sets the way the Fish looks. If the VertiFish is facing up, its drawChar should be ˆ and if it is facing down, it should be **V**.

- **void swim(Tank* tank)** - This function is called every frame by the Tank. The VertiFish should swim either one space up or one space down every frame, depending on which way it is facing. Use the Tank's moveFish() function to try to move.

HorizontaFish

- **void onWallCollision()** - Flips the direction of the Fish, represented by the ***facingRight*** boolean, then sets the way the Fish looks. If the HorizontaFish is facing right, its drawChar should be $>$ and if it is facing left, it should be $<$.

- **void swim(Tank* tank)** - Same as the VertiFish function, but should move the fish either one space left or one space right.

WallFish

- **void onWallCollision()** - Does nothing. The WallFish is immovable.

- **void swim(Tank* tank)** - Also does nothing.

To Do

Code should be added to three files: Fish.h, Fish.cpp, and Tank.cpp. You should see segments where your code is need with either "YOUR CODE HERE" or "TODO". The bulk of this program will be implementing the fish species swim() and onWallCollision() functions and the constructor/destructor for the Tank class.

If you are having trouble building with the makefile, you may need to remove any existing .o object files as well as any previously created executables. It is okay to write your initial code on Windows, **but you must test the final program on Linux.**

Sample Output

Included with the assignment is a text file called **static_output.txt** with sample output showing all frames of the virtual aquarium from a standard run. Aside from this, there are two executables that you can use to compare to your own fish tank. The **virt_aqua** program is an executable that can be used on Linux. It takes an optional command line argument, being a seed that you can use to generate random fish tank permutations. Your own program also supports this optional seed via main.cpp, so feel free to play around with your own, too. There is a Windows equivalent to this program included, as well.