Welcome to our journey through C++ programming! In this article, we will create a simplified virtual aquarium using C++'s Object Oriented Programming (OOP) techniques. This will be a great opportunity to learn how to use classes and objects to model real-world things and their relationships.

First, let's think about the design. Before we get our hands on the code, think about what's in this aquarium:

Fish: each fish has its specific species, color.

Aquarium: can hold multiple fish with a certain capacity limit.

Of course you can have more things in your aquarium when you build your own, but for now let's start with these two things.

1. Fish

Let's start by defining a fish class to represent the fish in our aquarium:

```
#include <iostream>
#include<string>
class Fish{
}
```

Then we will start writing some attributes for our fish, we will have 2 member variables, species,and color. Do not forget to put them under private, because you don't want other people to modify your fish in your aquarium, this is impossible in real life！

```
private:
    std::string species;
    std::string color;
    float size;
```

After that, we need to start writing the constructor for the Fish class, which is used to initialize the properties of the fish object when it is created.

It should accept three parameters: species(species), color(color), size(size): this is the initialization list used to set initial values for the data members of the class. Write the code under public.

```
Public:
    Fish(const std::string &species, const std::string &color,
float size): species(species), color(color), size(size) {}
```

Fish swim in the real world, so to make our fish closer to reality, we add a swim() method that simulates this behavior.

Write this code right under the constructor:

```
void swim() const {
```

```
    std::cout << species << " is swimming!" << std::endl;
}
```

void swim() const {:

this defines a method called swim. void means that the method does not return any value. the const keyword means that the method does not modify any member variables in the class, which is a way of assuring the compiler that the method does not change the state of the object.

std::cout << species << " is swimming!" << std::endl;:

this is the body of the method. It outputs information to the console using std::cout.

species:

outputs the species of the fish. Since species is a private member variable, but since we are inside the method of the class it belongs to, so we can access it directly.

Also we can write a getter for Species attribute, so we can access it from outside of the class.

```
std::string getSpecies() const { return species; }
```

With this method, we can access the species attribute by calling this method, instead of access the member variable directly.

You can try to write 2 more getters for ‘color’ and ‘size’! They should work the same as getSpecies().

Below is the code for the whole fish class, you can scroll down to check if you are doing it right.

```cpp
#include <string>
#include <iostream>

class Fish {
private:
    std::string species;
    std::string color;
    float size;

public:
    Fish(const std::string &species, const std::string &color,
float size)
        : species(species), color(color), size(size) {}

    void swim() const {
        std::cout << species << " is swimming!" << std::endl;
    }

    // getters
    std::string getSpecies() const { return species; }
    std::string getColor() const { return color; }
    float getSize() const { return size; }
};
```

In summary, the Fish class provides a simple but effective model to represent a fish. Through the swim method and a series of getters, we can simulate the behavior of a fish and access its properties. This provides us with a powerful tool to use in simulating an aquarium. Next we need to write an aquarium class.

2.Aquarium

Create a class called Aquarium with 2 member variables, a vector of fish and capcity. Don’t forget to include the Fish class that we just created!

```cpp
#include <iostream>
#include <vector>
#include  "Fish.h"
class Aquarium {
private:
  std::vector<Fish> fishes;
  int capacity;
```

As we said, you should put those variables under private.

Create a constructor for Aquarium class, It accepts an integer argument to initialize the capacity of the aquarium. The list initialization syntax capacity(capacity) is used to set the value of the capacity member variable.

```
#include <iostream>
#include <vector>
#include  "Fish.h"
class Aquarium {
private:
  std::vector<Fish> fishes;
  int capacity;


public:
  Aquarium(int capacity) : capacity(capacity) {}
```

Then we need to create a method for adding fish:

```
bool addFish(const Fish &fish)
{
  if(fishes.size() < capacity) {
      fishes.push_back(fish);
      return true; }
  std::cout << "Aquarium is full!" << std::endl; return false;
}
```

The *if(fishes.size() < capacity) { ... }* helps us to check if there is enough capacity for adding a new fish, if it does, use *fish.push_back(fish);* to add the fish to the fish vector and return true for success. If there is no capacity, output "Aquarium is full!" and return false for failure.

What else can we do now? We need to see how many fishes the aquarium have, right? So write a method to show information about all fish stored in the aquarium. We can use a range-based for loop for *(const Fish &fish : fishes) { ... }* Iterate over each fish in the fishes vector.
Output the species and color of each fish using *std::cout: fish.getSpecies()* and

fish.getColor().

Now the whole Aquarium class should look like this:

```
#include <iostream>
#include <vector>
#include  "Fish.h"

class Aquarium {
private:
    std::vector<Fish> fishes;
    int capacity;

public:
    Aquarium(int capacity) : capacity(capacity) {}

    // Adding fish into Aquarium
    bool addFish(const Fish &fish) {
        if(fishes.size() < capacity) {
            fishes.push_back(fish);
            return true;
        }
        std::cout << "Aquarium is full!" << std::endl;
        return false;
    }

    // Displaying fish in the Aquarium
    void showFishes() const {
        for (const Fish &fish : fishes) {
            std::cout << fish.getSpecies() << ", " <<
fish.getColor() << ", " << fish.getSize() << " cm" << std::endl;
        }
    }
};
```

3. Writing main function.

Are we done with our work? Not yet! we need to write a main function to execute our aquarium!

Create a cpp file called main, and don't forget to include these 2 classes that we just created.

```
#include "Aquarium.h"
#include "Fish.h"
int main() {
    // Create some fish
    Fish goldfish("Goldfish", "Golden", 10.0);
    Fish salmon("Salmon", "Silver", 50.0); // Create an aquarium
with a capacity of 2.

    // Create an aquarium with a capacity of 2
    Aquarium myAquarium(2); // Add fish to the aquarium.

    // Add fish to the aquarium
    myAquarium.addFish(goldfish); // add fish to the aquarium.
    myAquarium.addFish(goldfish); // add fish to the aquarium.
myAquarium.addFish(goldfish); // add fish to the aquarium.

    // Show the fish in the aquarium
    myAquarium.addFish(goldfish); myAquarium.addFish(salmon); //
show the fish in the aquarium.

    myAquarium.addFish(goldfish); // Show the fish in the
aquarium.
}
```

Click Run on your compiler to run the code and you should see the output :

```
Goldfish, Golden, 10 cm
Salmon, Silver, 50 cm
```

Congratulations, you have successfully built your first aquarium in c++ object-oriented programming!