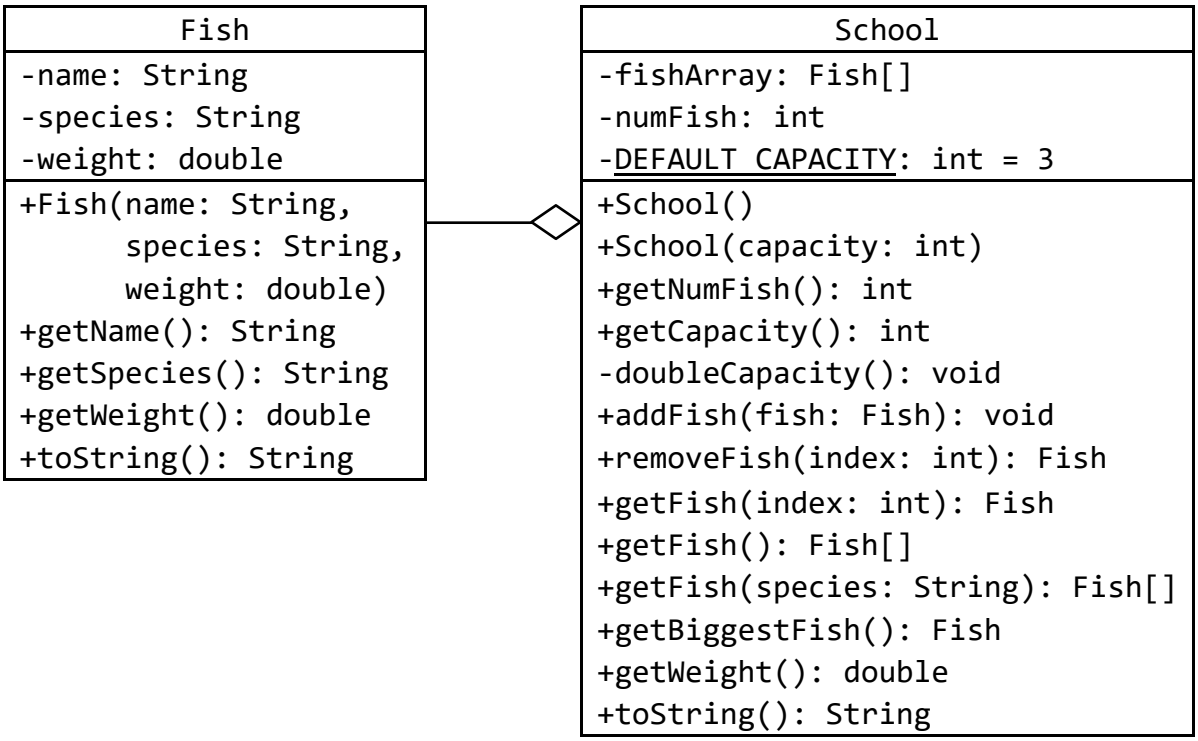
Lab 2: An Aggregation of Fish

Due: Tuesday, Jan. 28 at 11:59 PM

Description: In object-oriented programming, aggregation is a relationship between two classes where objects of one class contain objects of the other. For our second lab, we will implement two such classes: Fish and School. A Fish object represents a single fish, while a School object represents a school of fish. Each School object contains an array of Fish objects.

Class Diagram: Aggregation is represented in UML diagrams by a line with an empty diamond on one end that connects two classes. Objects of the class touching the diamond contain objects of the other class. The following is a class diagram for the Fish and School classes:



Notice that the diamond touches the School class. Just like in nature, a School is an aggregation of Fish.[1]

Fish Class: Each Fish object has a name, species, and weight (in pounds). The toString method returns a String with the values of these fields separated by commas. For example, suppose that a Fish object is constructed with the following line:

```
Fish fish = new Fish("Nemo", "clownfish", 0.2);
```

Calling toString on this object returns the String `"Nemo, clownfish, 0.2 lb"`.

[1] Actual schools usually consist only of fish that are the same species and size. Our School objects can contain Fish of any species and size.

Note that the weight, which is a double value, is rounded to the tenths place. This can be done by using the format method of the String class as follows:

```
String.format("%.1f", weight)
```

The first argument is a format String that tells the method how many digits to keep after the decimal place.

School Class: Each School object contains a reference to an array of Fish objects. The reference is stored in the field fishArray. When a School object is constructed, a new Fish array is also constructed. The array is initially empty, but Fish can be added by calling the addFish method. Each additional Fish is added to the next empty element in the array.

In CS 1323/1324, we used the adjective "oversize" to describe arrays such as this, which have extra space for adding data. An oversize array has both a capacity and a size. The capacity is the length of the array. The size is the number of elements we treat as non-empty. (For instance, if an array has a capacity of 10 and a size of 6, we treat the first 6 elements as valid data and the last 4 as empty space.) In a School object, the size of fishArray is stored in the field numFish.

The size of an oversize array cannot exceed its capacity. If addFish is called when fishArray is full, a new array must be constructed. The new array is made twice as long, and all the elements are copied from the old to the new array. The new Fish can then be added.

Below are descriptions of all the methods of the School class. Most methods deal with adding, removing, or retrieving a particular Fish. A few of them, however, return properties of the School, such as the total weight or largest Fish.

- School(): Construct a School object with capacity equal to DEFAULT_CAPACITY.
- School(int capacity): Construct a School object with the given capacity. The smallest allowed capacity is 1.
- getNumFish: Return the number of non-empty elements in fishArray (i.e., numFish).
- getCapacity: Return the length of fishArray.
- doubleCapacity: Double the capacity of fishArray. That is, create a new array with twice the length, copy the elements of the old array into the new array (at the same indices), and assign the new array to fishArray. (The old array will be garbage collected.)
- addFish: Add the given fish to the next empty element in fishArray. (Since arrays are indexed from 0, note that the index of this element is equal to the number of fish.) If the array is full, double the capacity before adding the new Fish.
- removeFish: Remove and return the Fish with the given index in fishArray. Any Fish with a larger index is shifted down to the next-lowest index. (For example, suppose fishArray has two Fish. If the Fish at index 0 is removed, the Fish at index 1 is shifted down to index 0.) If the index is outside the non-empty part of fishArray, return null.
- getFish(int index): Return the Fish with the given index in fishArray. If the index is outside the non-empty part of fishArray, return null.

- getFish(): Return an array with all the Fish in the school. (The length of the array should be equal to numFish.)
- getFish(String species): Return an array with all the Fish of a given species. The order of the Fish should be the same as in fishArray. (The length of this array will be less than or equal to numFish.)
- getBiggestFish: Return the Fish with the largest weight. If the method is called on an empty School, return null.
- getWeight: Return the sum of the weights of all the Fish.
- toString: Return a description of the School. The String is formed by calling toString on each Fish and concatenating the output. The substrings are numbered and separated by newline characters.

  For example, suppose a School is created with the following lines of code:

```
Fish fish1 = new Fish("Nemo", "clownfish", 0.2);
Fish fish2 = new Fish("Dory", "blue tang", 1.3);
School school = new School();
school.addFish(fish1);
school.addFish(fish2);
```

  Calling toString on the School returns this String:

```
"1. Nemo, clownfish, 0.2 lb\n2. Dory, blue tang, 1.3 lb"
```

Submission: Submit your code to the Zyante website for automated testing. We strongly suggest that you write some tests of your own before doing this.

Collaboration: If you collaborate with other students, include their names in a comment at the top of your source files. Make sure you implement any ideas you discuss yourself. Do not copy and paste someone else's code.

Pondering Immutability: Objects of the Fish class are immutable. If this were not the case, would your implementation of the School class need to change? If so, why and how? You do not need to submit answers to these questions, but they're worth thinking about!