# COMP1406 - Assignment #3

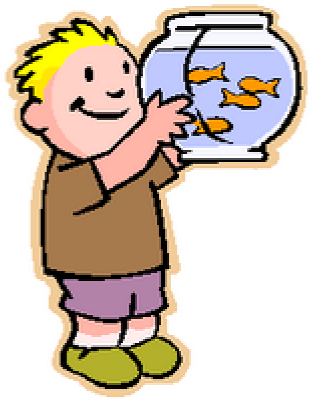
(Due: **Monday, February 3rd @ 11:30pm)**

In this assignment, you will simulate some people catching fish in a lake. The purpose of the assignment is to get used to using Arrays as well as getting more experience in having objects interact together.

## (1) The **Fish** class

Define a class called **Fish** that has the following **private** attributes:

- species - a **string** indicating the type of fish (e.g., "Pike", "Bass", "Sunfish")
- size - an **int** indicating the size of the fish in cm.



Write these **public** methods:
- get methods for each attribute
- a constructor that takes the size and species of the fish
- a **toString()** method that returns a string in the following format:

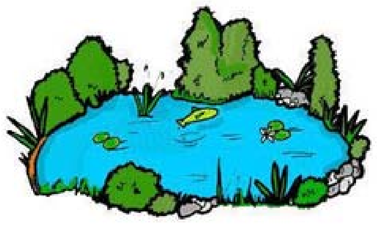
```
"A 10cm Pike"
```

Make sure that your code works before continuing by running the following code:

```
public class FishTestProgram {
    public static void main(String args[]) {
        Fish  f = new Fish(4, "Sunfish");
        System.out.println(f);              // should display A 4cm Sunfish
        System.out.println(f.getSize());    // should display 4
        System.out.println(f.getSpecies()); // should display Sunfish
    }
}
```

## (2) The **Pond** class

Define a class called **Pond** that defines the following **private** attributes:



- fish - an array which will contain all **Fish** objects that are in the pond
- numFish - an **int** indicating the # of fish that are currently in the pond

Write the following:
- a constructor that takes the capacity of the pond. The capacity is the maximum number of fish that can be stored in the pond at any time.
- a get method for the numFish attribute.
- a method called **isFull()** which returns a **boolean** indicating whether or not the pond has reached its capacity of fish.
- a **toString()** method that returns a string in the following format: `"Pond with 8 fish"`

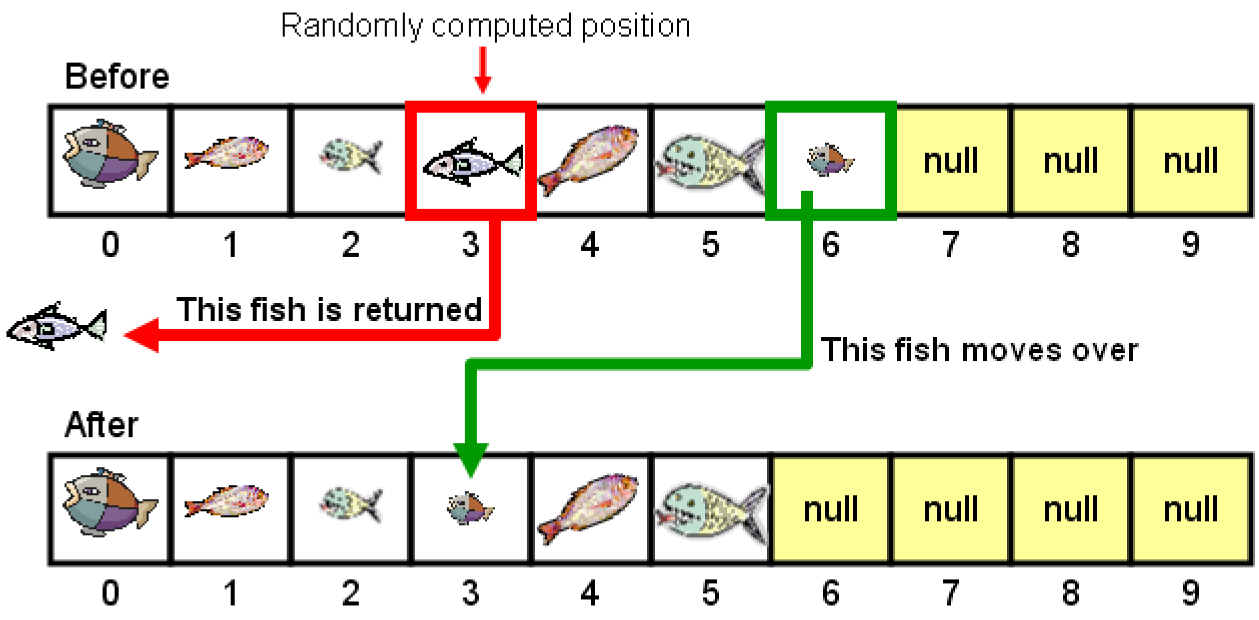
Also write the following more interesting methods:

- a method called **add()** which takes a single **Fish** parameter and adds that fish to the pond, provided that the pond is not full. If the pond is full to capacity, simply do nothing. You should assume that the fish is always added to the first available position in the array.

- a method called **listFish()** which lists (using System.out.println) all fish in the pond as follows:

```
Pond with 3 fish as follows:
A 35 cm Pike
A 6 cm Sunfish
A 10 cm Bass
```

- a method called **catchAFish()** which returns a ***random*** fish from the pond. The fish should be removed from the pond and returned from the method. If there are no fish, then **null** should be returned. Note however, that we cannot remove from the array, so simply fill in that position in the array with **null**. You should assume that the fish are all kept consecutively in the array from indices 0 through numFish-1 (i.e., no empty spaces in between). Use **Math.random()** to compute a random position in the array. Once you take the fish out, move the last fish in the array to fill in the gap so that the fish are stored consecutively again.



Here is a test program that creates 10 fish in a pond and then randomly catches 7. Run the program many times to make sure that it works properly without crashing:

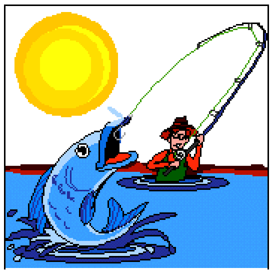
```java
public class PondTestProgram {
    public static void main(String args[]) {
        // Create a pond with 10 fish
        Pond pond = new Pond(15);
        pond.add(new Fish(4, "Sunfish"));
        pond.add(new Fish(25, "Pike"));
        pond.add(new Fish(20, "Bass"));
        pond.add(new Fish(30, "Perch"));
        pond.add(new Fish(4, "Sunfish"));
        pond.add(new Fish(15, "Pike"));
        pond.add(new Fish(9, "Pike"));
        pond.add(new Fish(12, "Bass"));
        pond.add(new Fish(5, "Sunfish"));
        pond.add(new Fish(12, "Sunfish"));
        pond.listFish();
        // Now catch 7 random fish
        System.out.println("Catching 7 random fish as follows ...");
        for (int i=0; i<7; i++)
            System.out.println(pond.catchAFish());
        System.out.println();

        // Now show what is left in the pond
        pond.listFish();

        // Now try to catch 5 random fish ... but only 3 actually left
        System.out.println("Catching 5 random fish as follows ...");
        for (int i=0; i<5; i++)
            System.out.println(pond.catchAFish());
        System.out.println();

        // Now show what is left in the pond
        pond.listFish();
    }
}
```

## (3) The **Fisher** class



Define a class called **Fisher** with the following **private** attributes:

- name - a String representing the name of the person who is fishing
- fishCaught - an array containing all **Fish** objects that were kept by this person
- numFishCaught - an **int** indicating the number of fish that were caught and are currently being kept by this person
- keepSize - an **int** indicating the minimum size of fish that this person is willing to keep. Anything less will be thrown back.

Make the following publicly accessible **static** variable:

- **LIMIT** = 3 (which is the maximum number of fish that all fishers are allowed to keep by "law").

Write the following:

- any get methods that you would like to have
- a constructor that takes the name of the person and the minimum size of fish that can be kept
- a **toString()** method that returns a string in the following format: `"Bob with 2 fish"`

Also write the following more interesting methods:

- a method called **keep()** which takes a single **Fish** parameter and causes the fisher to keep this fish. If the fisher has already reached the **LIMIT** with regards to how many fish can be kept, this method does nothing.

- a method called **likes()** which takes a single **Fish** parameter and returns a **boolean** indicating whether or not this fisher would like to keep this fish. A fisher likes a fish only if the fish is at least the desired keepSize for this person and the fish is NOT a **Sunfish**.

- a method called **listFish()** which lists (using System.out.println) all fish caught and kept by this fisher as follows:

```
                    Bob with 2 fish as follows:
                    A 4 cm Pike
                    A 15 cm Bass
```

At this point, test your **keep()**, **likes()** and **listFish()** methods using the following test program.   Do not continue until your code works:

```java
public class FisherTestProgram1 {
    public static void main(String [] args) {
        // Create two people to fish in the pond
        Fisher fred = new Fisher("Fred", 15);
        Fisher suzy = new Fisher("Suzy", 10);

        Fish f1 = new Fish(4, "Sunfish");
        Fish f2 = new Fish(25, "Sunfish");
        Fish f3 = new Fish(10, "Pike");
        Fish f4 = new Fish(15, "Pike");
        Fish f5 = new Fish(20, "Pike");

        System.out.println(fred.likes(f1));    // false
        System.out.println(fred.likes(f2));    // false
        System.out.println(fred.likes(f3));    // false
        System.out.println(fred.likes(f4));    // true
        System.out.println(fred.likes(f5));    // true

        System.out.println(suzy.likes(f1));    // false
        System.out.println(suzy.likes(f2));    // false
        System.out.println(suzy.likes(f3));    // true
        System.out.println(suzy.likes(f4));    // true
        System.out.println(suzy.likes(f5));    // true
        System.out.println();

        // FORCE the fishers to keep some fish (for testing purposes only)
        fred.keep(f1);
        fred.keep(f2);
        fred.keep(f3);
        fred.keep(f4);
        fred.keep(f5);
        suzy.keep(f2);
        suzy.keep(f3);

        System.out.println();

        // Now display the fish that were kept
        fred.listFish();   // should show 3 fish
        suzy.listFish();   // should show 2 fish
    }
}
```

Now create a method called **goFishingIn()** which takes a single **Pond** object parameter and returns no specific value.   The method should catch a random fish from the pond and then decide whether or not the fisher "likes" the fish.   If he/she "likes" the fish, the fish is removed from the pond and kept by the fisher.   If unable to keep the fish for any reason, the fish must be returned to the pond.   You MUST make use of your **keep()** , **likes()**, **catchAFish()** and **add()** methods that you wrote previously.

Change the **LIMIT** of fish that the fisher is able to catch from 3 to 10 and run the following test program:

```
public class FishingTestProgram2 {
    public static void main(String [] args) {
        // Create a big pond with 15 fish
        Pond   bigPond = new Pond(15);
        bigPond.add(new Fish(4, "Sunfish"));
        bigPond.add(new Fish(25, "Pike"));
        bigPond.add(new Fish(20, "Bass"));
        bigPond.add(new Fish(30, "Perch"));
        bigPond.add(new Fish(14, "Sunfish"));
        bigPond.add(new Fish(15, "Pike"));
        bigPond.add(new Fish(9, "Pike"));
        bigPond.add(new Fish(12, "Bass"));
        bigPond.add(new Fish(5, "Sunfish"));
        bigPond.add(new Fish(12, "Sunfish"));
        bigPond.add(new Fish(10, "Bass"));
        bigPond.add(new Fish(2, "Bass"));
        bigPond.add(new Fish(16, "Perch"));
        bigPond.add(new Fish(30, "Sunfish"));
        bigPond.add(new Fish(7, "Perch"));
        bigPond.listFish();

        // Create two people to fish in the pond
        Fisher fred = new Fisher("Fred", 15);
        Fisher suzy = new Fisher("Suzy", 10);

        System.out.println("First Fred catches 20 fish in the big pond ...");
        for (int i=0; i<20; i++)
            fred.goFishingIn(bigPond);
        fred.listFish();

        System.out.println("Suzy now catches 20 fish in the big pond ...");
        for (int i=0; i<20; i++)
            suzy.goFishingIn(bigPond);
        suzy.listFish();

        System.out.println("Here is what is left of the pond ...");
        bigPond.listFish();
    }
}
```

Run the program a few times. You should notice that **Fred** keeps an average of between 3 to 5 fish each time while **Suzy** keeps an average of between 2 and 4 ... depending on how many **Fred** keeps (because he fishes first). However, if you ran it many times and **Fred** kept catching sunfish and small fish, likely **Suzy** can get up to 6 or 7 fish. Notice as well that the fish remaining after the fishing are almost all **Sunfish** or small fish. Occasionally, some other fish are in there because they avoided being caught ;).

Finally, write a method called **giveAwayFish()** which represents a person quitting altogether and returning his/her fish to the pond and/or giving them away to another fisher. The method should take two parameters. The first parameter should be another **Fisher** representing the person that this fisher wants to give the fish to. The second parameter should be a **Pond**, representing the pond to return any unwanted fish to. The method works as follows. Go through all of this person's fish (i.e., the one giving the fish away) and see if the other fisher (i.e., the one who will be receiving the fish) is willing to keep any. If the other fisher wants any, they are to be given to that fisher. You should make use of the **likes()** and **keep()** methods. Otherwise, if the fisher is unwilling to keep the fish, then these fish must be returned to the pond. By the end of the method, the fisher giving away the fish should have no fish left and no fish may be lost (i.e., must be with the other fisher or in the pond).

Test your code by adding the following to the bottom of the **FishingTestProgram2**:

```
// Now simulate Suzy giving her fish to Fred
suzy.giveAwayFish(fred, bigPond);
fred.listFish();
suzy.listFish();
bigPond.listFish();
```

You should notice that **Fred** will keep some of **Suzy's** fish, but not the smaller ones...which will be added back to the pond. Make sure that the total fish in the pond and with **Fred** add up to **15** ;).

---

**NOTE:** Submit all **.java** and **.class** files needed to run. You **MUST NOT use packages** in your code, **nor projects**. Submit ALL of your files in one folder such that they can be opened and compiled individually in JCreator. Some IDEs may create packages and/or projects automatically. You **MUST** export the .java files and remove the package code at the top if it is there. Do NOT submit JCreator projects either. **JUST SUBMIT the JAVA and CLASS FILES**. Note that if your internet connection at home is down or does not work, we will not accept this as a reason for handing in an assignment late ... so make sure to submit the assignment WELL BEFORE it is due !

**Please NOTE that you WILL lose marks on this assignment if any of your files are missing. You will also lose marks if your code is not written neatly with proper indentation. See examples in the notes for proper style.**

---