COMP1406 - Assignment #4

(Due: **Thursday, February 13th @ 11:30pm)**

In this assignment, you will implement a Fish/Lake simulation similar to the previous assignment. You will then make adjustments to accommodate class hierarchies and make use of inheritance as well as a JAVA interface.

(1) The Basic Classes

Consider the following **Fish**, **Lake** and **Fisher** classes. Create a folder called **Part1** and copy all the code into unique files into that folder and then compile them:

```
public class Fish {
    // Any fish below this size must be thrown back into the lake
    public static int    THROW_BACK_SIZE = 18;

    protected int  size;
    protected float   weight;

    public Fish(int aSize, float aWeight) {
       size = aSize;
       weight = aWeight;
    }

    public boolean isDesirableTo(Fisher f) {
       // Replace the line below with your code
       return false;
    }

    public boolean canKeep() {
       // Replace the line below with your code
       return false;
    }

    public int getSize() { return size; }
    public float getWeight() { return weight; }

    public String toString () {
       return ("A " + size + "cm " + weight + "kg Fish");
    }
}
```

```
public class Lake {
    private Fish[]    catchableThings;
    private int       numThings;

    public Lake(int capacity) {
       catchableThings = new Fish[capacity];
       numThings = 0;
    }

    public int getNumCatchableThings() { return numThings; }
    public boolean isFull() { return numThings == catchableThings.length; }
    public String toString() { return "Lake with " + numThings + " catchable things"; }

    // Add the given thing to the lake
    public void add(Fish aCatchableThing) {
       if (numThings < catchableThings.length)
          catchableThings[numThings++] = aCatchableThing;
```

```
    }

    // Choose a random thing to be caught in the lake and return it
    public Fish catchSomething() {
       if (numThings == 0) return null;

       int   index = (int)(Math.random() * numThings);
       Fish f = catchableThings[index];
       catchableThings[index] = catchableThings[numThings-1];
       catchableThings[numThings-1] = null;
       numThings--;
       return f;
    }

    // List all things in the lake
    public void listAllThings() {
       System.out.println("  " + this + " as follows:");
       for (int i=0; i<numThings; i++)
          System.out.println("    " + catchableThings[i]);
       System.out.println();
    }
}
```

```
public class Fisher {
    public static int    LIMIT = 10; // max # of fish that can be caught

    private String    name;
    private Fish[]    thingsCaught;
    private int       numThingsCaught;
    private float     weightLimit;

    public Fisher(String aName, int wl) {
       name = aName;
       thingsCaught = new Fish[LIMIT];
       numThingsCaught = 0;
       weightLimit = wl;
    }

    public String getName() { return name; }
    public int getNumThingsCaught() { return numThingsCaught; }
    public float getWeightLimit() { return weightLimit; }

    public String toString() {
       return name + " with " + numThingsCaught + " things caught";
    }

    // List all the things caught and kept by this fisher
    public void listThingsCaught() {
       System.out.println("  " + this + " as follows:");
       if (numThingsCaught > 0) {
          for (int i=0; i<numThingsCaught; i++)
             System.out.println("    " + thingsCaught[i]);
       }
       System.out.println("    -------------------------");
       System.out.println("    Total fish weight: " + getWeightSoFar() + "kg");
       System.out.println();
    }

    // Cause this fisher to keep the thing caught.
    public void keep(Fish aThing) {
       thingsCaught[numThingsCaught++] = aThing;
    }

    // Simulate the fisher catching something in the lake and keeping what is caught if
    // it is desirable, otherwise throwing it back into the lake again.
```

```
    public void goFishingIn(Lake aLake) {
       Fish aThing = aLake.catchSomething();
       if (aThing != null) {
          if (aThing.isDesirableTo(this)) {
             this.keep(aThing);
          }
          else {
             aLake.add(aThing);
          }
       }
    }

    // Cause this fisher to give all fish to the given fisher, unless the given
    // fisher exceeds the wight or catch limit, then throw the fish back into the lake.
    public void giveAwayFish(Fisher f, Lake aLake) {
       for (int i=0; i<numThingsCaught; i++) {
          if (thingsCaught[i].isDesirableTo(f))
             f.keep(thingsCaught[i]);
          else
             aLake.add(thingsCaught[i]);
       }
       numThingsCaught = 0;
    }

    // Return the weight of all caught fish so far
    public float getWeightSoFar() {
       float total = 0;
       for (int i=0; i<numThingsCaught; i++)
          total += thingsCaught[i].getWeight();
       return total;
    }
}
```

---

1. Complete the **canKeep()** method in the **Fish** class. A fish can be "*kept*" if its size is greater than the "throw-back" size.

2. Complete the **isDesirableTo()** method in the **Fish** class. A fish is "*desirable*" to the given fisher if ALL of the following can be satisfied:
   a. it can be "kept" (i.e., all fish less than 18cm must be thrown back).
   b. the Fisher has less than his LIMIT of fish that may be caught.
   c. the weight of this fish combined with the "total weight of all fish caught by the Fisher so far" has not exceeded the Fisher's weight limit.

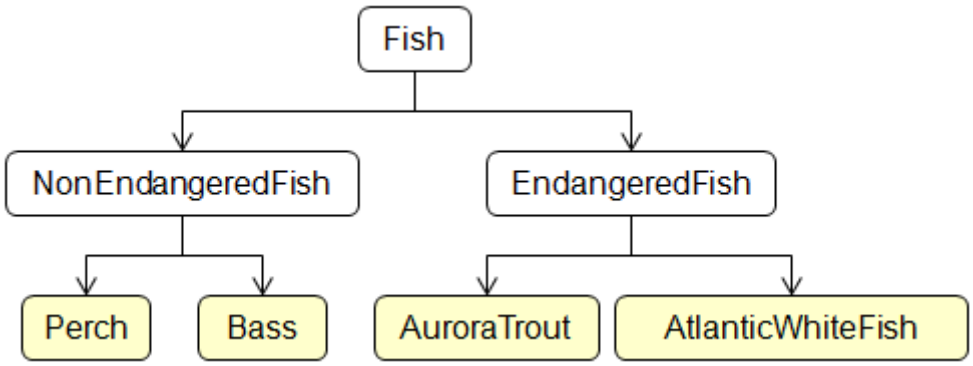
3. Test your code with the following program. Run it a few times and make sure that it is working.

```
public class FishingTestProgram1 {
   public static void main(String [] args) {
      // Create a big pond with 15 fish
      Lake   weirdLake = new Lake(15);
      weirdLake.add(new Fish(76, 6.1f));
      weirdLake.add(new Fish(32, 0.4f));
      weirdLake.add(new Fish(20, 0.9f));
      weirdLake.add(new Fish(30, 0.4f));
      weirdLake.add(new Fish(140, 7.4f));
      weirdLake.add(new Fish(15, 0.3f));
      weirdLake.add(new Fish(90, 5.9f));
      weirdLake.add(new Fish(120, 6.8f));
      weirdLake.add(new Fish(80, 4.8f));
      weirdLake.add(new Fish(42, 3.2f));
      weirdLake.add(new Fish(100, 5.6f));
      weirdLake.add(new Fish(45, 2.0f));
      weirdLake.add(new Fish(16, 0.2f));
```

```
        weirdLake.add(new Fish(30, 1.2f));
        weirdLake.add(new Fish(7, 0.1f));

        System.out.println("Here is what is in the lake to begin with ...");
        weirdLake.listAllThings();

        // Create two people to fish in the lake
        Fisher fred = new Fisher("Fred", 25);
        Fisher suzy = new Fisher("Suzy", 15);

        System.out.println("Fred casts his fishing line into the lake 10 times ...");
        for (int i=0; i<10; i++)
           fred.goFishingIn(weirdLake);
        fred.listThingsCaught();

        System.out.println("Suzy casts her fishing line into the lake 10 times ...");
        for (int i=0; i<10; i++)
           suzy.goFishingIn(weirdLake);
        suzy.listThingsCaught();

        System.out.println("Here is what remains in the lake ...");
        weirdLake.listAllThings();

        // Now simulate Suzy giving her fish to Fred
        System.out.print("Suzy now gives all her fish to Fred (if he can keep ");
        System.out.println("them), otherwise she returns them to  the lake ...");
        suzy.giveAwayFish(fred, weirdLake);
        suzy.listThingsCaught();
        fred.listThingsCaught();

        System.out.println("Here is the lake when Fred and Suzy go home ...");
        weirdLake.listAllThings();
    }
}
```

(2) The **Fish** Subclasses

Create a folder called **Part2** and copy all of the files from **Part1** into it. Create all additional subclasses necessary to construct the following hierarchy of classes:



Note that **Fish**, **NonEndangeredFish** and **EndangeredFish** are all **abstract** classes, while the other 4 are concrete classes. With the exception of the **Fish** class, none of the other classes here have any attributes defined within them. Follow the steps on the next page.

1. Change **Fish** to be an **abstract** class. Change the **canKeep()** method to be an abstract method.

2. Adjust the **toString()** method in the **Fish** class so that it displays the proper type of fish. You cannot use **instanceof** nor use any **IF** statements.

   (e.g., `"A 16cm 0.2kg Perch"` instead of `"A 16cm 0.2kg Fish"`)

3. Using maximum use of inheritance, create a **toString()** method in the **EndangeredFish** class so that endangered fish display as follows:

   ```
   "A 42cm 3.2kg AuroraTrout (ENDANGERED)"
   "A 80cm 4.8kg AtlanticWhiteFish (ENDANGERED)"
   ```

4. Write the **canKeep()** method in the **NonEndangeredFish** and **EndangeredFish** classes so that endangered fish can never be kept and non-endangered fish can be kept only if their size is greater than the "throw-back" size as defined in the **Fish** class.

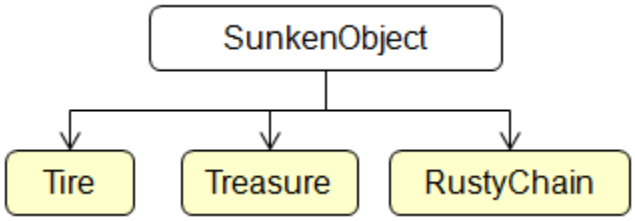
5. Test your code using the following program (copy the code into a file called **FishingTestProgram2.java**). You will need to create some constructors as well in order to get this to work.

```
public class FishingTestProgram2 {
    public static void main(String [] args) {
        // Create a big lake with 15 fish
        Lake   weirdLake = new Lake(15);
        weirdLake.add(new AuroraTrout(76, 6.1f));
        weirdLake.add(new Perch(32, 0.4f));
        weirdLake.add(new Bass(20, 0.9f));
        weirdLake.add(new Perch(30, 0.4f));
        weirdLake.add(new AtlanticWhiteFish(140, 7.4f));
        weirdLake.add(new Bass(15, 0.3f));
        weirdLake.add(new Bass(90, 5.9f));
        weirdLake.add(new Bass(120, 6.8f));
        weirdLake.add(new AtlanticWhiteFish(80, 4.8f));
        weirdLake.add(new AuroraTrout(42, 3.2f));
        weirdLake.add(new Bass(100, 5.6f));
        weirdLake.add(new Bass(45, 2.0f));
        weirdLake.add(new Perch(16, 0.2f));
        weirdLake.add(new Bass(30, 1.2f));
        weirdLake.add(new Perch(7, 0.1f));

        System.out.println("Here is what is in the lake to begin with ...");
        weirdLake.listAllThings();

        // Create two people to fish in the lake
        Fisher fred = new Fisher("Fred", 25);
        Fisher suzy = new Fisher("Suzy", 15);

        System.out.println("Fred casts his fishing line into the lake 10 times ...");
        for (int i=0; i<10; i++)
            fred.goFishingIn(weirdLake);
        fred.listThingsCaught();

        System.out.println("Suzy casts her fishing line into the lake 10 times ...");
        for (int i=0; i<10; i++)
            suzy.goFishingIn(weirdLake);
        suzy.listThingsCaught();

        System.out.println("Here is what remains in the lake ...");
        weirdLake.listAllThings();

        // Now simulate Suzy giving her fish to Fred
```

```
            System.out.print("Suzy now gives all her fish to Fred (if he can keep ");
            System.out.println("them), otherwise she returns them to  the lake ...");
            suzy.giveAwayFish(fred, weirdLake);
            fred.listThingsCaught();
            suzy.listThingsCaught();

            System.out.println("Here is the lake when Fred and Suzy go home ...");
            weirdLake.listAllThings();
        }
    }
```

(3) The **Other** Subclasses

Create a folder called **Part3** and copy all of the files from **Part2** into it. Create all additional classes necessary to form this hierarchy:



Abstract class **SunkenObject** should have a **weight** (i.e., a float) attribute associated with it.

1. Create a constructor that takes an initial weight.

2. Create a get method for the **weight**.

3. The **Tire**, **Treasure** and **RustyChain** classes should each have a zero-parameter constructor that calls the **SunkenObject** constructor with initial weights of 10, 20 and 15, respectively.

4. Write a single **toString()** method in the **SunkenObject** class so that **Tires**, **Treasures** and **RustyChains** look as follows when printed:

```
A 10kg Tire
A 20kg Treasure
A 15kg RustyChain
```

5. Create and compile an interface called **Catchable** which defines the following two methods:

```
public float getWeight();
public boolean isDesirableTo(Fisher f);
```

6. Have **SunkenObject** implement the **Catchable** interface. By default, **SunkenObjects** are NOT desirable. However, **Treasures** are desirable if the number of things caught by the fisher is below the **LIMIT** of items that are allowed to be caught (see **Fisher** class). Also, have Fish implement the **Catchable** interface.

7. Alter the **Lake** class so that they contain **Catchable** objects instead of just **Fish** objects.

8. Alter the **Fisher** class so that they keep **Catchable** objects as thingsCaught instead of just **Fish** objects.

9. Adjust the **giveAwayFish()** method in the **Fisher** class so that only **Fish** are given away, not **Treasures**.

10. Adjust the **getWeightSoFar()** method in the **Fisher** class so that it returns the weight of all **Fish** that have been caught and kept, ignoring the weight of any kept **Treasures**.

11. Test your code again by using the following test program. Run it a few times to be sure that everything is working. You will want to run until Suzy gets a treasure in order to make sure that she does not give it away to Fred, nor throw it back into the lake. Also, you will want to verify that Fred never exceeds his weight limit (especially when Suzy gives him her fish) and that no fish below 18cm are kept by anyone.

```
public class FishingTestProgram3 {
    public static void main(String [] args) {
        // Create a big lake with 15 fish, 2 tires, 2 treasures and 2 rusty chains
        Lake   weirdLake = new Lake(21);
        weirdLake.add(new AuroraTrout(76, 6.1f));
        weirdLake.add(new Tire());
        weirdLake.add(new Perch(32, 0.4f));
        weirdLake.add(new Bass(20, 0.9f));
        weirdLake.add(new Treasure());
        weirdLake.add(new Perch(30, 0.4f));
        weirdLake.add(new AtlanticWhiteFish(140, 7.4f));
        weirdLake.add(new RustyChain());
        weirdLake.add(new Bass(15, 0.3f));
        weirdLake.add(new Tire());
        weirdLake.add(new Bass(90, 5.9f));
        weirdLake.add(new Bass(120, 6.8f));
        weirdLake.add(new AtlanticWhiteFish(80, 4.8f));
        weirdLake.add(new AuroraTrout(42, 3.2f));
        weirdLake.add(new Bass(100, 5.6f));
        weirdLake.add(new Bass(45, 2.0f));
        weirdLake.add(new RustyChain());
        weirdLake.add(new Perch(16, 0.2f));
        weirdLake.add(new Bass(30, 1.2f));
        weirdLake.add(new Treasure());
        weirdLake.add(new Perch(7, 0.1f));

        System.out.println("Here is what is in the lake to begin with ...");
        weirdLake.listAllThings();

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

        System.out.println("Fred casts his fishing line into the lake 10 times ...");
        for (int i=0; i<10; i++)
            fred.goFishingIn(weirdLake);
        fred.listThingsCaught();

        System.out.println("Suzy casts her fishing line into the lake 10 times ...");
        for (int i=0; i<10; i++)
            suzy.goFishingIn(weirdLake);
        suzy.listThingsCaught();

        System.out.println("Here is what remains in the lake ...");
        weirdLake.listAllThings();

        // Now simulate Suzy giving her fish to Fred
        System.out.print("Suzy now gives all her fish to Fred (if he can keep ");
        System.out.println("them), otherwise she returns them to  the lake ...");
        suzy.giveAwayFish(fred, weirdLake);
        fred.listThingsCaught();
        suzy.listThingsCaught();

        System.out.println("Here is the lake when Fred and Suzy go home ...");
```

```
            weirdLake.listAllThings();
        }
    }
```

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.
