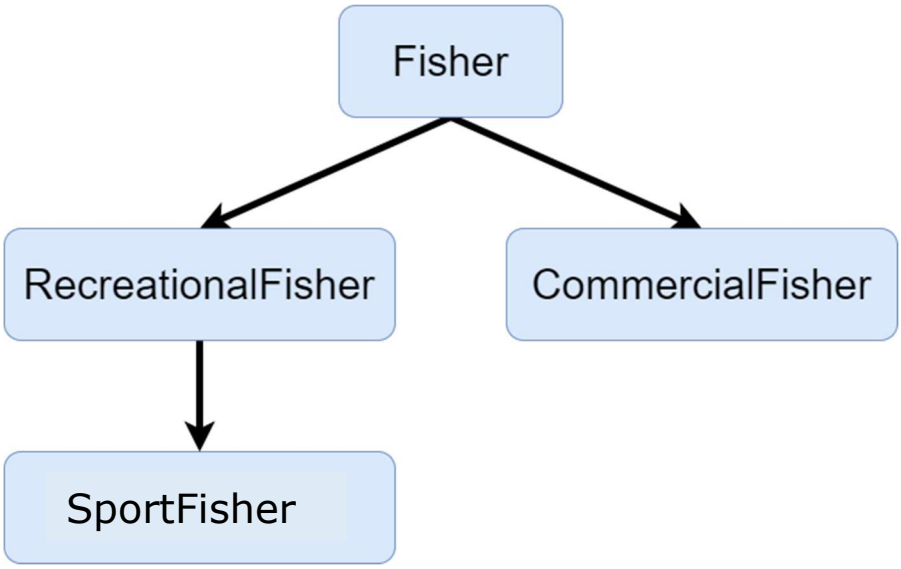
DUE DATE: NOVEMBER 24, 2021 AT 4:59PM

Instructions:

- You must complete the "**Blanket Honesty Declaration**" checklist on the course website before you can submit any assignment.
- Only submit the **java files**. Do ***not*** submit any other files, unless otherwise instructed.
- To submit the assignment, upload the specified files to the **Assignment 3** folder on the course website.
- Assignments must follow the **programming standards** document published on UMLearn.
- After the due date and time, assignments may be submitted but will be subject to a late penalty. Please see the ROASS document published on UMLearn for the course policy for late submissions.
- If you make multiple submissions, only the **most recent version** will be marked.
- These assignments are your chance to learn the material for the exams. Code your assignments independently. We use software to compare all submitted assignments to each other, and **pursue academic dishonesty vigorously**.
- Your Java programs must compile and run upon download, without requiring any modifications.
- Automated tests will be used to grade the output of your assignment. If you do not follow precisely the guidelines described below, these automated tests can fail and you will lose marks. In particular, **make sure that your methods are spelled exactly as described below**. Also**, make sure that your code produces exactly the example outputs (including correct spacing)** shown in these guidelines.

Assignment overview

In this assignment, you will implement a simplified Fishing Tracking program.
You will start by building classes to represent a type of fish, an actual fish, and a fishing trip.
In phase 2, you will build a hierarchy of Fisher classes (see the representation below).



Testing Your Code

We have provided sample test files for each phase to help you verify that your code is working to the assignment specifications (e.g., correct method headers). **These files are only *starting* points for testing your code.** Part of developing your skills as a programmer is to think through additional important test cases and to write your own code to test these cases. The test files should give you a sense what this additional code could consist of.

Phase 1: FishType, Fish, FishingTrips:

First, implement three simple classes: a **FishType** class, a **Fish** class. All methods should be public unless otherwise specified.

The **FishType** class should have:

- Four instance variables: the **name** of the fish (a String), the **rarity** of the fish type (a String), the **price** per pound (a double) and the minimum age **restriction** for catching a fish of this type (an int).
- A constructor that has four parameters (String, String, double, int), which are used to initialize the four instance variables described above (in the same order).
- Three accessor methods: **getRarity**() which returns the FishType's rarity, **getPrice**() which returns the price of the FishType, and **getRestriction**() which returns the minimum age restriction.
- A void method **changePrice**(double) which sets the FishType's current price to the value received as a parameter.
- A **getDescription**() method that return a String containing the FishType's name, and the rarity (in between parentheses).
- A standard **toString**() method, which returns a String containing the FishType's name, rarity (in between parentheses), price, and minimum age restriction (in between parentheses). Format this string as shown in the output below. Use the String.format method to insert the double values with only 2 digits after the decimal point.
- A **`boolean` equals(Object)** method that checks if the provided Object is equal to this FishType. Two FishTypes are equal if they have the same name, rarity, restriction, and that their difference in price is less than half a cent (0.005).

The **Fish** class should have:

- Three instance variables: the **type** of the Fish (a FishType), the **weight** of the fish in pounds (a double), and the **age** of the fish (an int).
- A constructor with three parameters (FishType, double, int), which are used to initialize the three instance variables described above (in the same order).
- Three accessor methods **getWeight**(), **getType**(), and **getAge**() that return the values of the corresponding instance variables.
- A **double getValue**() method, which returns the total value of the Fish.
- A **boolean isValid**() method, which will return whether or not this fish matches or is over the minimum age restriction and can therefore be kept, or must be released.
- A **toString**() method which returns a String containing the FishType's name, the FishType's rarity (given by FishType.getDescription()), the fish's weight, and the total value of the fish (given by getValue()) in between parentheses and whether it is a legal catch. See the example output below for the exact formatting.
- A **`boolean` equals(Object)** method that checks if the provided Object is equal to this Fish. Two Fish are equal if they have the same type, their difference in weight is less than 1 hundredth of a pound, and their **isValid**() result is identical.

You can begin testing your classes with **`TestFishTypes.java, and TestFish.java`**. You should get the output shown below.

`TestFishTypes.java`:

```
Created four FishType instances, let's see how they look:
FishType1 toString: Ayu, Sweetfish (Common) $5.00 (1)
FishType2 toString: Ayu, Sweetfish (Common) $5.01 (1)
FishType3 description: Ayu, Sweetfish (Common)
FishType4 description: Bicolor Goatfish, (Uncommon)
```

```
Let's take the Equals method for a test drive:
Are FishType1 and FishType2 equal? They should NOT be: false
Are FishType1 and FishType3 equal? They should be: true
Are FishType2 and FishType3 equal? They should be: true
Are FishType1 and FishType4 equal? They should NOT be: false

Time for some getters and setters:
How rare is FishType1? Common
How rare is FishType4? Uncommon
What's the price per pound of FishType1? $5.00
What's the price per pound of FishType4? $10.00
What's the new price per pound of FishType3? $5.25
What's the age restriction of FishType1? 1
What's the age restriction of FishType4? 2
```

`TestFish.java`:

```
Created five Fish instances, let's see how they look:
Fish1: Ayu, Sweetfish (Common), 2.10 pounds ($10.50), This is a legal catch
Fish2: Ayu, Sweetfish (Common), 2.11 pounds ($10.55), This is a legal catch
Fish3: Ayu, Sweetfish (Common), 1.10 pounds ($5.50), We should release this fish
Fish4: Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
Fish5: Bicolor Goatfish, (Uncommon), 1.51 pounds ($15.10), We should release this fish

Let's take the Equals method for a test drive:
Are Fish1 and Fish2 equal? They should be: true
Are Fish1 and Fish3 equal? They should NOT be: false
Are Fish4 and Fish5 equal? They should NOT be: false

Time for some getters and setters:
Fish1 is 2.10 pounds.
What type is Fish4? Bicolor Goatfish, (Uncommon) $10.00 (2)
Fish3 is 0 years old.
We could sell Fish2 for $10.55.
With the new price change, we could now sell Fish2 for $11.08.
Was Fish5 a valid catch? false
```

Phase 2 FishingTrip:

The **FishingTrip** class represents a fishing trip that has been made. It stores all the information about that fishing trip (an array of Fish caught). It should have:

- Three instance variables: a partially-filled array of fish **brought back** to shore (a Fish[]),the **amount** of fish being brought back (an int), and finally the **total** number of fish caught during the trip (an int), which may be greater than the amount of fish actually brought back. **We will add an additional instance variable in Phase 3.**
- A constructor with one parameter FishingTrip(int maxAmount), which is used to instantiate the partially-filled array. The amount and total number of fish caught are always initialized to 0. **We will create an additional constructor in Phase 3.**
- Two accessor methods **getTotal**() and **getAmount**() that return the value of the corresponding instance variables.
- A **double getTotalValue**() method, which returns the total value of all of the fish caught.
- A **double getTotalWeight**() method, which returns the total weight of all of the fish brought back.
- A **boolean catchFish(Fish newFish)** method, which adds the Fish object received as a parameter to the array of Fish to be brought back, **if there is room for it**. The method will return a boolean indicating if the fish was added to the array. This method should also increment **amount** if the fish was added to the array. The **total** instance variable should be incremented regardless of whether or not the fish was added.
- A **Fish releaseFish(int index)** method which removes the Fish located at the index parameter from the brought back array and returns it to the caller.
    - This method should check to ensure that the index it is passed is valid, if the index is invalid, the method should return null.

    - Once a fish is removed from the array, the elements after it in the array should be arranged to ensure there are no empty/null spots in the array.
    - This method should decrement **amount** accordingly if the fish was successfully released.
- A **toString**() method which returns a String containing the total amount of fish caught, the amount being brought back out of the maximum possible, and then listing each individual fish being brought back. See the example output below for the exact formatting.
- A **boolean equals(Object)** method that checks if the provided Object is equal to this FishingTrip. Two FishingTrips are considered equal if the difference between their **getTotalValue**() results is less than ten cents (0.1), the difference between their **getTotalWeight**() results is less than 1 tenth of a pound (0.1), and that their amount and total number of caught fish match.

You can begin testing your classes with **`TestFishingTrips.java`**. You should get the output shown below.

`TestFishingTrips.java`:

```
Created three FishingStrip instances, let's see how they look:
Trip1: During this FishingTrip, 0 total Fish were caught.
Sadly, the boat is empty.
Trip2: During this FishingTrip, 0 total Fish were caught.
Sadly, the boat is empty.
Trip3: During this FishingTrip, 0 total Fish were caught.
Sadly, the boat is empty.

Trip1 is trying to catch a fish, can it be brought back? true
Trip1 is trying to catch a fish, can it be brought back? true
Trip1 is trying to catch a fish, can it be brought back? true
Trip1 is trying to catch a fish, can it be brought back? true
Trip1 is trying to catch a fish, can it be brought back? true
Trip1 is trying to catch a fish, can it be brought back? false

Let's check the total value of the FishingTrips:
Trip1: $2.63.
Trip2: $63.43.
Trip3: $2.63.

Let's check the total weight of the FishingTrips:
Trip1: 10.50 pounds.
Trip2: 11.52 pounds.
Trip3: 10.51 pounds.

Let's see how they look now:
Trip1: During this FishingTrip, 6 total Fish were caught, and 5/5 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch

Trip2: During this FishingTrip, 5 total Fish were caught, and 5/99 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.11 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 1.10 pounds ($0.28), We should release this fish
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
 > Bicolor Goatfish, (Uncommon), 1.51 pounds ($15.10), We should release this fish

Trip3: During this FishingTrip, 5 total Fish were caught, and 5/15 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.11 pounds ($0.53), This is a legal catch

Time to try out releasing some fish:
Release the fist invalid Fish in Trip2: Ayu, Sweetfish (Common), 1.10 pounds ($0.28), We should
release this fish
Release the second invalid Fish in Trip2: Bicolor Goatfish, (Uncommon), 1.51 pounds ($15.10), We
should release this fish
```

```
Try to release an invalid Fish in Trip2: null

Check Trip2:
During this FishingTrip, 5 total Fish were caught, and 3/99 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.11 pounds ($0.53), This is a legal catch
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch

Let's release all the Fish in Trip2

Check Trip2 again:
During this FishingTrip, 5 total Fish were caught.
Sadly, the boat is empty.

Let's take the Equals method for a test drive:
Are Trip1 and Trip2 equal? They should NOT be: false
Are Trip1 and Trip3 equal? They should NOT be: false
Are Trip1 and Trip3 equal? They should be now: true
```

Phase 3 Fisher hierarchy:

Next, implement these classes: **Fisher**, **RecreatonalFisher**, **SportFisher**, and **CommercialFisher** following the descriptions below. All methods should be public unless otherwise specified.

The **Fisher** class must be an **abstract** class. It should have:

- Three main instance variables: the **name** of the fisher (a String), and the **limit** on the amount of fish this fisher can bring back (an int), and the **latest** FishingTrip this Fisher has gone on.
- A constructor that accepts the fisher's name (a String), and the fishing limit (an int), the latest FishingTrip starts off as null.
- Three accessor methods: **getName**(), **getLimit**(), and **getTrip**() which each return the appropriate instance variable.
- One mutator method **setTrip**(FishingTrip) which will set the appropriate instance variable.
- A abstract public **fishCaught**(Fish) method that will update the relevant values in the classes that implement it, this method will be called in the FishingTrip class's **catchFish**() method, detailed further below.
- A **toString**() method that returns a String containing the fisher's name, fishing limit, and latest FishingTrip (formatted exactly as shown in the sample output below).
- A **abstract String describeLatestTrip**() method that will return a String describing the latest FishingTrip, each child of Fisher will implement this method slightly differently.
- A **boolean equals(Object)** method that checks if the provided Object is equal to this Fisher. Two Fishers are considered equal if their names and limits are the same.

The **RecreationalFisher** class should be a subclass of **Fisher**. It should have:

- One additional instance variable, the **grand total** of all fish ever caught by this fisher (an int). Note that this number also includes fish that were caught and released.
- A constructor that receives only the name parameter, in which case the limit will be 0, and the grand total is initialized to 0. This constructor should call the superclass constructor to initialize the instance variables.
- The RecreationalFisher class should also have another constructor that accepts both a name and a limit. This constructor should call the superclass constructor to initialize the instance variables.
- A implementation of **fishCaught**(Fish) method that will increment the grand total of fish caught.
- One accessor methods: **getGrandTotal**(), which returns the appropriate instance variable.
- An overridden **toString**() method that returns a String containing the result from its parent class's toString(), as well as the total number of fish ever caught (formatted exactly as shown in the sample output below).
- An implementation of **describeLatestTrip**() which will just return the result of the **toString**() method of the FishingTrip (formatted exactly as shown in the sample output below).

The **SportFisher** class should be a subclass of **RecreationalFisher**. It should have:

- One additional instance variable, the **total weight** of all fish ever caught by this fisher (a double). Note that this number also includes fish that were caught and released.
- A constructor that accepts the fisher's name (a String), and the fishing limit (an int). The total weight should be initialized to 0. This constructor should call the superclass constructor to initialize the instance variables.
- A implementation of **fishCaught**(Fish) method that will add the weight of the caught fish to the grand total. The parent class's fishCaught() method should also be called.
- One accessor methods: **getTotalWeight**(), which returns the appropriate instance variable.
- An overridden **toString**() method that returns a String containing the result from its parent class' toString(), as well as the total weight of fish ever caught (formatted exactly as shown in the sample output below).
- An overridden of **describeLatestTrip**() which will return the result of the **toString**() method of the FishingTrip and add the total weight of fish brought back (formatted exactly as shown in the sample output below).

The **CommercialFisher** class should be a subclass of **Fisher**. It should have:

- One additional instance variable, the **grand total value** of all fish ever caught by this fisher (a double). Note that this number also includes fish that were caught and released.
- A constructor that receives the name parameter, and the fishing limit (an int), the grand total is always initialized to 0. This constructor should call the superclass constructor to initialize the instance variables.
- One accessor methods: **getTotalValue** (), which returns the appropriate instance variable.
- A implementation of **fishCaught**(Fish) method that will add the value of the caught fish to the grand total.
- An overridden **toString**() method that returns a String containing the result from its parent class' toString(), as well as the total value of fish ever caught (formatted exactly as shown in the sample output below).
- An implementation of **describeLatestTrip**() which (unlike the other Fisher classes) will **NOT** return the result of the **toString**() method of the FishingTrip and rather only return the **getTotalValue**() for the trip (formatted exactly as shown in the sample output below).

Phase 3 FishingTrip additions

In this phase we will also extend the FishingTrip class as follows:

- We will add an additional instance variable, the fisher who's fishing trip this is (a Fisher object).
- A new constructor with one parameter: **FishingTrip**(Fisher), which sets the corresponding instance variable and calls the previously defined constructor (**FishingTrip**(int size)) using the Fisher object's fish limit value, the fisher's latest trip should be set to this one as well.

The **catchFish**(Fish) method should now call the **fishCaught**(Fish) method of it's fisher instance variable, if fisher is not null.You can begin testing your classes with **`TestFishers.java`**. You should get the output shown below.

`TestFishers.java`:

```
Created three Fisher instances, let's see how they look:
Fisher1: Gabriel, limited to 0 fish.
Latest Trip: During this FishingTrip, 0 total Fish were caught.
Sadly, the boat is empty.
In total, this Fisher has caught: 0 fish.

Fisher2: Jon, limited to 2 fish.
Latest Trip: During this FishingTrip, 0 total Fish were caught.
Sadly, the boat is empty.
In total, this Fisher has caught: 0 fish.
In total, this Fisher has caught: 0.00 pounds of fish.

Fisher3: Frédéric, limited to 5 fish.
Latest Trip: During this FishingTrip, 0 total Fish were caught.
Sadly, the boat is empty.
```

```
In total, this Fisher has caught: $0.00 worth of fish.


Catch a Fish for trip1, attached to a Recreational Fisher, they should all be false:
Trip1 is trying to catch a fish, can it be brought back? false
Trip1 is trying to catch a fish, can it be brought back? false

Fisher1 should now have two fish caught in total:
Gabriel, limited to 0 fish.
Latest Trip: During this FishingTrip, 2 total Fish were caught.
Sadly, the boat is empty.
In total, this Fisher has caught: 2 fish.

Fisher1 grand total: 2

Catch a Fish for trip1, attached to a Recreational Fisher, the first 2 should be true:
Trip2 is trying to catch a fish, can it be brought back? true
Trip2 is trying to catch a fish, can it be brought back? true
Trip2 is trying to catch a fish, can it be brought back? false

Fisher2 should now have three fish caught in total, and two brought back:
Jon, limited to 2 fish.
Latest Trip: During this FishingTrip, 3 total Fish were caught, and 2/2 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
In total, this Fisher has caught: 3 fish.
In total, this Fisher has caught: 8.31 pounds of fish.

Fisher2 describes their latest trip:
During this FishingTrip, 3 total Fish were caught, and 2/2 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
That's 6.80 pounds of fish!

Fisher2 grand total: 3
Fisher2 total weight: 8.31 pounds

Catch a Fish for trip1, attached to a Recreational Fisher, the first 5 should be true:
Trip3 is trying to catch a fish, can it be brought back? true
Trip3 is trying to catch a fish, can it be brought back? true
Trip3 is trying to catch a fish, can it be brought back? true
Trip3 is trying to catch a fish, can it be brought back? true
Trip3 is trying to catch a fish, can it be brought back? true
Trip3 is trying to catch a fish, can it be brought back? false

Fisher2 should now have six fish caught in total, and five brought back:
Frédéric, limited to 5 fish.
Latest Trip: During this FishingTrip, 6 total Fish were caught, and 5/5 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 2.11 pounds ($0.53), This is a legal catch
 > Ayu, Sweetfish (Common), 1.10 pounds ($0.28), We should release this fish
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
 > Bicolor Goatfish, (Uncommon), 1.51 pounds ($15.10), We should release this fish
In total, this Fisher has caught: $95.93 worth of fish.

Fisher2 total value: $95.93

Fisher1 describes their latest trip:
During this FishingTrip, 2 total Fish were caught.
Sadly, the boat is empty.

Fisher2 describes their latest trip:
During this FishingTrip, 3 total Fish were caught, and 2/2 were brought back:
 > Ayu, Sweetfish (Common), 2.10 pounds ($0.53), This is a legal catch
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
That's 6.80 pounds of fish!

Fisher3 describes their latest trip:
The latest Fishing Expedition resulted in $63.43 in profits.

Gave each Fisher a new FishingTrip:
```

```
Trip1 is trying to catch a fish, can it be brought back? false
Trip2 is trying to catch a fish, can it be brought back? true
Trip3 is trying to catch a fish, can it be brought back? true

Last look at the Fisher instances:
Fisher1: Gabriel, limited to 0 fish.
Latest Trip: During this FishingTrip, 1 total Fish were caught.
Sadly, the boat is empty.
In total, this Fisher has caught: 3 fish.

Fisher2: Jon, limited to 2 fish.
Latest Trip: During this FishingTrip, 1 total Fish were caught, and 1/2 were brought back:
 > Bicolor Goatfish, (Uncommon), 1.51 pounds ($15.10), We should release this fish
In total, this Fisher has caught: 4 fish.
In total, this Fisher has caught: 9.82 pounds of fish.

Fisher3: Frédéric, limited to 5 fish.
Latest Trip: During this FishingTrip, 1 total Fish were caught, and 1/5 were brought back:
 > Bicolor Goatfish, (Uncommon), 4.70 pounds ($47.00), This is a legal catch
In total, this Fisher has caught: $142.93 worth of fish.
```

Hand in

Submit your 7 Java files (**`FishType.java`**, **`Fish.java`**, **`FishingTrip.java`**, **`Fisher.java`**, **`RecreationalFisher.java`**, **SportFisher.java, CommercialFisher.java**). ***Do not submit .class or .java~ files!*** You do ***not*** need to submit the **`Test[N].java`** files that were given to you. If you did not complete both phases of the assignment, use the Comments field when you hand in the assignment to tell the marker which phases were completed, so that only the appropriate tests can be run. For example, if you say that you completed Phase 1, then the marker will compile your files and Test[N].java, and then run the main method in Test[N]. If it fails to compile and run, you will lose ***all*** of the marks for the test runs. The marker will ***not*** try to run anything else, and will ***not*** edit your files in any way. (***Make sure none of your files specify a package at the top)***