EXERCISE 6, ABSTRACT CLASSES AND INTERFACES

Exercise 6, Abstract classes

Objective

The objective of this lab is to provide you with the skills necessary to be able to:

- Create and work with abstract classes
- Define and implement interfaces

Overview

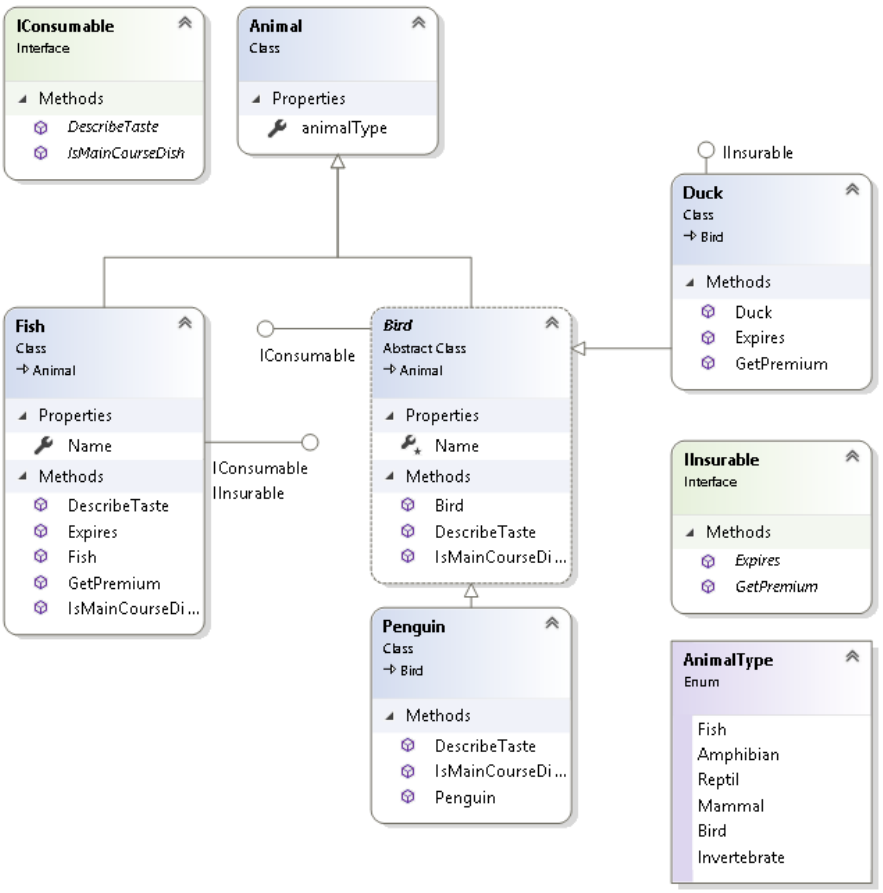
You are going to have Penguins, Ducks and Fish in a collection.

Penguins and Ducks will be derived from the Bird class. We have decided that "Bird" is too abstract for what we want.

All birds are consumable; and Ducks are insurable (Penguins aren't).

Fish (which is not derived from Bird) are also insurable but are consumable.

We'll iterate round the collection and call operations on all consumable things, then on all insurable things regardless of whether they are inherited from Bird or not.
The class diagram after completing the lab can be seen below.



Step by step

1. Create a new Console application called Lab06.
2. Add the following classes to the project
   a. Animal, Bird, Fish, Penguin and Duck
3. Add the following interfaces to the package
   a. Insurable, Consumable. You'll add code to these interface later on.
4. Create an enum to define the 6 animal types:

```
enum AnimalType {
    Fish,
    Amphibian,
    Reptil,
    Mammal,
    Bird,
    Invertebrate
}
```

5. Add code to the Animal class
   a. Make this class **abstract**
   b. Add a public property to define the type, like:
      AnimalType animalType { get; set;}
   c. Read only (private set) property method called Name

d. Create a constructor to set the *Name* and *animalType* properties

6. Add code to the **Bird** class

   a. Make this class **`abstract`**
   we want to make different kinds of birds, not a generic bird!

   b. **`Extends (inherits from)`** the Animal class

   c. You'll need a constructor as the base class (*Animal*) has a constructor with parameters. Set the type of the animal to ***`AnimalType.Bird`***

7. Write code for the **Duck** class

   a. **`Extends`** Bird
   By default a **Duck** will also be an **Animal** (Bird extends Animal)

   b. You'll need to provide a constructor because the base class (Bird) has constructor with a **string** parameter to set the Name.

8. Write code for the **Penguin** class

   a. **`Extends`** Bird

   b. Provide a constructor similar to the Duck class

9. Write code for the interfaces:

```
public interface Insurable {
    string getPremium();
    string expires();
}

public interface Consumable {
    string describeTaste();
    string isMainCourseDish();
}
```

> **Note** The **expires()** method should return a DateTime object but it returns a string to life easier for testing it in this exercise.

10. Check that everything compiles.

11. Implement the interfaces. Please view the **class diagram** on page one.

    a. For the **describeTaste()** and **isMainCourseDish()**,
    you could write something like:

```
public string describeTaste() {
    return Name + ": " + "Delicate";
}

public string isMainCourseDish() {
    return Name + ": " + true;
}
```

12. Open class **`Program`**.

    Now you write some code that invokes the just defined interface methods.

a. Create an array of Animal called animals
   Place different Animal instances in the array

b. Write an enhanced '**for**' loop that iterates over the animals array.
   This is an example of Polymorphism. Every Fish, Penguin, Duck can be referred to as an Animal and can therefore be placed in an array/List of type *Animal*.

c. Inside the loop test **if** the Animal object is **Consumable** and if it is, cast it as Consumable in order to call its describeTaste() and isMainCourseDish() methods.

13. Run your code to make sure it works.
14. Return the Main() method.

a. Write another enhanced '**for**' loop that iterates over the animals array.

b. Inside the loop, test **if** the Animal object is **Insuable** and if it is, cast it as Insurable in order to call its getPremium() and expires() methods.

** End **