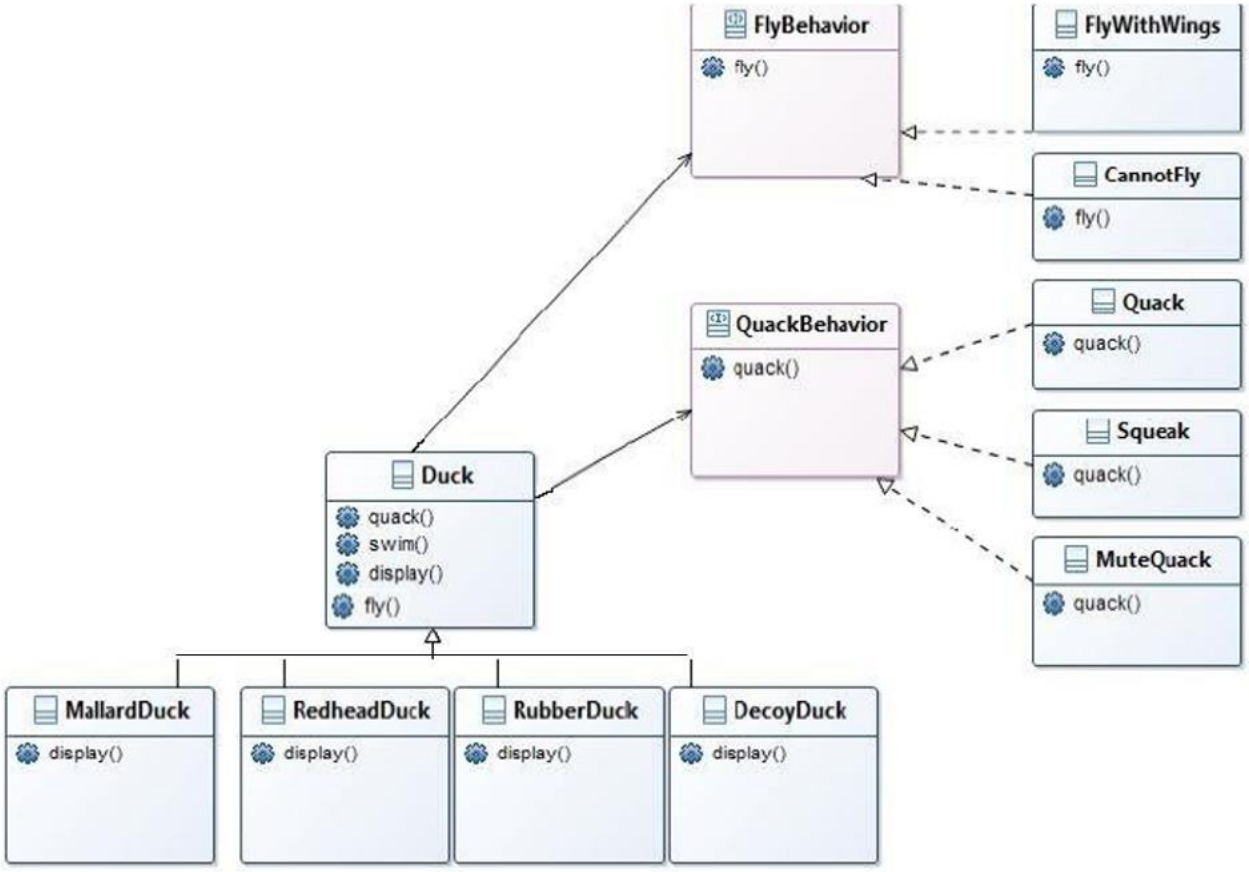
Lab 5

2. Implement the diagram in Java. To implement the methods like fly() and quack(), just print a phrase to the console, like “Flying with wings” or “Quack by squeaking.”



To test your code, create a Main class like the following:

```
public class Main {
    public static void main(String[] args) {
        Duck[] ducks =
            {new MallardDuck(), new DecoyDuck(), new RedheadDuck(), new RubberDuck()};
        for(Duck d: ducks) {
            System.out.println(d.getClass().getSimpleName() + ":");
            d.display();
            d.fly();
            d.quack();
            d.swim();
        }
    }
}
```

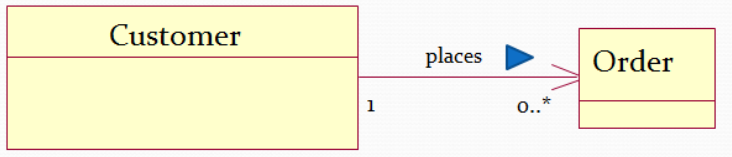
Output should look like this:

```
MallardDuck:
  display
  fly with wings
  quacking
  swimming
DecoyDuck:
  displaying
  cannot fly
  cannot quack
  swimming
RedheadDuck:
  displaying
  fly with wings
  quacking
  swimming
RubberDuck:
  displaying
  cannot fly
  squeaking
  swimming
```

4. In Lesson 2, one way of maintaining a unidirectional one-many relationship was mentioned – in that approach, creation of secondary objects (in the example, these were of type Order) was controlled by limiting visibility of the constructor to *package* level. For this problem, modify the code mentioned there (which can be found in the code folder for Lab 5) so that construction of Order objects is controlled by a factory method. The guidelines given in Lesson 2 for maintaining a one-many unidirectional association are reproduced here:

One-many Multiplicity:
- Associated with each Customer, there are zero or more Orders
- A Customer object maintains a collection of Order objects.
- Associated with each Order, there is exactly one Customer
- It is possible to navigate from a Customer to any of his Orders, but not from an Order to the owning Customer.
- *Maintaining the relationship* means:
  - when new Customer object is created, it is equipped with a (possibly empty) collection of Orders
  - it is not possible to create an Order object independent of a Customer; each new Order object must belong to the collection of Orders for some Customer.

Note: An example of using a factory method to maintain an association relationship was given in the Lesson 5 slides (see lesson5.lecture.factorymethods6 for the code showing a factory method, and see lesson2.lecture.unidirectional.onemany to see another way to code a one-many unidirectional relationship).



In your implementation, make sure that Customer, Order, and Item all belong to the same package and that the only way any of these classes can be instantiated is by using a factory method in a factory class (which you may wish to name as CustOrderFactory).