1. **Caterpillar** and **Bird** extend **Creature**. A Caterpillar can be decorated to become a **Butterfly**. Every kind of Creature has method `public void move()` which uses some **LocomotionStrategy** that provides a `public void performMove()` method. A Bird uses `Fly` (which just prints “I am flying”). A Caterpillar uses `Crawl` (which prints “I am crawling”), but it changes to `Fly` when it becomes a Butterfly.

   ```java
   Creature c1 = new Caterpillar();
   Creature c2 = new Bird();
   c1.move();               // prints “I am crawling”
   c2.move();               // prints “I am flying”
   c1 = new Butterfly(c1);
   c1.move();               // now prints “I am flying”
   ```

A Bird can observe other Creatures and reacts whenever the observed Creature moves(). If it is observing another Bird, it will move whenever the other bird moves. If it is observing a Caterpillar, it will eat it as soon as it moves, using the Bird’s method `protected void eat(Caterpillar c)`.

Use the “pure” Strategy, Decorator, and Observer Patterns (and use Java’s Observer and Observable).

1. [5%] Just below, write driver code so that `c2` would eat `c1` automatically if `c1` subsequently moves:

2. [5%] If Creature `c3 = new Bird();` and `c2` and `c3` observe one another, what happens after `c2.move()`? Explain (a Sequence Diagram would be useful but not essential to your explanation).

3. [30%] Use a UML class diagram to show the hierarchies for Creature and LocomotionStrategy and show their relationships. Make sure you cannot accidentally decorate a Bird to become a Butterfly. Show your design using a UML class diagram.

4. [60%] Implement your design.