Please study the following, which will then be the basis for questions to follow. A **Duck** has the following methods:

```java
public void eat();
public void move();
public void sleep();
```

a) The Duck class is **abstract**. A concrete instance of a baby duck is a **Duckling**, with its two adult parents passed to the constructor:

```
Duck louie = new Duckling(donald, daisy);
```

b) When the Duckling grows up, it becomes an **AdultDuck**, and receives its distinctive decoration of colorful plumage, plus some new capabilities not available to it when it was a mere Duckling (discussed momentarily).

```
louie = new AdultDuck(louie);
```

c) A Duck is either **Hungry** or **Sleepy** (but not both at the same time).

d) A Duck has two ways to **eat()**; by **Gobbling()** and by **Nibbling()**. When **Hungry** it eats by **Gobbling**; when **Sleepy** it eats by **Nibbling**.

e) **Turkeys** also eat by Gobbling and Nibbling (i.e., use the same methods). Turkey and Duck are sibling subclasses of **Bird**.

f) An adult duck has three ways to **move()**: **Paddling** (when floating on water), and **Waddling** (how ducks walk on land) and **Flying** (adults Ducks can fly, but not Ducklings).

g) A duckling copies the behaviors of its parents (either one or both): When a parent decides to eat, the ducklings also begin to eat; when a parent begins to move, the ducklings follow.

h) The paired adult ducks also have cooperative behavior, and when either one starts to eat or move, the other copies.

i) Ducks get **Sleepy** when the **Night** comes, and **Hungry** when the **Day** comes. (They also get **Hungry** or **Sleepy** on their own, but Night and Day are major influences on the Ducks.)

j) Additional code is planned for the Ducks (e.g., graphics, sound effects, and artificial intelligence for choosing how to move).
1) [45%] Going through points a) through j), use UML class diagrams and code fragments to supplement your description of how Design Patterns would apply. When you see an application of a pattern, 1) name and briefly describe the pattern (in general), then 2) describe how each pattern applies, 3) how to use it, 4) the benefits, and 5) any restrictions or undesired consequences you find.

Providing code is a nice way to be specific and clear in your descriptions. Add whatever methods and classes you need. You are not expected to write a complete implementation. Just be efficient, clear, and specific enough to show your application of design patterns.

(Note, this is not just a repetition of the text’s SimUDuck example.)

Use the space below for an overall UML diagram. Provide your discussion and code fragments on separate pages.
2) [35%] A **Train** has an **Engine** followed by one or more **Cars**. **Engine** and **Car** are doubly-linked using `setNext()` and `setPrev()`. In the following, e1 and c1 will be coupled, then c1 and c2, and c2 and c3.

```java
Train e1 = new Engine(); // note: all these are cast to Train
Train c1 = new Car();
Train c2 = new Car();
Train c3 = new Car();
```

After all are properly coupled, the following would be true:

```java
c1.getNext() == c2 && c1.getPrev() == e1
```

2a) [10%] Provide code to form **Train t** out of e1, c1, c2, and c3:

```java
Train t;
```
Engine has a method `int getMaxPull()` which returns the maximum weight of loaded Cars it can pull. Car has `int getWeight()` that returns the weight of that instance. If the train is too heavy to climb a mountain, a `ShortenV(int maxWeight)` (subclass of Visitor) visits the Train and keeps as many Cars (starting just after the Engine) as the Engine can pull. For instance, suppose e1 can only pull the combined weight of Cars c1 and c2, then:

```java
ShortenV v = new ShortenV(e1.getMaxPull());
e1.acceptVisitor(v);
```

would disconnect Car c3 from the train.

Assume `acceptVisitor(Visitor v)` is abstract in Train and concrete in Engine and Car (each calling `v.visit(this)`).

Write both Visitor and ShortenV below:
2c [10%] Draw a UML sequence diagram of the visitation (note that v and e1 and the rest of the train are already created):

driver e1 v c1 c2 c3

3. [20%] Write a 1-2 page essay discussing 1) the computer science concepts you learned this quarter, 2) the important engineering principles you learned, and 3) the particular aspects you found most appealing and 4) least appealing, from high level Object Oriented philosophy to the low-level implementation. Be sure to answer all 4 issues.