The following simple interface **Item** has no methods

```java
public interface Item {}
```

And a simple **List** class (that does not use generics) and its Iterator:

```java
public class List {
    public void add(Item item) {... // Add the given Item to end of List.
    public Item get(int index) {... // Return the Item at a given index;
    public Iterator iterator() {... // Get an Iterator
    public Item remove(int index) {... // Removes and return Item at index
    public boolean remove(Item item) {... // Remove item & return true else false
    public int size(); // Returns the size of the List.
}
```

```java
public class Iterator {
    private List list;
    private int index = 0; // index of next element to be returned

    public Iterator(List l) { list = l; }
    public boolean hasNext() { return index < list.size(); }
    public Item next() { return list.get(index++); }
    public void remove() { list.remove(index--); }
}
```

**Presume:** 1) **Lion** and **Deer** extend **Animal**, 2) **Animal** extends **Observable** and implements **Observer** and **Item**.

```java
Deer bambi = new Deer();
Deer nambi = new Deer();
Deer gambi = new Deer();
List herd = new List();
Lion leo = new Lion();

herd.add(bambi);
herd.add(nambi);
herd.add(gambi);
```

1. [15%] Using an Iterator and the addObserver method, write a loop to 1) make leo observe all the deer in the herd, and 2) each deer to observe leo:
2) Every Animal has a characteristic behavior, when its behave() method is called. Here are but two possible behaviors for the Strategy Pattern:

```java
public interface Behavior { public void behave(); }

public class EscapeBehavior implements Behavior
    public void behave() { System.out.println("I’m running"); }

public class EatBehavior implements Behavior
    public void behave() { System.out.println("munch munch"); }
```

In the following, bambi is eating, then leo’s roar makes the deer run away. The deer actually changes its behavior (a good strategy to adopt)

```java
Animal bambi = new Deer(); // the cast forces you to use the Strategy pattern
Lion   leo   = new Lion();
leo.addObserver(bambi);
bambi.behave(); // results in printing “munch munch”
leo.roar();     // results in printing “I’m running”
```

As a reminder, here is the Observer interface:

```java
public interface Observer { public void update(Observable o, Object obj); }
```

[20%] Finish Animal and Deer. Be careful to extend all necessary classes:

```java
abstract public class Animal extends Observable implements Observer {

    abstract public void update(Observable o, Object obj);

public class Deer
    public Deer() {
```
3. [35%] Using the interface Item, and the classes List and Iterator from question 1, write a Java-like Observable class (see comments above methods)

```java
public class Observable {
    private boolean changed = false;

    public Observable() {

        // Add an observer of this object, provided that it is not already:
        public void addObserver(Observer o) {

            protected void clearChanged() { changed = false; }
            protected void setChanged()   { changed = true; }
            public boolean hasChanged()   { return changed; }

            // returns the number of observers of this Observable:
            public int countObservers() {

                // deletes a specified observer from the set of observers of this object.
                public void deleteObserver(Observer o) {

                    // clears the observer list so this object no longer has any observers:
                    public void deleteObservers() {

                        void notifyObservers() { notifyObservers(null); }

                        // If this Observable has changed, notify all of its observers then call the
                        // clearChanged method to indicate that this object has no longer changed:
                        void notifyObservers(Object arg) {
```
4 [15%] Concerning the design principle “Favor composition over inheritance”:

4.1 [5%] Rephrase it to be more easily understood.

4.2 [5%] Describe the concrete benefits of following this advice.

4.3 [5%] Use a UML class diagram and describe a specific example from this midterm of this principle.
5 [15%] Concerning “Program to an interface, not an implementation”:

5.1 [5%] Rephrase it to be more easily understood.

5.2 [5%] Describe the concrete benefits of following this advice.

5.3 [5%] Use a UML class diagram and describe a specific example from this midterm of this principle.