Problem 1

In the class Circle, add a data field

```java
private double recordedArea = -1.0;
```

Modify the `area` method as follows:

```java
public double area() {
    if (recordedArea < 0.0)
        recordedArea = Math.PI * radius * radius;
    return recordedArea;
}
```

Change data field `radius` to be `private`. Add methods

```java
public double getRadius() { return radius; }
public void setRadius(double R) {
    recordedArea = -1.0;
    radius = R;
}
```

Problem 2

Here is a solution as a static method.

```java
public static Node DoubleList(L) {
    Node N1 = new Node(L.data);
    Node N2 = new Node(L.data, N1);
    if (next == null) return N2; // Base case of the recursion
    else N1.next = DoubleList(L.next); // Recur to the next node of the
}
```

Here is a solution as a dynamic method, recurring on the owner.

```java
public Node DoubleList() {
    Node N1 = new Node(data);
    Node N2 = new Node(data, N1);
    if (next == null) return N2; // Base case of the recursion
    else N1.next = next.DoubleList(); // Recur to the next node of the
        // owner
}
```

Honors Problem

```java
public static int HailstoneNumber(int X) {
    if (X==1) return 1; // Base case of the recursion
    else {
        if (X%2 ==1) X = 3*X+1;
        else X = X/2;
    }
```
What’s unusual here is that recursion is usually described as expressing the solution to a complex problem in terms of simpler problems. But here the argument to the recursive call is “simpler” only in the sense that it has a smaller hailstone number.