A “necklace” is comprised of three distinct Elements: a Left-Clasp, a number of Links, and a RightClasp. To form a closed loop, the RightClasp is connected back to the LeftClasp.

The elements all have a public void connect method, however:
- A LeftClasp can only connect to a Link (not a RightClasp).
- A Link can connect to another Link or to a RightClasp.
- A RightClasp can only connect to a LeftClasp.

usage (with reference to the above diagram):

```
LeftClasp leftClasp = new LeftClasp();
Link link1 = new Link();
leftClasp.connect(link1); // and etc.
```

Each Link (but not a LeftClasp or RightClasp) can have a Stone attached (or detached!). A Diamond is a kind of Stone.

```
public class Stone {}
public class Diamond extends Stone {}
```

The Link class has methods for attaching and detaching a Stone:
```
public void attach(Stone s)
public Stone detach(); // returns the given Stone or null
```

A “necklace” (held by its LeftClasp) can accept NecklaceVisitors, such as a ThiefV, which, detaches the first Diamond it finds. The ThiefV keeps at most one Diamond, to later be passed by its getDiamond method. Note that the necklace remains intact, but one link might be missing its Diamond. You can get the Diamond from a thief by the following:

```
ThiefV t = new ThiefV();
leftClasp.accept(t); // start visit on the left
Diamond d = t.getDiamond(); // or null if n had Diamond
```
1) [15%] Write Java code to create the specific necklace with attached instances of Stone in the order: sharon, neil, oliver, and irving (that is, sharon should be just to the right of the LeftClasp, and neil to the right of sharon, etc.).

```java
LeftClasp leftClasp = new LeftClasp();
RightClasp rightClasp = new RightClasp();

Stone sharon = new Stone();
Diamond neil = new Diamond();
Stone oliver = new Stone();
Stone irving = new Stone();
```

2) [10%] Now, to start to design your necklace hierarchy, create a UML diagram showing the classes and methods (remember it must accept NecklaceVisitors).

3) [10%] Next, draw the UML diagram for your NeckVisitor class and ThiefV. Careful with the method names needed.

4) [40%] Write the entire “necklace” class hierarchy. Do not use any pre-built collection classes (such as List, ListC, or arrays, etc.).

5) [25%] Write ThiefV.