We continue to work on the program you finished in homework #2. In homework #3, some new features are continually added, and some Design Patterns have to be applied in your design, too. Also, you have to draw the class diagram for your program. In the end you have to write new unit tests for your additional program and pass all the test cases you wrote before.

1. Apply the state pattern to your program. (30%)
   
   In our program, we have to parse a file to create the objects which composes a class diagram. In general, there is a set of strings called tokens we defined in the input. In figure 1, we use a finite state machine to simulate the whole parsing process, and each state is defined as an action parsing different elements in a class diagram. There are some specified tokens that could change state of process.

   ![Finite State Machine Diagram]

   **Figure 1: finite state machine of parsing a class diagram**

   For example, we use a simple input in the list 1 to demonstrate how the finite state machine works in our parsing process. We start at the initial state which does nothing. Then, we get the first token “//classes & interfaces” and we are at the
state 0 now. The next token should be the “//interface”, and we enter state 1 and begin to create and read the class name for a class. The next token is “//operations,” and we change to state 3. It means that we start collecting all the data of the operations in a class. After collecting the data, we would read the token “//class” again. We go back to the state 1 to create another class. Until reading the “//generalizations” token, the machine changes to the state 4. But in this example we have no generalization, we continue to read the next token. The next token is “//relationships”, it means we would parse all the association in this state. Finally, we get the “EOF” token, and the machine changes to the state 6. It’s an accepting state in this machine. You should practice an example on you own.

```java
//classes & interfaces
//-interface
List
//-operations
add  public void 1
//-parameters
  T

//-class
Diagram

//generalizations

//relationships
Diagram   List   dependency
```

In this problem, you should apply the state pattern in your parser program. The figure 2 shows that it’s a conceptual structure of the state pattern. You could see the context class as our parser, and all the concrete state classes could map to the states in our finite state machine. Also, the concrete state classes are derived from a base state class and override the handle operation. You should write different logic for different elements of a diagram. We hope the state pattern would make your code clearer and more easily to add new features in the future. Please make sure that you apply the state pattern correctly.
2. We add some new features into our class diagram, including **dependency**, **composition**, **aggregation**, and **template class**. In Figure 3, there exist a dependency between Diagram and List, and it means the Diagram have to change when the List interface changes. And you have to separate this definition from association. In Figure 4 and 5, there are two examples to explain the composition and aggregation separately. In Figure 6, this is a template class called Set. It needs to define the template parameter called T for the template class. You have to integrate all the new features to the previous functions. Also, you should make the composition and aggregation are derived the same class called association. (30%)
3. Please keep your unit tests running. You do not need to add any new tests for the new programs written in this homework. However, you should maintain your unit test programs written for homework #2. (20%)

4. Draw class diagrams for the program in UML. To do this they need to have every functions encapsulated in a class, if not already so. The class diagrams will be drawn in two perspectives: design class diagram and implementation class diagram. (note: you do not need to draw the class diagram for unit tests). (20%)