设计模式实验（6）

一、实验目的

1. 结合实例，熟练绘制设计模式结构图。
2. 结合实例，熟练使用 Java 语言实现设计模式。
3. 通过本实验，理解每一种设计模式的模式动机，掌握模式结构，学习如何使用代码实现这些设计模式。

二、实验要求

1. 结合实例，绘制设计模式的结构图。
2. 使用 Java 语言实现设计模式实例，代码运行正确。

三、实验内容

1.状态模式

在某网络管理软件中， TCP 连接（TCP Connection）具有建立（Established）、监听（Listening）、关闭（Closed）等多种状态，在不同的状态下 TCP 连接对象具有不同的行为，连接对象还可以从一个状态转换到另一个状态。当一个连接对象收到其他对象的请求时，它根据自身的当前状态做出不同的反应。现采用状态模式对 TCP 连接进行设计，绘制对应的类图并编程模拟实现。

2.享元模式

某 OA 系统采用享元模式设计权限控制与管理模块，在该模块中，将与系统功能相对应的业务类设计为享元类并将相应的业务对象存储到享元池中（提示；可使用 Map 实现，key 为业务对象对应的权限编码， value 为业务对象）。用户身份验证成功后，系统通过存储在数据库中的该用户的权限编码集从享元池获取相应的业务对象并构建权限列表，在界面上显示用户所拥有的权限。根据以上描述，绘制对应的类图并编程模拟实现。

3.代理模式

在某电子商务系统中，为了提高查询性能，需要将一些频繁查询的数据保存到内存的辅助存储对象中（提示：可使用 Map 实现）。用户在执行查询操作时，先判断辅助存储对象中是否存在待查询的数据，如果不存在，则通过数据操作对象查询并返回数据，然后将效据保存到辅助存储对象中，否则直接返回存储在辅助存储对象中的数据。现采用代理模式中的缓冲代理实现该功能，要求绘制对应的类图并编程模拟实现。

4.命令模式

某灯具厂商要生产一个智能灯具遥控器，该遥控器具有5 个可编程的插槽，每个插槽都有一个控制灯具的开关，这5个开关可以通过蓝牙技术控制5个不同房间灯光的打开和关闭，用户可以自行设置每一个开关所对应的房间。现采用命令模式实现该智能遥控器的软件部分，绘制对应的类图并编程模拟实现。

5.解释器模式

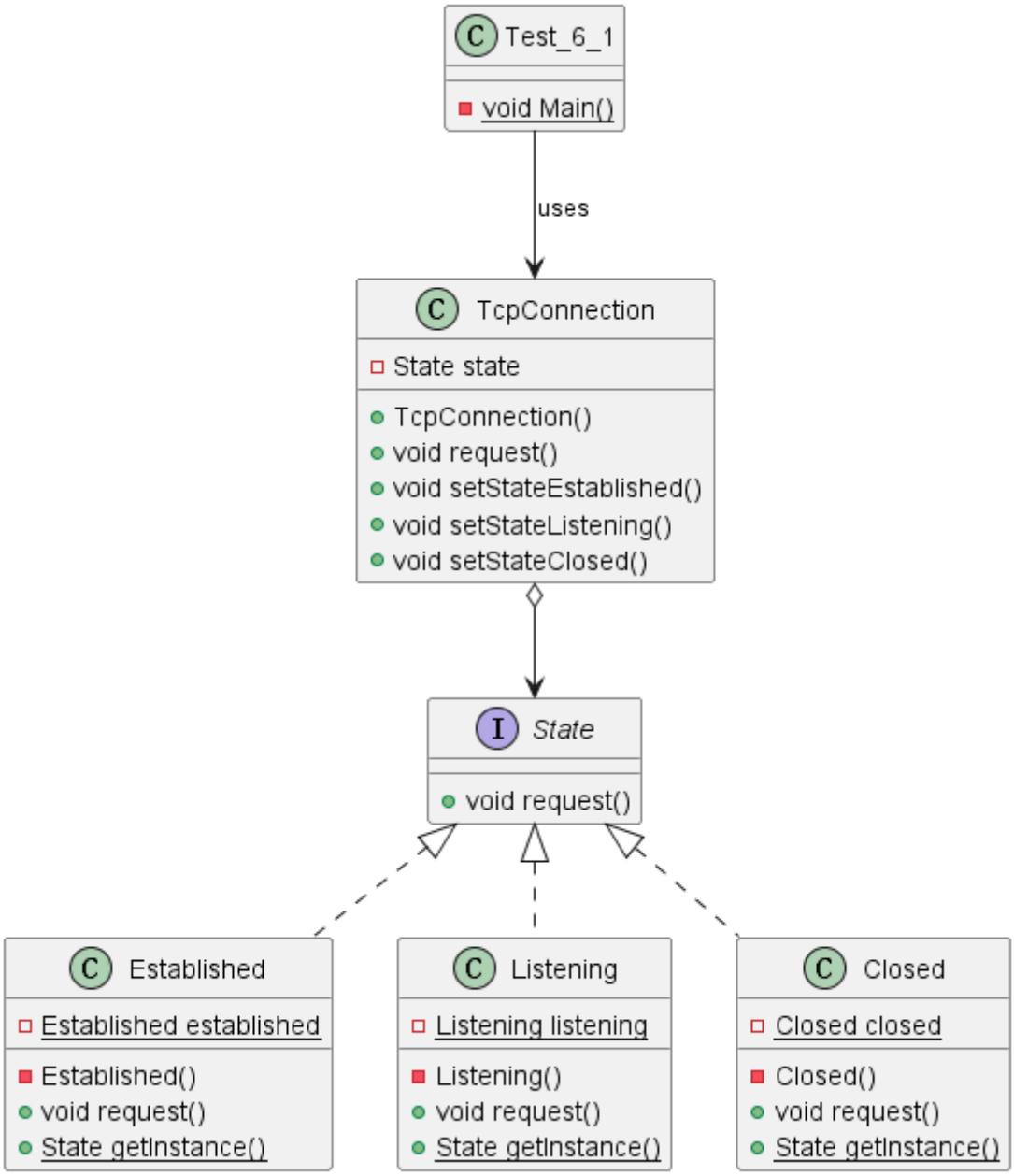
某软件公司要为数据库备份和同步开发一套简单的数据库同步指令，通过指令可以对数据库中的数据和结构进行备份。例如，输入指令“ COPY VIEW FROM srcDB TO desDB ”，表示将数据库 srcDB 中的所有视图（View）对象都拷贝至数据库 desDB ； 输人指令“ MOVETABLE Student FROM srcDB TO desDB ”，表示将数据库 srcDB 中的 Student 表移动至数据库desDB 。现使用解释器模式来设计并编程

模拟实现该数据库同步指令系统

三、实验结果

1.外观模式

类图



实现代码

- Closed.java

```
public class Closed implements State {

    private static Closed closed = new Closed();

    private Closed(){}

    @Override
    public void request() {
```

```
        System.out.println("Requsted : Current State is [ "+
this.getClass().getName() + " ]");
    }

    public static State getInstance(){
        return Closed.closed;
    }
}
```

- Established.java

```
public class Established implements State {

    private static Established established = new Established();

    private Established(){}

    @Override
    public void request() {
        System.out.println("Requsted : Current State is [ "+
this.getClass().getName() + " ]");
    }

    public static State getInstance(){
        return Established.established;
    }
}
```

- Listening.java

```
public class Listening implements State {

    private static Listening listening = new Listening();

    private Listening(){}

    @Override
    public void request() {
        System.out.println("Requsted : Current State is [ "+
this.getClass().getName() + " ]");
    }

    public static State getInstance(){
        return Listening.listening;
    }
}
```

- State.java

```
public interface State {
    public void request();
}
```

- TcpConnection.java

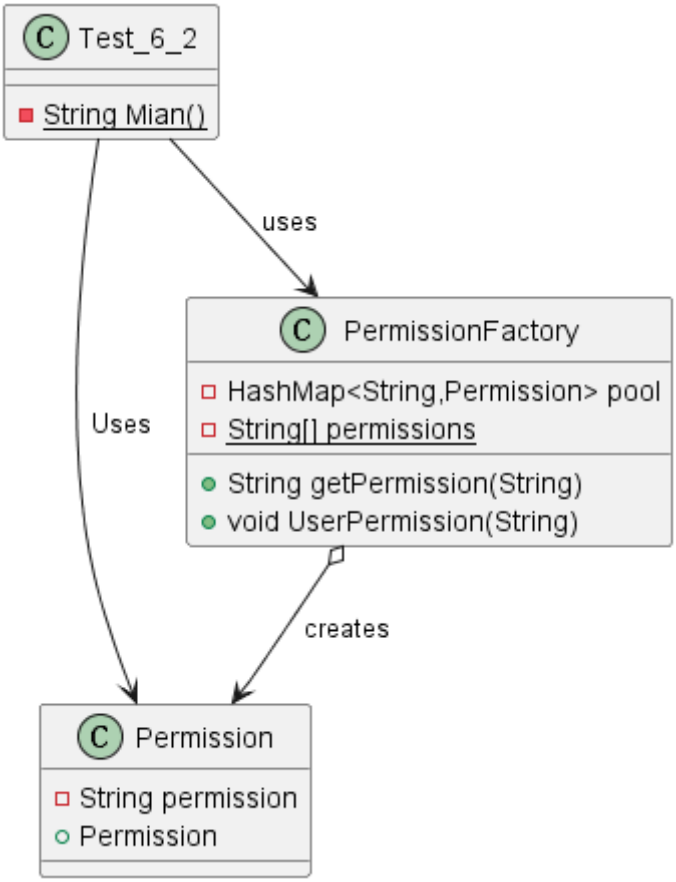
```
public class TcpConnection {
    private State state;
    public TcpConnection(){}
    public void setStateEstablished(){
        state = Established.getInstance();
    }
    public void setStateListening(){
        state = Listening.getInstance();
    }
    public void setStateClosed(){
        state = Closed.getInstance();
    }
    public void request(){
        state.request();
    }
}
```

- Test_6_1.java

```
public class Test_6_1 {
    public static void main(String[] args) {
        TcpConnection tcpConnection = new TcpConnection();
        tcpConnection.setStateClosed();
        tcpConnection.request();
        tcpConnection.setStateEstablished();
        tcpConnection.request();
        tcpConnection.setStateListening();
        tcpConnection.request();
        tcpConnection.setStateClosed();
        tcpConnection.request();
    }
}
```

2.享元模式

类图



实现代码

- Permission.java

```
public class Permission {
    private String permission ;
    public Permission(String permission){
        this.permission=permission;
    }



    public String toString(){
        return "[ " + permission + " ]";
    }

}
```

- PermissioonFactory.java

```
import java.util.HashMap;

public class PermissioonFactory {
    private HashMap<String,Permission> pool ;
    private static String[] permissions = {"Query","Add","Modify","Delete"};
    public PermissioonFactory(){
        pool=new HashMap<>();
    }
```

```
    public String getPermission( String permission ){
        if(pool.containsKey(permission)){
            return pool.get(permission).toString();
        } else {
            pool.put(permission,new Permission(permission));
            return pool.get(permission).toString();
        }
    }

    public void UsersPermission(String persmissons){
        System.out.println("The Permissions of Current User are shown below :");
        for(int i=0;i<4;i++){
            if (persmissons.charAt(i)=='1'){

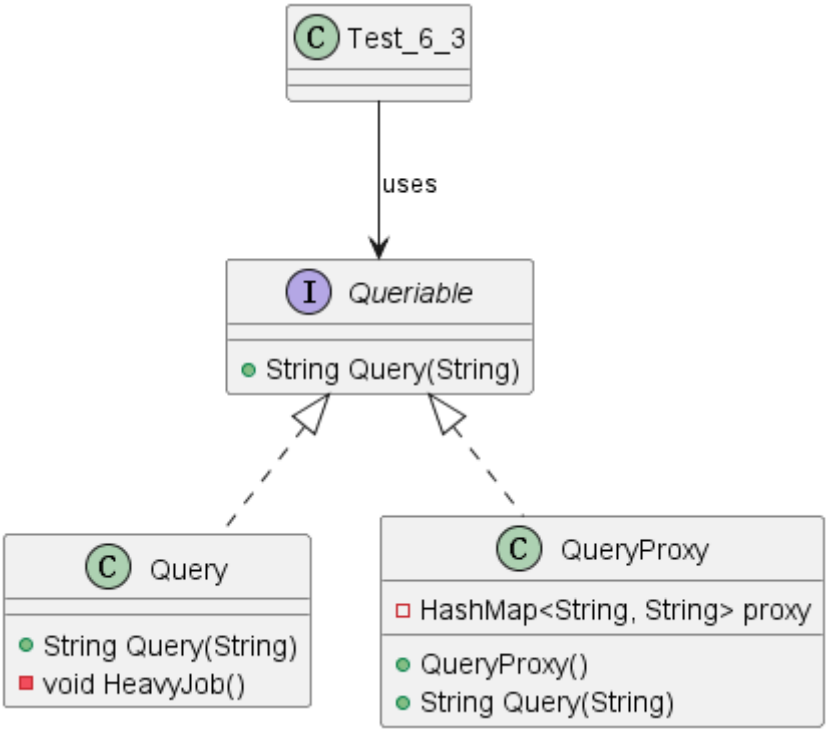
 System.out.println(this.getPermission(PermissioonFactory.permissions[i]));
            }
        }
        System.out.println("*****************************************\n");
    }
}
```

- Test_6_2.java

```
public class Test_6_2 {
    public static void main(String[] args) {
        PermissioonFactory permissioonFactory = new PermissioonFactory();
        permissioonFactory.UsersPermission("0000");
        permissioonFactory.UsersPermission("1111");
        permissioonFactory.UsersPermission("1010");
        permissioonFactory.UsersPermission("0111");
        permissioonFactory.UsersPermission("1001");
    }
}
```

3.代理模式

类图



实现代码

- Queriable.java

```
public interface Queriable {
    public String Query(String query);
}
```

- Query.java

```
public class Query implements Queriable{
    @Override
    public String Query(String query) {
        HeavyJob();
        return "[ " + query + " ]";
    }

    private void HeavyJob(){
        System.out.println("Asking from Query");
        for(int i=0;i<10;i++){
            try {
                Thread.sleep(300);
            } catch (Exception e) {
            }
            System.out.print(".");
        }
        System.out.println(" Done");
    }
}
```

- QueryProxy.java

```
import java.util.HashMap;
```

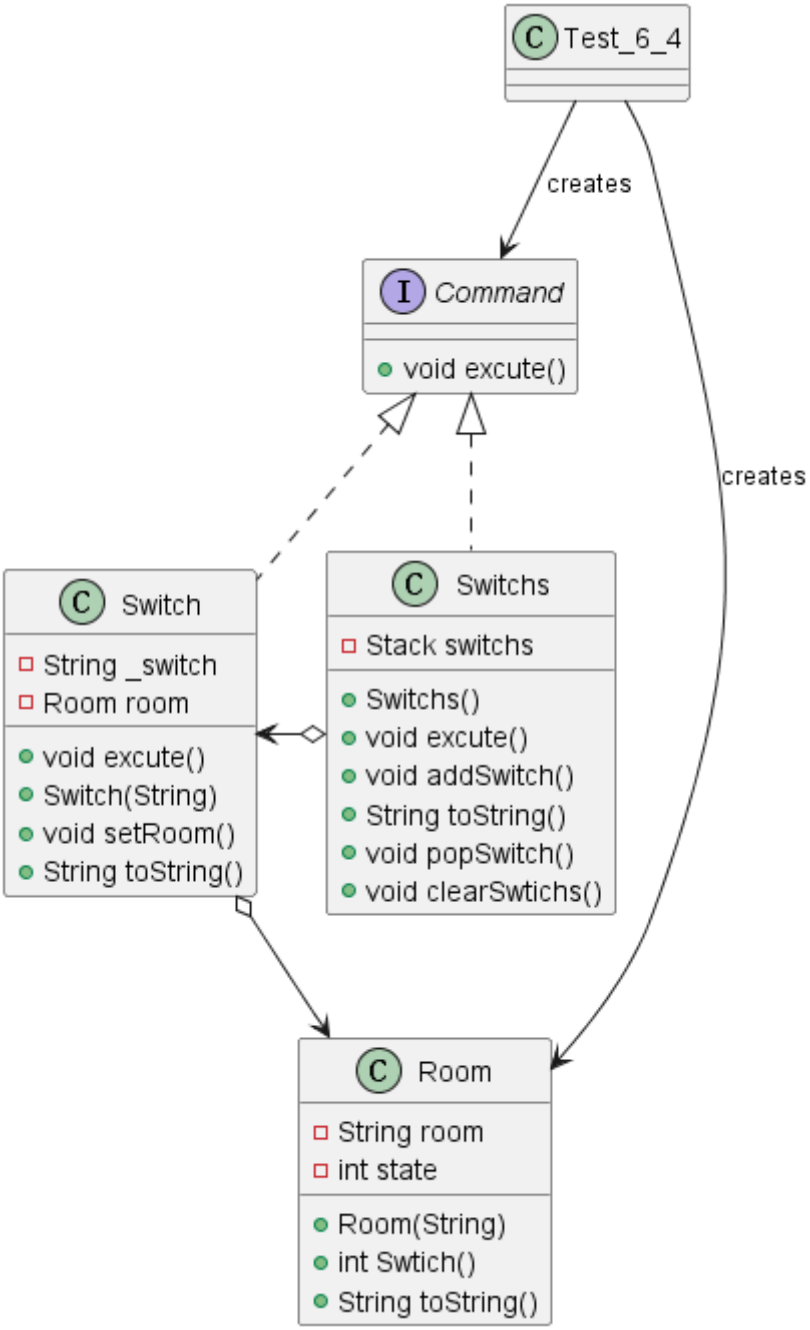
```
public class QueryProxy implements Queriable{
    private HashMap<String,String> proxy;
    public QueryProxy(){
        proxy = new HashMap<>();
    }
    @Override
    public String Query(String query) {
        String re = null;
        if(proxy.containsKey(query)){
            re =proxy.get(query);
        } else {
            re = new Query().Query(query);
            proxy.put(query,re);
        }
        System.out.println( "[ proxy Answered " + re + " ]");
        return re;
    }
}
```

- Test_6_3.java

```
public class Test_6_3 {
    public static void main(String[] args) {
        Queriable queryProxy = new QueryProxy();
        queryProxy.Query("Query");
        queryProxy.Query("Problem");
        queryProxy.Query("Query");
        queryProxy.Query("Question");
        queryProxy.Query("Question");
        queryProxy.Query("Problem");
        queryProxy.Query("Answer");
    }
}
```

4.命令模式

类图



实现代码

- Command.java

```
public interface Command {
    public void excute();
}
```

- Room.java

```
public class Room {
    private String room;
    private int state;
    public Room(String room){
        this.room = room;
        this.state = 0;
    }

    public int Switch(){
```

```
        state^=1;
        return state;
    }

    public String toString(){
        return "Room [ " + this.room + " ]";
    }

}
```

- Switch.java

```
public class Switch implements Command{
    private String _switch;
    private Room room;
    public Switch(String _switch){
        this._switch = _switch;
        this.room = null;
    }

    @Override
    public void excute() {
        System.out.println(this.room.toString() + " has been turned " +
(this.room
                .Switch()==1?"on":"off") +  " by " +  this.toString()  );
    }

    public void setRoom(Room room){
        System.out.println("Switch [ " + this._switch + " ] controlling " +
room.toString());
        this.room = room;
    }

    public String toString(){
        return "Switch [ " + this._switch + " ]";
    }
}
```

- Switchs.java

```
import java.util.Iterator;
import java.util.Stack;

public class Switchs implements Command{

    private Stack<Command> switches;

    public Switchs(){
        this.switches = new Stack<>();
    }

    @Override
    public void excute() {
        Iterator<Command> it = switches.iterator();
        while(it.hasNext()){
            it.next().excute();
```

```
        }
    }

    public void addSwitch(Command switch_){
        if(switch_ != this){
            switches.push(switch_);
        }
    }

    public void popSwitch(){
        if(!switches.empty())switches.pop();
    }

    public void clearSwitchs(){
        switches.clear();
    }

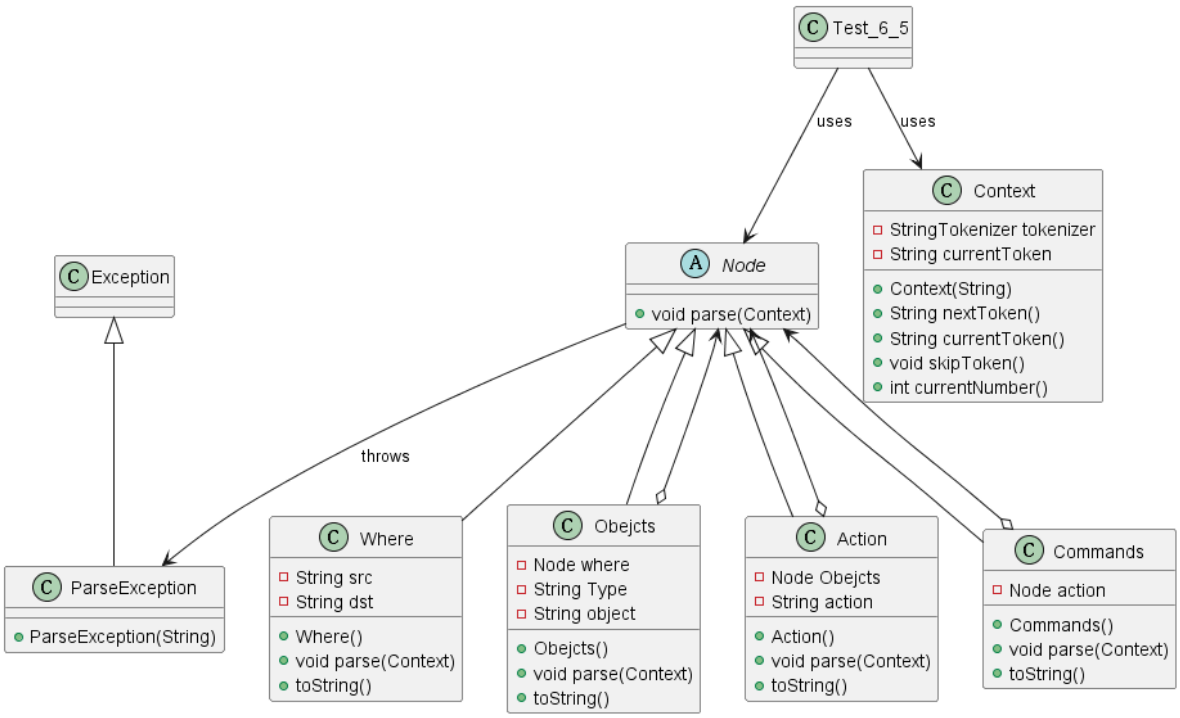
}
```

- Test_6_4.java

```
public class Test_6_4 {
    public static void main(String[] args) {
        Switch s1 = new Switch("s1");
        Room r1 = new Room("room1");
        s1.setRoom(r1);
        s1.excute();
        s1.excute();
        s1.excute();
        Switch s2 = new Switch("s2"),s3 = new Switch("s3"),
                s4 = new Switch("s4"),s5 = new Switch("r5");
        Room r2=new Room("room2"),r3 = new Room("room3"),r4 = new Room("room4"),
                r5 = new Room("room5"),r6 = new Room("room6");
        s2.setRoom(r2);s3.setRoom(r3);s4.setRoom(r4);s5.setRoom(r5);
        s2.excute();s4.excute();
        Switchs switchs = new Switchs();
        switchs.addSwitch(s1);switchs.addSwitch(s2);switchs.addSwitch(s3);
        switchs.addSwitch(s4);switchs.addSwitch(s5);
        switchs.excute();
        s5.setRoom(r6);
        s5.excute();s5.excute();s5.excute();
    }
}
```

5.解释器模式

类图



实现代码

- Action.java

```
public class Action extends Node{
    private Node Objects;
    private String action;
    @Override
    public void parse(Context context) throws ParseException {
        if(context.currentToken()==null){
            throw new ParseException("Missing 'Action' ");
        } else if(context.currentToken().equals("COPY")||
context.currentToken().equals("MOVE")){
            action = context.currentToken();
            Objects = new Objects();
            context.nextToken();
            Objects.parse(context);
        } else {
            throw new ParseException(context.currentToken() + " Action if
undefiend");
        }
    }

    public String toString(){
        return "[ " + action + " " + Objects + "] ";
    }
}
```

- command.txt

```
COPY VIEW FROM srcDB TO desDB
MOVE TABLE Student FROM srcDB TO desDB```

- Commands.java
```java
public class Commands extends Node{
```
```

```
    private Node action;
    @Override
    public void parse(Context context) throws ParseException {
        action = new Action();
        action.parse(context);
    }

    @Override
    public String toString() {
        return "[ " + action + "] ";
    }
}
```

- Context.java

```
import java.util.*;

public class Context {
    private StringTokenizer tokenizer;
    private String currentToken;
    public Context(String text) {
        tokenizer = new StringTokenizer(text);
        nextToken();
    }
    public String nextToken() {
        if (tokenizer.hasMoreTokens()) {
            currentToken = tokenizer.nextToken();
        } else {
            currentToken = null;
        }
        return currentToken;
    }
    public String currentToken() {
        return currentToken;
    }
    public void skipToken(String token) throws ParseException {
        if (!token.equals(currentToken)) {
            throw new ParseException("Warning: " + token + " is expected, but "
+ currentToken + " is found.");
        }
        nextToken();
    }
    public int currentNumber() throws ParseException {
        int number = 0;
        try {
            number = Integer.parseInt(currentToken);
        } catch (NumberFormatException e) {
            throw new ParseException("Warning: " + e);
        }
        return number;
    }
}
```

- Node.java

```
public abstract class Node {
    public abstract void parse(Context context) throws ParseException;
}
```

- Objects.java

```
public class Objects extends Node{
    private String object = null ;
    private String Type;
    private Node where;
    @Override
    public void parse(Context context) throws ParseException {
        if(context.currentToken()==null){
            throw new ParseException("Missing DataBase Type");
        }
        else
if(context.currentToken().equals("VIEW")||context.currentToken().equals("TABLE")
){
            Type = context.currentToken();
            context.nextToken();
            if(!context.currentToken().equals("FROM")){
              object = context.currentToken();
              context.skipToken(object);
            }
            where = new Where();
            where.parse(context);
        } else {
            throw new ParseException(context.currentToken() + " Database Type is
undefined ");
        }
    }

    @Override
    public String toString() {
        return "[ " + (object==null?("All " + Type + "s "):(Type + " " + object
+ " ")) + where + "] ";
     }
}
```

- ParseException.java

```
public class ParseException extends Exception {
    public ParseException(String msg) {
        super(msg);
    }
}
```

- Test_6_5.java

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.lang.management.BufferPoolMXBean;
```

```
public class Test_6_5 {
    public static void main(String[] args) throws IOException, ParseException {
        String command ;
        BufferedReader reader = new BufferedReader(new FileReader("homework6/6-
5Interpreter/command.txt"));
        while((command=reader.readLine())!=null){
            System.out.println("command = " + command);
            Node cmd = new Commands();
            cmd.parse(new Context(command));
            System.out.println("Node = " + cmd);
        }
    }
}
```

- Where.java

```
public class Where extends Node{
    private String src;
    private String dst;
    @Override
    public void parse(Context context) throws ParseException {
        if(context.currentToken()==null){
            throw new ParseException("Command not complete");
        }
        if(!context.currentToken().equals("FROM")){
            throw new ParseException("Missign 'FROM'");
        }
        context.nextToken();
        if(context.currentToken()==null){
            throw new ParseException("Command not complete");
        }
        src = context.currentToken();
        context.nextToken();
        if(!context.currentToken().equals("TO")){
            throw new ParseException("Missign 'TO'");
        }
        context.nextToken();
        if(context.currentToken()==null){
            throw new ParseException("Command not complete");
        }
        dst = context.currentToken();
        context.nextToken();
        if(context.currentToken()!=null){
            throw new ParseException("command is wrong");
        }
    }

    @Override
    public String toString() {
        return "[ FROM [ " + src + " ] TO [ " + dst + " ] ] ";
    }
}
```

- command.txt

```
COPY VIEW FROM srcDB TO desDB
MOVE TABLE Student FROM srcDB TO desDB
```

五、实验小结

本次实验学会了状态模式，享元模式,代理模式,命令模式和解释器模式，在做这些作业的过程中，不仅对这5类设计模式有了直接的理解和体会，同时对java代码更加熟悉了，也更深入的了解了面向对象的编程的思想，对代码整体的设计的把握也更得心应手了