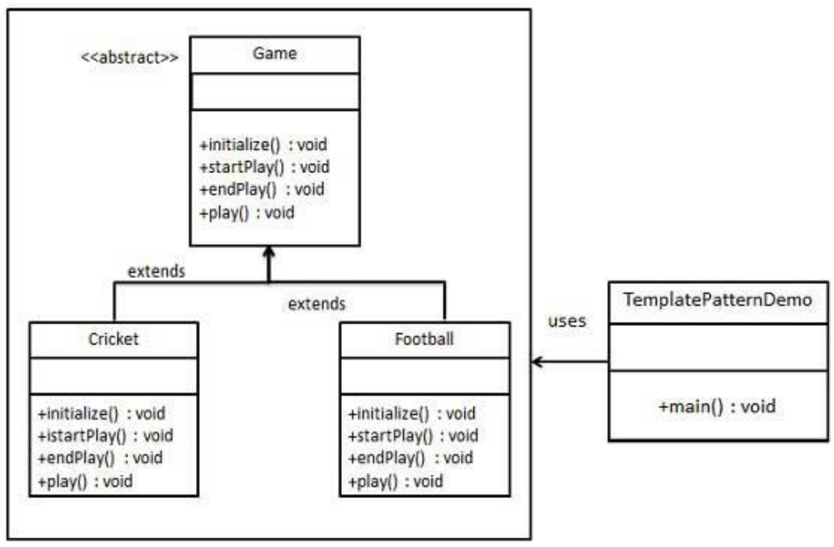
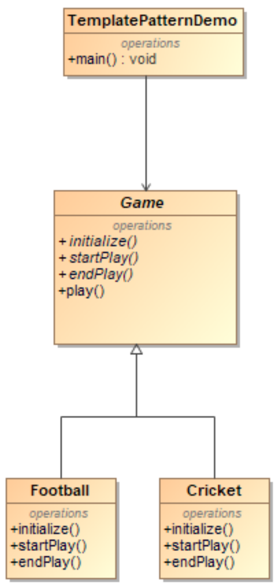
DESIGN PATTERNS

K2

Contents

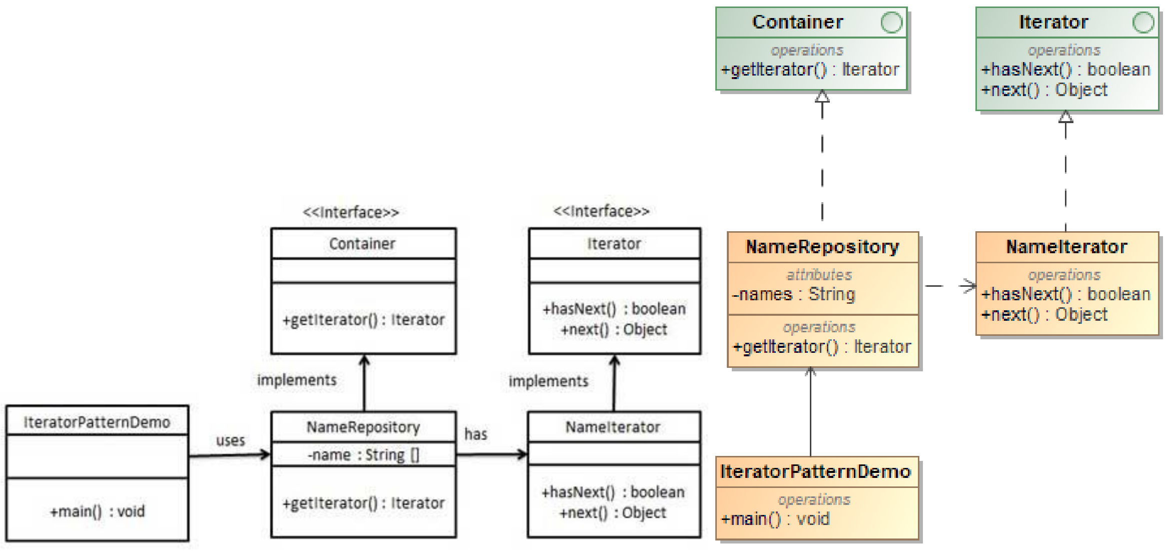
1. Template Method





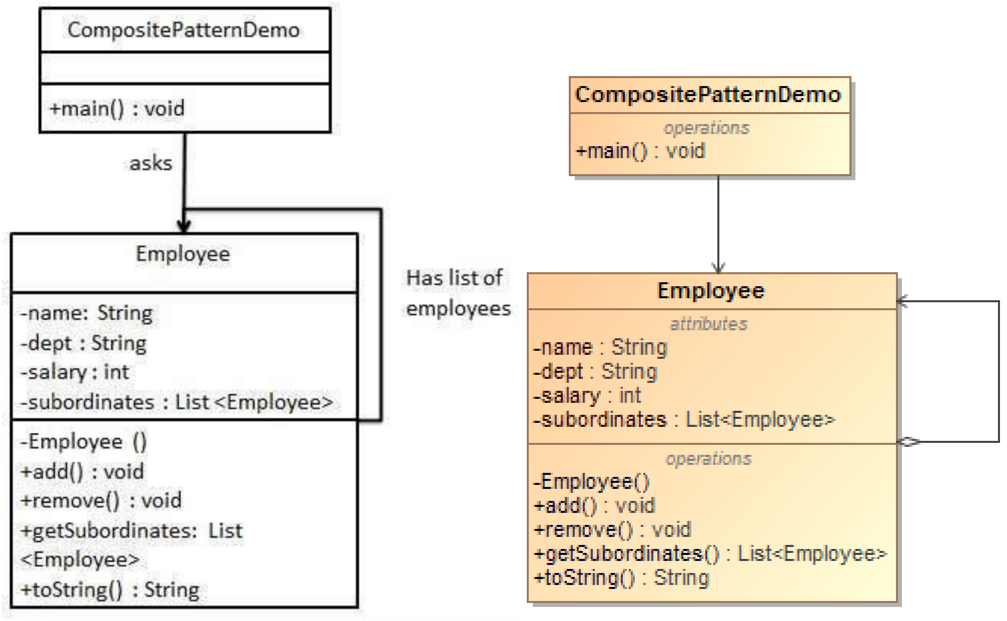
```java
abstract class Game {
    public abstract void initialize();
    public abstract void startPlay();
    public abstract void endPlay();

    public final void play() {
        initialize();
        startPlay();
        endPlay();
    }
}
class Football extends Game {
    @Override
    public void initialize() {
        System.out.println("Football game initialized. Start playing!");
    }
    @Override
    public void startPlay() {
        System.out.println("Football game started. Enjoy!");
    }
    @Override
    public void endPlay() {
        System.out.println("Football game finished!");
    }
}
class Cricket extends Game {
    @Override
    public void initialize() {
        System.out.println("Cricket game initialized. Start playing!");
    }
    @Override
    public void startPlay() {
        System.out.println("Cricket game started!");
    }
    @Override
    public void endPlay() {
        System.out.println("Cricket game finished.");
    }
}
class TemplatePatternDemo {
    public static void main(String[] args) {
        Game game = new Football();
        game.play();
        System.out.println();
        game = new Cricket();
        game.play();
    }
}
```

2. Iterator



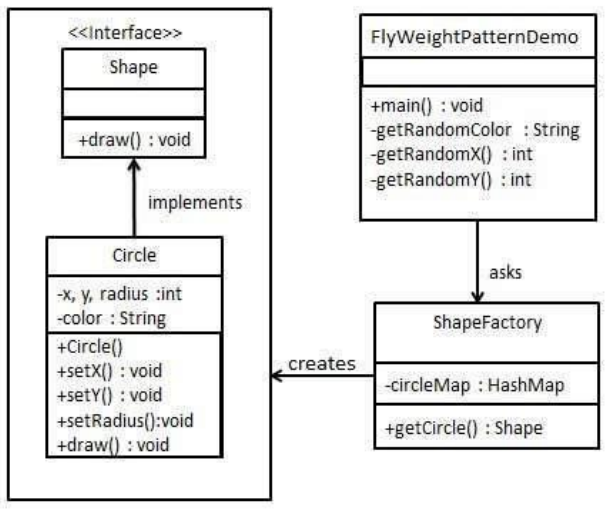
```java
public interface Iterator {
    public boolean hasNext();
    public Object next();
}

public interface Container {
    public Iterator getIterator();
}

public class NameRepository implements Container {
    public String names[] = {
        "Robert", "John", "Julie", "Lora"
    };
    @Override
    public Iterator getIterator() {
        return new NameIterator();
}
    // Klasė klasėje.
    private class NameIterator implements Iterator {
        int index;
        @Override
        public boolean hasNext() {
            if (index < names.length) {
                return true;
            }
            return false;
        }
        @Override
        public Object next() {
            if (this.hasNext()) {
                return names[index++];
            }
            return null;
        }
    }
}

public class IteratorPatternDemo {
    public static void main(String[] args) {
        NameRepository namesRepository = new NameRepository();
        for (Iterator iter = namesRepository.getIterator(); iter.hasNext();) {
            String name = (String) iter.next();
            System.out.println("Name : " + name);
        }
    }
}
```

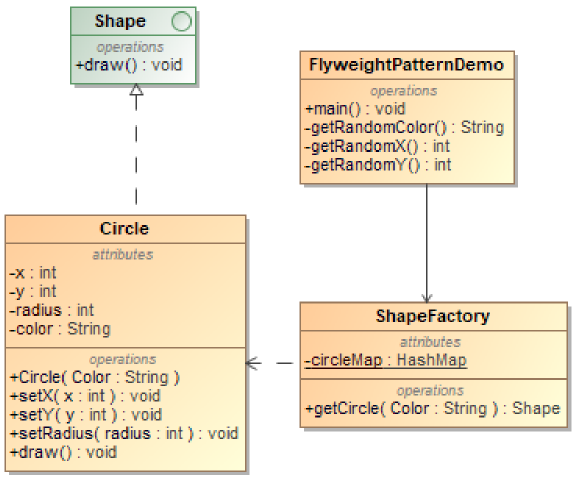
3. Composite



```java
import java.util.ArrayList;
import java.util.List;
public class Employee {
    private String name;
    private String dept;
    private int salary;
    private List < Employee > subordinates; // constructor
    public Employee(String name, String dept, int sal) {
        this.name = name;
        this.dept = dept;
        this.salary = sal;
        subordinates = new ArrayList < Employee > ();
    }
    public void add(Employee e) {
        subordinates.add(e);
    }
    public void remove(Employee e) {
        subordinates.remove(e);
    }
    public List < Employee > getSubordinates() {
        return subordinates;
    }
    public String toString() {
        return ("Employee :[ Name : " + name + ", dept : " + dept + ", salary :" + salary +
 " ]");
    }
}
public class CompositePatternDemo {
    public static void main(String[] args) {
        Employee CEO = new Employee("John", "CEO", 30000);
        Employee headSales = new Employee("Robert", "Head Sales", 20000);
        Employee headMarketing = new Employee("Michel", "Head Marketing", 20000);
        Employee clerk1 = new Employee("Laura", "Marketing", 10000);
        Employee clerk2 = new Employee("Bob", "Marketing", 10000);
        Employee salesExecutive1 = new Employee("Richard", "Sales", 10000);
        Employee salesExecutive2 = new Employee("Rob", "Sales", 10000);
        CEO.add(headSales);
        CEO.add(headMarketing);
        headSales.add(salesExecutive1);
        headSales.add(salesExecutive2);
        headMarketing.add(clerk1);
        headMarketing.add(clerk2); //print all employees of the organization
        System.out.println(CEO);
        for (Employee headEmployee: CEO.getSubordinates()) {
            System.out.println(headEmployee);
            for (Employee employee: headEmployee.getSubordinates()) {
                System.out.println(employee);
            }
        }
    }
}
```

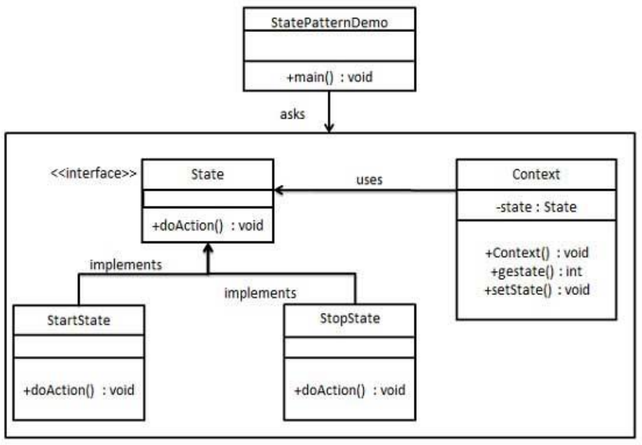
4. Flyweight

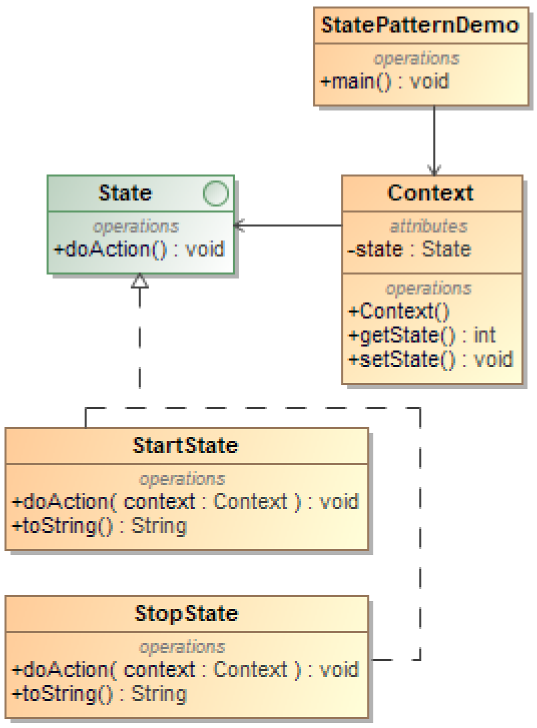




```
public interface Shape {
    void draw();
}
public class Circle implements Shape {
    private String color;
    private int x;
    private int y;
    private int radius;
    public Circle(String color) {
        this.color = color;
    }
    public void setX(int x) {
        this.x = x;
    }
    public void setY(int y) {
        this.y = y;
    }
    public void setRadius(int radius) {
        this.radius = radius;
    }
    @Override
    public void draw() {
        System.out.println("Circle: Draw() [Color : " + color + ", x : " + x + ", y :" + y
+ ", radius :" + radius);
    }
}
public class ShapeFactory {
    private static final HashMap < String, Shape > circleMap = new HashMap();
    public static Shape getCircle(String color) {
        Circle circle = (Circle) circleMap.get(color);
        if (circle == null) {
            circle = new Circle(color);
            circleMap.put(color, circle);
            System.out.println("Creating circle of color : " + color);
        }
        return circle;
    }
}
public class FlyweightPatternDemo {
    private static final String colors[] = {
        "Red", "Green", "Blue", "White", "Black"
    };
    public static void main(String[] args) {
        for (int i = 0; i < 20; ++i) {
            Circle circle = (Circle) ShapeFactory.getCircle(getRandomColor());
            circle.setX(getRandomX());
            circle.setY(getRandomY());
            circle.setRadius(100);
            circle.draw();
        }
    }
    private static String getRandomColor() {
        return colors[(int)(Math.random() * colors.length)];
    }
    private static int getRandomX() {
        return (int)(Math.random() * 100);
    }
    private static int getRandomY() {
        return (int)(Math.random() * 100);
    }
}
```

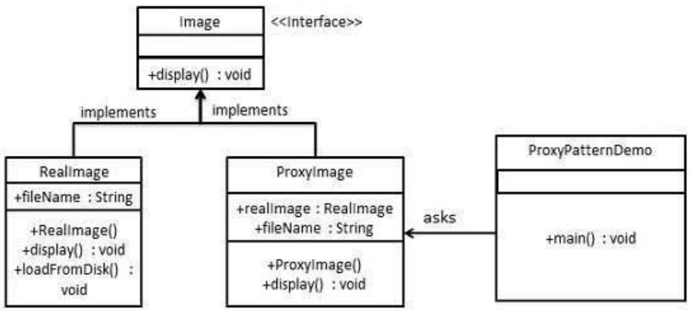
5. State

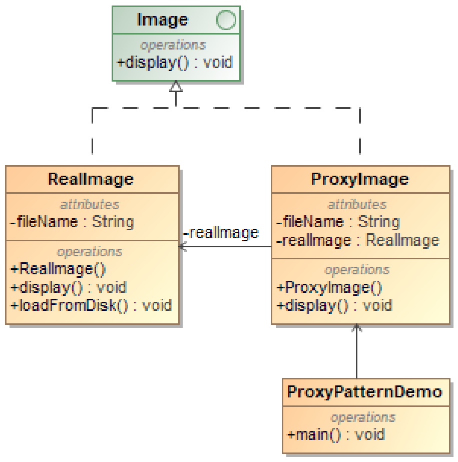




```java
public interface State {
    public void doAction(Context context);
}
public class StartState implements State {
    public void doAction(Context context) {
        System.out.println("Player is in start state");
        context.setState(this);
    }
    public String toString() {
        return "Start State";
    }
}
public class StopState implements State {
    public void doAction(Context context) {
        System.out.println("Player is in stop state");
        context.setState(this);
    }
    public String toString() {
        return "Stop State";
    }
}
public class Context {
    private State state;
    public Context() {
        state = null;
    }
    public void setState(State state) {
        this.state = state;
    }
    public State getState() {
        return state;
    }
}
public class StatePatternDemo {
    public static void main(String[] args) {
        Context context = new Context();
        StartState startState = new StartState();
        startState.doAction(context);
        System.out.println(context.getState().toString());
        StopState stopState = new StopState();
        stopState.doAction(context);
        System.out.println(context.getState().toString());
    }
}
```

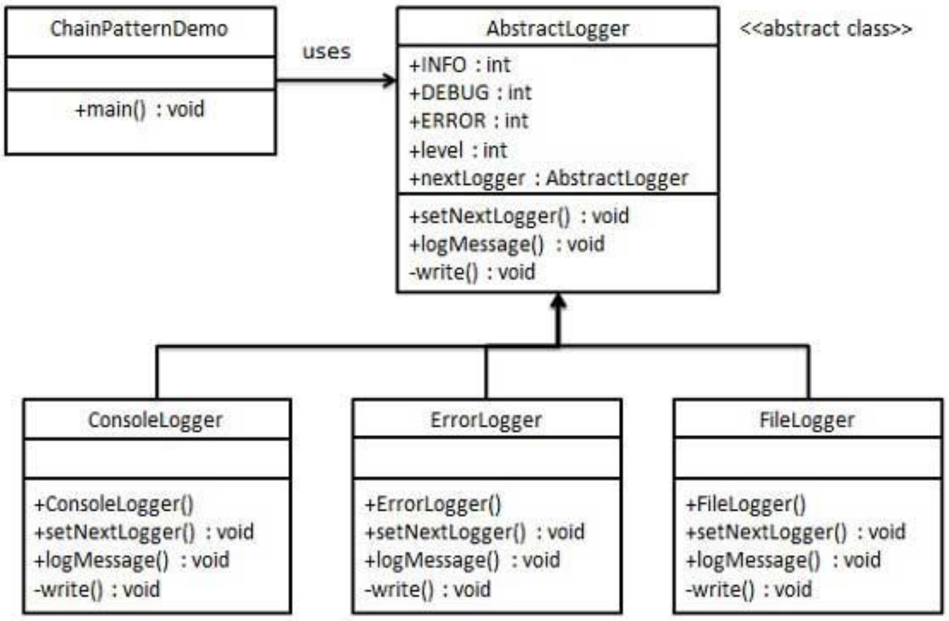
6. Proxy

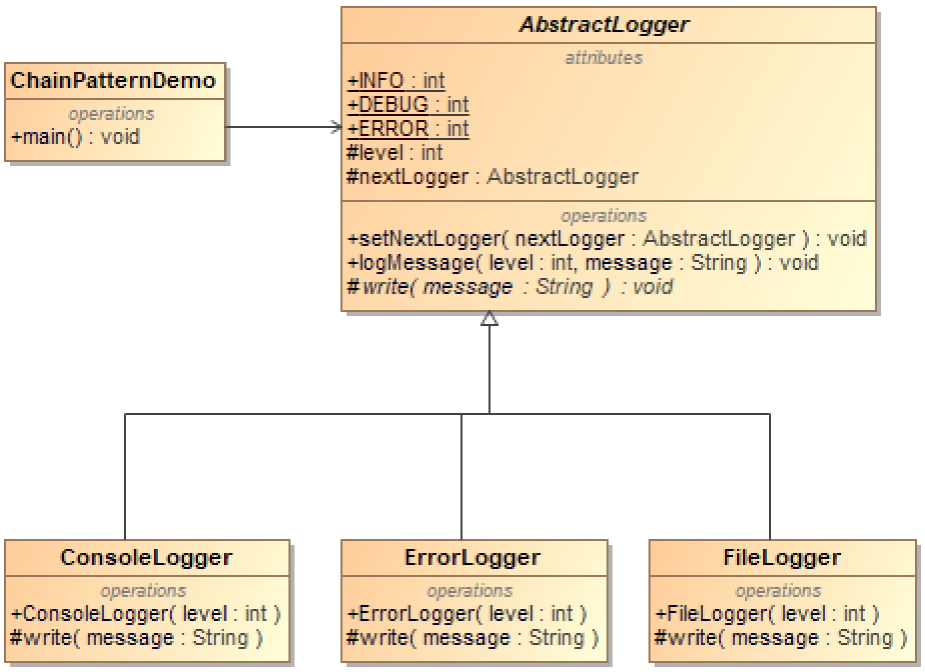




```java
public interface Image {
    void display();
}
public class RealImage implements Image {
    private String fileName;
    public RealImage(String fileName) {
        this.fileName = fileName;
        loadFromDisk(fileName);
    }
    @Override
    public void display() {
        System.out.println("Displaying " + fileName);
    }
    private void loadFromDisk(String fileName) {
        System.out.println("Loading " + fileName);
    }
}
public class ProxyImage implements Image {
    private RealImage realImage;
    private String fileName;
    public ProxyImage(String fileName) {
        this.fileName = fileName;
    }
    @Override
    public void display() {
        if (realImage == null) {
            realImage = new RealImage(fileName);
        }
        realImage.display();
    }
}
public class ProxyPatternDemo {
    public static void main(String[] args) {
        Image image = new ProxyImage("test_10mb.jpg"); //image will be loaded from disk
        image.display();
        System.out.println(""); //image will not be loaded from disk
        image.display();
    }
}
```

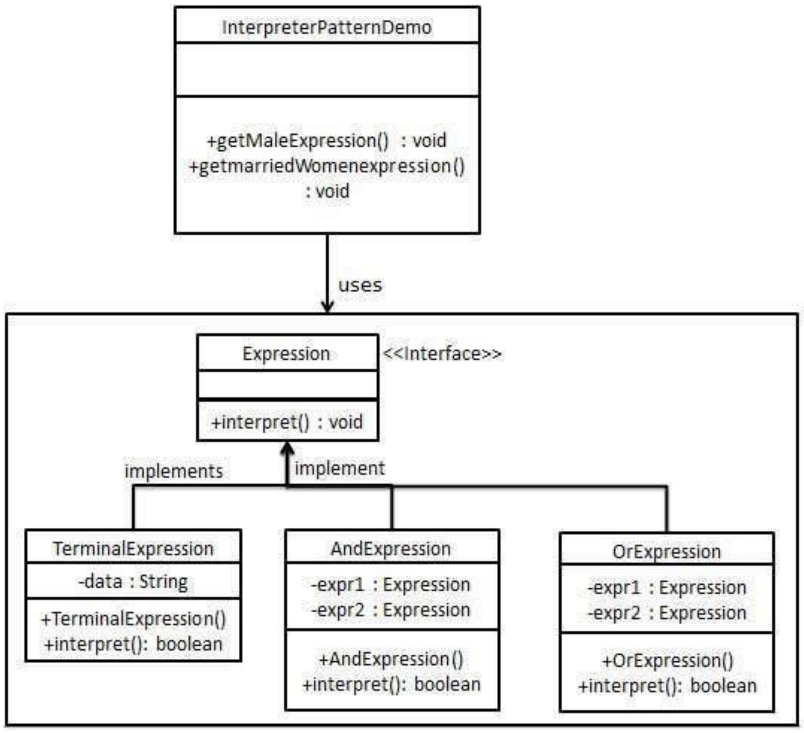
7. Chain of Responsibility

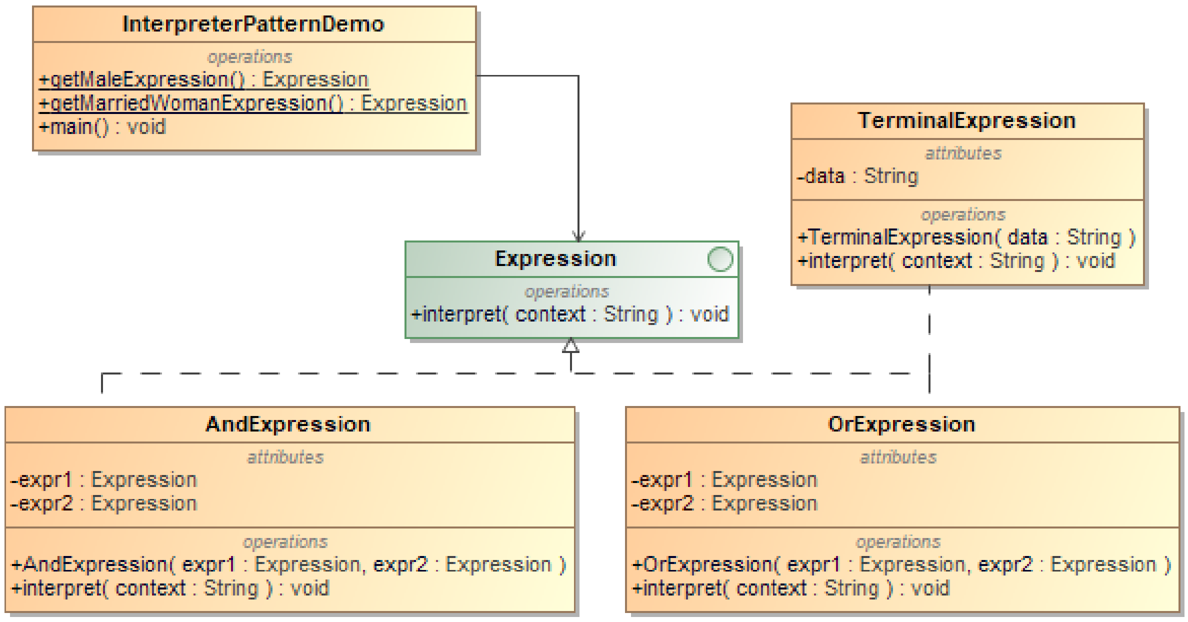



AbstractLogger
attributes
+INFO : int
+DEBUG : int
+ERROR : int
#level : int
#nextLogger : AbstractLogger
operations
+setNextLogger(nextLogger : AbstractLogger) : void
+logMessage(level : int, message : String) : void
#write(message : String) : void
ChainPatternDemo
operations
+main() : void
ConsoleLogger
operations
+ConsoleLogger(level : int)
#write(message : String)
ErrorLogger
operations
+ErrorLogger(level : int)
#write(message : String)
FileLogger
operations
+FileLogger(level : int)
#write(message : String)

```java
public abstract class AbstractLogger {
    public static int INFO = 1;
    public static int DEBUG = 2;
    public static int ERROR = 3;
    protected int level; //next element in chain of responsibility
    protected AbstractLogger nextLogger;
    public void setNextLogger(AbstractLogger nextLogger) {
        this.nextLogger = nextLogger;
    }
    public void logMessage(int level, String message) {
        if (this.level <= level) {
            write(message);
        }
        if (nextLogger != null) {
            nextLogger.logMessage(level, message);
        }
    }
    abstract protected void write(String message);
}
public class ConsoleLogger extends AbstractLogger {
    public ConsoleLogger(int level) {
        this.level = level;
    }
    @Override
    protected void write(String message) {
        System.out.println("Standard Console::Logger: " + message);
    }
}
public class ErrorLogger extends AbstractLogger {
    public ErrorLogger(int level) {
        this.level = level;
    }
    @Override
    protected void write(String message) {
        System.out.println("Error Console::Logger: " + message);
    }
}
public class FileLogger extends AbstractLogger {
    public FileLogger(int level) {
        this.level = level;
    }
    @Override
    protected void write(String message) {
        System.out.println("File::Logger: " + message);
    }
}
public class ChainPatternDemo {
    private static AbstractLogger getChainOfLoggers() {
        AbstractLogger errorLogger = new ErrorLogger(AbstractLogger.ERROR);
        AbstractLogger fileLogger = new FileLogger(AbstractLogger.DEBUG);
        AbstractLogger consoleLogger = new ConsoleLogger(AbstractLogger.INFO);
        errorLogger.setNextLogger(fileLogger);
        fileLogger.setNextLogger(consoleLogger);
        return errorLogger;
    }
    public static void main(String[] args) {
        AbstractLogger loggerChain = getChainOfLoggers();
        loggerChain.logMessage(AbstractLogger.INFO, "This is an information.");
        loggerChain.logMessage(AbstractLogger.DEBUG, "This is an debug level information.");
        loggerChain.logMessage(AbstractLogger.ERROR, "This is an error information.");
    }
}
```

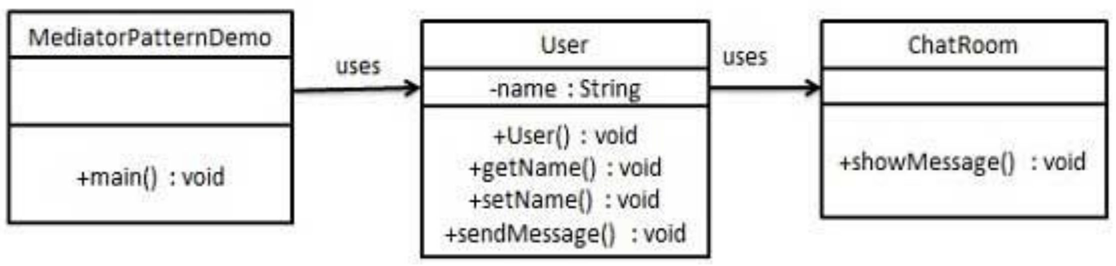
8. Interpreter

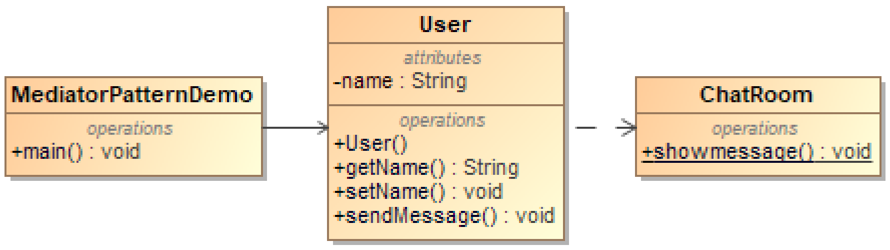



InterpreterPatternDemo
operations
+getMaleExpression() : Expression
+getMarriedWomanExpression() : Expression
+main() : void
TerminalExpression
attributes
-data : String
operations
+TerminalExpression(data : String)
+interpret(context : String) : void
Expression
operations
+interpret(context : String) : void
AndExpression
attributes
-expr1 : Expression
-expr2 : Expression
operations
+AndExpression(expr1 : Expression, expr2 : Expression)
+interpret(context : String) : void
OrExpression
attributes
-expr1 : Expression
-expr2 : Expression
operations
+OrExpression(expr1 : Expression, expr2 : Expression)
+interpret(context : String) : void

```java
public interface Expression {
    public boolean interpret(String context);
}
public class TerminalExpression implements Expression {
    private String data;
    public TerminalExpression(String data) {
        this.data = data;
    }
    @Override
    public boolean interpret(String context) {
        if (context.contains(data)) {
            return true;
        }
        return false;
    }
}
public class OrExpression implements Expression {
    private Expression expr1 = null;
    private Expression expr2 = null;
    public OrExpression(Expression expr1, Expression expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
    @Override
    public boolean interpret(String context) {
        return expr1.interpret(context) || expr2.interpret(context);
    }
}
public class AndExpression implements Expression {
    private Expression expr1 = null;
    private Expression expr2 = null;
    public AndExpression(Expression expr1, Expression expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
    @Override
    public boolean interpret(String context) {
        return expr1.interpret(context) && expr2.interpret(context);
    }
}
public class InterpreterPatternDemo { //Rule: Robert and John are male
    public static Expression getMaleExpression() {
            Expression robert = new TerminalExpression("Robert");
            Expression john = new TerminalExpression("John");
            return new OrExpression(robert, john);
        } //Rule: Julie is a married women
    public static Expression getMarriedWomanExpression() {
        Expression julie = new TerminalExpression("Julie");
        Expression married = new TerminalExpression("Married");
        return new AndExpression(julie, married);
    }
    public static void main(String[] args) {
        Expression isMale = getMaleExpression();
        Expression isMarriedWoman = getMarriedWomanExpression();
        System.out.println("John is male? " + isMale.interpret("John"));
        System.out.println("Julie is a married women? " + isMarriedWoman.interpret("Married
 Julie"));
    }
}
```

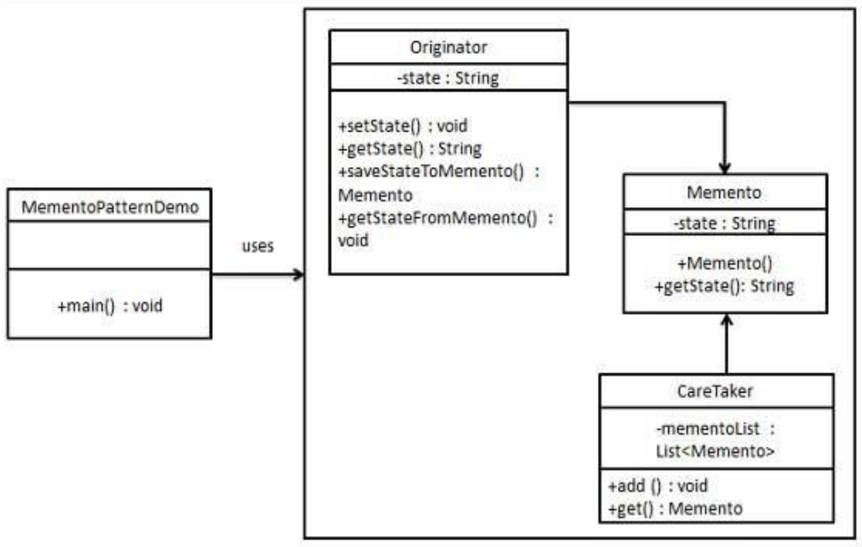
9. Mediator

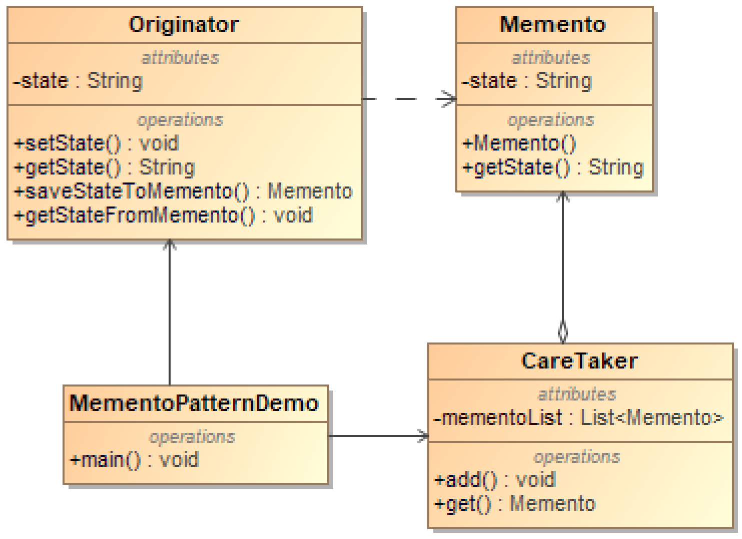




```java
public class ChatRoom {
    public static void showMessage(User user, String message) {
        System.out.println(new Date().toString() + " [" + user.getName() + "] : " + message
);
    }
}
public class User {
    private String name;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public User(String name) {
        this.name = name;
    }
    public void sendMessage(String message) {
        ChatRoom.showMessage(this, message);
    }
}
public class MediatorPatternDemo {
    public static void main(String[] args) {
        User robert = new User("Robert");
        User john = new User("John");
        robert.sendMessage("Hi! John!");
        john.sendMessage("Hello! Robert!");
    }
}
```

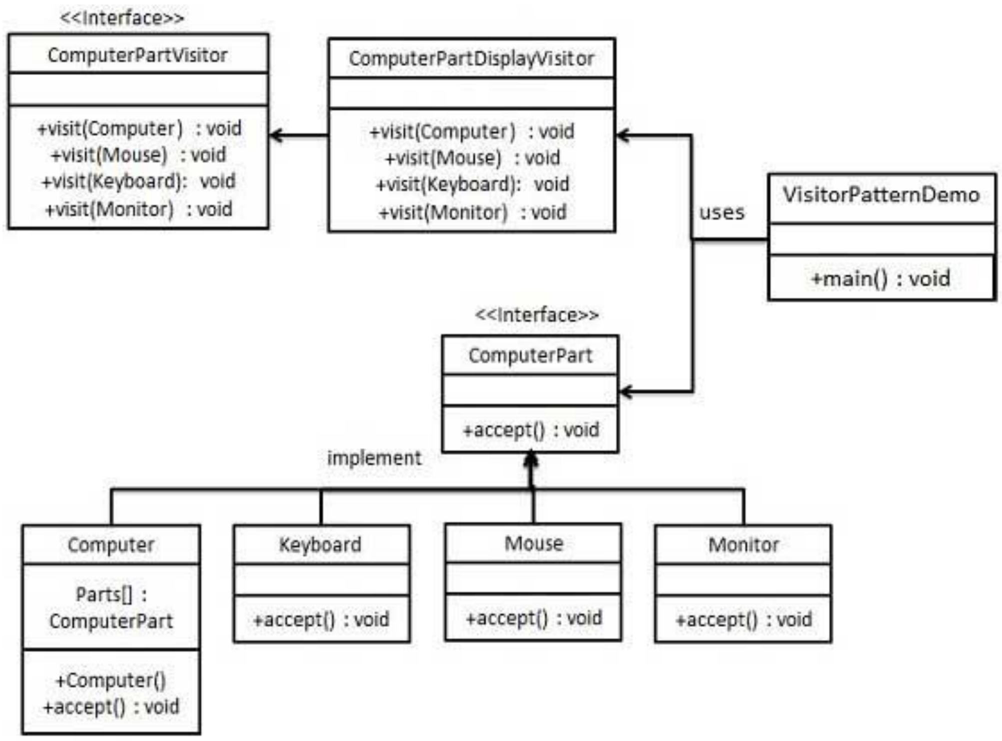
10. Memento

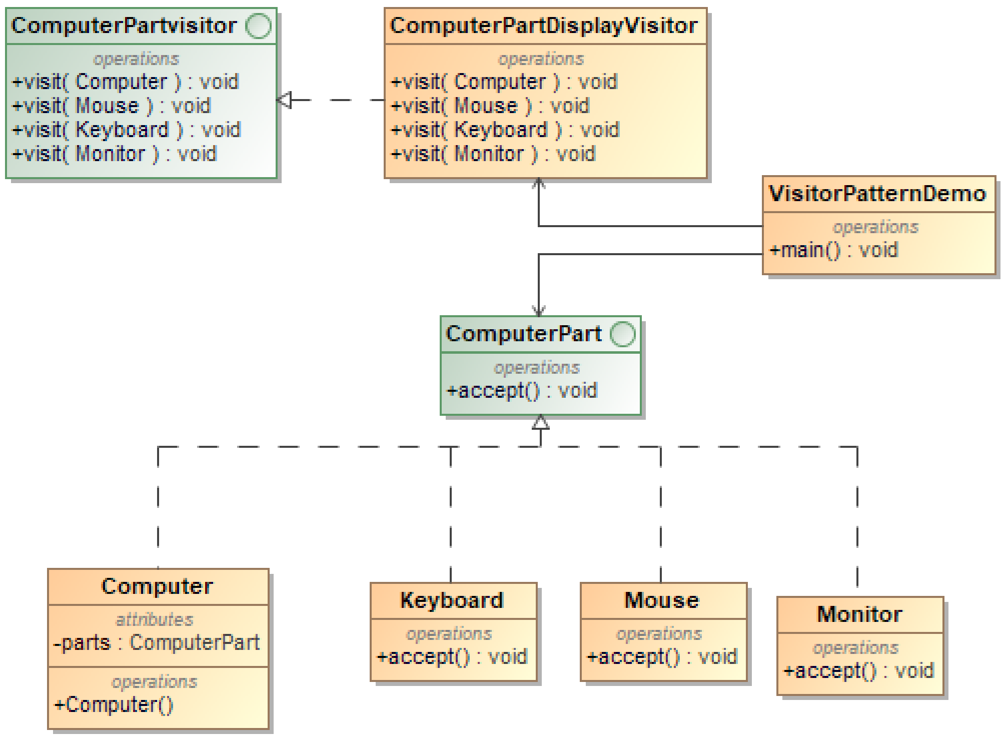




```java
public class Memento {
    private String state;
    public Memento(String state) {
        this.state = state;
    }
    public String getState() {
        return state;
    }
}
public class Originator {
    private String state;
    public void setState(String state) {
        this.state = state;
    }
    public String getState() {
        return state;
    }
    public Memento saveStateToMemento() {
        return new Memento(state);
    }
    public void getStateFromMemento(Memento memento) {
        state = memento.getState();
    }
}
public class CareTaker {
    private List < Memento > mementoList = new ArrayList < Memento > ();
    public void add(Memento state) {
        mementoList.add(state);
    }
    public Memento get(int index) {
        return mementoList.get(index);
    }
}
public class MementoPatternDemo {
    public static void main(String[] args) {
        Originator originator = new Originator();
        CareTaker careTaker = new CareTaker();
        originator.setState("State #1");
        originator.setState("State #2");
        careTaker.add(originator.saveStateToMemento());
        originator.setState("State #3");
        careTaker.add(originator.saveStateToMemento());
        originator.setState("State #4");
        System.out.println("Current State: " + originator.getState());
        originator.getStateFromMemento(careTaker.get(0));
        System.out.println("First saved State: " + originator.getState());
        originator.getStateFromMemento(careTaker.get(1));
        System.out.println("Second saved State: " + originator.getState());
    }
}
```

11. Visitor





```java
public interface ComputerPart {
    public void accept(ComputerPartVisitor computerPartVisitor);
}
public class Keyboard implements ComputerPart {
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor) {
        computerPartVisitor.visit(this);
    }
}
```

```java
public class Monitor implements ComputerPart {
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor) {
        computerPartVisitor.visit(this);
    }
}
public class Mouse implements ComputerPart {
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor) {
        computerPartVisitor.visit(this);
    }
}
public class Computer implements ComputerPart {
    ComputerPart[] parts;
    public Computer() {
        parts = new ComputerPart[] {
            new Mouse(), new Keyboard(), new Monitor()
        };
    }
    @Override
    public void accept(ComputerPartVisitor computerPartVisitor) {
        for (int i = 0; i < parts.length; i++) {
            parts[i].accept(computerPartVisitor);
        }
        computerPartVisitor.visit(this);
    }
}
public interface ComputerPartVisitor {
    public void visit(Computer computer);
    public void visit(Mouse mouse);
    public void visit(Keyboard keyboard);
    public void visit(Monitor monitor);
}
public class ComputerPartDisplayVisitor implements ComputerPartVisitor {@
    Override public void visit(Computer computer) {
        System.out.println("Displaying Computer.");
    }
    @Override
    public void visit(Mouse mouse) {
        System.out.println("Displaying Mouse.");
    }
    @Override
    public void visit(Keyboard keyboard) {
        System.out.println("Displaying Keyboard.");
    }
    @Override
    public void visit(Monitor monitor) {
        System.out.println("Displaying Monitor.");
    }
}
public class VisitorPatternDemo {
    public static void main(String[] args) {
        ComputerPart computer = new Computer();
        computer.accept(new ComputerPartDisplayVisitor());
    }
}
```