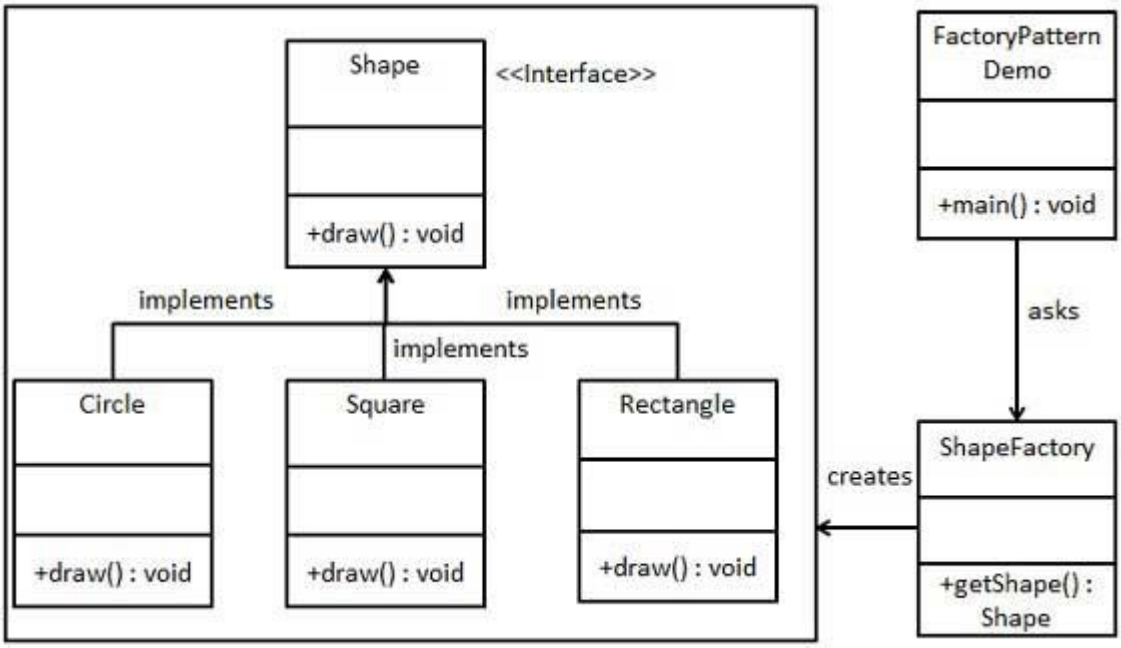
Creational Design Patterns:

1.Factory



Code:

```
public interface Shape
{
   void draw();
}


public class Rectangle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Rectangle::draw() method.");
   }
}


public class Square implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Square::draw() method.");
   }
```

```
}


public class Circle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Circle::draw() method.");
   }
}


public class ShapeFactory
{
   //use getShape method to get object of type shape
   public Shape getShape(String shapeType)
   {
      if(shapeType == null)
      {
         return null;
      }
      if(shapeType.equalsIgnoreCase("CIRCLE"))
      {
         return new Circle();

      }
else if(shapeType.equalsIgnoreCase("RECTANGLE"))
      {
         return new Rectangle();
      }
else if(shapeType.equalsIgnoreCase("SQUARE"))
      {
         return new Square();
      }
      return null;
   }
}


public class FactoryPatternDemo
{
   public static void main(String[] args)
{
      ShapeFactory shapeFactory = new ShapeFactory();
      //get an object of Circle and call its draw method.
      Shape shape1 = shapeFactory.getShape("CIRCLE");
      //call draw method of Circle
      shape1.draw();
      //get an object of Rectangle and call its draw method.
      Shape shape2 = shapeFactory.getShape("RECTANGLE");
```

```
        //call draw method of Rectangle
        shape2.draw();
        //get an object of Square and call its draw method.
        Shape shape3 = shapeFactory.getShape("SQUARE");
        //call draw method of square
        shape3.draw();
    }
}
```

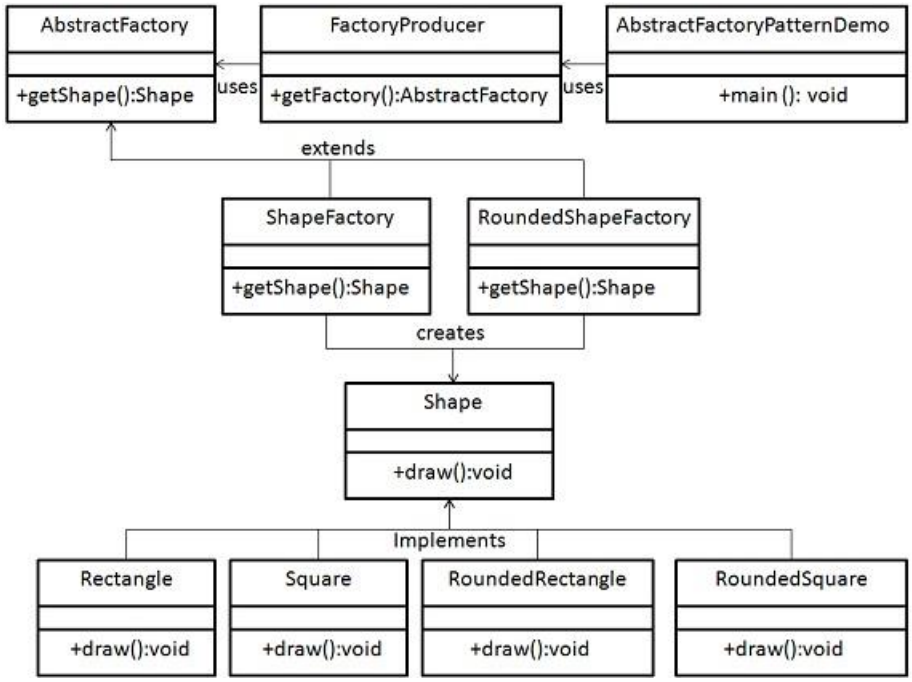
Output:

Inside Circle::draw() method.

Inside Rectangle::draw() method.

Inside Square::draw() method.

2.Abstract Factory



Code:

```
public interface Shape
{
   void draw();
}
```

```
public class RoundedRectangle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside RoundedRectangle::draw()
method.");
   }
}


public class RoundedSquare implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside RoundedSquare::draw()
method.");
   }
}


public class Rectangle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Inside Rectangle::draw() method.");
   }
}


public class Square implements Shape
{
@override
     public void draw()
     {
     System.out.println("Inside Square::draw() method.");
     }
}



public abstract class AbstractFactory
{
   abstract Shape getShape(String shapeType) ;
}
```

```
public class ShapeFactory extends AbstractFactory
{
   @Override
   public Shape getShape(String shapeType)
     {
      if(shapeType.equalsIgnoreCase("RECTANGLE"))
      {
         return new Rectangle();
      }
else if(shapeType.equalsIgnoreCase("SQUARE"))
      {
         return new Square();
      }
      return null;
   }
}


public class RoundedShapeFactory extends AbstractFactory
{
   @Override
   public Shape getShape(String shapeType)
      {
      if(shapeType.equalsIgnoreCase("RECTANGLE"))
      {
         return new RoundedRectangle();
      }
else if(shapeType.equalsIgnoreCase("SQUARE"))
      {
         return new RoundedSquare();
      }
      return null;
   }
}


public class FactoryProducer
{
   public static AbstractFactory getFactory(boolean rounded){
      if(rounded)
      {
         return new RoundedShapeFactory();
      }
     else
      {
         return new ShapeFactory();
      }
   }
}
```

```
public class AbstractFactoryPatternDemo
{
   public static void main(String[] args)
{
      //get shape factory
      AbstractFactory shapeFactory =
FactoryProducer.getFactory(false);
      //get an object of Shape Rectangle
      Shape shape1 = shapeFactory.getShape("RECTANGLE");
      //call draw method of Shape Rectangle
      shape1.draw();
      //get an object of Shape Square
      Shape shape2 = shapeFactory.getShape("SQUARE");
      //call draw method of Shape Square
      shape2.draw();
      //get shape factory
      AbstractFactory shapeFactory1 =
FactoryProducer.getFactory(true);
      //get an object of Shape Rectangle
      Shape shape3 = shapeFactory1.getShape("RECTANGLE");
      //call draw method of Shape Rectangle
      shape3.draw();
      //get an object of Shape Square
      Shape shape4 = shapeFactory1.getShape("SQUARE");
      //call draw method of Shape Square
      shape4.draw();
   }
}
```

Output:

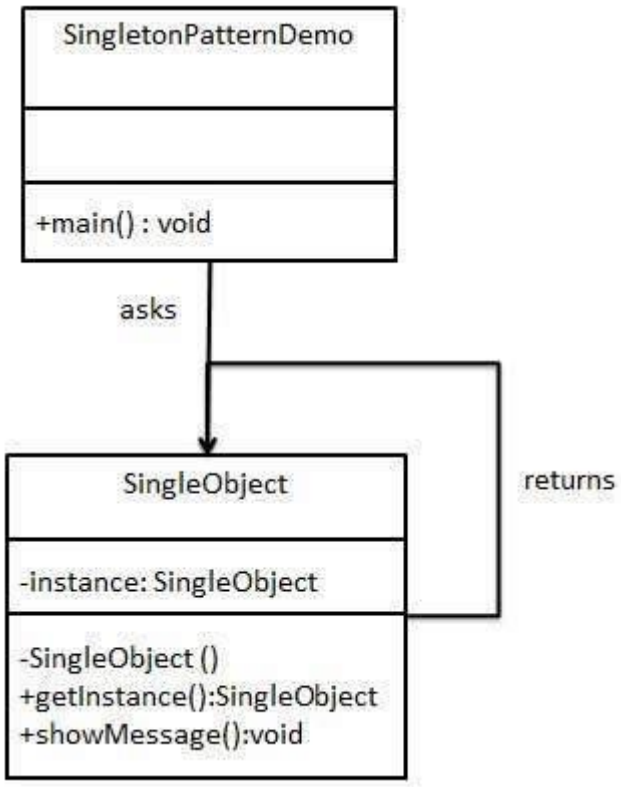
Inside Rectangle::draw() method.

Inside Square::draw() method.

Inside RoundedRectangle::draw() method.

Inside RoundedSquare::draw() method.

3.Singleton

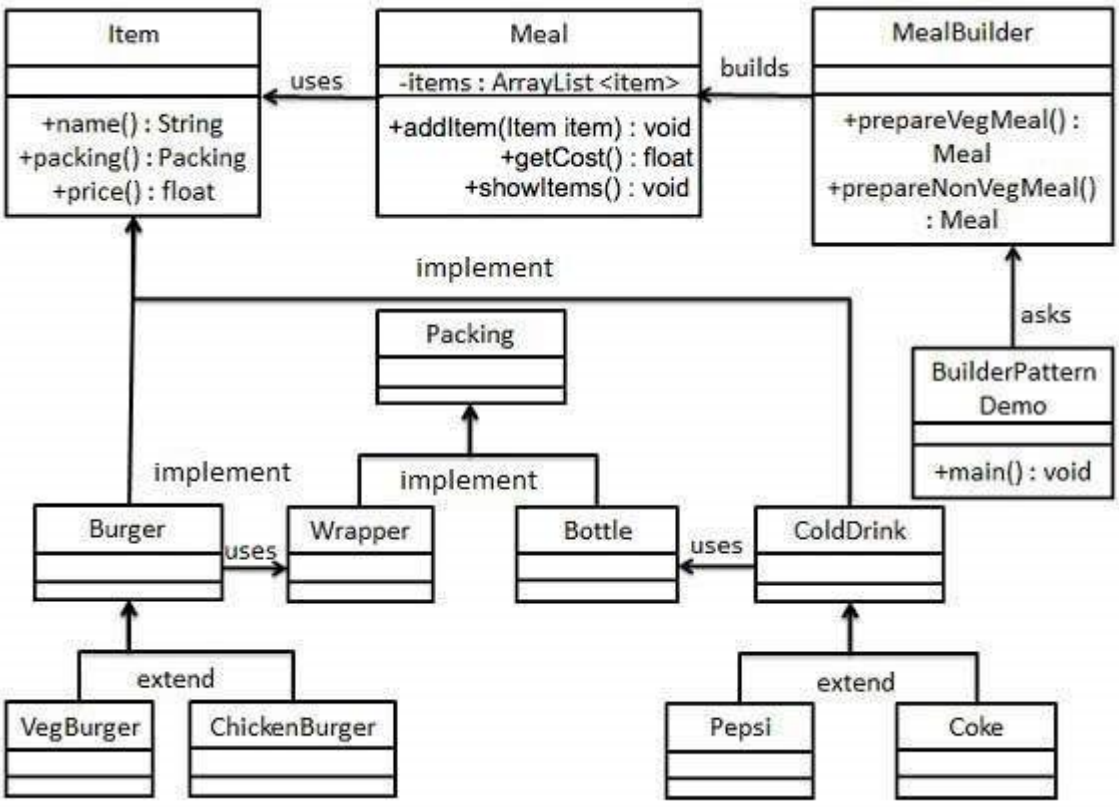


Code:

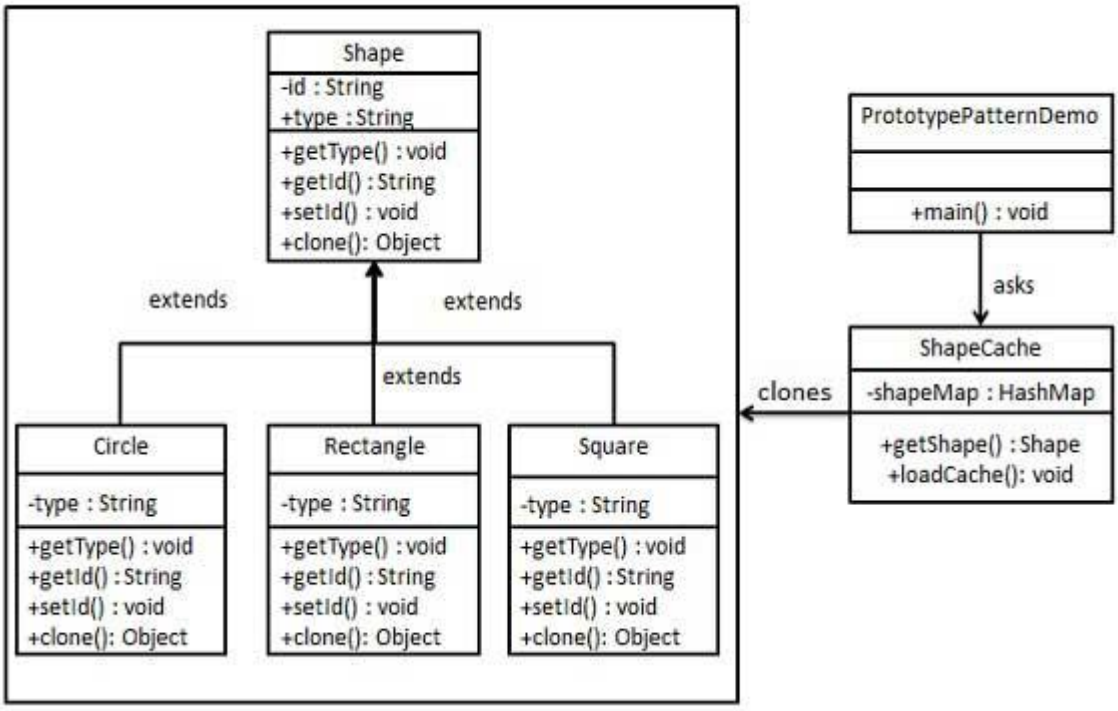
```
public class SingleObject
{
   //create an object of SingleObject
   private static SingleObject instance = new SingleObject();

   //make the constructor private so that this class cannot be
instantiated
   private SingleObject()
   {
   }

   //Get the only object available
   public static SingleObject getInstance()
{
      return instance;
   }

   public void showMessage()
{
      System.out.println("Hello World!");
   }
```

```
}


public class SingletonPatternDemo
{
   public static void main(String[] args)
{

      //Get the only object available
      SingleObject object = SingleObject.getInstance();

      //show the message
      object.showMessage();
   }
}
```

4.Builder

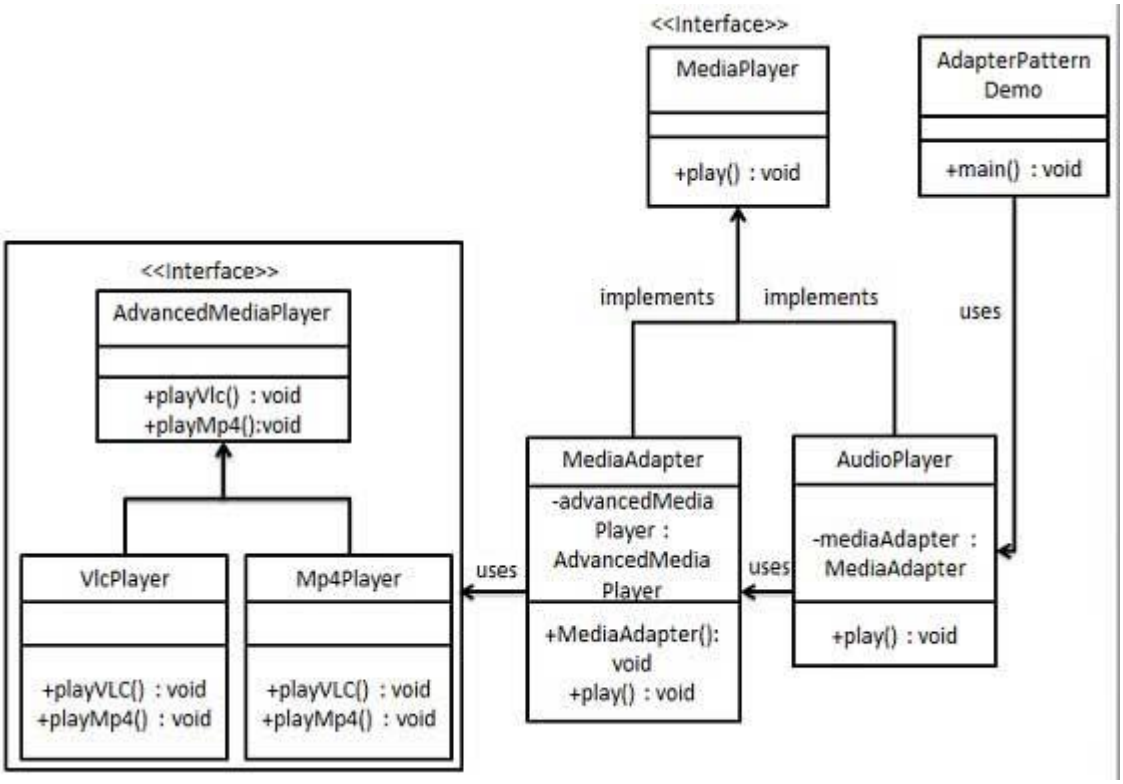


5.Prototype



Structural Design Patterns:

1.Adapter



Code:

```
public interface MediaPlayer
{
   public void play(String audioType, String fileName);
}


public interface AdvancedMediaPlayer
{
   public void playVlc(String fileName);
   public void playMp4(String fileName);
}


public class VlcPlayer implements AdvancedMediaPlayer
{
   @Override
   public void playVlc(String fileName)
   {
      System.out.println("Playing vlc file. Name: "+
fileName);
   }
   @Override
   public void playMp4(String fileName)
   {
      //do nothing
   }
}


public class Mp4Player implements AdvancedMediaPlayer
{
   @Override
   public void playVlc(String fileName)
   {
      //do nothing
   }
   @Override
   public void playMp4(String fileName)
   {
      System.out.println("Playing mp4 file. Name: "+
fileName);
   }
}
```

```
public class MediaAdapter implements MediaPlayer
{

   AdvancedMediaPlayer advancedMusicPlayer;

   public MediaAdapter(String audioType)
   {

      if(audioType.equalsIgnoreCase("vlc") )
      {
         advancedMusicPlayer = new VlcPlayer();

      }
     else if (audioType.equalsIgnoreCase("mp4"))
      {
         advancedMusicPlayer = new Mp4Player();
      }
   }

   @Override
   public void play(String audioType, String fileName)
   {
      if(audioType.equalsIgnoreCase("vlc"))
      {
         advancedMusicPlayer.playVlc(fileName);
      }
      else if(audioType.equalsIgnoreCase("mp4"))
      {
         advancedMusicPlayer.playMp4(fileName);
      }
   }
}


public class AudioPlayer implements MediaPlayer
{
   MediaAdapter mediaAdapter;
   @Override
   public void play(String audioType, String fileName)
   {
      //inbuilt support to play mp3 music files
      if(audioType.equalsIgnoreCase("mp3"))
      {
         System.out.println("Playing mp3 file. Name: " +
fileName);
      }
      //mediaAdapter is providing support to play other file
formats
      else if(audioType.equalsIgnoreCase("vlc") ||
audioType.equalsIgnoreCase("mp4"))
```

```
        {
            mediaAdapter = new MediaAdapter(audioType);
            mediaAdapter.play(audioType, fileName);
        }
        else
        {
            System.out.println("Invalid media. " + audioType + "
format not supported");
        }
    }
}


public class AdapterPatternDemo
{
    public static void main(String[] args)
    {
        AudioPlayer audioPlayer = new AudioPlayer();
        audioPlayer.play("mp3", "beyond the horizon.mp3");
        audioPlayer.play("mp4", "alone.mp4");
        audioPlayer.play("vlc", "far far away.vlc");
        audioPlayer.play("avi", "mind me.avi");
    }
}
```

Output:

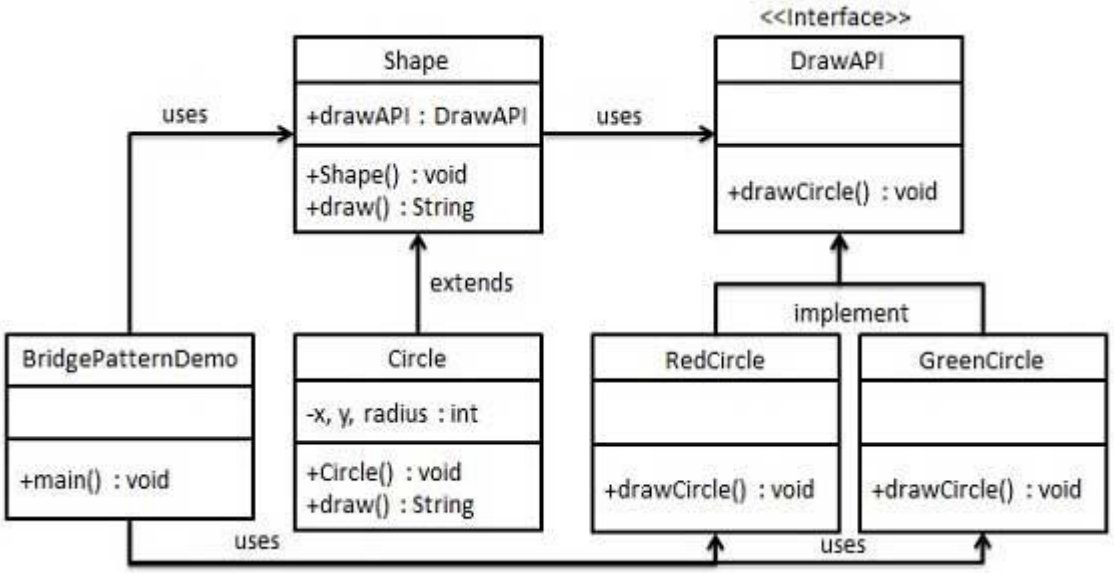
Playing mp3 file. Name: beyond the horizon.mp3

Playing mp4 file. Name: alone.mp4

Playing vlc file. Name: far far away.vlc

Invalid media. avi format not supported

2.Bridge



Code:

```
public interface DrawAPI
      {
         public void drawCircle(int radius, int x, int y);
      }

      public class RedCircle implements DrawAPI
      {
         @Override
         public void drawCircle(int radius, int x, int y)
         {
            System.out.println("Drawing Circle[ color: red,
radius: " + radius + ", x: " + x + ", " + y + "]");
         }
      }
      public class GreenCircle implements DrawAPI
      {
         @Override
         public void drawCircle(int radius, int x, int y)
         {
            System.out.println("Drawing Circle[ color: green,
radius: " + radius + ", x: " + x + ", " + y + "]");
         }
      }
```

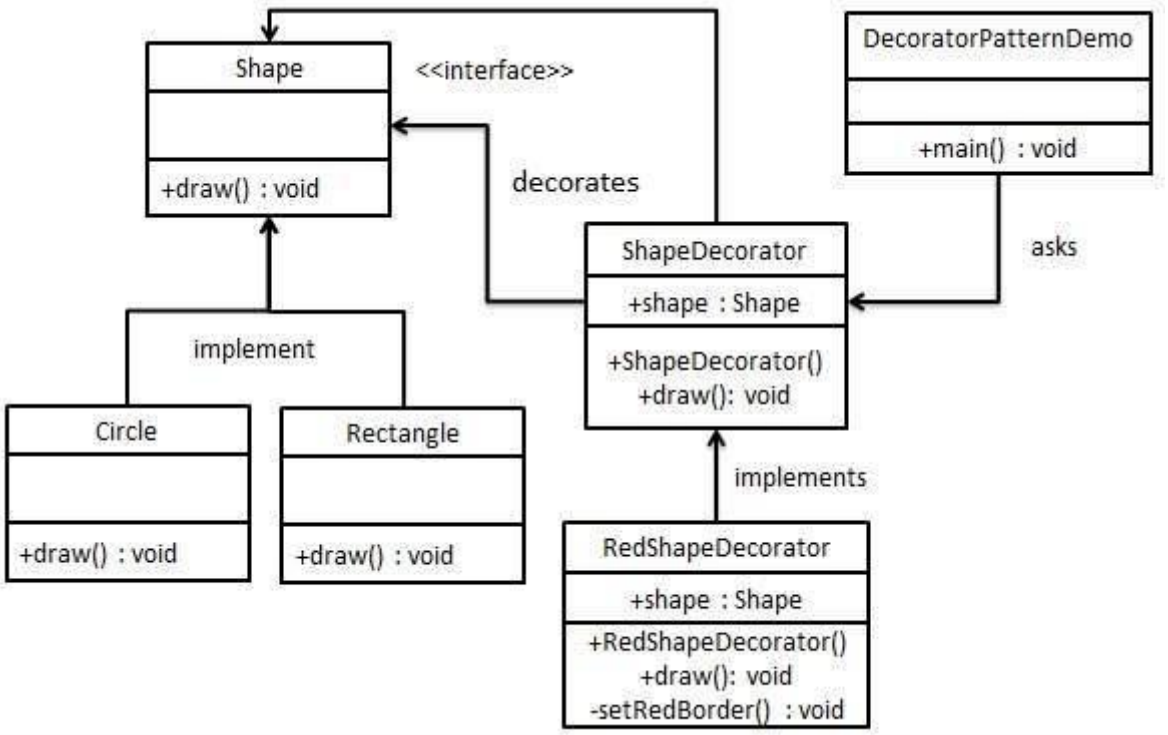
```
public abstract class Shape
{
   protected DrawAPI drawAPI;
   protected Shape(DrawAPI drawAPI)
   {
      this.drawAPI = drawAPI;
   }
   public abstract void draw();
}


public class Circle extends Shape
{
   private int x, y, radius;
   public Circle(int x, int y, int radius, DrawAPI
drawAPI)
   {
      super(drawAPI);
      this.x = x;
      this.y = y;
      this.radius = radius;
   }
   public void draw()
{
      drawAPI.drawCircle(radius,x,y);
   }
}


public class BridgePatternDemo
 {
   public static void main(String[] args)
   {

Shape redCircle = new Circle(100,100, 10, new RedCircle());
Shape greenCircle = new Circle(100,100, 10, new
GreenCircle());
      redCircle.draw();
      greenCircle.draw();
   }
}
```

Output:

Drawing Circle[color: red, radius: 10, x: 100, 100]

Drawing Circle[color: green, radius: 10, x: 100, 100]

3.Decorator



Code:

```
public interface Shape
{
   void draw();
}
```

```
public class Rectangle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Shape: Rectangle");
   }
}
```

```
public class Circle implements Shape
{
   @Override
   public void draw()
   {
      System.out.println("Shape: Circle");
```

```
    }
}


public abstract class ShapeDecorator implements Shape
{
   protected Shape decoratedShape;
   public ShapeDecorator(Shape decoratedShape)
   {
      this.decoratedShape = decoratedShape;
   }
   public void draw()
   {
      decoratedShape.draw();
   }
}


public class RedShapeDecorator extends ShapeDecorator
{
   public RedShapeDecorator(Shape decoratedShape)
   {
      super(decoratedShape);
   }

   @Override
   public void draw()
   {
      decoratedShape.draw();
      setRedBorder(decoratedShape);
   }

   private void setRedBorder(Shape decoratedShape)
    {
      System.out.println("Border Color: Red");
   }
}


public class DecoratorPatternDemo
{
   public static void main(String[] args)
   {
      Shape circle = new Circle();
      Shape redCircle = new RedShapeDecorator(new Circle());
      Shape redRectangle = new RedShapeDecorator(new
Rectangle());
      System.out.println("Circle with normal border");
      circle.draw();
      System.out.println("\nCircle of red border");
      redCircle.draw();
```

```
        System.out.println("\nRectangle of red border");
        redRectangle.draw();
    }
}
```

Output:

Circle with normal border

Shape: Circle

Circle of red border

Shape: Circle

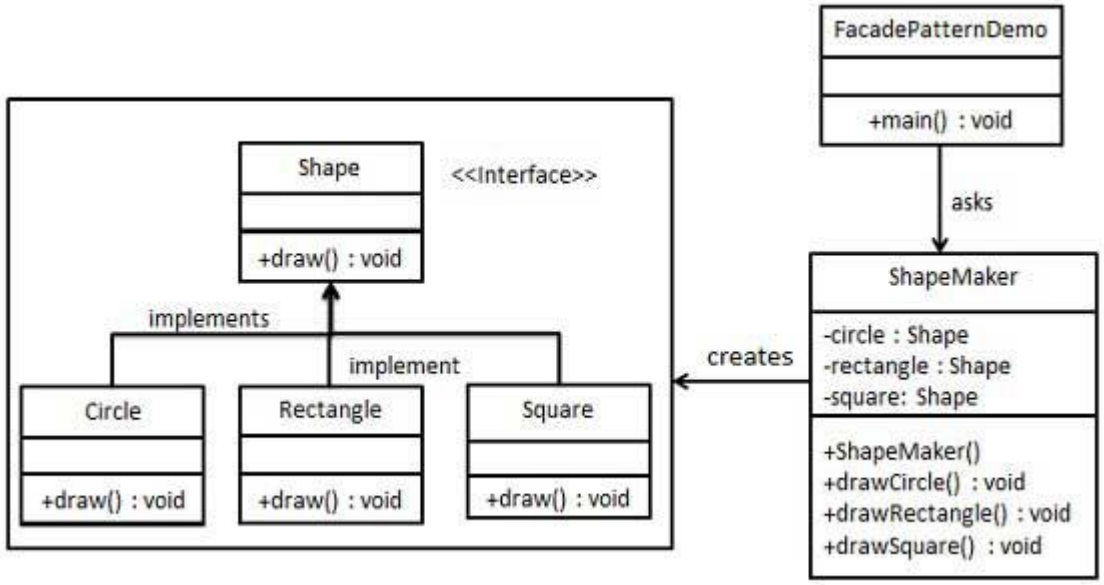
Border Color: Red

Rectangle of red border

Shape: Rectangle

Border Color: Red

4.Facade



Code:

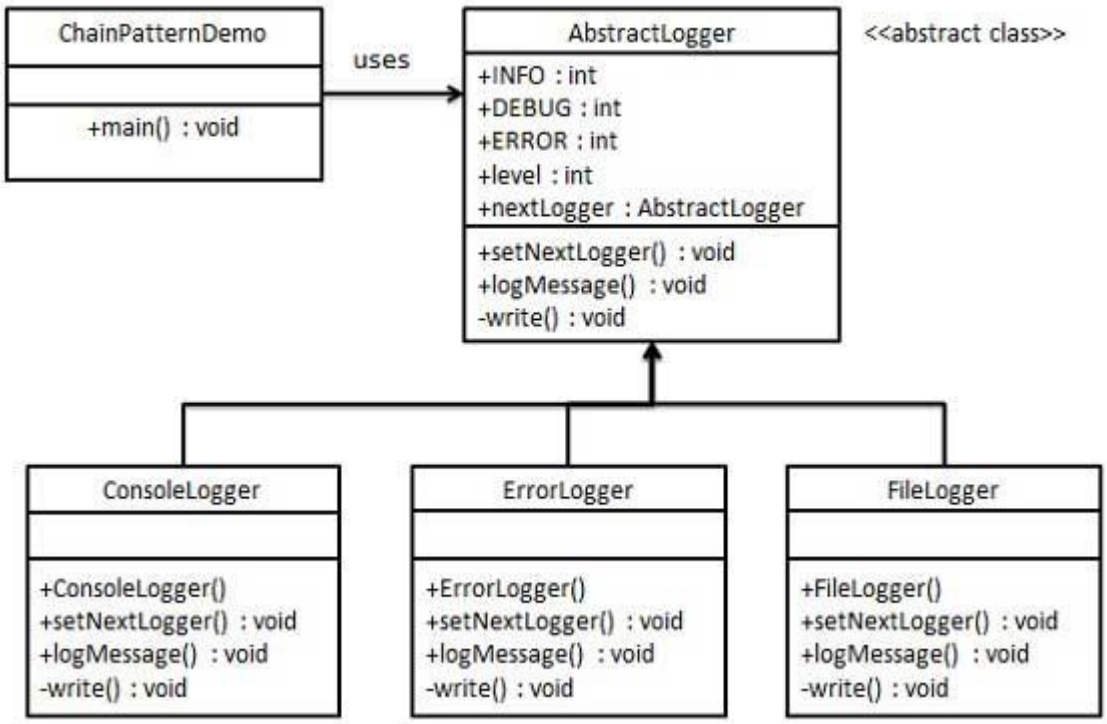
```
public interface Shape
{
    void draw();
}
```

```java
//Step 2: Create concrete classes implementing the same
interface. (Rectangle.java & Circle.java & Square.java)

public class Square implements Shape
{
    @Override
    public void draw()
    {
        System.out.println("Square::draw()");

    }

}


//Step 2: Create concrete classes implementing the same
interface. (Rectangle.java & Circle.java & Square.java)

public class Rectangle implements Shape
{
    @Override
    public void draw()
    {
        System.out.println("Rectangle::draw()");

    }

}

//Step 2: Create concrete classes implementing the same
interface. (Rectangle.java & Circle.java & Square.java)

public class Circle implements Shape
{
    @Override
    public void draw()
    {
        System.out.println("Circle::draw()");

    }

}

//Step 3: Create a facade class. (ShapeMaker.java)

public class ShapeMaker
{
    private Shape circle;
       private Shape rectangle;
       private Shape square;
```

```
        public ShapeMaker()
        {
           circle = new Circle();
           rectangle = new Rectangle();
           square = new Square();
        }

        public void drawCircle()
        {
           circle.draw();
        }
        public void drawRectangle()
        {
           rectangle.draw();
        }
        public void drawSquare()
        {
           square.draw();
        }
}

//Step 4: Use the facade to draw various types of shapes.
(FacadePatternDemo.java)

public class FacadePatternDemo
{
    public static void main(String[] args)
    {
        ShapeMaker shapeMaker = new ShapeMaker();

          shapeMaker.drawCircle();
          shapeMaker.drawRectangle();
          shapeMaker.drawSquare();
    }

}
```

Behavioural Design Patterns:

1.Chain of Responsibility



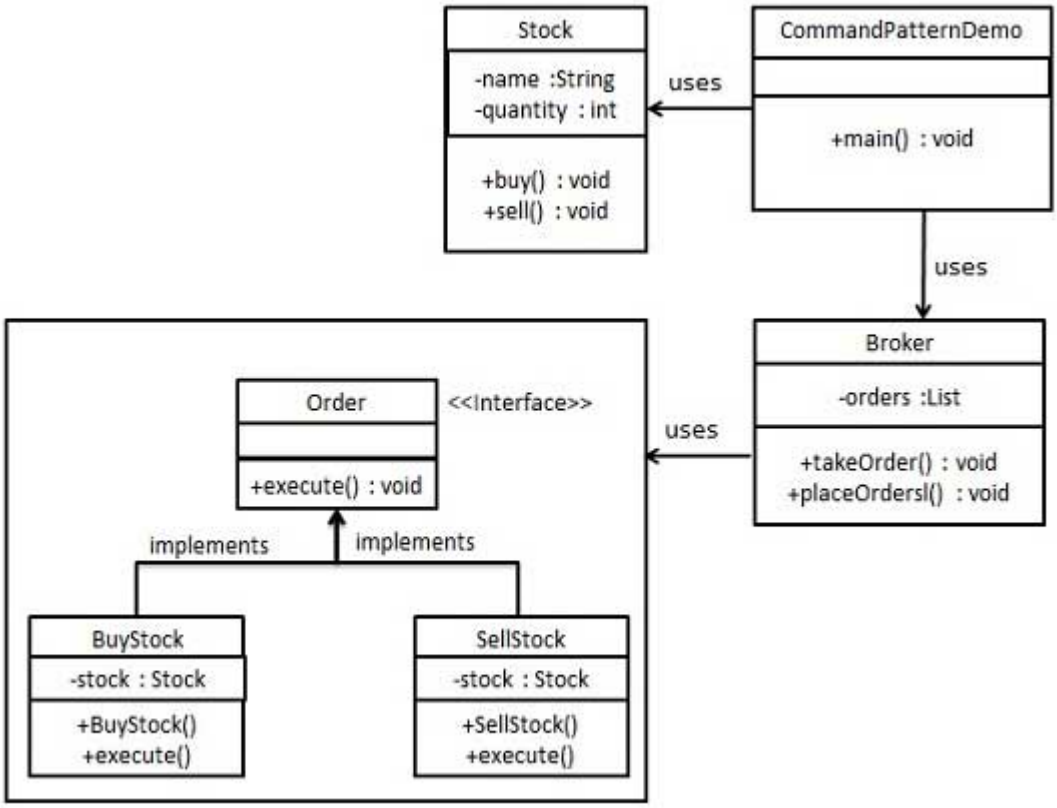
Code:

```
public abstract class AbstractLogger
{
        public static int MDLevel = 1;
        public static int RMLevel = 2;
        public static int EMPLevel = 3;

        protected int level;
        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 ChainPatternDemo
{
      private static AbstractLogger getChainOfLoggers()
      {

            AbstractLogger EMPlg = new
EmpLogger(AbstractLogger.EMPLevel);
            AbstractLogger RMlg = new
RMLogger(AbstractLogger.RMLevel);
            AbstractLogger MDlg = new
MDLogger(AbstractLogger.MDLevel);

            EMPlg.setNextLogger(RMlg);
            RMlg.setNextLogger(MDlg);

            return EMPlg;
         }

         public static void main(String[] args)
         {
            AbstractLogger loggerChain = getChainOfLoggers();

            loggerChain.logMessage(AbstractLogger.EMPLevel,
               "This is an information from Employee.");

            loggerChain.logMessage(AbstractLogger.RMLevel,
               "This is an Approval of information from Regional
Manager.");

            loggerChain.logMessage(AbstractLogger.MDLevel,
               "This is an Approval of information from Managing
Director.");
         }
      }
```

```
public class EmpLogger extends AbstractLogger
{
    public EmpLogger(int level)
    {
          this.level = level;
    }

       @Override
       protected void write(String message)
       {
          System.out.println("Standard Employee::Logger: " +
message);
       }
}



public class MDLogger extends AbstractLogger
{
    public MDLogger(int level)
       {
          this.level = level;
       }

       @Override
       protected void write(String message)
       {
          System.out.println("Managing Director::Logger: " +
message);
       }
}



public class RMLogger extends AbstractLogger
{
    public RMLogger(int level)
       {
          this.level = level;
       }

       @Override
       protected void write(String message)
       {
          System.out.println("Regional Manager::Logger: " +
message);
       }
}
```

2.Command



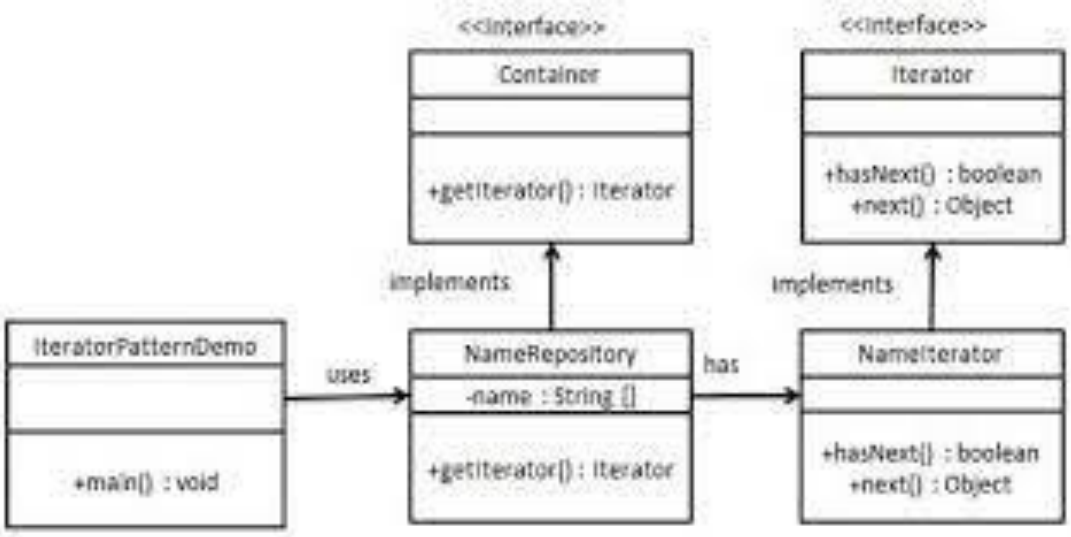
Code:

```
public interface Order
{
    void execute();
}


public class BuyStock implements Order
{
     private Stock abcStock;

       public BuyStock(Stock abcStock)
       {
          this.abcStock = abcStock;
       }

       public void execute()
       {
          abcStock.buy();
```

```
        }
}

public class SellStock implements Order
{
    private Stock abcStock;

        public SellStock(Stock abcStock)
        {
            this.abcStock = abcStock;
        }

        public void execute()
        {
            abcStock.sell();
        }

}


public class Stock
{
        private String name = "Creov Inc";
        private int quantity = 95;

        public void buy()
        {
            System.out.println("Stock [ Name: "+name+",
Quantity: " + quantity +" ] bought");
        }
        public void sell()
        {
            System.out.println("Stock [ Name: "+name+",
Quantity: " + quantity +" ] sold");
        }
}


package CDPDemo;
import java.util.ArrayList;
import java.util.List;

public class Broker
{
     private List<Order> orderList = new ArrayList<Order>();

        public void takeOrder(Order order)
        {
            orderList.add(order);
        }
```

```
        public void placeOrders()
        {

           for (Order order : orderList)
           {
              order.execute();
           }
           orderList.clear();
        }
}

public class CommandPatternDemo
{
    public static void main(String[] args)
     {
         Stock abcStock = new Stock();

          BuyStock buyStockOrder = new BuyStock(abcStock);
          SellStock sellStockOrder = new SellStock(abcStock);

          Broker broker = new Broker();
          broker.takeOrder(buyStockOrder);
          broker.takeOrder(sellStockOrder);

          broker.placeOrders();

    }

}
```

3.Iterator



Code:

```
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();
   }


   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);
      }
   }
}
```

Output:

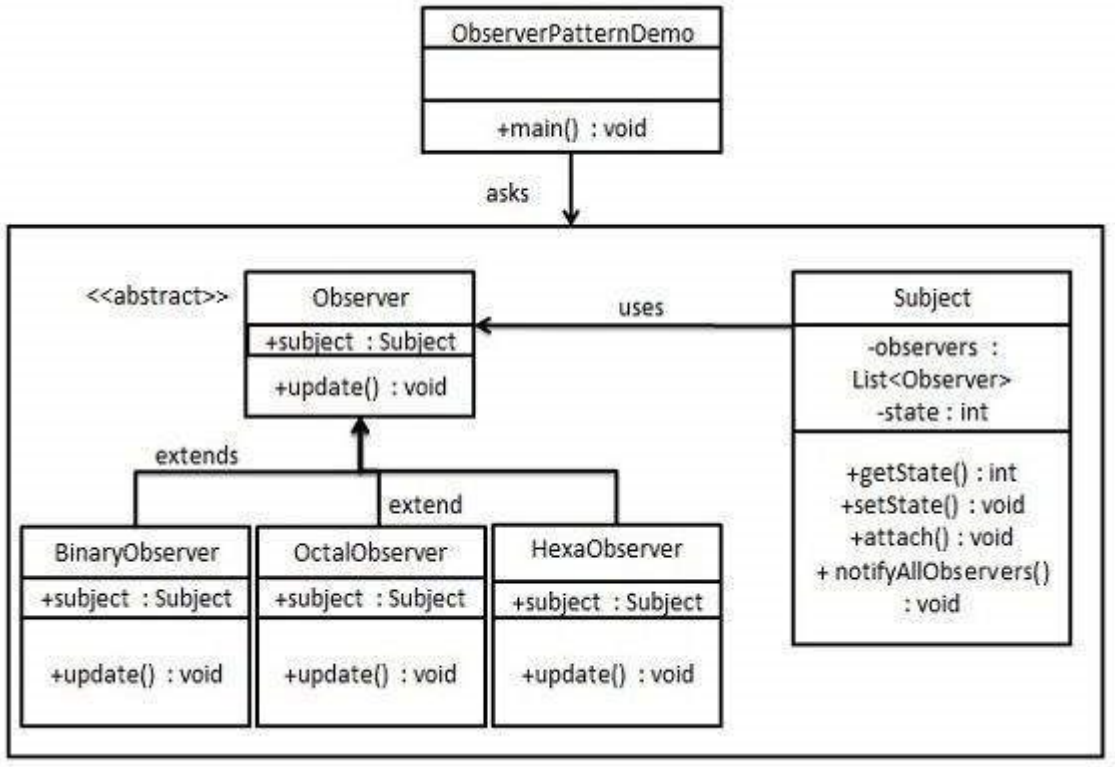
Name : Robert

Name : John

Name : Julie

Name : Lora

4.Observer



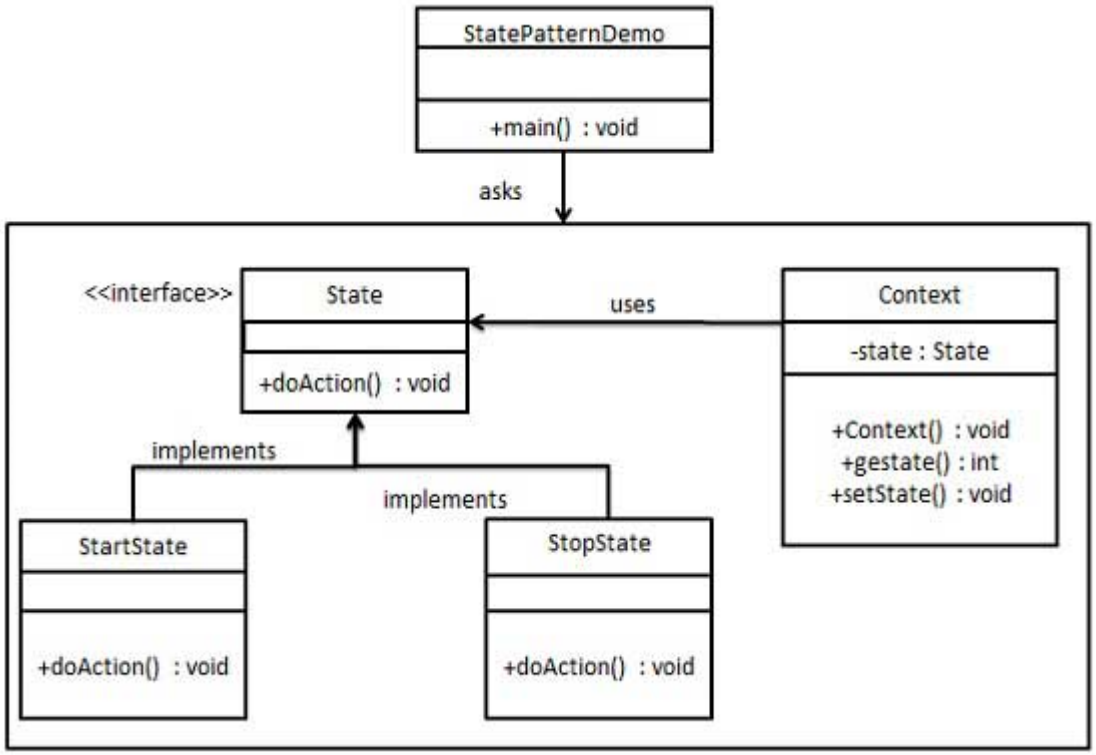
Code:

```
package ODBDemo;

public class BinaryObserver extends Observer
{
    public BinaryObserver(Subject subject)
      {
          this.subject = subject;
          this.subject.attach(this);
       }

       @Override
       public void update()
       {
          System.out.println( "Binary String: " +
Integer.toBinaryString( subject.getState() ) );
       }
}


public class HexaObserver extends Observer
      {

       public HexaObserver(Subject subject)
       {
          this.subject = subject;
          this.subject.attach(this);
       }

       @Override
       public void update()
       {
          System.out.println( "Hex String: " +
Integer.toHexString( subject.getState() ).toUpperCase() );
       }
    }


public abstract class Observer
{
       protected Subject subject;
       public abstract void update();
}

public class ObserverPatternDemo
{
    public static void main(String[] args)
     {
```

```
        Subject subject = new Subject();

        new HexaObserver(subject);
        new OctalObserver(subject);
        new BinaryObserver(subject);

        System.out.println("First state change: 15");
        subject.setState(15);
        System.out.println("Second state change: 10");
        subject.setState(10);
    }

}


public class OctalObserver extends Observer
{
    public OctalObserver(Subject subject)
        {
           this.subject = subject;
           this.subject.attach(this);
        }

        @Override
        public void update()
        {
          System.out.println( "Octal String: " +
Integer.toOctalString( subject.getState() ) );
        }
}


package ODBDemo;
import java.util.ArrayList;
import java.util.List;

public class Subject
{

   private List<Observer> observers = new
ArrayList<Observer>();
   private int state;

   public int getState()
   {
      return state;
   }

   public void setState(int state)
   {
      this.state = state;
```

```
        notifyAllObservers();
    }

    public void attach(Observer observer)
    {
        observers.add(observer);
    }

    public void notifyAllObservers()
    {
        for (Observer observer : observers)
        {
            observer.update();
        }
    }
}
```

5.State



Code:

```
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());
    }
}
```

Output:

Player is in start state

Start State

Player is in stop state

Stop State

AVL Tree:

```
class AVLNode
{

    AVLNode left, right;
    int data;
    int height;

    public AVLNode()
  {
        left = null;
        right = null;
        data = 0;
        height = 0;
    }

    public AVLNode(int n)
  {
        left = null;
        right = null;
        data = n;
        height = 0;
    }
}


// AVL Tree Class

class AVLTree
{
    private AVLNode root;
    public AVLTree()
   {
        root = null;
    }


    private int height(AVLNode avlNode)
  {
        return avlNode == null ? -1 : avlNode.height;
    }

    private int max(int lHeight, int rHeight)
   {
        return lHeight > rHeight ? lHeight : rHeight;
    }

    public void insert(int data)
```

```
  {
        root = insert(data, root);
    }


    private AVLNode insert(int data, AVLNode avlNode)
    {
        if (avlNode == null)
            avlNode = new AVLNode(data);
        else if (data < avlNode.data)
       {
            avlNode.left = insert(data, avlNode.left);

if (height(avlNode.left) - height(avlNode.right) == 2)
                if (data < avlNode.left.data)
                    avlNode = leftRotation(avlNode);
                else
                    avlNode = leftRightRotation(avlNode);
        }
 else if (data > avlNode.data)
           {
            avlNode.right = insert(data, avlNode.right);

if (height(avlNode.right) - height(avlNode.left) == 2)
                if (data > avlNode.right.data)
                    avlNode = rightRotation(avlNode);
                else
                    avlNode = rightLeftRotation(avlNode);
          } else
        ;
        avlNode.height = max(height(avlNode.left),
height(avlNode.right)) + 1;
        return avlNode;
    }

    private AVLNode leftRotation(AVLNode avlNode)
   {
        AVLNode k1 = avlNode.left;
        avlNode.left = k1.right;
        k1.right = avlNode;
        avlNode.height = max(height(avlNode.left),
height(avlNode.right)) + 1;
        k1.height = max(height(k1.left), avlNode.height) + 1;
        return k1;
    }


 private AVLNode rightRotation(AVLNode avlNode)
   {
        AVLNode node = avlNode.right;
        avlNode.right = node.left;
```

```
        node.left = avlNode;
        avlNode.height = max(height(avlNode.left),
height(avlNode.right)) + 1;
        node.height = max(height(node.right), avlNode.height)
+ 1;
        return node;
    }

    private AVLNode leftRightRotation(AVLNode avlNode)
  {
        avlNode.left = rightRotation(avlNode.left);
        return leftRotation(avlNode);
    }

    private AVLNode rightLeftRotation(AVLNode avlNode)
   {
        avlNode.right = leftRotation(avlNode.right);
        return rightRotation(avlNode);
    }

    public int countNodes()
  {
        return countNodes(root);
    }

    private int countNodes(AVLNode avlNode)
    {
        if (avlNode == null)
            return 0;
        else
          {
            int l = 1;
            l += countNodes(avlNode.left);
            l += countNodes(avlNode.right);
            return l;
        }
    }

    public boolean search(int data)
   {
        return search(root, data);
    }


    private boolean search(AVLNode avlNode, int data)
   {
        boolean found = false;
        while ((avlNode != null) && !found)
        {
            int rval = avlNode.data;
            if (data < rval)
```

```
                avlNode = avlNode.left;
            else if (data > rval)
                avlNode = avlNode.right;
            else
             {
                found = true;
                break;
            }
            found = search(avlNode, data);
        }
        return found;
    }


    public void inorder() {
        inorder(root);
    }


    private void inorder(AVLNode avlNode)
   {
        if (avlNode != null)
          {
            inorder(avlNode.left);
            System.out.print(avlNode.data + " ");
            inorder(avlNode.right);
        }
    }
}

import java.util.Scanner;

public class AVLTreeHelper
{
    public static void main(String[] args)
     {
        Scanner scanner = new Scanner(System.in);
        AVLTree avlTree = new AVLTree();

        char ch;
        do
        {
            System.out.println("\nAVLTree Operations\n");
            System.out.println("1. insert ");
            System.out.println("2. search");
            System.out.println("3. count nodes");
            int choice = scanner.nextInt();
            switch (choice)
          {
                case 1:
```

```
System.out.println("Enter integer element to insert");
                    avlTree.insert(scanner.nextInt());
                    break;
                case 2:

System.out.println("Enter integer element to search");
                    System.out.println("Search result : " +
avlTree.search(scanner.nextInt()));
                    break;
                case 3:

System.out.println("Nodes = " + avlTree.countNodes());
                    break;
                default:

System.out.println("Wrong Entry \n ");
                    break;
            }


 System.out.print("\nIn order : ");
            avlTree.inorder();


System.out.println("\nDo you want to continue (Type y or n)
\n");
            ch = scanner.next().charAt(0);
        }
while (ch == 'Y' || ch == 'y');
    }
}
```