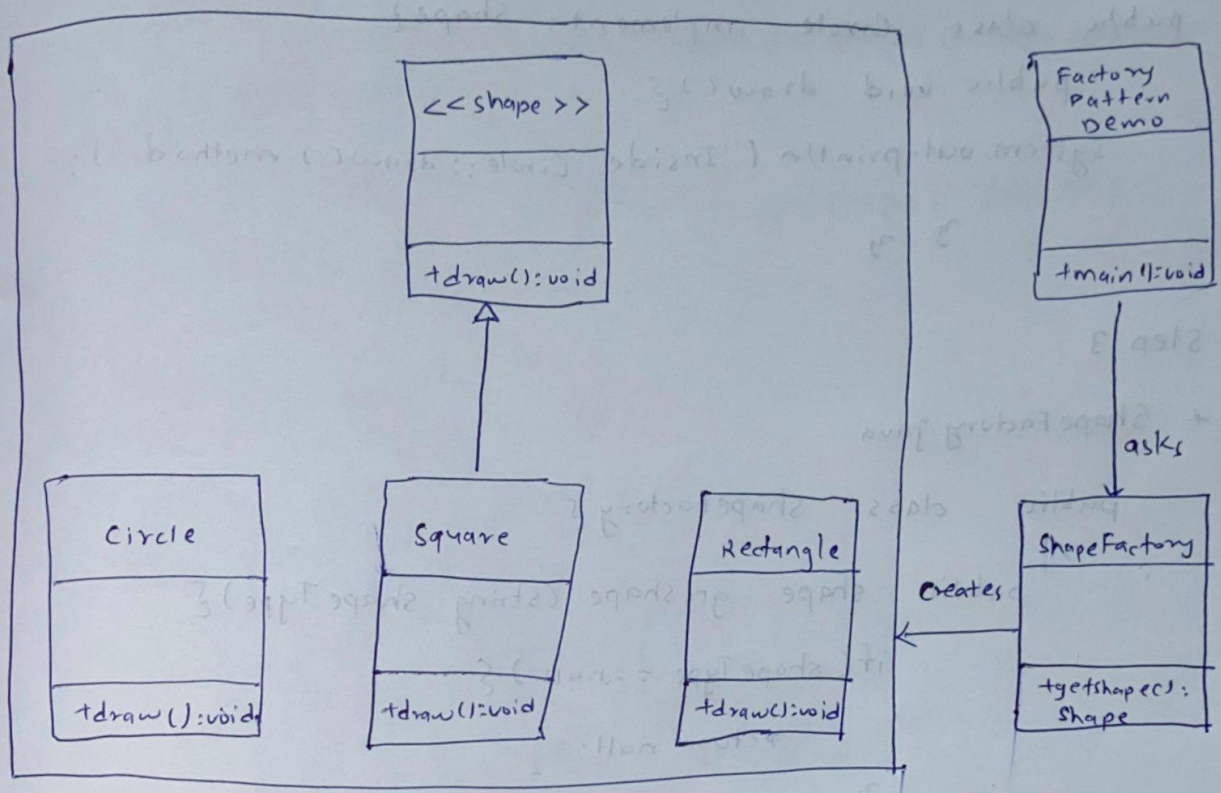
AOOP - Advanced object oriented programming

LAB - 1

* Design patterns

① Factory pattern.



Step 1

Create an Interface

* Shape.java

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

Step 2

* Rectangle.java

```
public class Rectangle implements shape {
    public void draw() {
        system.out.println("Inside Rectangle::draw() method");
    }
}
```

* Square.java

```
public class Square implements
    public void draw() {
system.out.println(" Inside Square :: draw() method.");
    }
}
```

* Circle.java

```
public class Circle implements Shape{
    public void draw(){
system.out.println("Inside Circle :: draw() method.");
    } }
```

Step 3

* ShapeFactory.java

```
public class ShapeFactory {
    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;
    }
}
```

Step 4

* FactoryPatternDemo.java

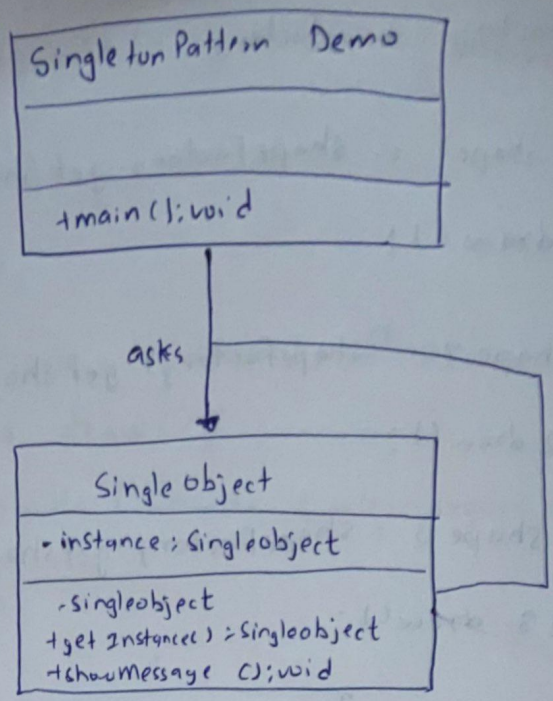
```
public static void main(String[] args) {
   ShapeFactory shapeFactory = new shapeFactory();

   Shape shape1 = shapeFactory.getShape("CIRCLE");
   shape1.draw();

   Shape shape2 = shapeFactory.getShape("RECTANGLE");
   shape2.draw();

   Shape shape3 = shapeFactory.getshape("SQUARE");
   shape3.draw();

                }
                }
```

Step 5

Output

```
Inside Circle :: draw() method.
Inside Rectangle :: draw() method
Inside Square :: draw() method
```

① Singleton pattern



Step 1

* singleObject.java

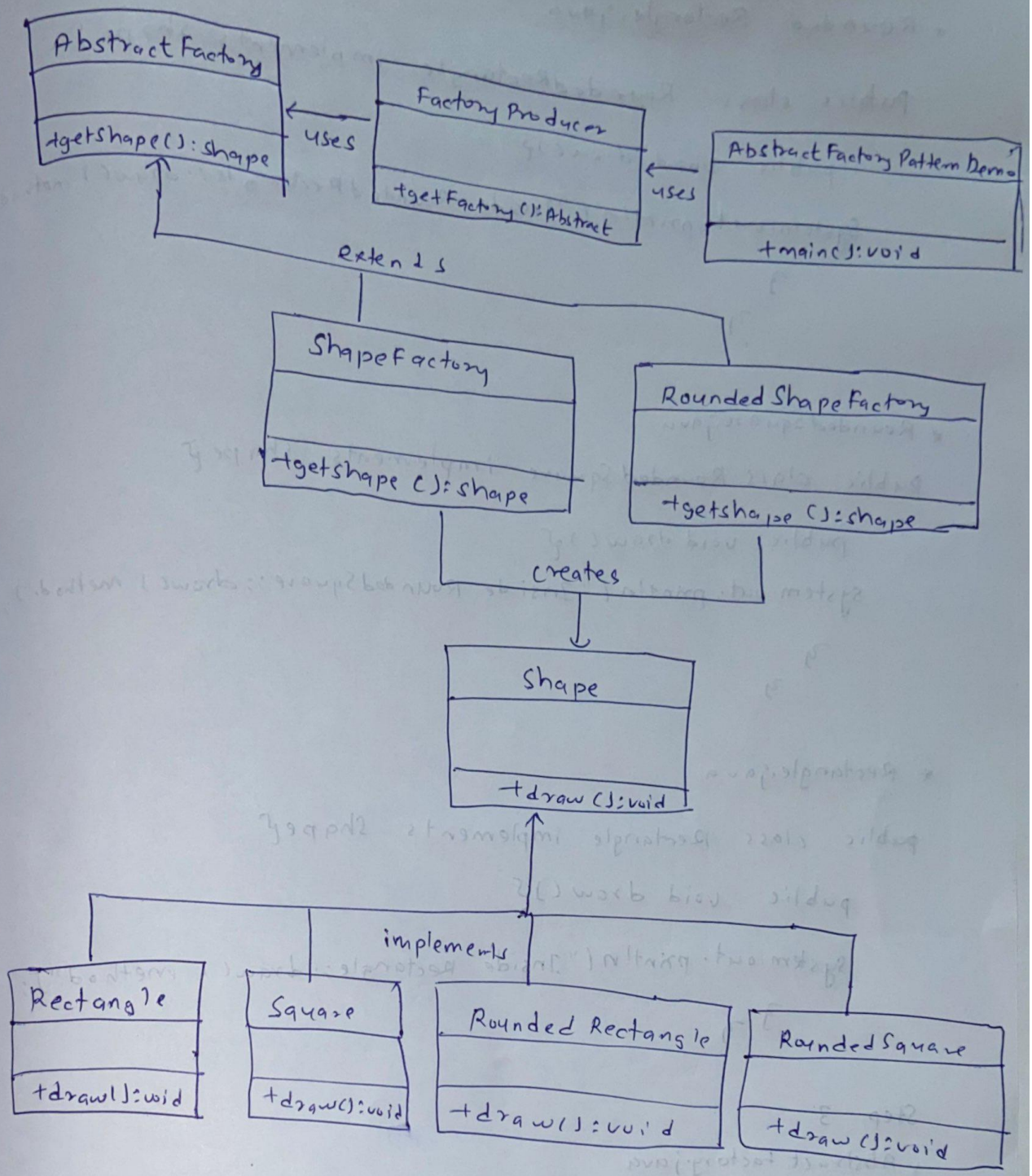
```
public class SingleObject {
   private static SingleObject instance = new SingleObject();
   private SingleObject(){}
   public static SingleObject getInstance(){
      return instance;
   }

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

Step 2

* SingletonPatternDemo.java

```
public class SingletonPatternDemo {
   public static void main(String[] args) {
      SingleObject object = SingleObject.getInstance();
      object.showMessage();
   }
}
```

(3) Factory Pattern



Step 1.

* Shape.java

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

Step 2

* RoundedRectangle.java

```
public class RoundedRectangle implements Shape {

    public void draw(){
System.out.println("Inside RoundedRectangle::draw() meth
    }
}
```

* RoundedSquare.java

```
Public class RoundedSquare implements Shape{

    public void draw(){
system.out.println("Inside RoundedSquare::draw() method.");
    }
}
```

* Rectangle.java

```
public class Rectangle implements Shape{

    public void draw(){
system.out.println("Inside Rectangle::draw() method");
    }
}
```

Step 3

* AbstractFactory.java

```
public abstract class AbstractFactory {

    abstract shape getshape(String shapeType);

}
```

step 4.

* ShapeFactory.java

```
public class ShapeFactory extends Abstractfactory{

    public shape getshape(String shapeType)
    {
        if(shapeType.equalsIgnoreCase("RECTANGLE"))
        {
            return new Rectangle();
```

```
{
    return new Square();
    }
    return null;
    }
}
```

* RoundedShapeFactory.java

```
public class RoundedShapeFactory extends AbstractFactory
{
  public shape getShape(String shapeType)
  {
    if (shapeType.equalsIgnoreCase("RECTANGLE"))
    {
      return new RoundedRectangle();
    }
    else if (shapeType.equalsIgnoreCase("SQUARE"))
    {
      return new RoundedSquare();
    }
    return NULL;
  }
}
```

Step 5

* FactoryProducer.java

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

Step 6

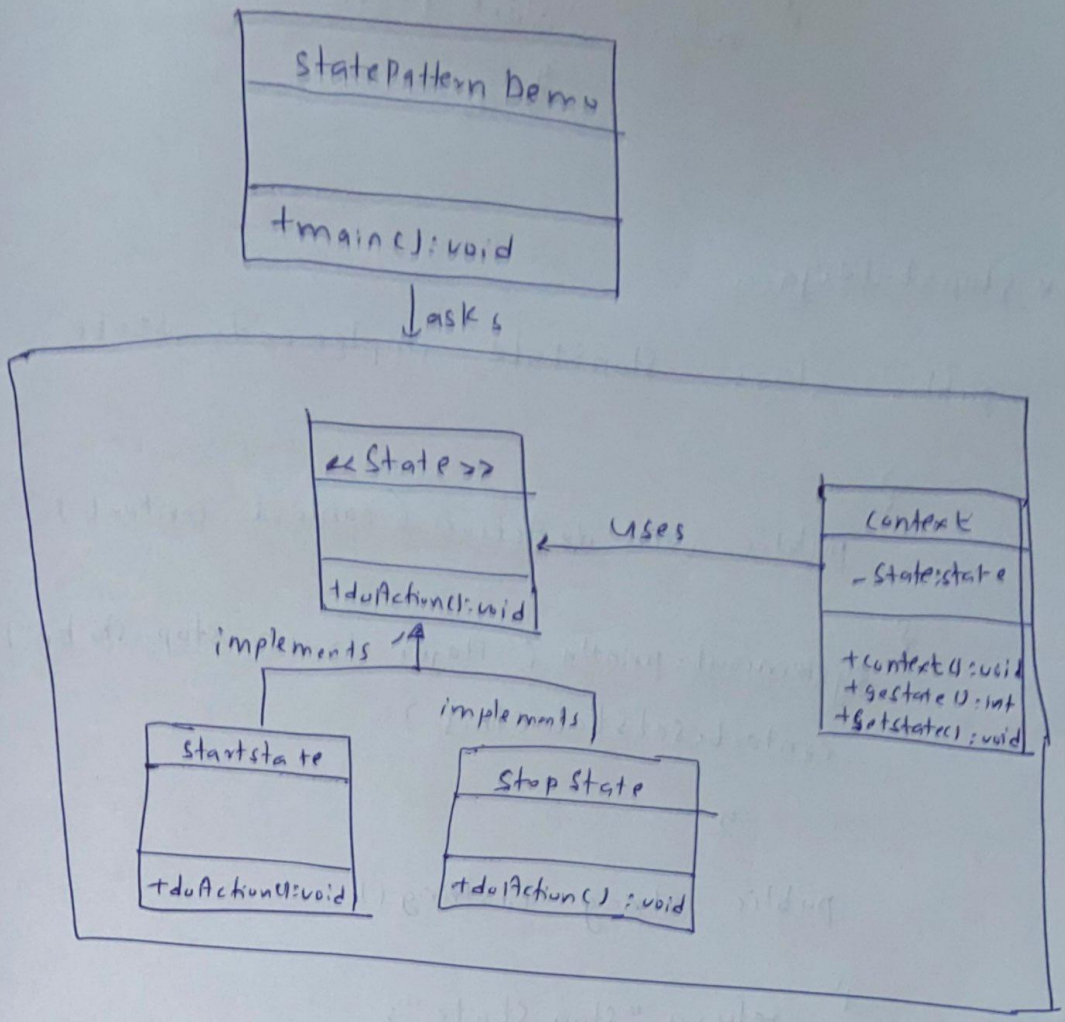
* AbstractFactoryPatternDemo.java

```
public class AbstractFactoryPatternDemo{
    public static void main (String[] args)
    {
AbstractFactory shapeFactory = FactoryProducer.getFactory(
Shape shape1 = shapefactory.getshape ("RECTANGLE");
shape1.draw ();
Shape shape2 = shapefactory.getshape ("SQUARE");
Shape2.draw ();
    }
}
```

Step 7

output

```
Inside Rectangle::draw() method
Inside Square:: draw() method
```



Step 1.

* State.java

```
public interface State
{ public void doAction(Context context);
}
```

Step 2

* Startstate.java

```
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";
}
}
```

* StopState.java

```
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";
    }
}
```

* Context.java

```
public class Context
{   private State state;
    public Context()
    {   state = null;
    }
public void setState(State state)
{
    this.state = state;
}
public State getState()
{   return state;
}
}
```

* StatePatternDemo.java

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

Step 5 :

Output

```
player is in start state
start state
player is in stop state
stop state
```