Lehrstuhl für Angewandte Softwaretechnik

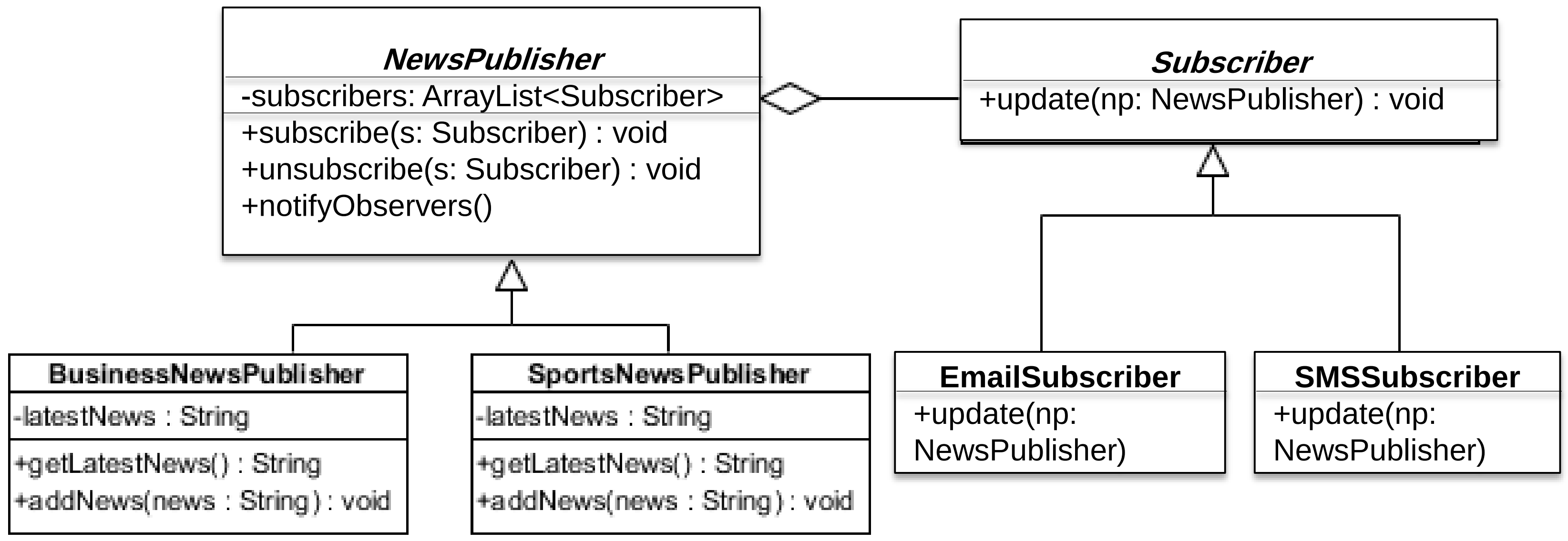
Technische Universität München

Homework #7

Bernd Brügge Ph.D. , Jan Knobloch, Emitzá Guzmán

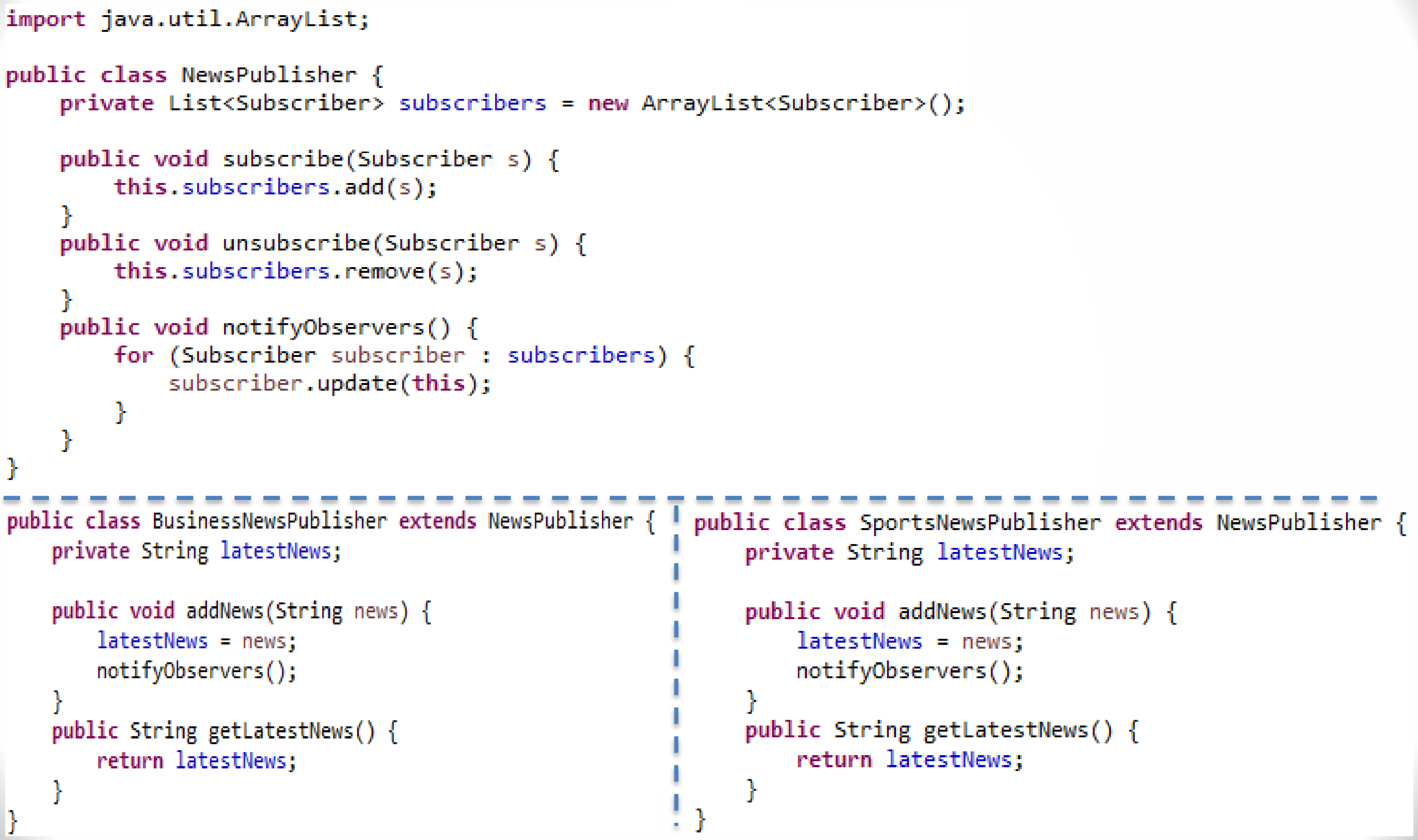
Questions

1. Name the pattern used in the following UML diagram


NewsPublisher
-subscribers: ArrayList<Subscriber>
+subscribe(s: Subscriber) : void
+unsubscribe(s: Subscriber) : void
+notifyObservers()
Subscriber
+update(np: NewsPublisher) : void
BusinessNewsPublisher
-latestNews : String
+getLatestNews() : String
+addNews(news : String) : void
SportsNewsPublisher
-latestNews : String
+getLatestNews() : String
+addNews(news : String) : void
EmailSubscriber
+update(np: NewsPublisher)
SMSSubscriber
+update(np: NewsPublisher)

Answer: Observer Pattern (Publish-Subscribe)

2. Implement the previous UML diagram in Java



```java
import java.util.ArrayList;

public class NewsPublisher {
    private List<Subscriber> subscribers = new ArrayList<Subscriber>();

    public void subscribe(Subscriber s) {
        this.subscribers.add(s);
    }
    public void unsubscribe(Subscriber s) {
        this.subscribers.remove(s);
    }
    public void notifyObservers() {
        for (Subscriber subscriber : subscribers) {
            subscriber.update(this);
        }
    }
}
```

```java
public class BusinessNewsPublisher extends NewsPublisher {
    private String latestNews;

    public void addNews(String news) {
        latestNews = news;
        notifyObservers();
    }
    public String getLatestNews() {
        return latestNews;
    }
}
```

```java
public class SportsNewsPublisher extends NewsPublisher {
    private String latestNews;

    public void addNews(String news) {
        latestNews = news;
        notifyObservers();
    }
    public String getLatestNews() {
        return latestNews;
    }
}
```

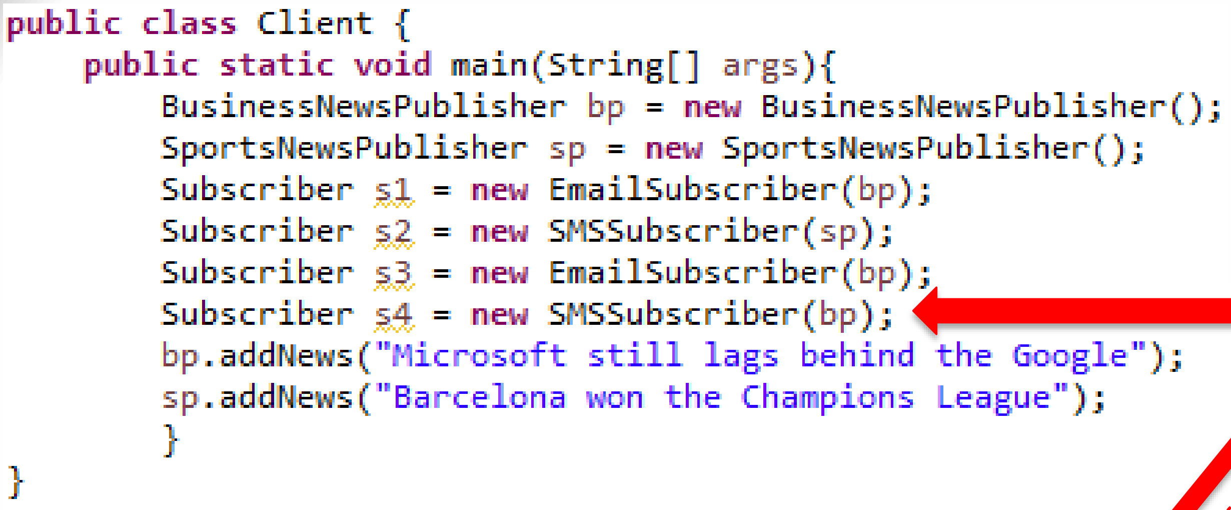
```java
public interface Subscriber {
    public void update(NewsPublisher np);
}
```

```java
public class SMSSubscriber implements Subscriber {
    String news;

    public SMSSubscriber(NewsPublisher np) {
        np.subscribe(this);
    }
    public void update(NewsPublisher np) {
        if (np instanceof SportsNewsPublisher) {
            SportsNewsPublisher snp = (SportsNewsPublisher) np;
            news = snp.getLatestNews();
            System.out.println("Latest news :" + news);
        }
    }
}
```

We decided that just Sport news should be published via SMS

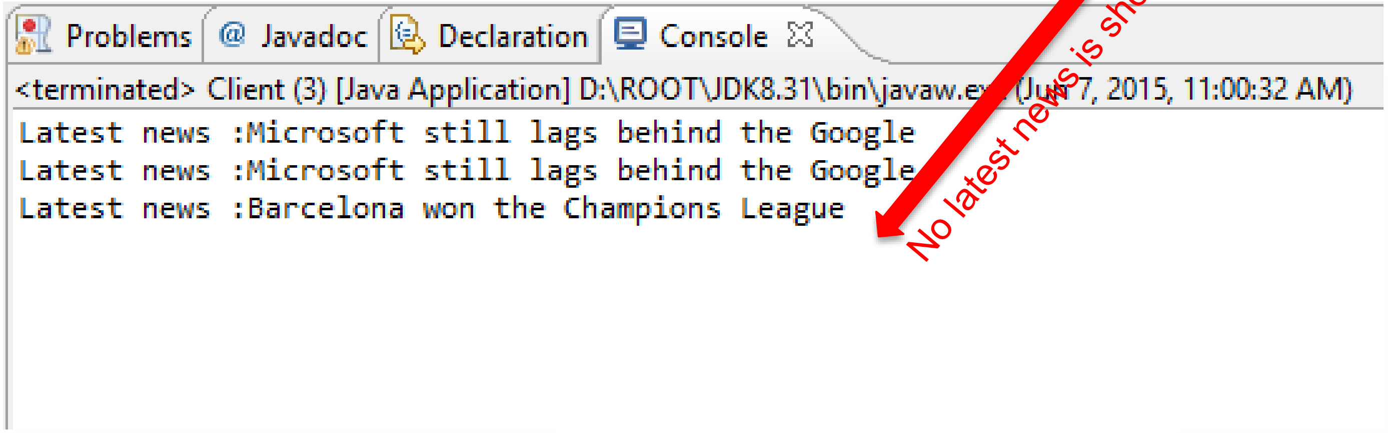
```java
public class EmailSubscriber implements Subscriber {
    String news;

    public EmailSubscriber(NewsPublisher np) {
        np.subscribe(this);
    }
    public void update(NewsPublisher np) {
        if (np instanceof BusinessNewsPublisher) {
            BusinessNewsPublisher bnp = (BusinessNewsPublisher) np;
            news = bnp.getLatestNews();
            System.out.println("Latest news :" + news);
        }
    }
}
```

… and just Business news should be published via Email



```
public class Client {
    public static void main(String[] args){
        BusinessNewsPublisher bp = new BusinessNewsPublisher();
        SportsNewsPublisher sp = new SportsNewsPublisher();
        Subscriber s1 = new EmailSubscriber(bp);
        Subscriber s2 = new SMSSubscriber(sp);
        Subscriber s3 = new EmailSubscriber(bp);
        Subscriber s4 = new SMSSubscriber(bp);
        bp.addNews("Microsoft still lags behind the Google");
        sp.addNews("Barcelona won the Champions League");
        }
}
```

Since s4 is a SMS Subscriber which subscribed to Business NewsPublisher, the latest news is not shown for that



```
<terminated> Client (3) [Java Application] D:\ROOT\JDK8.31\bin\javaw.exe (Jun 7, 2015, 11:00:32 AM)
Latest news :Microsoft still lags behind the Google
Latest news :Microsoft still lags behind the Google
Latest news :Barcelona won the Champions League
```

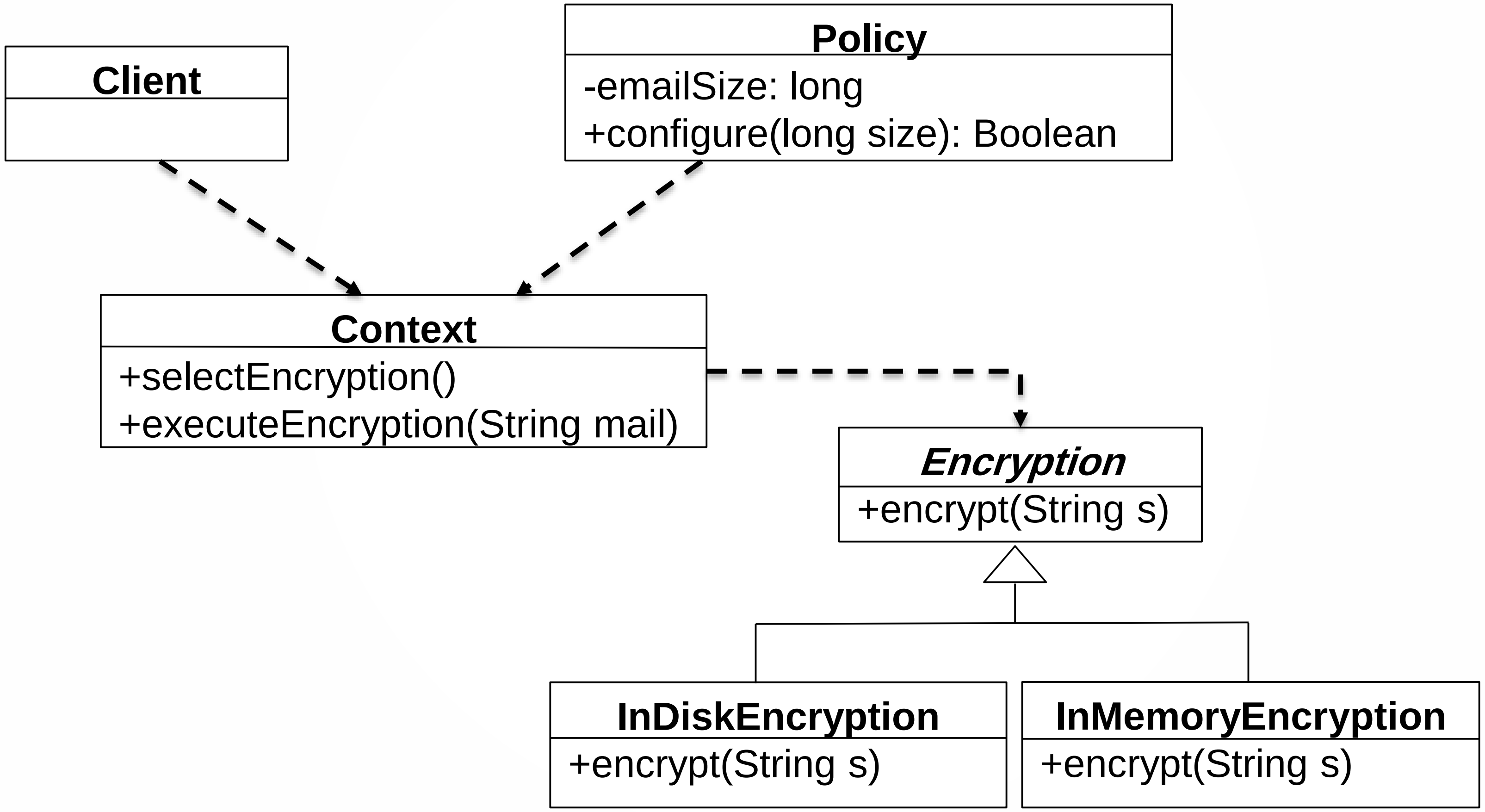
No latest news is shown here for s4

You are currently implementing a system for encrypting emails and are planning to use two different encryption mechanisms: *InMemoryEncryption* and *InDiskEncryption*. *InMemoryEncryption* will be used when the email size is below 1 GB and the file can be read and encrypted in memory. On the other hand, *InDiskEncryption* will be used when the email size is over 1GB and the encryption needs to take place in batches and part of the encryption results will be stored in disk.

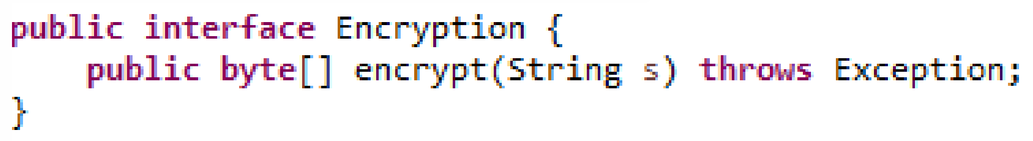
3. Which pattern would you use to solve this problem?

   Answer: Strategy pattern

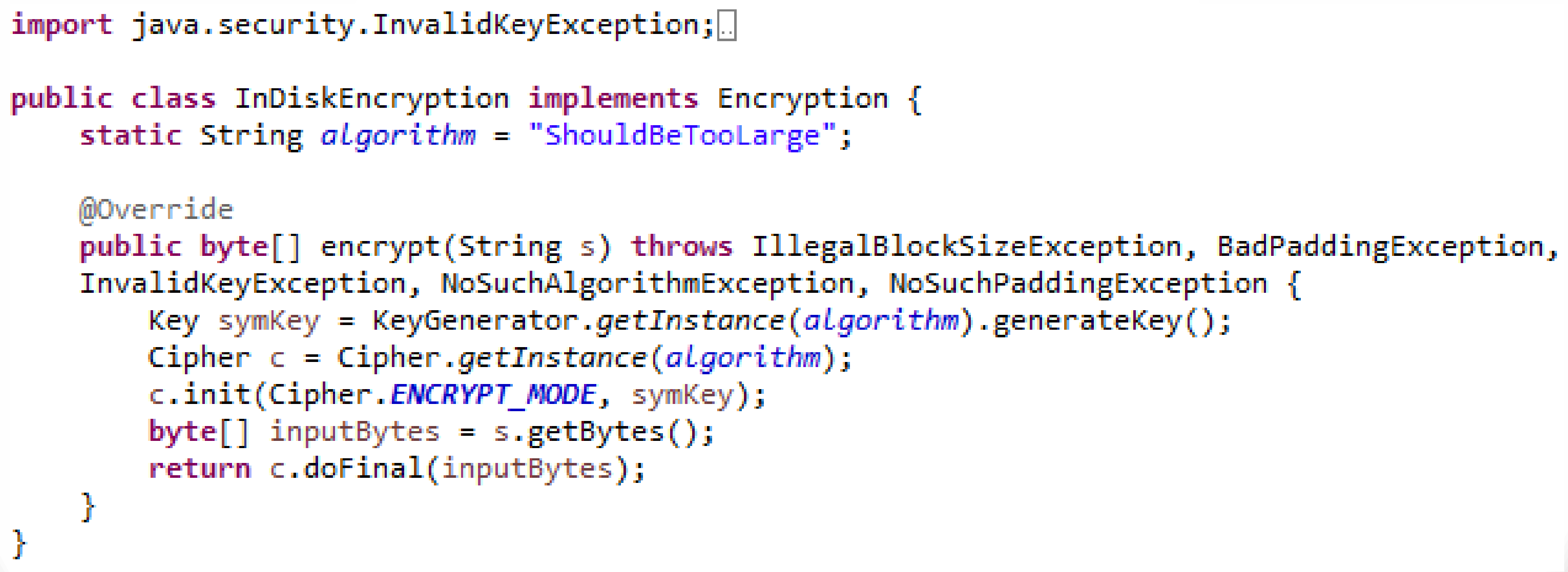
4. Model how the chosen pattern could be used to solve the problem. Use a UML class diagram for your answer.



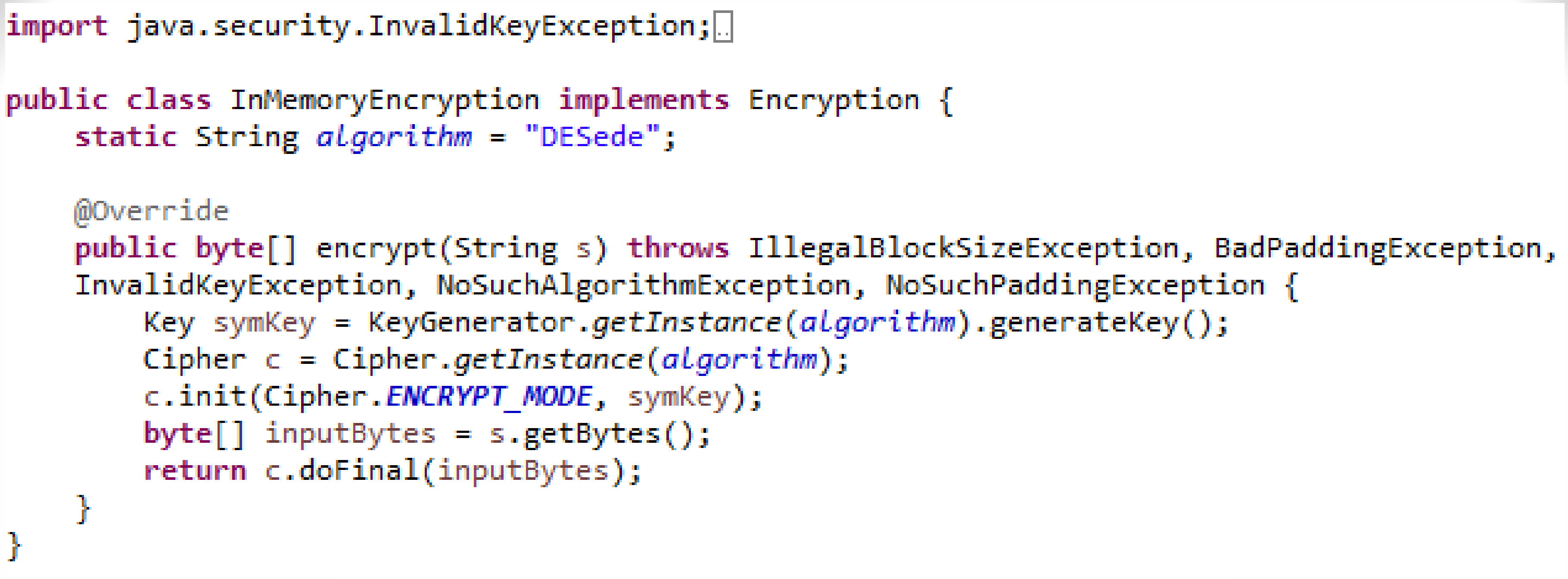
5. Implement the UML class diagram in Java.



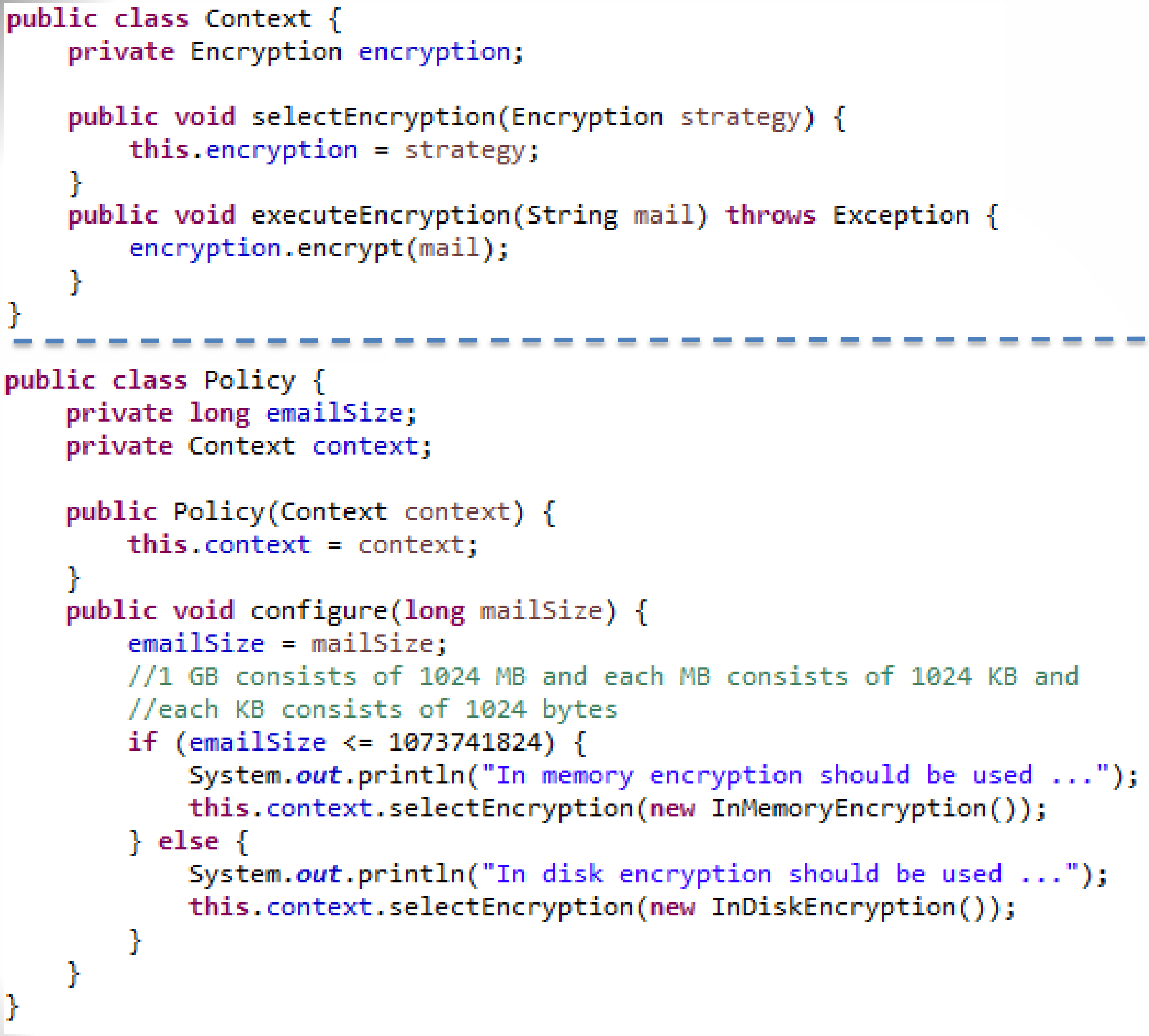
```java
public interface Encryption {
    public byte[] encrypt(String s) throws Exception;
}
```



```java
import java.security.InvalidKeyException;..

public class InDiskEncryption implements Encryption {
    static String algorithm = "ShouldBeTooLarge";

    @Override
    public byte[] encrypt(String s) throws IllegalBlockSizeException, BadPaddingException,
    InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException {
        Key symKey = KeyGenerator.getInstance(algorithm).generateKey();
        Cipher c = Cipher.getInstance(algorithm);
        c.init(Cipher.ENCRYPT_MODE, symKey);
        byte[] inputBytes = s.getBytes();
        return c.doFinal(inputBytes);
    }
}
```

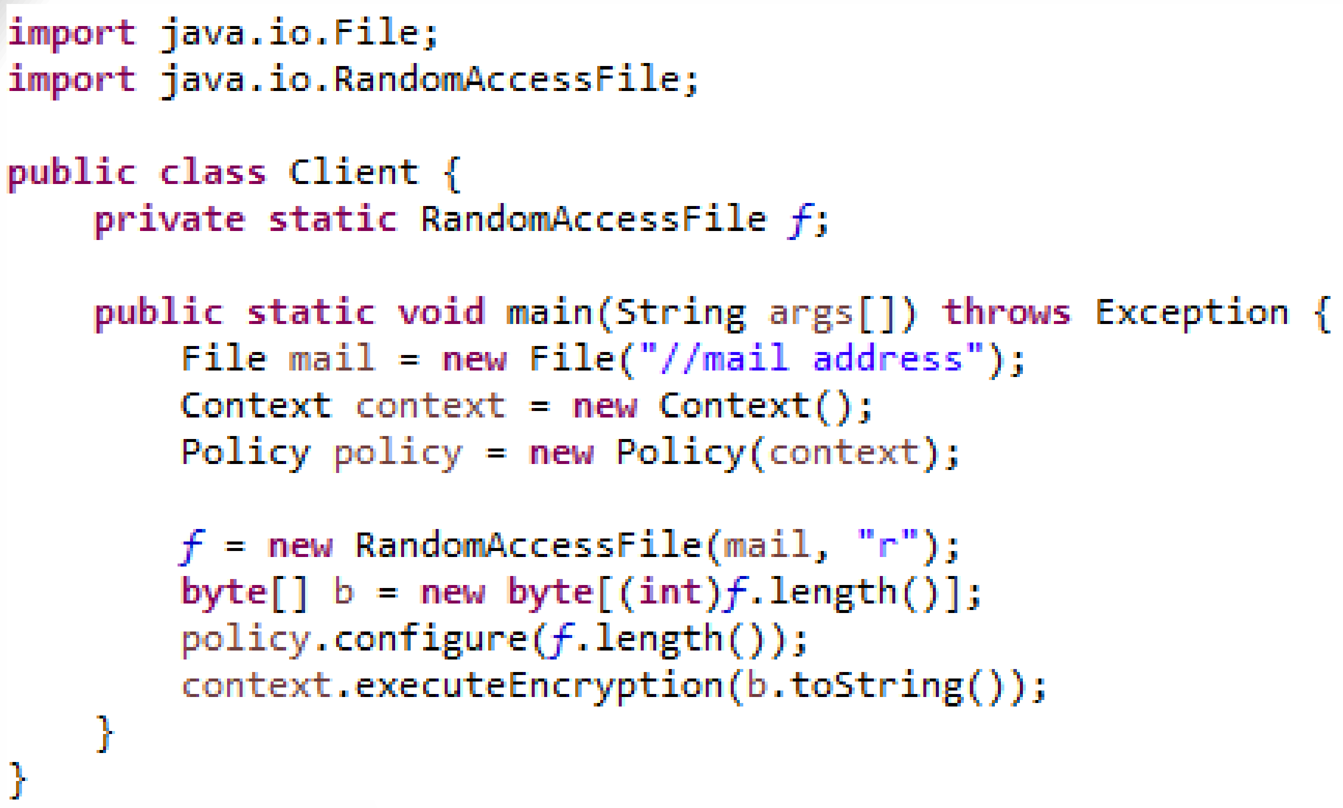


```java
import java.security.InvalidKeyException;

public class InMemoryEncryption implements Encryption {
    static String algorithm = "DESede";

    @Override
    public byte[] encrypt(String s) throws IllegalBlockSizeException, BadPaddingException,
    InvalidKeyException, NoSuchAlgorithmException, NoSuchPaddingException {
        Key symKey = KeyGenerator.getInstance(algorithm).generateKey();
        Cipher c = Cipher.getInstance(algorithm);
        c.init(Cipher.ENCRYPT_MODE, symKey);
        byte[] inputBytes = s.getBytes();
        return c.doFinal(inputBytes);
    }
}
```



```java
public class Context {
    private Encryption encryption;

    public void selectEncryption(Encryption strategy) {
        this.encryption = strategy;
    }
    public void executeEncryption(String mail) throws Exception {
        encryption.encrypt(mail);
    }
}
```

```java
public class Policy {
    private long emailSize;
    private Context context;

    public Policy(Context context) {
        this.context = context;
    }
    public void configure(long mailSize) {
        emailSize = mailSize;
        //1 GB consists of 1024 MB and each MB consists of 1024 KB and
        //each KB consists of 1024 bytes
        if (emailSize <= 1073741824) {
            System.out.println("In memory encryption should be used ...");
            this.context.selectEncryption(new InMemoryEncryption());
        } else {
            System.out.println("In disk encryption should be used ...");
            this.context.selectEncryption(new InDiskEncryption());
        }
    }
}
```



```java
import java.io.File;
import java.io.RandomAccessFile;

public class Client {
    private static RandomAccessFile f;

    public static void main(String args[]) throws Exception {
        File mail = new File("//mail address");
        Context context = new Context();
        Policy policy = new Policy(context);

        f = new RandomAccessFile(mail, "r");
        byte[] b = new byte[(int)f.length()];
        policy.configure(f.length());
        context.executeEncryption(b.toString());
    }
}
```

5. Describe a concrete example where you would use the state pattern.

In general, the systems which are always running in different definite states can be modeled with state pattern.

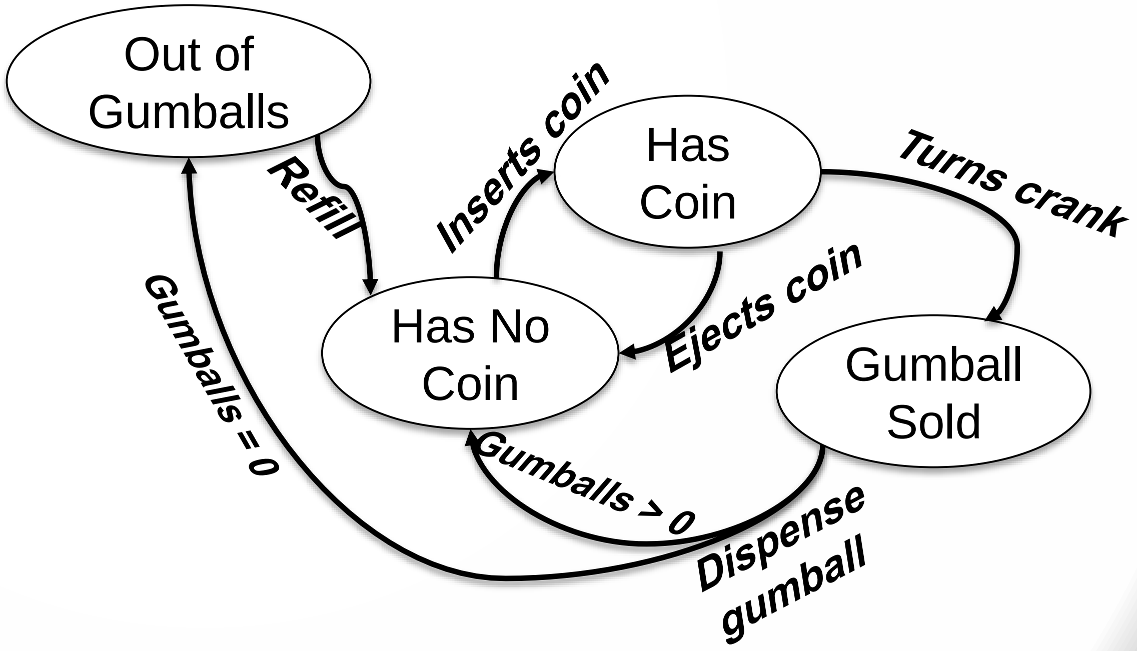
For instance, a gumball machine (a machine that you insert your coin and gives you a gumball in exchange) consists of these four states:

HasNoCoin

HasCoin

GumballSold

OutOfGumballs


Out of Gumballs
Has Coin
Has No Coin
Gumball Sold
Refill
Inserts coin
Turns crank
Ejects coin
Gumballs = 0
Gumballs > 0
Dispense gumball