**Statement of Work:**

Nowadays, internet security has become a very popular subject. If we explain this issue a bit, the security of personal data is very important for everyone. Nobody wants to share their private data by third parties. These people may use the other people's personal information in illegal affairs and/or they can restrict the free will of the people. People can be threatened with pictures or messages, or it can harm people's dignity. Encryption of sent messages is also included in this topic. It is necessary to make an application that can encrypt every word of sentence.

Thus, chats becomes more secure and reliable. In this way, people's perspectives on applications can change. Knowing that the texts written by people will not be stolen gives people confidence.

In this direction, I want to make an application that encrypts messages.

**Design Pattern:**

- **Factory pattern** introduces **loose coupling** between classes which is the most important principle one should consider and apply while designing the application architecture. Loose coupling can be introduced in application architecture by programming against abstract entities rather than concrete implementations. This not only makes our architecture more flexible but also less fragile.

- The creation of an object precludes its reuse without significant **duplication of code**.

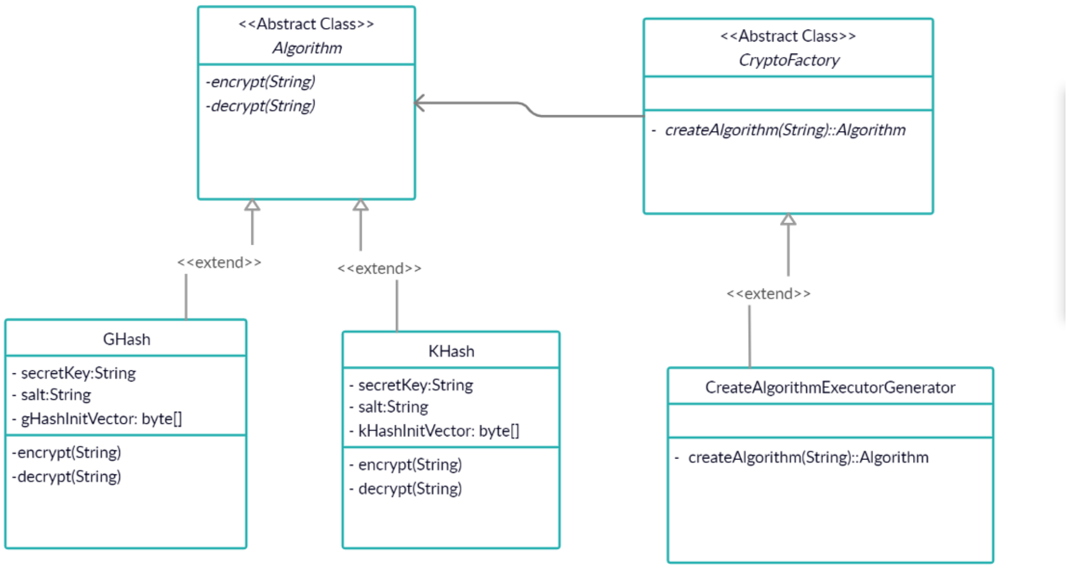
- The creation of an object requires access to information or resources that should not be contained within the **composing class**.

- The lifetime management of the generated objects must be centralized to ensure a consistent behavior within the application.

- **Factory pattern** is most suitable where there is some **complex object creation** steps are involved. To ensure that these steps are centralized and not exposed to composing classes, factory pattern should be used.

In the project that I mentioned above, The application have complex object creation and. So it's good candidate for **Factory Pattern.**

**UML:**



**Research**:

Let's use the example of Calculate Electricity Bill : A Real World Example of Factory Method

- We are going to create a Plan abstract class and concrete classes that extends the Plan abstract class. A factory class GetPlanFactory is defined as a next step.
- GenerateBill class will use GetPlanFactory to get a Plan object. It will pass information (DOMESTICPLAN / COMMERCIALPLAN / INSTITUTIONALPLAN) to GetPalnFactory to get the type of object it needs.

**Step 1:** Create a Plan abstract class.

```
import java.io.*;
abstract class Plan{
        protected double rate;
        abstract void getRate();
public void calculateBill(int units){
            System.out.println(units*rate);
        }
}//end of Plan class.
```

**Step 2:** Create the concrete classes that extends Plan abstract class.

```
class  DomesticPlan extends Plan{
       //@override
        public void getRate(){
           rate=3.50;
       }
  }//end of DomesticPlan class.
class  CommercialPlan extends Plan{
  //@override
   public void getRate(){
      rate=7.50;
  }
/end of CommercialPlan class.
class  InstitutionalPlan extends Plan{
  //@override
   public void getRate(){
      rate=5.50;
  }
/end of InstitutionalPlan class.
```

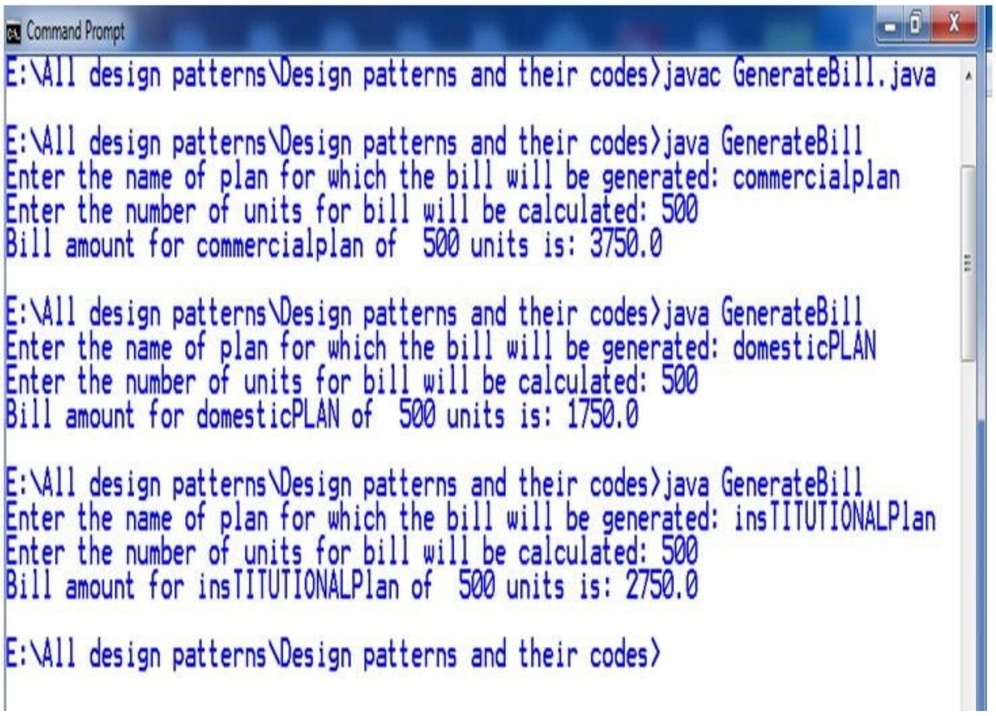
**Step 3:** Create a GetPlanFactory to generate object of concrete classes based on given information..

```
class GetPlanFactory{

  //use getPlan method to get object of type Plan
      public Plan getPlan(String planType){
          if(planType == null){
           return null;
          }
        if(planType.equalsIgnoreCase("DOMESTICPLAN")) {
             return new DomesticPlan();
           }
         else if(planType.equalsIgnoreCase("COMMERCIALPLAN")){
             return new CommercialPlan();
         }
        else if(planType.equalsIgnoreCase("INSTITUTIONALPLAN")) {
            return new InstitutionalPlan();
        }
      return null;
    }
}//end of GetPlanFactory class.
```

**Step 4:** Generate Bill by using the GetPlanFactory to get the object of concrete classes by passing an information such as type of plan DOMESTICPLAN or COMMERCIALPLAN or INSTITUTIONALPLAN.

```
import java.io.*;
class GenerateBill{
  public static void main(String args[])throws IOException{
    GetPlanFactory planFactory = new GetPlanFactory();

    System.out.print("Enter the name of plan for which the bill will be generated: "
);
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in)
);

    String planName=br.readLine();
    System.out.print("Enter the number of units for bill will be calculated: ");
    int units=Integer.parseInt(br.readLine());

    Plan p = planFactory.getPlan(planName);
    //call getRate() method and calculateBill()method of DomesticPaln.

     System.out.print("Bill amount for "+planName+" of  "+units+" units is: ");
        p.getRate();
        p.calculateBill(units);
         }
  }//end of GenerateBill class.
```

**Output:**



```
E:\All design patterns\Design patterns and their codes>javac GenerateBill.java

E:\All design patterns\Design patterns and their codes>java GenerateBill
Enter the name of plan for which the bill will be generated: commercialplan
Enter the number of units for bill will be calculated: 500
Bill amount for commercialplan of  500 units is: 3750.0

E:\All design patterns\Design patterns and their codes>java GenerateBill
Enter the name of plan for which the bill will be generated: domesticPLAN
Enter the number of units for bill will be calculated: 500
Bill amount for domesticPLAN of  500 units is: 1750.0

E:\All design patterns\Design patterns and their codes>java GenerateBill
Enter the name of plan for which the bill will be generated: insTITUTIONALPlan
Enter the number of units for bill will be calculated: 500
Bill amount for insTITUTIONALPlan of  500 units is: 2750.0

E:\All design patterns\Design patterns and their codes>
```

## Code Implementation of My Application:

### Algorithm.java

```
package ceng.anadolu.bim492.Crypto.Algorithm;

public abstract class Algorithm {
   //First I defined the contract for each of the cryptography algorithm
   public abstract String encrypt(String str);

   public abstract String decrypt(String str);
}
```

**GHash.java**

```
package ceng.anadolu.bim492.Crypto.Algorithm;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.spec.KeySpec;
import java.util.Base64;

public class GHash extends Algorithm {
  //I defined an Algorithm call GHash, which is a fictitious cryptography algorithm
that i devised
  private static String secretKey = "GHash!!!!";
  private static String salt = "GHash!!!!";
  private byte[] gHashInitVector = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  private IvParameterSpec gHashInitVectorHash = new
IvParameterSpec(gHashInitVector);

  @Override
  public String encrypt(String str) {
    System.out.println("Encrypting String with GHash");
    try {
      SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
      KeySpec spec = new PBEKeySpec(secretKey.toCharArray(), salt.getBytes(),
65536, 256);
      SecretKey tmp = factory.generateSecret(spec);
      SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, gHashInitVectorHash);
      return
Base64.getEncoder().encodeToString(cipher.doFinal(str.getBytes("UTF-8")));
    } catch (Exception e) {
      System.out.println("Error while encrypting with GHash: " +
e.toString());
    }
    return null;
  }

  @Override
  public String decrypt(String str) {
    System.out.println("Decrypting String with GHash");
    try {
      SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
      KeySpec spec = new PBEKeySpec(secretKey.toCharArray(), salt.getBytes(),
65536, 256);
            SecretKey tmp = factory.generateSecret(spec);
            SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey, gHashInitVectorHash);
            return new String(cipher.doFinal(Base64.getDecoder().decode(str)));
        } catch (Exception e) {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }
}
```

**KHash.java**

```
package ceng.anadolu.bim492.Crypto.Algorithm;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.spec.KeySpec;
import java.util.Base64;

    public class KHash extends Algorithm {
        //Next I created yet another fictitious cryptographic algorithm call KHash
        private static String secretKey = "KHash!!!!";
        private static String salt = "KHash!!!!";
        private byte[] kHashInitVector = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
        private IvParameterSpec kHashInitVectorHash = new
IvParameterSpec(kHashInitVector);

        @Override
        public String encrypt(String str) {
            System.out.println("Encrypting String with KHash");
            try {
                SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
                KeySpec spec = new PBEKeySpec(secretKey.toCharArray(), salt.getBytes(),
65536, 256);
                SecretKey tmp = factory.generateSecret(spec);
                SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

                Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
                cipher.init(Cipher.ENCRYPT_MODE, secretKey, kHashInitVectorHash);
            return
Base64.getEncoder().encodeToString(cipher.doFinal(str.getBytes("UTF-8")));
        } catch (Exception e) {
            System.out.println("Error while encrypting with KHash: " +
e.toString());
        }
        return null;
    }

    @Override
    public String decrypt(String str) {
        System.out.println("Decrypting String with KHash");
        try {
            SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
            KeySpec spec = new PBEKeySpec(secretKey.toCharArray(), salt.getBytes(),
65536, 256);
            SecretKey tmp = factory.generateSecret(spec);
            SecretKeySpec secretKey = new SecretKeySpec(tmp.getEncoded(), "AES");

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
            cipher.init(Cipher.DECRYPT_MODE, secretKey, kHashInitVectorHash);
            return new String(cipher.doFinal(Base64.getDecoder().decode(str)));
        } catch (Exception e) {
            System.out.println("Error while decrypting: " + e.toString());
        }
        return null;
    }

}
```

**CreateAlgorithmExecutorGenerator.java**

```
package ceng.anadolu.bim492.Crypto;

import ceng.anadolu.bim492.Crypto.Algorithm.Algorithm;
import ceng.anadolu.bim492.Crypto.Algorithm.GHash;
import ceng.anadolu.bim492.Crypto.Algorithm.KHash;

public class CreateAlgorithmExecutorGenerator extends CryptoFactory {
  //I extended the contract (abstract class) defined in CryptoFactory class in
  //CreateAlgorithmExecutorGenerator class to implement the createAlgorithm()
function
  // to generate object for the crypto algorithm as per the type passed as params.
  public Algorithm createAlgorithm(String type) {
    switch (type) {
      case "KHASH":
        return new KHash();
      case "GHASH":
            return new GHash();
          default:
            return new GHash();
        }
    }
}
```

**CryptoFactory.java**

```
package ceng.anadolu.bim492.Crypto;

import ceng.anadolu.bim492.Crypto.Algorithm.Algorithm;

abstract class CryptoFactory {
   //I created the factory abstract class called CryptoFactory,
   // which exposes the contract required to create object of the individual
algorithm.
   abstract Algorithm createAlgorithm(String type);
}
```

**App.java**

```
package ceng.anadolu.bim492;

import ceng.anadolu.bim492.Crypto.Algorithm.Algorithm;
import ceng.anadolu.bim492.Crypto.CreateAlgorithmExecutorGenerator;

public class App {
   public static void main(String[] args) {
       System.out.println("Test project");
       //I showed the driver class which shows the encrypt/decrypt algorithms for
each of the existing cryptography algorithm in action.

       CreateAlgorithmExecutorGenerator cl = new
CreateAlgorithmExecutorGenerator();
       System.out.println("Encrypting using GHash");
       Algorithm gHashAlgo = cl.createAlgorithm("GHASH");
       String SampleStringForGHash = "Irem";
       System.out.println("Sample String before Encryption :: " +
SampleStringForGHash);
       String encryptedGHash = gHashAlgo.encrypt(SampleStringForGHash);
       System.out.println("Encrypted String after encryption with GHash :: " +
encryptedGHash);
       String decryptedGHash = gHashAlgo.decrypt(encryptedGHash);
       System.out.println("Decrypted String after encryption with GHash :: " +
decryptedGHash);
```

```
        System.out.println("Encrypting using KHash");
        Algorithm kHashAlgo = cl.createAlgorithm("KHASH");
        String SampleStringForKHash = "Caliskan";
        System.out.println("Sample String before Encryption :: " +
SampleStringForKHash);
        String encryptedKHash = kHashAlgo.encrypt(SampleStringForKHash);
        System.out.println("Encrypted String after encryption with KHash :: " +
encryptedKHash);
        String decryptedKHash = kHashAlgo.decrypt(encryptedKHash);
        System.out.println("Decrypted String after encryption with KHash :: " +
decryptedKHash);
    }
}
```

**Output:**

```
C:\Users\ASUS\Desktop\bim492-19-20-p2-iremcaliskan-master>java -jar
target\project.jar

Test project

Encrypting using GHash

Sample String before Encryption :: Irem

Encrypting String with GHash

Encrypted String after encryption with GHash :: Ue2OnwcSyQ+rWfLdWJOx8A==

Decrypting String with GHash

Decrypted String after encryption with GHash :: Irem

Encrypting using KHash

Sample String before Encryption :: Caliskan

Encrypting String with KHash

Encrypted String after encryption with KHash :: RgX2EVIZOW7u1XfOFh3M+g==

Decrypting String with KHash

Decrypted String after encryption with KHash :: Caliskan
```