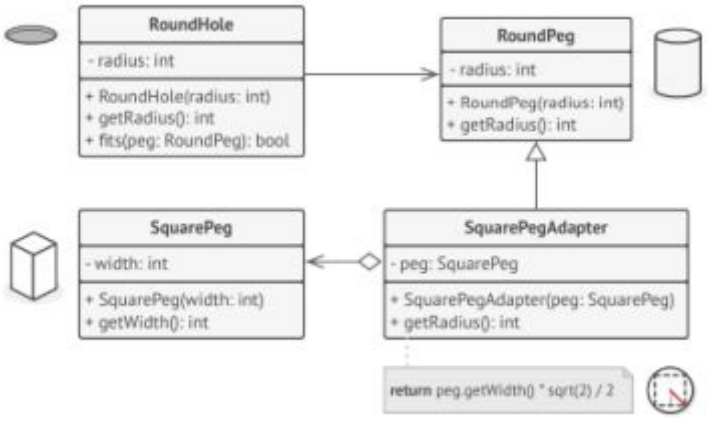
Laboratorium 6
Adrian Wójciak, Lena Obertyńska

1. **Adapter**



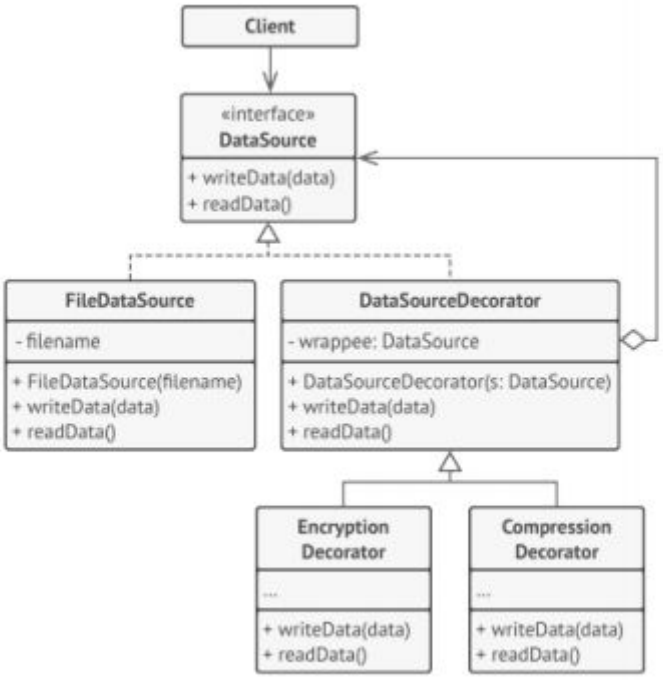
Na początku mieliśmy dwie inkotabilne klasy które nie możemy zmieniać. Wykorzystaliśmy wzorzec projektowy Adapter.

```
public class SquarePeg {
    private int width;

    public SquarePeg(int width){
        this.width = width;
    }

    public int getWidth() {
        return this.width;
    }
}

public class RoundPeg {
    private int radius;

    public RoundPeg(int radius){
        this.radius=radius;
    }

    public int getRadius() {
        return this.radius;
    }

}
```

```
public class SquarePegAdapter extends RoundPeg {
   private SquarePeg peg;

   public SquarePegAdapter(SquarePeg peg){
       super(peg.getWidth());
       this.peg = peg;
   }

   public int getRadius() {
       return (int)(this.peg.getWidth()*sqrt(2)/2);
   }

}

public class Main {

   public static void main(String[] args) {
       RoundHole hole = new RoundHole (5);
       RoundPeg rpeg = new RoundPeg (5);

       hole.fits ( rpeg ); // true

       SquarePeg small_sqpeg = new SquarePeg (5);
       SquarePeg large_sqpeg = new SquarePeg (10);

       SquarePegAdapter small_sqpeg_adapter = new
SquarePegAdapter ( small_sqpeg );
       SquarePegAdapter large_sqpeg_adapter = new
SquarePegAdapter ( large_sqpeg );

       System.out.println(hole.fits ( small_sqpeg_adapter
)); // true
       System.out.println(hole.fits ( large_sqpeg_adapter
)); // false
   }
}
```

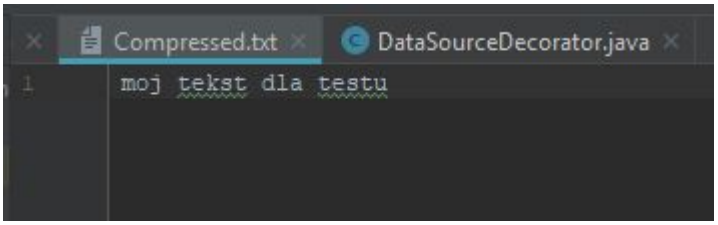
```
true
false

Process finished with exit code 0
```

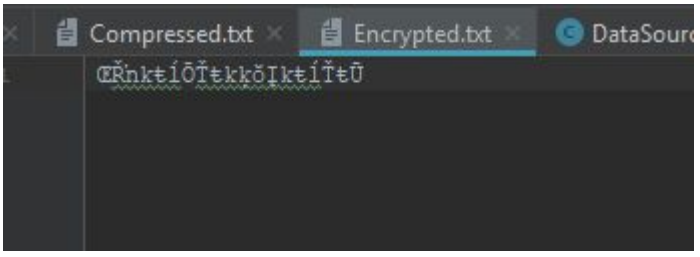
2. Decorator



W danym zadaniu 2 zaimplementowaliśmy powyższy schemat dla kompresji danych.

```
"C:\Program Files\Java\jdk1.8.0_251\bin\java.exe" .
moj tekst dla testu

moj tekst dla testu
moj tekst dla testu


Process finished with exit code 0
```





```
public class Main {
   private static final String text = "moj tekst dla
testu";
   public static void main(String[] args) throws
IOException, IllegalBlockSizeException,
NoSuchPaddingException, BadPaddingException,
NoSuchAlgorithmException, InvalidKeyException {
       DataSource dataSource = new
FileDataSource("NormalWrite.txt");
       dataSource.writeData(text);
       System.out.println(dataSource.readData());

       FileDataSource dataSource1 = new
FileDataSource("Encrypted.txt");
       DataSource encryptionDecorator = new
EncryptionDecorator(dataSource1);
       encryptionDecorator.writeData(text);
       System.out.println(
encryptionDecorator.readData());

       DataSource dataSource2 = new
FileDataSource("Compressed.txt");
       DataSource compressionDecorator = new
CompressionDecorator(dataSource2);
       compressionDecorator.writeData(text);
       System.out.println(
compressionDecorator.readData());
   }
}


public interface DataSource {

   public void writeData(String data) throws IOException,
NoSuchAlgorithmException, BadPaddingException,
IllegalBlockSizeException, NoSuchPaddingException,
InvalidKeyException;
   public String readData() throws  IOException,
NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException, BadPaddingException,
IllegalBlockSizeException;
}
```

```
public class FileDataSource implements  DataSource {
   private final String filename;

   public FileDataSource(String filename){
       this.filename=filename;
   }

   public String getFilename() {
       return filename;
   }

   @Override
   public void writeData(String data) throws IOException,
NoSuchAlgorithmException, BadPaddingException,
IllegalBlockSizeException, NoSuchPaddingException {
       FileWriter fw = new FileWriter(filename);

       fw.write(data);
       fw.close();
   }

   @Override
   public String readData() throws IOException,
NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException, BadPaddingException,
IllegalBlockSizeException {
       FileReader fr = new FileReader(filename);
       BufferedReader br = new BufferedReader(fr);
       StringBuilder result = new StringBuilder();
       String line = null;
       while((line = br.readLine()) != null){
               result.append(line);
               result.append("\n");
       }
       br.close();
       return result.toString();
   }
}
```

```
public class DataSourceDecorator implements DataSource {

   DataSource wrapper;

   public DataSourceDecorator( DataSource wrapper){
       this.wrapper=wrapper;
   }

   @Override
   public void writeData(String data) throws IOException,
NoSuchAlgorithmException, BadPaddingException,
IllegalBlockSizeException, NoSuchPaddingException,
InvalidKeyException {

   }

   @Override
   public String readData() throws IOException,
NoSuchAlgorithmException, NoSuchPaddingException,
InvalidKeyException, BadPaddingException,
IllegalBlockSizeException {
       return null;
   }
}
```

```
public class EncryptionDecorator extends DataSourceDecorator {

   public EncryptionDecorator(DataSource wrapper) {
       super(wrapper);
   }

   private final int multiplier = 3;
   private final int adder = 11;

   @Override
   public void writeData(String data) throws
BadPaddingException, NoSuchAlgorithmException, IOException,
IllegalBlockSizeException, NoSuchPaddingException,
InvalidKeyException {
       wrapper.writeData(
               data.chars()
                       .mapToObj(ch -> (char) ch)
                       .map(character -> character *
multiplier + adder)
                       .map(integer ->(char)((int)integer))

.collect(StringBuilder::new,StringBuilder::appendCodePoint,Str
ingBuilder::append)
                       .toString());
   }
   @Override
   public String readData() throws NoSuchPaddingException,
NoSuchAlgorithmException, IOException, BadPaddingException,
IllegalBlockSizeException, InvalidKeyException {
       return wrapper
               .readData()
               .chars()
               .mapToObj(ch -> (char) ch)
               .map(character -> (character-adder) /
multiplier )
               .map(integer ->(char)((int)integer))

.collect(StringBuilder::new,StringBuilder::appendCodePoint,Str
ingBuilder::append)
               .toString();
   }
}
```

```
public class CompressionDecorator extends DataSourceDecorator
{
   public CompressionDecorator(DataSource wrapper) {
       super(wrapper);
   }

   @Override
   public void writeData(String data) throws
BadPaddingException, NoSuchAlgorithmException,
IllegalBlockSizeException, IOException,
NoSuchPaddingException, InvalidKeyException {

       StringBuilder compressed = new StringBuilder();

       for (int i = 0; i < data.length(); i++) {

           char currLetter = data.charAt(i);
           int j = i + 1;
           for (; j < data.length() && data.charAt(j) ==
currLetter; j++) {
           }

           if (j - i > 2) {
               compressed.append(currLetter).append(j - i);
               i = j - 1;
           } else {
               compressed.append(currLetter);
           }
       }
       wrapper.writeData(compressed.toString());
   }

   @Override
   public String readData() throws NoSuchPaddingException,
IOException, NoSuchAlgorithmException,
IllegalBlockSizeException, BadPaddingException,
InvalidKeyException {
       String readData = wrapper.readData();
       StringBuilder uncompressed = new StringBuilder();

       for (int i = 0; i < readData.length(); i++) {
```

```
            char currLetter = readData.charAt(i);
            if (Character.isDigit(currLetter)) {
                int nrOFOcc =
Integer.parseInt(String.valueOf(currLetter));

                char[] repeat = new char[nrOFOcc - 1];
                Arrays.fill(repeat, readData.charAt(i - 1));
                uncompressed.append(new String(repeat));
            } else {
                uncompressed.append(currLetter);
            }
        }
        return uncompressed.toString();
    }
}
```