Nazar Kordiumov

Laboratorium 6

Wzorce projektowe 2

1. Adapter
    a. Stworzyłem według UML trzy klasy SquarePeg, RoundPeg RoundHole.

```java
public class RoundPeg {

    private double radius;

    public RoundPeg() {
    }

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

    public double getRadius() {
        return radius;
    }
}
```

```java
public class RoundHole {
    private final double radius;

    public RoundHole(double radius) {
        this.radius = radius;
    }

    public double getRadius() {
        return radius;
    }

    public boolean fits(RoundPeg roundPeg) {
        return this.getRadius() >= roundPeg.getRadius();
    }
}
```

```java
public class SquarePeg {
    private final double width;

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

    public double getWidth() {
        return width;
    }
}
```

    b. Dwie z nich są kompatybilne: RoundPeg o RoundHole

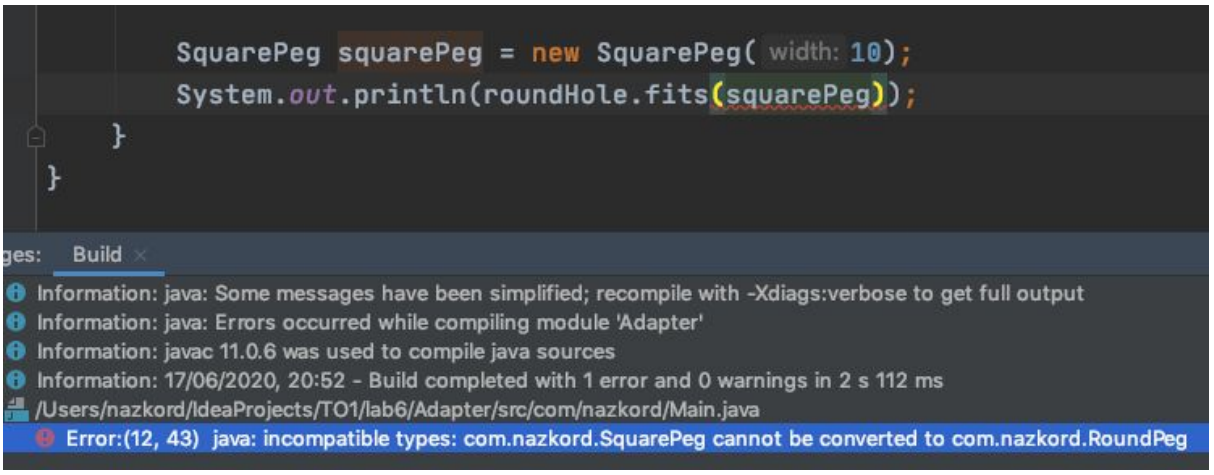
```java
public class Main {

    public static void main(String[] args) {
    // write your code here
        RoundHole roundHole = new RoundHole( radius: 10);
        RoundPeg roundPeg = new RoundPeg( radius: 10);
        System.out.println(roundHole.fits(roundPeg));
    }
}
```

```
Main
/Library/Java/JavaVirtualMachines/jdk-11.0.6.jdk/Content
true

Process finished with exit code 0
```

c. Natomiast klasy RoundPeg i SquarePeg nie są ze sobą kompatybilne



d. Stworzyłem więc adapter SquarePegAdapter, który rozwiązuje powyższy problem

```
public class SquarePegAdapter extends RoundPeg {

    private SquarePeg squarePeg;

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

    @Override
    public double getRadius() {
        return squarePeg.getWidth() * Math.sqrt(2) / 2;
    }
}
```

e. Ostatecznie (wynik to: true, false)

```
        RoundHole roundHole = new RoundHole( radius: 10);

        SquarePegAdapter squarePegAdapter = new SquarePegAdapter(new SquarePeg( width: 10));
        System.out.println(roundHole.fits(squarePegAdapter));

        SquarePegAdapter squarePegAdapter2 = new SquarePegAdapter(new SquarePeg( width: 20));
        System.out.println(roundHole.fits(squarePegAdapter2));
    }
```

2. Decorator
    a. Na początku stworzyłem DataSource, który będzie wspólny dla obiektu wrapowanego i klasy dekorującej

```
public interface DataSource {
    void writeData(String data);
    String readData();
}
```

    b. Następnie stworzyłem implementacje tego interfejsu, czyli klasę, która później będzie dekorowana.

```
public class FileDataSource implements DataSource {
    private String filePath;

    public FileDataSource(String name) {
        this.filePath = name;
    }

    @Override
    public void writeData(String data) {
        try {
            Files.write(Paths.get(filePath), data.getBytes());
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
    }

    @Override
    public String readData() {
        try {
            return new String(Files.readAllBytes(Paths.get(filePath)));
        } catch (IOException e) {
            System.out.println(e.getMessage());
            return null;
        }
    }
}
```

c. Następnie stworzyłem BaseDecator. Jest to klasa, z której poprzez dziedziczenie będą tworzone konkretne dekoratory, zmieniające działanie klasy FileDataSource.

```java
public class DataSourceDecorator implements DataSource {
    private DataSource wrappee;

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

    @Override
    public void writeData(String data) {
        wrappee.writeData(data);
    }

    @Override
    public String readData() {
        return wrappee.readData();
    }
}
```

d. Następnie należało stworzyć dwa dekoratory rozszerzające działania klasy dekorującej, czyli klasy FileDataSource

i. EncryptionDecorator

```java
public class EncryptionDecorator extends DataSourceDecorator {
    public EncryptionDecorator(DataSource source) { super(source); }

    @Override
    public void writeData(String data) {
        super.writeData(encode(data));
    }

    @Override
    public String readData() { return decode(super.readData()); }

    private String encode(String data) { return Base64.getEncoder().encodeToString(data.getBytes()); }

    private String decode(String data) { return new String(Base64.getDecoder().decode(data)); }
}
```

ii. CompressionDecorator (z logika (de)kompresującą String'a)

```java
public class CompressionDecorator extends DataSourceDecorator {
    private int compLevel = 6;

    public CompressionDecorator(DataSource source) {
        super(source);
    }

    @Override
    public void writeData(String data) {
        super.writeData(compress(data));
    }

    @Override
    public String readData() {
        return decompress(super.readData());
    }
```

```java
    private String compress(String stringData) {
        byte[] input;
        input = stringData.getBytes(StandardCharsets.UTF_8);

        // Compress the bytes
        byte[] output = new byte[input.length];
        Deflater compressor = new Deflater();
        compressor.setInput(input);
        compressor.finish();
        compressor.deflate(output);
        compressor.end();
        return new String(Base64.getEncoder().encode(output));
    }
```

```java
    private String decompress(String stringData) {
        byte[] output2 = Base64.getDecoder().decode(stringData);

        // Decompress the bytes
        Inflater decompressor = new Inflater();
        decompressor.setInput(output2);
        byte[] result = stringData.getBytes();
        int resultLength = 0;
        try {
            resultLength = decompressor.inflate(result);
        } catch (DataFormatException e) {
            System.out.println(e.getMessage());
        }
        decompressor.end();

        // Decode the bytes into a String
        return new String(result, offset: 0, resultLength, StandardCharsets.UTF_8);
    }
```

e. Ostateczny test działania wzorca (widać, że wszystko działa tak jak powinno)

```java
public static void main(String[] args) {
    String filePath = "/Users/nazkord/IdeaProjects/TO1/lab6/Decorator/test.txt";

    String salaryRecords = "Testowy string \n wartosc 9090909090";
    DataSourceDecorator encoded = new CompressionDecorator(
            new EncryptionDecorator(
                    new FileDataSource(filePath)));
    encoded.writeData(salaryRecords);
    DataSource plain = new FileDataSource(filePath);

    System.out.println("- Input ----------------");
    System.out.println(salaryRecords);
    System.out.println("- Encoded --------------");
    System.out.println(plain.readData());
    System.out.println("- Decoded --------------");
    System.out.println(encoded.readData());
}
```

```
- Input ----------------
Testowy string
 wartosc 9090909090
- Encoded --------------
ZUp3TFNTMHV5Uyt2VkNndUtjck1TMWZnVWloUExDckpMMDVXc0RTQVFRRGlZUXM9
- Decoded --------------
Testowy string
 wartosc 9090909090

Process finished with exit code 0
```