Paweł Kocimski

Lab 6 wzorce projektowe II

1.Wzorzec adapter

Klasa RoundHole współdziała z obiektami w kształcie koła(mających promień) i implementujących interfejs klasy RoundPeg. W klasach zamiast tak jak we schemacie użycia typu int do oznaczenia promienia i długości boku kwadratu użyto typ double.

```
3   public class RoundHole {
4       private double radius;
5
6
7       public RoundHole(int radius) { this.radius = radius; }
10
11      public double getRadius() { return radius; }
14
15  @   public boolean fits(RoundPeg peg) { return this.getRadius() >= peg.getRadius(); }
18  }
```

Klasa RoundPeg definiuje specyficzny interfejs używany przez RoundHole.

```
3   public class RoundPeg {
4       private double radius;
5
6       public RoundPeg(double radius) { this.radius = radius; }
9
10      public RoundPeg() {
11      }
12
13      public double getRadius() { return radius; }
16  }
```

Klasa SquarePeg jest elementem dostosywanym czyli definuje interfejs, który trzeba dostosować.

```
3   public class SquarePeg {
4       private double width;
5
6       public SquarePeg(double width) { this.width = width; }
9
10      public double getWidth() { return width; }
13
```

Klasa SquarePegAdapter jest adapterem, czyli dostosowywuje element klasy SquarePeg do do interfejsu klasy RoundHole.

```
4   public class SquarePegAdapter extends RoundPeg{
5       private SquarePeg peg;
6
7       public SquarePegAdapter(SquarePeg peg) { this.peg = peg; }
10
11      @Override
12      public double getRadius() { return peg.getWidth() * Math.sqrt(2)/2; }
15  }
```

Kod klienta pokazuje działanie adaptera. Sprawdza czy umożliwia on sprawdzić czy można włozyć do okrągłej dziury kwadratowe elementy.

```
3   public class Main {
4
5       public static void main(String[] args) {
6       RoundHole roundHole = new RoundHole( radius: 10);
7       RoundPeg roundPeg = new RoundPeg( radius: 10);
8       if(roundHole.fits(roundPeg)){
9           System.out.println("Round hole with radius 10 suit to round peg with the same radius");
10      }
11      SquarePeg smallSqPeg = new SquarePeg( width: 14);
12      SquarePeg hugeSqPeg = new SquarePeg( width: 15);
13
14      SquarePegAdapter smallPegAdapter = new SquarePegAdapter(smallSqPeg);
15      SquarePegAdapter hugePegAdapter = new SquarePegAdapter(hugeSqPeg);
16
17
18       if (roundHole.fits(smallPegAdapter)) {
19          System.out.println("Square peg with width = 14 fits round hole with radius 10.");
20      }
21          if (!roundHole.fits(hugePegAdapter)) {
22              System.out.println("Square peg with width = 15 does not fit into round hole with radius = 10 .");
23          }
24
25      }
26  }
```

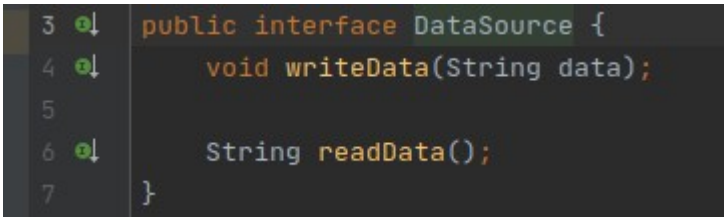
Wynik działania prostego testu. Wszystko działa poprawnie

14 * sqrt(2) = 19.7989898732 < 20 można włożyć element
15 * sqrt(2) = 21.2132034356 > 20 nie można włożyć elementu

```
Round hole with radius 10 suit to round peg with the same radius
Square peg with width = 14 fits round hole with radius 10.
Square peg with width = 15 does not fit into round hole with radius = 10 .
```

2.Wzorzec dekorator

Interfejs DataSource definiuje interfejs obiektów do których można dynamicznie dodawać obsługę zdarzeń



```
3  public interface DataSource {
4      void writeData(String data);
5
6      String readData();
7  }
```

Klasa FieldDataSource jest obiektem, do którego można dołączyć obsługę dodatkowych zdarzeń

```
5   public class FileDataSource implements DataSource {
6       private String fileName;
7
8       public FileDataSource(String fileName) { this.fileName = fileName; }
11
12      @Override
13      public void writeData(String data) {
14          File file = new File(fileName);
15          try(OutputStream fos = new FileOutputStream(file)){
16              fos.write(data.getBytes(),  off: 0, data.length());
17          }catch(IOException ex){
18              System.out.println(ex.getMessage());
19          }
20      }
21
22      @Override
23      public String readData() {
24          char[] buffer = null;
25          File file = new File(fileName);
26          try(FileReader reader = new FileReader(file)){
27              buffer = new char[(int) file.length()];
28              reader.read(buffer);
29          }catch(IOException ex){
30              System.out.println(ex.getMessage());
31          }
32          return  new String(buffer);
33      }
34  }
```

Dekorator DataSourceDecorator przechowuje referencję do obiektu DataSource i implementuje interfejs klasy DataSource

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

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

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

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

Klasa EncryptionDecorator dodaje konkretne zadanie do DataSource – umożliwia szyfrowanie I deszyfrowanie wiadomości.

```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){
        byte[] bytes = data.getBytes();
        for(int i = 0; i < bytes.length; i++){
            bytes[i] += (byte) 1;
        }
        return Base64.getEncoder().encodeToString(bytes);
    }

    private String decode(String data){
        byte[] bytes = Base64.getDecoder().decode(data);
        for(int i= 0; i < bytes.length; i++){
            bytes[i] -= (byte)1;
        }
        return new String(bytes);
    }
}
```

Klasa CompressionDecorator również dodaje konkretne zadanie do dataSource – umożliwia kompresowanie I dekompresowanie

```java
public class CompressionDecorator extends DataSourceDecorator{
    private int compressionLevel = 4;

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

    public int getCompressionLevel() { return compressionLevel; }

    public void setCompressionLevel(int compressionLevel) { this.compressionLevel = compressionLevel; }

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

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

    private String compress(String stringData){
        byte[] data = stringData.getBytes();
        try {
            ByteArrayOutputStream bout = new ByteArrayOutputStream( size: 512);
            DeflaterOutputStream dos = new DeflaterOutputStream(bout, new Deflater(compressionLevel));
            dos.write(data);
            dos.close();
            bout.close();
            return Base64.getEncoder().encodeToString(bout.toByteArray());
        }catch(IOException ex){
            System.out.println(ex.getMessage());
            return null;
        }
    }

    private String decompress(String stringData){
        byte[] data = Base64.getDecoder().decode(stringData);
        try{
            InputStream in = new ByteArrayInputStream(data);
            InflaterInputStream iin = new InflaterInputStream(in);
            ByteArrayOutputStream bout = new ByteArrayOutputStream( size: 512);
            int byteNo;
            while((byteNo = iin.read()) != -1){
                bout.write(byteNo);
            }
            in.close();
            iin.close();
            bout.close();
            return new String(bout.toByteArray());
        }catch (IOException ex){
            System.out.println(ex.getMessage());
            return null;
        }
    }
}
```

Klient sprawdzajacy dzialnie aplikacji

```java
public class Main {
    public static void main(String[] args) {
        String d17 = "Al. A. Mickiewicza 30\n " +
                "Pawilon D-17 \n" +
                "30-059 Kraków";

        DataSourceDecorator encoded = new EncryptionDecorator(
                                        new CompressionDecorator(
                                                new FileDataSource( fileName: "output/address.txt")));

        encoded.writeData(d17);
        DataSource file = new FileDataSource( fileName: "output/address.txt");

        System.out.println("INPUT:\n " + d17);
        System.out.println("ENCODED:\n " + file.readData());
        System.out.println("DECODED:\n " + encoded.readData());
    }
}
```

Wynik działania aplikacji

```
INPUT:
 Al. A. Mickiewicza 30
 Pawilon D-17
30-059 Kraków
ENCODED:
 eF4LzDUo8wz1BGLTwij3isKovIzCKI+STM+QwArnSregSCA/KcIJKB9a6puVnuFcFVjhk+Vq5J/p5pts5FVc4RNkAgBHJhgE
DECODED:
 Al. A. Mickiewicza 30
 Pawilon D-17
30-059 Kraków
```

3.Wzorzec command

Klasa abstrakcyjna Command

```java
public abstract class Command {
    public Editor editor;
    private String backup;

    Command(Editor editor) { this.editor = editor; }

    void backup() { backup = editor.textField.getText(); }

    public void undo() { editor.textField.setText(backup); }

    public abstract boolean execute();
}
```

Klasy dziedziczące po klasie Comand:
CopyCommand

```java
public class CopyCommand extends Command {

    public CopyCommand(Editor editor) { super(editor); }

    @Override
    public boolean execute() {
        editor.clipboard = editor.textField.getSelectedText();
        return false;
    }
}
```

PasteCommand

```java
public class PasteCommand extends Command{
    public PasteCommand(Editor editor) { super(editor); }

    @Override
    public boolean execute() {
        if (editor.clipboard == null || editor.clipboard.isEmpty()) return false;

        backup();
        editor.textField.insert(editor.clipboard, editor.textField.getCaretPosition());
        return true;
    }
}
```

CutCommand

```java
public class CutCommand extends Command{

    public CutCommand(Editor editor) {
        super(editor);
    }

    @Override
    public boolean execute() {
        if (editor.textField.getSelectedText().isEmpty()) return false;

        backup();
        String source = editor.textField.getText();
        editor.clipboard = editor.textField.getSelectedText();
        editor.textField.setText(cutString(source));
        return true;
    }

    private String cutString(String source) {
        String start = source.substring(0, editor.textField.getSelectionStart());
        String end = source.substring(editor.textField.getSelectionEnd());
        return start + end;
    }
}
```

Klasa CommandHistory umożliwiająca przechowywanie historii komend

```java
public class CommandHistory {
    private Stack<Command> history = new Stack<>();

    public void push(Command c) { history.push(c); }

    public Command pop() { return history.pop(); }

    public boolean isEmpty() { return history.isEmpty(); }
}
```

3. Edytor Graficzny

```java
public class Editor {
    public JTextArea textField;
    public String clipboard;
    private CommandHistory history = new CommandHistory();

    public void init() {
        JFrame frame = new JFrame( title: "Text editor");
        JPanel content = new JPanel();
        frame.setContentPane(content);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
        textField = new JTextArea();
        textField.setLineWrap(true);
        content.add(textField);
        JPanel buttons = new JPanel(new FlowLayout(FlowLayout.CENTER));
        JButton copy = new JButton( text: "Copy");
        JButton cut = new JButton( text: "Cut");
        JButton paste = new JButton( text: "Paste");
        JButton undo = new JButton( text: "Undo");
        Editor editor = this;
        copy.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) { executeCommand(new CopyCommand(editor)); }
        });
        cut.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) { executeCommand(new CutCommand(editor)); }
        });
        paste.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) { executeCommand(new PasteCommand(editor)); }
        });
        undo.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) { undo(); }
        });
        buttons.add(copy);
        buttons.add(cut);
        buttons.add(paste);
        buttons.add(undo);
        content.add(buttons);
        frame.setSize( width: 600,  height: 250);
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
```

```
@       private void executeCommand(Command command) {
            if (command.execute()) {
                history.push(command);
            }
        }

        private void undo() {
            if (history.isEmpty()) return;

            Command command = history.pop();
            if (command != null) {
                command.undo();
            }
        }
    }
```

Klient aplikacji

```
3   public class Main {
4       public static void main(String[] args) {
5           Editor editor = new Editor();
6           editor.init();
7       }
8   }
```

Aplikacja