Projektowanie obiektowe

Lab 6 Wzorce projektowe cz. 2

Dawid Białka, Piotr Tekielak

1. Adapter

Tworzymy klasę RoundPeg.

```
public class RoundPeg {
    private int radius;

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

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

Tworzymy klasę RoundHole. Umożliwia ona sprawdzenie czy dany kołek może zmieścić się w dziurze.

```
public class RoundHole {
    private int radius;

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

    int getRadius()
    {
        return radius;
    }

    boolean fits(RoundPeg peg)
    {
        System.out.println("Trying to fit peg into the hole");
        return peg.getRadius() <= radius;
    }
}
```

Tworzymy klasę SquarePeg.

```
public class SquarePeg {
    private int width;

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

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

Obecnie nie można sprawdzić czy kwadratowy kołek zmieści się w okrągłej dziurze. Nie posiada on promienia. Aby to umożliwić tworzymy klasę SquarePegAdapter.

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

    public SquarePegAdapter(SquarePeg peg)
    {
        super(peg.getWidth());
        System.out.println("Creating an adapter for square with width of " +
peg.getWidth());
        this.peg = peg;
    }

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

Od teraz możemy wywołać metodę fits nie tylko dla okrągłego kołka, lecz także dla kwadratowego za pośrednictwem adaptera. W metodzie głównej testujemy całą aplikację.

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

    System.out.println(hole.fits(roundPeg));

    SquarePeg smallSquarePeg = new SquarePeg(5);
    SquarePeg largeSquarePeg = new SquarePeg(10);

    //System.out.println(hole.fits(smallSquarePeg)); //will not work

    SquarePegAdapter smallSquarePegAdapter = new SquarePegAdapter(smallSquarePeg);
    SquarePegAdapter largeSquarePegAdapter = new SquarePegAdapter(largeSquarePeg);

    System.out.println(hole.fits(smallSquarePegAdapter));
    System.out.println(hole.fits(largeSquarePegAdapter));
}
```

Wynik wywołania programu:

```
Trying to fit peg into the hole
true
Creating an adapter for square with width of 5
Creating an adapter for square with width of 10
Trying to fit peg into the hole
true
Trying to fit peg into the hole
false
```

2. Decorator

Ponieważ praca z plikami, kompresja i szyfrowanie wymaga łapania wielu wyjątków, pominąłem je w poniższych fragmentach kodu.

Tworzymy interfejs DataSource.

```
public interface DataSource
{
    public void writeData(Data data);

    public Data readData();
}
```

Dane będą reprezentowane jako tablica bajtów.

```
public class Data
{
    private byte[] data;

    public Data(byte[] data)
    {
        this.data = data;
    }

    public byte[] getData()
    {
        return data;
    }

    public void setData(byte[] data)
    {
        this.data = data;
    }
}
```

FileDataSource będzie konkretną reprezentacją interfejsu DataSource.

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

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

    @Override
    public void writeData(Data data) throws IOException
    {
        Files.write(Paths.get(filename), data.getData());
    }

    @Override
    public Data readData() throws IOException
    {
        byte[] bytes = Files.readAllBytes(Paths.get(filename));
        Data data = new Data(bytes);
        return data;
    }
}
```

Tworzymy klasę DataSourceDecorator. Umożliwi ona „zawijanie” obiektów typu DataSource w dekoratory.

```
public class DataSourceDecorator implements DataSource
{
    private DataSource wrapper;

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

    @Override
    public void writeData(Data data)
    {
        wrapper.writeData(data);
    }

    @Override
    public Data readData()
    {
        return wrapper.readData();
    }
}
```

Tworzymy klasę EncryptionDecorator. Będzie ona odpowiedzialna za szyfrowanie i deszyfrowanie danych. W tym celu będzie wykorzystywać klasę Cipher z pakietu javax.crypto.

```
public class EncryptionDecorator extends DataSourceDecorator
{
    private static SecretKeySpec secretKey;
    private static byte[] key;

    public static void setKey(String myKey)
    {
        MessageDigest sha = null;
        try {
            key = myKey.getBytes("UTF-8");
            sha = MessageDigest.getInstance("SHA-1");
            key = sha.digest(key);
            key = Arrays.copyOf(key, 16);
            secretKey = new SecretKeySpec(key, "AES");
        }
        catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
    }

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

    @Override
    public void writeData(Data data)
    {
        System.out.println("Encryptor received:" +
Arrays.toString(data.getData()));
        setKey("sssh!!!!");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        data.setData(cipher.doFinal(data.getData()));
        System.out.println("Encryptor sent:" + Arrays.toString(data.getData()));
        super.writeData(data);
    }

    @Override
    public Data readData()
    {
        Data data = super.readData();
        System.out.println("Decryptor received:" +
Arrays.toString(data.getData()));
        setKey("sssh!!!!");
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        data.setData(cipher.doFinal(data.getData()));
        System.out.println("Decryptor sent:" + Arrays.toString(data.getData()));
        return data;
    }
}
```

Tworzymy klasę CompressionDecorator. Będzie ona kompresować i dekompresować dane za pomocą klas GZIP z pakietu java.util.zip.

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

    @Override
    public void writeData(Data data)
    {
        System.out.println("Compressor received: " +
Arrays.toString(data.getData()));
        ByteArrayOutputStream bos = new
ByteArrayOutputStream(data.getData().length);
        GZIPOutputStream gzip = new GZIPOutputStream(bos);
        gzip.write(data.getData());
        gzip.close();
        byte[] compressed = bos.toByteArray();
        bos.close();
        data.setData(compressed);
        System.out.println("Compressor sent: " + Arrays.toString(data.getData()));
        super.writeData(data);
    }

    @Override
    public Data readData()
    {
        Data data = super.readData();
        System.out.println("Decompressor received: " +
Arrays.toString(data.getData()));
        ByteArrayInputStream bis = new ByteArrayInputStream(data.getData());
        GZIPInputStream gis = new GZIPInputStream(bis);
        data.setData(gis.readAllBytes());
        gis.close();
        bis.close();
        System.out.println("Decompressor sent: " +
Arrays.toString(data.getData()));
        return data;
    }
}
```

W klasie Main tworzymy metody, które tworzą odpowiednio zagnieżdżone struktury i wykonują na nich operacje. W funkcji głównej testujemy działanie programu.

```
public class Main
{
    public static void main(String[] args)
    {
        Data data = new Data("javaJestSuper".getBytes("UTF-8"));
        writeData(data, "test.txt");
        data = readData("test.txt");

        String text = new String(data.getData(), "UTF-8");
        System.out.println(text);
    }
```

```
    public static void writeData(Data data, String filename)
    {
        DataSource dataSource = new EncryptionDecorator(new
CompressionDecorator(new FileDataSource(filename)));
        dataSource.writeData(data);
    }

    public static Data readData(String filename)
    {
        DataSource dataSource = new EncryptionDecorator(new
CompressionDecorator(new FileDataSource(filename)));
        return dataSource.readData();
    }
}
```

Wynik działania programu:

```
Encryptor received:[106, 97, 118, 97, 74, 101, 115, 116, 83, 117, 112, 101, 114]
Encryptor sent:[-24, -11, -47, -103, -58, -35, 5, -2, -23, 69, 23, -128, -8, -99, 115, -46]
Compressor received: [-24, -11, -47, -103, -58, -35, 5, -2, -23, 69, 23, -128, -8, -99, 115, -46]
Compressor sent: [31, -117, 8, 0, 0, 0, 0, 0, 0, 0, 123, -15, -11, -30, -52, 99, 119, 89, -1, -67, 116, 21, 111, -8, 49, -73, -8, 18, 0, -38, 13, -57, 23, 16, 0, 0, 0]
Decompressor received: [31, -117, 8, 0, 0, 0, 0, 0, 0, 0, 123, -15, -11, -30, -52, 99, 119, 89, -1, -67, 116, 21, 111, -8, 49, -73, -8, 18, 0, -38, 13, -57, 23, 16, 0, 0, 0]
Decompressor sent: [-24, -11, -47, -103, -58, -35, 5, -2, -23, 69, 23, -128, -8, -99, 115, -46]
Decryptor received:[-24, -11, -47, -103, -58, -35, 5, -2, -23, 69, 23, -128, -8, -99, 115, -46]
Decryptor sent:[106, 97, 118, 97, 74, 101, 115, 116, 83, 117, 112, 101, 114]
javaJestSuper
```

3. Command

```
public abstract class Command
{
    protected Application app;
    protected Editor editor;
    protected String backup;

    public Command(Application app, Editor editor)
    {
        this.app = app;
        this.editor = editor;
    }

    public void saveBackup()
    {
        this.backup = editor.text.getText();
    }

    public abstract void execute();

    public abstract void undo();
}
```

Klasa abstrakcyjna Command, po której dziedziczą wszystkie następujące klasy:

Klasa CopyCommand:

```
public class CopyCommand extends Command
{
    public CopyCommand(Application app, Editor editor)
    {
        super(app, editor);
    }

    @Override
    public void execute()
    {
        app.clipboard = editor.getSelection();
        System.out.println("Copied " + app.clipboard);
    }

    @Override
    public void undo()
    {
        app.clipboard = null;
        System.out.println("Copy undone");
    }
}
```

Klasa PasteCommand:

```
public class PasteCommand extends Command
{
    public PasteCommand(Application app, Editor editor)
    {
        super(app, editor);
    }

    @Override
    public void execute()
    {
        this.saveBackup();
        if(app.clipboard != null)
        {
            editor.replaceSelection(app.clipboard);
            System.out.println("Pasted " + app.clipboard);
        }
        else
        {
            System.out.println("Clipboard empty");
        }
    }

    @Override
    public void undo()
    {
        editor.text.setText(this.backup);
        System.out.println("Paste undone");
    }
}
```

Klasa CutCommand:

```
public class CutCommand extends Command
{
    public CutCommand(Application app, Editor editor)
    {
        super(app, editor);
    }

    @Override
    public void execute()
    {
        this.saveBackup();
        app.clipboard = editor.getSelection();
        editor.deleteSelection();
        System.out.println("Cut " + app.clipboard);
    }

    @Override
    public void undo()
    {
        this.app.clipboard = null;
        editor.text.setText(this.backup);
        System.out.println("Cut undone");
    }
}
```

Klasa CommandHistory przechowująca wszystkie wywołane dotychczas komendy, które nie zostały cofnięte:

```
ublic class CommandHistory
{
    private Stack<Command> history;

    public CommandHistory()
    {
        history = new Stack<>();
    }

    public void push(Command command)
    {
        this.history.push(command);
    }

    public Command pop()
    {
        if(history.isEmpty())
        {
            throw new NullPointerException("No command to undo");
        }
        else
        {
            return history.pop();
        }
    }
}
```

Klasa Application, w której przechowujemy schowek i instancję klasy CommandHistory oraz jest to klasa, która wywołuje wykonanie komend.

```
public class Application
{
    public Editor editor = null;
    public String clipboard = null;
    public CommandHistory history;
    public GUI gui;

    public Application()
    {
        editor = new Editor();
        history = new CommandHistory();
    }

    public void createGUI()
    {
        this.gui = new GUI(this);
    }

    public void executeCommand(Command command)
    {
        command.execute();
        history.push(command);
    }

    public void undo()
    {
        history.pop().undo();
    }

}
```

Klasa Editor, która przechowuje aktualny tekst.

```
public class Editor
{
    JTextPane text;

    public Editor()
    {
        text = new JTextPane();
        text.setText("Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                Donec eget dictum odio, eget interdum nisl. Aliquam " +
               "vitae ante eu enim sodales ultricies ac id libero.");
    }

    public void deleteSelection()
    {
        this.text.replaceSelection("");
    }

    public void replaceSelection(String text)
    {
        this.text.replaceSelection(text);
    }
```

```
    public String getSelection()
    {
        return this.text.getSelectedText();
    }
}
```

Klasa GUI.

```
public class GUI
{
    Application app;
    private JFrame frame;
    JButton copyCommandButton;
    JButton pasteCommandButton;
    JButton cutCommandButton;
    JButton undoCommandButton;

    public GUI(Application app)
    {
        this.app = app;
        this.frame = new JFrame();

        this.copyCommandButton = new JButton("Copy");
        this.pasteCommandButton = new JButton("Paste");
        this.cutCommandButton = new JButton("Cut");
        this.undoCommandButton = new JButton("Undo");
        this.copyCommandButton.setVisible(true);
        this.pasteCommandButton.setVisible(true);
        this.cutCommandButton.setVisible(true);
        this.undoCommandButton.setVisible(true);
        this.copyCommandButton.setPreferredSize(new Dimension(110, 24));
        this.pasteCommandButton.setPreferredSize(new Dimension(110, 24));
        this.cutCommandButton.setPreferredSize(new Dimension(110, 24));
        this.undoCommandButton.setPreferredSize(new Dimension(110, 24));
        this.copyCommandButton.setBounds(300, 10, 110, 24);
        this.pasteCommandButton.setBounds(300, 44, 110, 24);
        this.cutCommandButton.setBounds(300, 78, 110, 24);
        this.undoCommandButton.setBounds(300, 112, 110, 24);

        JTextPane text =  this.app.editor.text;
        text.setEditable(false);
        text.setPreferredSize(new Dimension(200, 200));
        text.setBounds(10, 10, 200, 200);

        copyCommandButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent event)
            {
                Command cmd = new CopyCommand(app, app.editor);
                app.executeCommand(cmd);
            }
        });

        pasteCommandButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent event)
```

```
            {
                Command cmd = new PasteCommand(app, app.editor);
                text.setEditable(true);
                app.executeCommand(cmd);
                text.setEditable(false);
            }
        });

        cutCommandButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent event)
            {
                Command cmd = new CutCommand(app, app.editor);
                text.setEditable(true);
                app.executeCommand(cmd);
                text.setEditable(false);
            }
        });

        undoCommandButton.addActionListener(new ActionListener()
        {
            @Override
            public void actionPerformed(ActionEvent event)
            {
                text.setEditable(true);
                app.undo();
                text.setEditable(false);
            }
        });

        this.frame.add(copyCommandButton);
        this.frame.add(pasteCommandButton);
        this.frame.add(cutCommandButton);
        this.frame.add(undoCommandButton);
        this.frame.add(this.app.editor.text);


        frame.setLayout(null);
        frame.pack();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setTitle("Text editor");
        frame.setResizable(false);
        Dimension screenDimension = Toolkit.getDefaultToolkit().getScreenSize();
        frame.setLocation((int) (screenDimension.getWidth() / 2 - 220),
                (int) (screenDimension.getHeight() /2 - 250));
        frame.setVisible(true);
        frame.setSize(440, 500);
    }

}
```