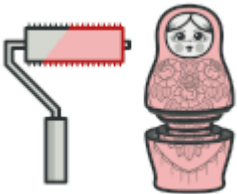


HELP UKRAINE STOP RUSSIA

[🏠](#) / [Паттерны проектирования](#) / [Декоратор](#) / [Java](#)

# Декоратор на Java

**Декоратор** — это структурный паттерн, который позволяет добавлять объектам новые поведения на лету, помещая их в объекты-обёртки.

Декоратор позволяет оборачивать объекты бесчисленное количество раз благодаря тому, что и обёртки, и реальные оборачиваемые объекты имеют общий интерфейс.

[📖 Подробнее о паттерне Декоратор →](#)

## Навигация

[📖 Интро](#)[📖 Шифрование и сжатие данных](#)[📁 decorators](#)[📄 DataSource](#)[📄 FileDataSource](#)[📄 DataSourceDecorator](#)[📄 EncryptionDecorator](#)[📄 CompressionDecorator](#)[📄 Demo](#)[📄 OutputDemo](#)

Сложность: ★★☆☆

Популярность: ★★☆☆

**Применимость:** Паттерн можно часто встретить в Java-коде, особенно в коде, работающем с потоками данных.

Примеры Декораторов в стандартных библиотеках Java:

- Все подклассы `java.io.InputStream`, `OutputStream`, `Reader` и `Writer` имеют конструктор, принимающий объекты этих же классов.
- `java.util.Collections`, методы `checkedXXX()`, `synchronizedXXX()` и `unmodifiableXXX()`.
- `javax.servlet.http.HttpServletRequestWrapper` и `HttpServletResponseWrapper`

**Признаки применения паттерна:** Декоратор можно распознать по создающим методам, которые принимают в параметрах объекты того же абстрактного типа или интерфейса, что и текущий класс.

## Шифрование и сжатие данных

Пример показывает, как можно добавить новую функциональность объекту, не меняя его класса.

Сначала класс бизнес-логики мог считывать и записывать только чистые данные напрямую из/в файлы. Применив паттерн Декоратор, мы создали небольшие классы-обёртки, которые добавляют новые поведения до или после основной работы вложенного объекта бизнес-логики.

Первая обёртка шифрует и расшифрует данные, а вторая — сжимает и распакует их.

Мы можем использовать обёртки как отдельно друг от друга, так и все вместе, обернув один декоратор другим.

### decorators

#### decorators/DataSource.java: Интерфейс, задающий базовые операции чтения и записи данных

```
package refactoring_guru.decorator.example.decorators;

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

```
String readData();  
}
```

## decorators/FileDataSource.java: Класс, реализующий прямое чтение и запись данных

```
package refactoring_guru.decorator.example.decorators;  
  
import java.io.*;  
  
public class FileDataSource implements DataSource {  
    private String name;  
  
    public FileDataSource(String name) {  
        this.name = name;  
    }  
  
    @Override  
    public void writeData(String data) {  
        File file = new File(name);  
        try (OutputStream fos = new FileOutputStream(file)) {  
            fos.write(data.getBytes(), 0, data.length());  
        } catch (IOException ex) {  
            System.out.println(ex.getMessage());  
        }  
    }  
  
    @Override  
    public String readData() {  
        char[] buffer = null;  
        File file = new File(name);  
        try (FileReader reader = new FileReader(file)) {  
            buffer = new char[(int) file.length()];  
            reader.read(buffer);  
        } catch (IOException ex) {  
            System.out.println(ex.getMessage());  
        }  
        return new String(buffer);  
    }  
}
```

## decorators/DataSourceDecorator.java: Базовый декоратор

```
package refactoring_guru.decorator.example.decorators;
```

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

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

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

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

## decorators/EncryptionDecorator.java: Декоратор шифрования

```
package refactoring_guru.decorator.example.decorators;

import java.util.Base64;

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

    private String decode(String data) {

```

```

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

```

## decorators/CompressionDecorator.java: Декоратор сжатия

```

package refactoring_guru.decorator.example.decorators;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.util.Base64;
import java.util.zip.Deflater;
import java.util.zip.DeflaterOutputStream;
import java.util.zip.InflaterInputStream;

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

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

    public int getCompressionLevel() {
        return compLevel;
    }

    public void setCompressionLevel(int value) {
        compLevel = value;
    }

    @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(512);
        DeflaterOutputStream dos = new DeflaterOutputStream(bout, new Deflater(comple
        dos.write(data);
        dos.close();
        bout.close();
        return Base64.getEncoder().encodeToString(bout.toByteArray());
    } catch (IOException ex) {
        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(512);
        int b;
        while ((b = iin.read()) != -1) {
            bout.write(b);
        }
        in.close();
        iin.close();
        bout.close();
        return new String(bout.toByteArray());
    } catch (IOException ex) {
        return null;
    }
}
}

```

## Demo.java: Клиентский код

```

package refactoring_guru.decorator.example;

import refactoring_guru.decorator.example.decorators.*;

public class Demo {
    public static void main(String[] args) {
        String salaryRecords = "Name,Salary\nJohn Smith,100000\nSteven Jobs,912000";
        DataSourceDecorator encoded = new CompressionDecorator(
            new EncryptionDecorator(
                new FileDataSource("out/OutputDemo.txt")));
        encoded.writeData(salaryRecords);
        DataSource plain = new FileDataSource("out/OutputDemo.txt");

        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());
    }
}
```

### OutputDemo.txt: Результат выполнения

```
- Input -----
Name,Salary
John Smith,100000
Steven Jobs,912000
- Encoded -----
Zkt7e1Q5eU8yUm1Qe0ZsdHJ2VXp6dDBKVnhrUhtUe0sxRUYxQkJIdjVLTvZ0dVI5Q2IwOXFISmVUMU5rcENCQmdxF
- Decoded -----
Name,Salary
John Smith,100000
Steven Jobs,912000
```

**ЧИТАЕМ ДАЛЬШЕ**

Фасад на Java →

**ВЕРНУТЬСЯ НАЗАД**

← Компоновщик на Java

## Декоратор на других языках программирования

[Главная](#)



[Рефакторинг](#)



[Паттерны](#)




[Премиум контент](#)

[Форум](#)

[Связаться](#)

© 2014-2023 Refactoring.Guru. Все права защищены.

 Иллюстрации нарисовал Дмитрий Жарт

 Хмельницкое шоссе 19 / 27, Каменец-Подольский, Украина, 32305

 Email: [support@refactoring.guru](mailto:support@refactoring.guru)

[Условия использования](#)

[Политика конфиденциальности](#)

[Использование контента](#)