

## Examples

## Adapter Pattern

**Szenario:** Altbestand an Drittanbieter-Bibliothek integrieren

```
1 // Bestehende Schnittstelle
2 interface ModernPrinter {
3     void printDocument(String content);
4 }
5
6 // Alte Drittanbieter-Klasse
7 class LegacyPrinter {
8     public void print(String[] pages) {
9         for(String page : pages) {
10             System.out.println(page);
11         }
12     }
13 }
14
15 // Adapter
16 class PrinterAdapter implements ModernPrinter {
17     private LegacyPrinter legacyPrinter;
18
19     public PrinterAdapter(LegacyPrinter printer) {
20         this.legacyPrinter = printer;
21     }
22
23     public void printDocument(String content) {
24         String[] pages = content.split("\n");
25         legacyPrinter.print(pages);
26     }
27 }
```

## Simple Factory

**Szenario:** Erzeugung von verschiedenen Datenbankverbindungen

```
1 class DatabaseFactory {
2     public static Database createDatabase(String type)
3     {
4         switch(type) {
5             case "MySQL":
6                 return new MySQLDatabase();
7             case "PostgreSQL":
8                 return new PostgreSQLDatabase();
9             default:
10                 throw new
11                     IllegalArgumentException("Unknown
12                     DB type");
13         }
14     }
15 }
16
17 // Verwendung
18 Database db = DatabaseFactory.createDatabase("MySQL");
```

## Singleton

**Szenario:** Globale Konfigurationsverwaltung

```
1 public class Configuration {
2     private static Configuration instance;
3     private Map<String, String> config;
4
5     private Configuration() {
6         config = new HashMap<>();
7     }
8
9     public static Configuration getInstance() {
10         if(instance == null) {
11             instance = new Configuration();
12         }
13         return instance;
14     }
15 }
```

## Dependency Injection

**Szenario:** Flexible Logger-Implementation

```
1 interface Logger {
2     void log(String message);
3 }
4
5 class FileLogger implements Logger {
6     public void log(String message) {
7         // Log to file
8     }
9 }
10
11 class UserService {
12     private final Logger logger;
13
14     public UserService(Logger logger) { // Dependency
15         Injection
16         this.logger = logger;
17     }
18 }
```

## Proxy

**Szenario:** Verzögertes Laden eines großen Bildes

```
1 interface Image {
2     void display();
3 }
4
5 class RealImage implements Image {
6     private String filename;
7
8     public RealImage(String filename) {
9         this.filename = filename;
10         loadFromDisk();
11     }
12
13     private void loadFromDisk() {
14         System.out.println("Loading " + filename);
15     }
16
17     public void display() {
18         System.out.println("Displaying " + filename);
19     }
20 }
21
22 class ImageProxy implements Image {
23     private RealImage realImage;
24     private String filename;
25
26     public ImageProxy(String filename) {
27         this.filename = filename;
28     }
29
30     public void display() {
31         if(realImage == null) {
32             realImage = new RealImage(filename);
33         }
34         realImage.display();
35     }
36 }
```

## Chain of Responsibility

**Szenario:** Authentifizierungskette

```
1 abstract class AuthHandler {
2     protected AuthHandler next;
3
4     public void setNext(AuthHandler next) {
5         this.next = next;
6     }
7
8     public abstract boolean handle(String username,
9         String password);
10 }
11
12 class LocalAuthHandler extends AuthHandler {
13     public boolean handle(String username, String
14         password) {
15         if(checkLocalDB(username, password)) {
16             return true;
17         }
18         return next != null ? next.handle(username,
19             password) : false;
20     }
21 }
22
23 class LDAPAuthHandler extends AuthHandler {
24     public boolean handle(String username, String
25         password) {
26         if(checkLDAP(username, password)) {
27             return true;
28         }
29         return next != null ? next.handle(username,
30             password) : false;
31     }
32 }
```

## Decorator

**Szenario:** Dynamische Erweiterung eines Text-Editors

```
1 interface TextComponent {
2     String render();
3 }
4
5 class SimpleText implements TextComponent {
6     private String text;
7
8     public SimpleText(String text) {
9         this.text = text;
10    }
11
12    public String render() {
13        return text;
14    }
15 }
16
17 class BoldDecorator implements TextComponent {
18     private TextComponent component;
19
20     public BoldDecorator(TextComponent component) {
21         this.component = component;
22    }
23
24    public String render() {
25        return "<b>" + component.render() + "</b>";
26    }
27 }
```

## Observer

**Szenario:** News-Benachrichtigungssystem

```
1 interface NewsObserver {
2     void update(String news);
3 }
4
5 class NewsAgency {
6     private List<NewsObserver> observers = new
7         ArrayList<>();
8
9     public void addObserver(NewsObserver observer) {
10         observers.add(observer);
11    }
12
13    public void notifyObservers(String news) {
14        for(NewsObserver observer : observers) {
15            observer.update(news);
16        }
17    }
18 }
19
20 class NewsChannel implements NewsObserver {
21     private String name;
22
23     public NewsChannel(String name) {
24         this.name = name;
25    }
26
27     public void update(String news) {
28         System.out.println(name + " received: " +
29             news);
30    }
31 }
```

## Strategy

**Szenario:** Verschiedene Zahlungsmethoden

```
1 interface PaymentStrategy {
2     void pay(int amount);
3 }
4
5 class CreditCardPayment implements PaymentStrategy {
6     private String cardNumber;
7
8     public void pay(int amount) {
9         System.out.println("Paid " + amount + " using
10             Credit Card");
11    }
12 }
13
14 class PayPalPayment implements PaymentStrategy {
15     private String email;
16
17     public void pay(int amount) {
18         System.out.println("Paid " + amount + " using
19             PayPal");
20    }
21 }
```

## Composite

**Szenario:** Dateisystem-Struktur

```
1 interface FileSystemComponent {
2     void list(String prefix);
3 }
4
5 class File implements FileSystemComponent {
6     private String name;
7
8     public void list(String prefix) {
9         System.out.println(prefix + name);
10    }
11 }
12
13 class Directory implements FileSystemComponent {
14     private String name;
15     private List<FileSystemComponent> children = new
16         ArrayList<>();
17
18     public void add(FileSystemComponent component) {
19         children.add(component);
20    }
21
22     public void list(String prefix) {
23         System.out.println(prefix + name);
24         for(FileSystemComponent child : children) {
25             child.list(prefix + " ");
26         }
27    }
28 }
```

## State

**Szenario:** Verkaufsautomat

```
1 interface VendingMachineState {
2     void insertCoin();
3     void ejectCoin();
4     void selectProduct();
5     void dispense();
6 }
7
8 class HasCoinState implements VendingMachineState {
9     private VendingMachine machine;
10
11     public void selectProduct() {
12         System.out.println("Product selected");
13         machine.setState(machine.getSoldState());
14    }
15
16     public void insertCoin() {
17         System.out.println("Already have coin");
18    }
19 }
20
21 class VendingMachine {
22     private VendingMachineState currentState;
23
24     public void setState(VendingMachineState state) {
25         this.currentState = state;
26    }
27
28     public void insertCoin() {
29         currentState.insertCoin();
30    }
31 }
```

### Visitor

**Szenario:** Dokumentstruktur mit verschiedenen Operationen

```
1 interface DocumentElement {
2     void accept(Visitor visitor);
3 }
4
5 interface Visitor {
6     void visit(Paragraph paragraph);
7     void visit(Heading heading);
8 }
9
10 class HTMLVisitor implements Visitor {
11     public void visit(Paragraph p) {
12         System.out.println("<p>" + p.getText() +
13             "</p>");
14     }
15
16     public void visit(Heading h) {
17         System.out.println("<h1>" + h.getText() +
18             "</h1>");
19     }
20 }
```

### Facade

**Szenario:** Vereinfachte Multimedia-Bibliothek

```
1 class MultimediaFacade {
2     private AudioSystem audio;
3     private VideoSystem video;
4     private SubtitleSystem subtitles;
5
6     public void playMovie(String movie) {
7         audio.initialize();
8         video.initialize();
9         subtitles.load(movie);
10        video.play(movie);
11        audio.play();
12    }
13 }
```

### Abstract Factory

**Szenario:** GUI-Elemente für verschiedene Betriebssysteme

```
1 interface GUIFactory {
2     Button createButton();
3     Checkbox createCheckbox();
4 }
5
6 class WindowsFactory implements GUIFactory {
7     public Button createButton() {
8         return new WindowsButton();
9     }
10
11     public Checkbox createCheckbox() {
12         return new WindowsCheckbox();
13     }
14 }
15
16 class MacFactory implements GUIFactory {
17     public Button createButton() {
18         return new MacButton();
19     }
20
21     public Checkbox createCheckbox() {
22         return new MacCheckbox();
23     }
24 }
```

### Factory Method Implementation

**Aufgabe:** Implementieren Sie eine Factory für verschiedene Dokumenttypen (PDF, Word, Text)

**Lösung:**

```
1 // Interface fuer Produkte
2 interface Document {
3     void open();
4     void save();
5 }
6
7 // Konkrete Produkte
8 class PdfDocument implements Document {
9     public void open() { /* ... */ }
10    public void save() { /* ... */ }
11 }
12
13 // Factory Method Pattern
14 abstract class DocumentCreator {
15     abstract Document createDocument();
16
17     // Template Method
18     final void processDocument() {
19         Document doc = createDocument();
20         doc.open();
21         doc.save();
22     }
23 }
24
25 // Konkrete Factory
26 class PdfDocumentCreator extends DocumentCreator {
27     Document createDocument() {
28         return new PdfDocument();
29     }
30 }
```

### Observer Pattern Implementation

**Aufgabe:** Implementieren Sie ein Benachrichtigungssystem für Aktienkurse

**Lösung:**

```
1 interface StockObserver {
2     void update(String stock, double price);
3 }
4
5 class StockMarket {
6     private List<StockObserver> observers = new
7         ArrayList<>();
8
9     public void attach(StockObserver observer) {
10         observers.add(observer);
11     }
12
13     public void notifyObservers(String stock, double
14         price) {
15         for(StockObserver observer : observers) {
16             observer.update(stock, price);
17         }
18     }
19 }
20
21 class StockDisplay implements StockObserver {
22     public void update(String stock, double price) {
23         System.out.println("Stock: " + stock +
24             " updated to " + price);
25     }
26 }
```

**Vorher:**

Nachher:

Unit Test

**Zu testende Klasse:**

**Test:**

### BDD Test

**Feature File:**

**Step Definitions:**

## Client-Server Implementation

**Aufgabe:** Implementieren Sie einen einfachen Echo-Server mit Java.

**Lösung:**

```

1 // Server
2 public class EchoServer {
3     public static void main(String[] args) {
4         try (ServerSocket server = new
5             ServerSocket(8080)) {
6             while (true) {
7                 Socket client = server.accept();
8                 new Thread(() ->
9                     handleClient(client)).start();
10            }
11        }
12
13        private static void handleClient(Socket client) {
14            try (
15                BufferedReader in = new BufferedReader(
16                    new
17                        InputStreamReader(client.getInputStream(),
18                            "UTF-8")) {
19                PrintWriter out = new PrintWriter(
20                    client.getOutputStream(), true)
21            ) {
22                String line;
23                while ((line = in.readLine()) != null) {
24                    out.println("Echo: " + line);
25                }
26            } catch (IOException e) {
27                e.printStackTrace();
28            }
29        }
30    }
31}
32
33// Client
34public class EchoClient {
35    public static void main(String[] args) {
36        try (
37            Socket socket = new Socket("localhost",
38                8080);
39            PrintWriter out = new PrintWriter(
40                socket.getOutputStream(), true);
41            BufferedReader in = new BufferedReader(
42                new
43                    InputStreamReader(socket.getInputStream(),
44                        "UTF-8")) {
45            } {
46                out.println("Hello Server!");
47                System.out.println(in.readLine());
48            } catch (IOException e) {
49                e.printStackTrace();
50            }
51        }
52    }
53}

```

## Publish-Subscribe Pattern

**Aufgabe:** Implementieren Sie ein einfaches Event-System.

**Lösung:**

```
1 public class EventBus {
2     private Map<String, List<EventHandler>> handlers =
3         new HashMap<>();
4
5     public void subscribe(String event, EventHandler
6         handler) {
7         handlers.computeIfAbsent(event, k -> new
8             ArrayList<>())
9             .add(handler);
10
11     }
12
13     public void publish(String event, String data) {
14         if (handlers.containsKey(event)) {
15             handlers.get(event)
16                 .forEach(handler ->
17                     handler.handle(data));
18         }
19     }
20
21     // Verwendung
22     EventBus bus = new EventBus();
23     bus.subscribe("userLogin", data ->
24         System.out.println("User logged in: " + data));
25     bus.publish("userLogin", "john_doe");
26 }
```

## JDBC Basisbeispiel

```
1 import java.sql.*;
2
3 public class DbTest {
4     public static void main(String[] args)
5         throws SQLException {
6         // Verbindung aufbauen
7         Connection con = DriverManager.getConnection(
8             "jdbc:postgresql://test.zhaw.ch/testdb",
9             "user", "password");
10
11         // Statement erstellen und ausfuehren
12         Statement stmt = con.createStatement();
13         ResultSet rs = stmt.executeQuery(
14             "SELECT * FROM test ORDER BY name");
15
16         // Ergebnisse verarbeiten
17         while (rs.next()) {
18             System.out.println(
19                 "Name: " + rs.getString("name"));
20         }
21
22         // Aufräumen
23         rs.close();
24         stmt.close();
25         con.close();
26     }
27 }
```

## DAO Implementation

```
1 public interface ArticleDAO {
2     void insert(Article item);
3     void update(Article item);
4     void delete(Article item);
5     Article findById(int id);
6     Collection<Article> findAll();
7     Collection<Article> findByName(String name);
8 }
9
10 public class Article {
11     private long id;
12     private String name;
13     private float price;
14
15     // Getter/Setter
16 }
17
18 public class JdbcArticleDAO implements
19     ArticleDAO {
20     private Connection conn;
21
22     public void insert(Article item) {
23         PreparedStatement stmt =
24             conn.prepareStatement(
25                 "INSERT INTO articles (name, price)
26                 VALUES (?, ?)");
27         stmt.setString(1, item.getName());
28         stmt.setFloat(2, item.getPrice());
29         stmt.executeUpdate();
30     }
31
32     // weitere Implementierungen
33 }
```

## Parent-Child Beziehung mit JPA

```
1 @Entity
2 public class Department {
3     @Id @GeneratedValue
4     private Long id;
5
6     private String name;
7
8     @OneToMany(mappedBy = "department")
9     private List<Employee> employees;
10 }
11
12 @Entity
13 public class Employee {
14     @Id @GeneratedValue
15     private Long id;
16
17     @ManyToOne
18     @JoinColumn(name = "department_id")
19     private Department department;
20
21     private String name;
22     private double salary;
23 }
```

## Spring Data Repository

```
1 @Repository
2 public interface SaleRepository
3     extends CrudRepository<Sale, String> {
4
5     List<Sale> findOrderByDateTime();
6
7     List<Sale> findByDateTime(
8         final LocalDateTime dateTime);
9 }
10
11 @Service
12 public class ProcessSaleHandler {
13     private final ProductDescriptionRepository catalog;
14     private final SaleRepository saleRepository;
15
16     @Transactional
17     public void endSale() {
18         assert(currentSale != null
19             && !currentSale.isComplete());
20         this.currentSale.becomeComplete();
21         this.saleRepository.save(currentSale);
22     }
23 }
```

## Abstract Factory: POS Terminal

```
1 public interface IJavaPOSDevicesFactory {
2     CashDrawer getNewCashDrawer();
3     CoinDispenser getNewCoinDispenser();
4     // weitere Methoden
5 }
6
7 public class IBMJavaPOSDevicesFactory
8     implements IJavaPOSDevicesFactory {
9     public CashDrawer getNewCashDrawer() {
10         return new com.ibm.pos.jpos.CashDrawer();
11     }
12     // weitere Implementierungen
13 }
```

## Command: Persistenz

```
1 public interface ICommand {
2     void execute();
3     void undo();
4 }
5
6 public class DBUpdateCommand implements ICommand {
7     private PersistentObject object;
8
9     public void execute() {
10         // Update in Datenbank
11     }
12
13     public void undo() {
14         // Änderung rückgaengig machen
15     }
16 }
```

## Template Method: GUI Framework

```
1 public abstract class GUIComponent {
2     // Template Method
3     public final void update() {
4         clearBackground();
5         repaint(); // Hook Method
6     }
7
8     protected abstract void repaint();
9 }
10
11 public class MyButton extends GUIComponent {
12     protected void repaint() {
13         // Button-spezifische Implementation
14     }
15 }
```

## Spring Data Repository

```
1 @Repository
2 public interface UserRepository
3     extends JpaRepository<User, Long> {
4     // Methode wird automatisch implementiert
5     List<User> findByLastNameOrderByFirstNameAsc(
6         String lastName);
7
8     // SQL-Query via Annotation
9     @Query("SELECT u FROM User u WHERE u.active =
10         true")
11     List<User> findActiveUsers();
12 }
```