ADAPTER

// Adapter für MP3Player

class MP3PlayerAdapter implements AudioPlayer {

private MP3Player mp3Player;

public MP3PlayerAdapter(MP3Player mp3Player) {

this.mp3Player = mp3Player;

}

@Override

public void playAudio(String audioType, String fileName) {

if (audioType.equalsIgnoreCase("mp3")) {

mp3Player.playMP3(fileName);

} else {

System.out.println("Unsupported audio type: " + audioType);

}}}

// Adapter für WAVPlayer

class WAVPlayerAdapter implements AudioPlayer {

private WAVPlayer wavPlayer;

public WAVPlayerAdapter(WAVPlayer wavPlayer) {

this.wavPlayer = wavPlayer;

}

@Override

public void playAudio(String audioType, String fileName) {

if (audioType.equalsIgnoreCase("wav")) {

wavPlayer.playWAV(fileName);

} else {

System.out.println("Unsupported audio type: " + audioType);

}}}

public class Main {

public static void main(String[] args) {

AudioPlayer mp3PlayerAdapter = new MP3PlayerAdapter(new MP3Player());

mp3PlayerAdapter.playAudio("mp3", "song.mp3");

AudioPlayer wavPlayerAdapter = new WAVPlayerAdapter(new WAVPlayer());

wavPlayerAdapter.playAudio("wav", "sound.wav");

}}

FACADE

// Zimmerverfügbarkeits-Subsystem

class RoomAvailability {

boolean checkAvailability(int roomNumber) {

// Hier würde die Logik stehen, um die Verfügbarkeit des Zimmers zu überprüfen

return roomNumber % 2 == 0; // Beispiel: Gerade Zimmer sind verfügbar, ungerade Zimmer sind nicht verfügbar

}}

// Buchungsverwaltungs-Subsystem

class BookingManagement {

void bookRoom(int roomNumber) {

// Hier würde die Logik stehen, um das Zimmer zu buchen

System.out.println("Booking room: " + roomNumber);}}

// Zahlungsabwicklungs-Subsystem

class PaymentProcessing {

void processPayment(double amount) {

// Hier würde die Logik stehen, um die Zahlung abzuwickeln

System.out.println("Processing payment of amount: " + amount);}}

// HotelBookingFacade Klasse

class HotelBookingFacade {

private RoomAvailability roomAvailability;

private BookingManagement bookingManagement;

private PaymentProcessing paymentProcessing;

public HotelBookingFacade() {

this.roomAvailability = new RoomAvailability();

this.bookingManagement = new BookingManagement();

this.paymentProcessing = new PaymentProcessing(); }

public void bookRoom(int roomNumber, double amount) {

if (roomAvailability.checkAvailability(roomNumber)) {

bookingManagement.bookRoom(roomNumber);

paymentProcessing.processPayment(amount);

System.out.println("Room booked successfully.");

} else {

System.out.println("Room " + roomNumber + " is not available.");

} }}

public class Main {

public static void main(String[] args) {

HotelBookingFacade hotelBookingFacade = new HotelBookingFacade();

hotelBookingFacade.bookRoom(101, 200); // Beispiel: Zimmer 101 reservieren für 200 Einheiten

}}

ITERATOR

// Iterator Schnittstelle

interface Iterator {

boolean hasNext();

Object next();

}

// Konkreter Iterator für MyCollection

class MyCollectionIterator implements Iterator {

private String[] elements;

private int position;

public MyCollectionIterator(String[] elements) {

this.elements = elements;

this.position = 0;

}

@Override

public boolean hasNext() {

return position < elements.length;

}

@Override

public Object next() {

if (hasNext()) {

return elements[position++]; }

return null;

}}

// MyCollection Klasse

class MyCollection {

private String[] elements;

public MyCollection(String[] elements) {

this.elements = elements;

}

public Iterator createIterator() {

return new MyCollectionIterator(elements);

}}

public class Main {

public static void main(String[] args) {

String[] data = {"A", "B", "C", "D", "E"};

MyCollection collection = new MyCollection(data);

// Erzeugen des Iterators

Iterator iterator = collection.createIterator();

// Iteration durch die Elemente

while (iterator.hasNext()) {

String element = (String) iterator.next();

System.out.println("Element: " + element);

} }}