**SOLID Principles assignment document:**

1. **Single Responsibility :**

**// Class to book restaurant**

**package com.solid.entity;**

**import com.solid.interfaces.Bookable;**

**public class Restaurant extends Hospitality implements Bookable {**

**String restId;**

**String restType;**

**public Restaurant(String restId, String restType,String name,String place) {**

**super();**

**this.restId = restId;**

**this.restType = restType;**

**this.name = name;**

**this.place = place;**

**}**

**@Override**

**public void book(int identifier) {**

**System.*out*.println("Table " + identifier + " reserved in the restaurant.");**

**System.*out*.println("Restaurant type is " + this.restType);**

**System.*out*.println("Restaurant name is " + this.name);**

**System.*out*.println("Restaurant place is " + this.place);**

**}**

**}**

**// Class to book hotel**

**package com.solid.entity;**

**import com.solid.interfaces.Bookable;**

**import com.solid.interfaces.ConferenceHall;**

**public class Hotel extends Hospitality implements Bookable,ConferenceHall {**

**String hotelId;**

**String hotelType;**

**public Hotel(String hotelId, String hotelType,String name, String place) {**

**super();**

**this.hotelId = hotelId;**

**this.hotelType = hotelType;**

**this.name = name;**

**this.place = place;**

**}**

**@Override**

**public void book(int identifier) {**

**System.*out*.println("Room " + identifier + " booked in the hotel.");**

**System.*out*.println("Hotel type is " + this.hotelType);**

**System.*out*.println("Hotel name is " + this.name);**

**System.*out*.println("Hotel place is " + this.place);**

**}**

**@Override**

**public void bookConferenceHall() {**

**System.*out*.println("Conference hall is booked");**

**}**

**}**

**Here 2 different classes are used for booking a hotel / restaurant both with a single functionality only.**

1. **Open/Closed Principle :**

**package com.solid.entity;**

**abstract public class Hospitality {**

**String name;**

**String place;**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getPlace() {**

**return place;**

**}**

**public void setPlace(String place) {**

**this.place = place;**

**}**

**}**

**//Here the hospitality class has been extended to form hotel and restaurant classes.**

**3.Liskov Substitution Principle :**

**package com.solid.entity;**

**abstract public class Hospitality {**

**String name;**

**String place;**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getPlace() {**

**return place;**

**}**

**public void setPlace(String place) {**

**this.place = place;**

**}**

**}**

**package com.solid.entity;**

**import com.solid.interfaces.Bookable;**

**public class Restaurant extends Hospitality implements Bookable {**

**String restId;**

**String restType;**

**public Restaurant(String restId, String restType,String name,String place) {**

**super();**

**this.restId = restId;**

**this.restType = restType;**

**this.name = name;**

**this.place = place;**

**}**

**@Override**

**public void book(int identifier) {**

**System.*out*.println("Table " + identifier + " reserved in the restaurant.");**

**System.*out*.println("Restaurant type is " + this.restType);**

**System.*out*.println("Restaurant name is " + this.name);**

**System.*out*.println("Restaurant place is " + this.place);**

**}**

**}**

**package com.solid.entity;**

**import com.solid.interfaces.Bookable;**

**import com.solid.interfaces.ConferenceHall;**

**public class Hotel extends Hospitality implements Bookable,ConferenceHall {**

**String hotelId;**

**String hotelType;**

**public Hotel(String hotelId, String hotelType,String name, String place) {**

**super();**

**this.hotelId = hotelId;**

**this.hotelType = hotelType;**

**this.name = name;**

**this.place = place;**

**}**

**@Override**

**public void book(int identifier) {**

**System.*out*.println("Room " + identifier + " booked in the hotel.");**

**System.*out*.println("Hotel type is " + this.hotelType);**

**System.*out*.println("Hotel name is " + this.name);**

**System.*out*.println("Hotel place is " + this.place);**

**}**

**@Override**

**public void bookConferenceHall() {**

**System.*out*.println("Conference hall is booked");**

**}**

**}**

**// Here the hotel and restaurant classes are extended from the hospitality class and they implement a ‘Bookable’ interface. Book method is defined in the Bookable interface and both the hotel and restaurant class provide their own implementation of the ‘book’ method.**

**4. Interface Segregation Principle :**

**public interface Bookable {**

**void book(int identifier);**

**}**

**public interface ConferenceHall {**

**public void bookConferenceHall();**

**}**

**//The interface ‘Bookable’ is used for booking hotel/ restaurant and ‘ConferenceHall’ is used for booking conference hall in a hotel.**

**5. Dependency Inversion :**

**package com.solid.utility;**

**import com.solid.entity.Hotel;**

**import com.solid.entity.Restaurant;**

**import com.solid.services.BookingSystem;**

**public class Main {**

**public static void main(String[] args) {**

**Hotel hotel = new Hotel("001","2 star", "Taj", "Kochi");**

**Restaurant restaurant = new Restaurant("002","Chinese","Kanthari","Trivandrum");**

**BookingSystem bookingSystem = new BookingSystem();**

**// Example bookings**

**bookingSystem.makeBooking(hotel,101); // Book room in the hotel**

**bookingSystem.makeBooking(restaurant,5); // Reserve table in the restaurant**

**hotel.bookConferenceHall();**

**}**

**}**

**package com.solid.services;**

**import com.solid.interfaces.Bookable;**

**public class BookingSystem {**

**public void makeBooking(Bookable place,int identifier) {**

**place.book(identifier);**

**}**

**}**

**//In the provided code, the Dependency Inversion Principle (DIP) is applied through the BookingSystem class and the ‘Bookable’ interface. The BookingSystem class depends on the abstraction (Bookable) rather than concrete implementations. This allows for flexibility and extensibility, as different types of objects implementing the Bookable interface can be passed to the makeBooking method without changing the high-level logic. This adherence to abstraction and decoupling enhances maintainability, extensibility, and testability of the code.**