

# SOLID Documentation

ARJUN S

January 2024

## 1 LISKOV SUBSTITUTION PRINCIPLE

Base class:

```
1
2 public class FoodRecommendation {
3     public List<String> recommendFood() {
4         return List.of();
5     }
6 }
```

Child classes :

```
1
2 public class UserBasedFoodRecommendation extends
    FoodRecommendation {
3     @Override
4     public
5     List<String> recommendFood() {
6         return List.of("Pizza", "Burger", "Salad");
7     }
}
```

```
1
2 public class AIModelBasedFoodRecommendation extends
    UserBasedFoodRecommendation {
3     //using function of the UserBasedFoodRecommendation
4 }
```

```

1
2 public class NewRecommendation extends
   FoodRecommendation { //for new users to avoid cold
   start problem
3 @Override
4 public List<String> recommendFood() {
5     System.out.println("This is a new method specific
       to NewRecommendation.");
6     return List.of("New Food 1", "New Food 2", "New
       Food 3");
7 }
8 }

```

## 2 INTERFACE SEGREGATION PRINCIPLE

Interface:

```

1
2
3 public interface BasicFoodBooking {
4     void bookFood(String foodItem, int quantity, Date
       deliveryDate);
5
6     //void bookSpecialFood(String specialFoodItem, int
       quantity, Date deliveryDate);
7     //If i add bookSpecialFood function also here it
       violates ISP.
8 }

```

```

1 import java.util.Date;
2
3 public
4 interface VIPFoodBooking {
5     void bookSpecialFood(String specialFoodItem, int
       quantity, Date deliveryDate);
6 }

```

### Implementation classes

```
1 import java.util.Date;
2
3 public
4 class RegularFoodBooking implements BasicFoodBooking {
5     @Override
6     public void bookFood(String foodItem, int
7         quantity, Date deliveryDate) {
8         System.out.println("Regular_user_booked_" +
9             quantity + "_" + foodItem + "(s)_for_"
10             delivery_on_" + deliveryDate);
11     }
12 }
```

```
1 import java.util.Date;
2
3 public
4 class VIPFoodBookingImpl implements VIPFoodBooking {
5     @Override
6     public void bookSpecialFood(String
7         specialFoodItem, int quantity, Date
8         deliveryDate) {
9
10         System.out.println("VIP_user_booked_" +
11             quantity + "_" + specialFoodItem + "(s)_
12             for_delivery_on_" + deliveryDate);
13     }
14 }
```

### 3 DEPENDENCY INVERSION PRINCIPLE

INTERFACE

```
1
2 import java.util.Date;
3
4 public
5
6 interface FoodBooking {
7     void bookFood(String foodItem, int quantity, Date
        deliveryDate);
8 }
```

```
1
2 import java.util.Date;
3
4 public
5 class RegularFoodBooking implements FoodBooking {
6     @Override
7     public void bookFood(String foodItem, int
        quantity, Date deliveryDate) {
8
9         System.out.println("Regular_user_booked_" +
            quantity + "_" + foodItem + "(s)_for_"
            delivery_on_" + deliveryDate);
10    }
11 }
```

```
1
2 import java.util.Date;
3
4 public
5 class VIPFoodBookingImpl implements FoodBooking {
6     @Override
7     public void bookFood(String foodItem, int
        quantity, Date deliveryDate) {
8
9         System.out.println("VIP_user_booked_" +
            quantity + "_" + foodItem + "(s)_for_"
            delivery_on_" + deliveryDate);
10    }
11 }
```

Main page

```
1 import java.util.Date;
2
3 import com.ilp.service.*;
4
5 public class FoodBookingUtility {
6
7     public static void main(String[] args) {
8
9         FoodBooking regularBooking = new
10             RegularFoodBooking();
11         regularBooking.bookFood("Pizza", 2, new
12             Date());
13
14         FoodBooking vipBooking = new
15             VIPFoodBookingImpl();
16         vipBooking.bookFood("Exclusive_Meal", 1, new
17             Date());
18
19         // RegularFoodBooking regularFoodBooking = new
20         // RegularFoodBooking();
21         // regularFoodBooking.bookFood("Pizza", 2, new
22         // Date());
23         // this code violates the DIP
24     }
25 }
```

## 4 OPEN CLOSE PRINCIPLE

Interface

```
1
2 package com.ilp.service;
3
4 import java.util.Date;
5
6 public interface FoodBooking {
7     void bookFood(String foodItem, int quantity, Date
8         deliveryDate);
9
10    //public void VegBookFood(String foodItem, int
11    //    quantity, Date deliveryDate) {}
12
13    //public void NonVegBookFood(String foodItem, int
14    //    quantity, Date deliveryDate) {}
15 }
```

```
12 }
```

Implementation of interface

```
1
2 import java.util.Date;
3
4 public class NonVegFoodBooking implements FoodBooking
5 {
6     @Override
7     public void bookFood(String foodItem, int quantity,
8         Date deliveryDate) {
9
10         System.out.println("Non_veg_Food_Booking: " +
11             "Item: " + foodItem +
12             ", Quantity: " + quantity +
13             ", Delivery Date: " + deliveryDate);
14     }
15 }
```

```
1 package com.ilp.service;
2
3 import java.util.Date;
4
5 public class VegFoodBooking implements FoodBooking {
6     @Override
7     public void bookFood(String foodItem, int quantity,
8         Date deliveryDate) {
9         System.out.println("Regular_Food_Booking: " +
10             "Item: " + foodItem +
11             ", Quantity: " + quantity +
12             ", Delivery Date: " + deliveryDate);
13     }
14
15     // public void VegBookFood(String foodItem, int
16     // quantity, Date deliveryDate) {
17     //     System.out.println("Regular Food Booking: " +
18     //         "Item: " + foodItem +
19     //         ", Quantity: " + quantity +
20     //         ", Delivery Date: " + deliveryDate);
21     // }
22
23     // public void NonVegBookFood(String foodItem, int
24     // quantity, Date deliveryDate) {
25     //     System.out.println("Regular Food Booking: " +
26     //         "Item: " + foodItem +
```

```
24 //           ", Quantity: " + quantity +  
25 //           ", Delivery Date: " + deliveryDate);  
26 // }  
27  
28 //violates Open Close principle  
29 }
```

## 5 SINGLE RESPONSIBILITY PRINCIPLE

All the above programs follows Single Responsibility principle.