SOLID PRINCIPLES

My project was centered around “lamborghinistore.com”.

The set of Classes and interfaces chosen for implementing SOLID Principles are as given below:-

- Address
- AddressBook
- Product
- AddressRepositoryServices
- CartServices
- DeliveryServices
- InternationalAddress
- WishlistServices
- AddressRepository (Interface)
- CartOperations (Interface)
- WishlistOperations (Interface)

Single Responsibility Principle (SRP)

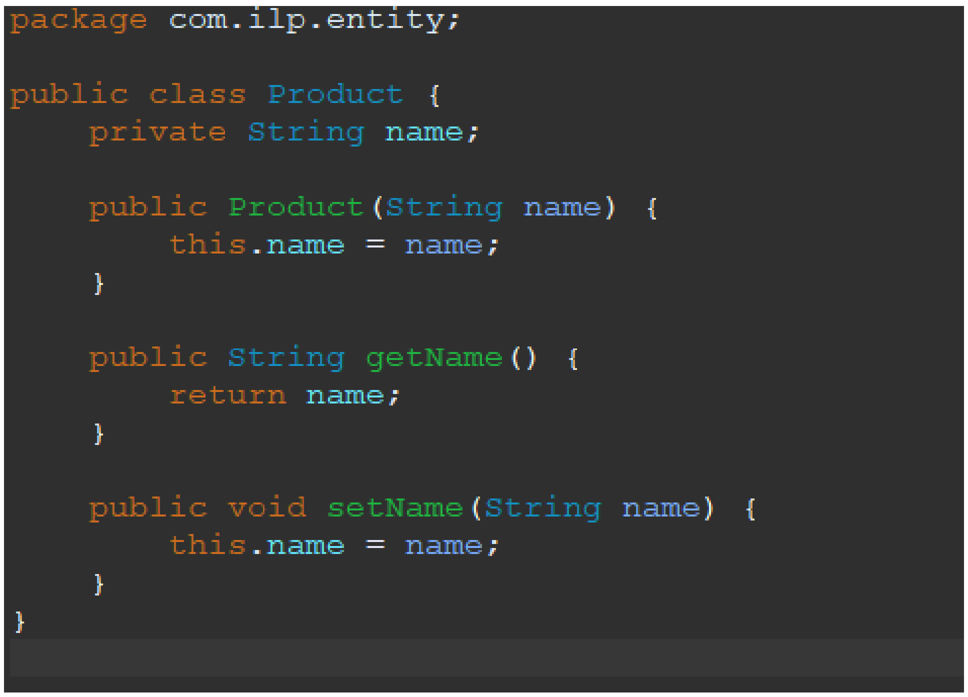
Statement: - The single responsibility principle (SRP) states that Every module (such as a class, function, or microservice) should have one and only one reason to change.

- Product: Represents a product with its name.
- Address: Represents a general address with its components.
- AddressRepository: Defines operations for managing addresses (adding, retrieving).

- CartOperations: Defines operations for managing a cart (adding items).
- WishlistOperations: Defines operations for managing a wishlist (adding items).
- AddressRepositoryServices: Implements the address repository functionality using a list.
- CartServices: Implements cart operations (adding to cart).
- WishlistServices: Implements wishlist operations (adding to wishlist).
- DeliveryAddress: Extends Address to add delivery instructions.
- InternationalAddress: Extends Address to add a country code and specialized validation.

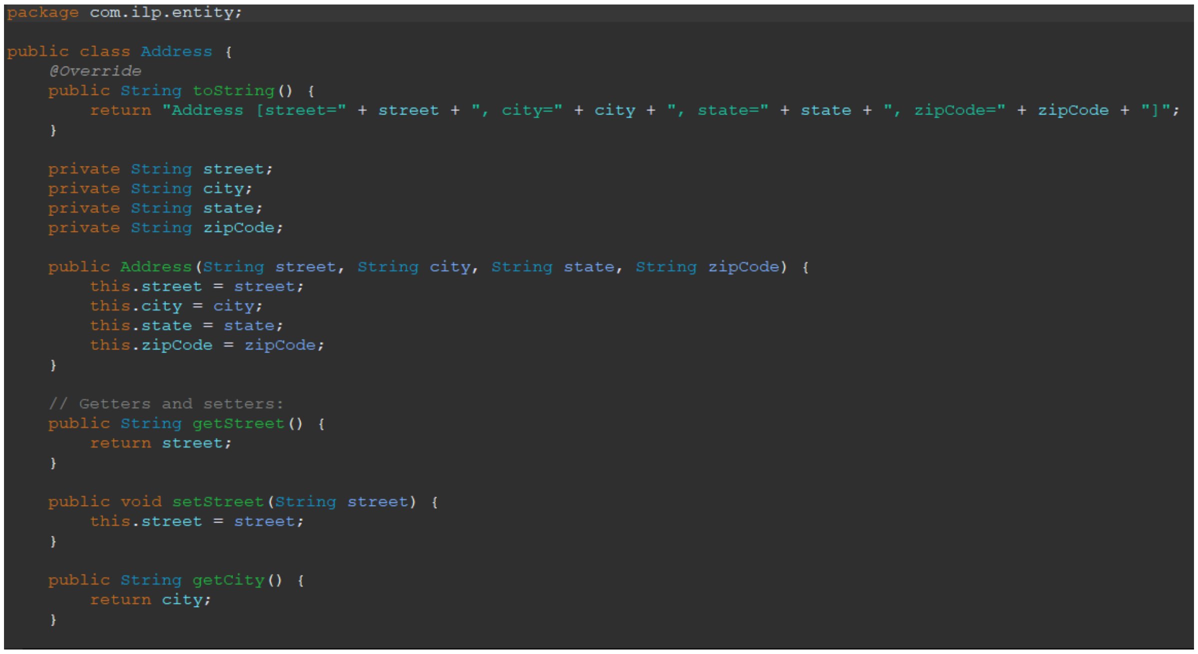
Examples -

Product.java

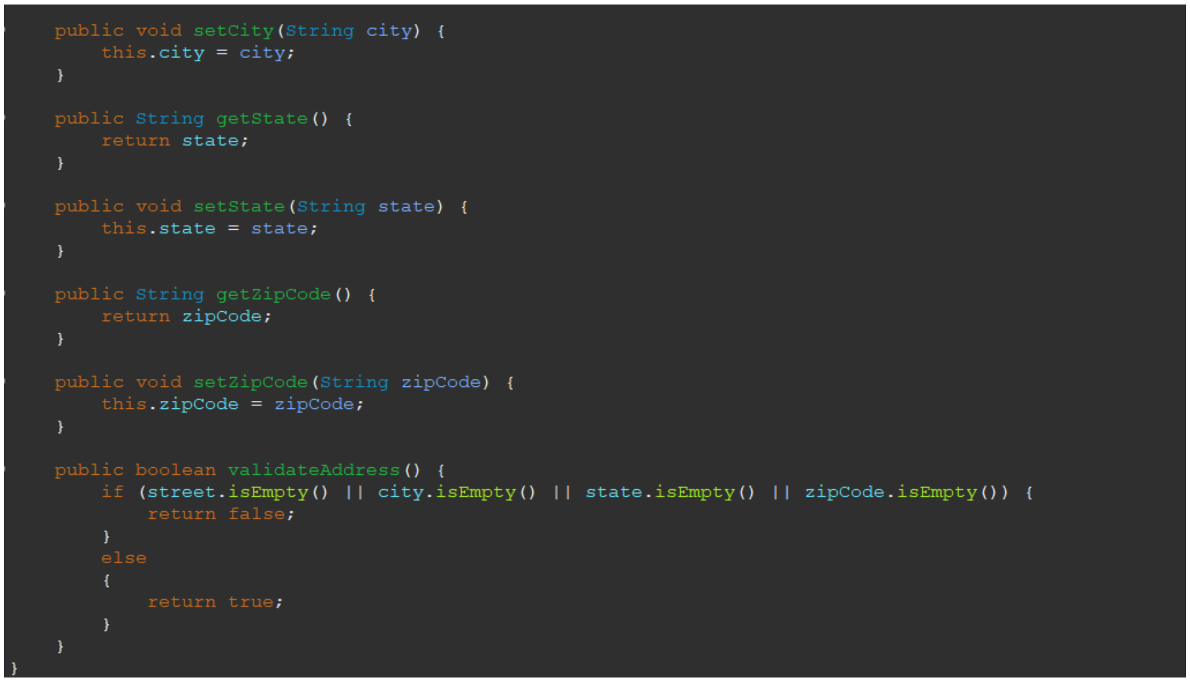


```
package com.ilp.entity;

public class Product {
    private String name;

    public Product(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Address.java



```java
package com.ilp.entity;

public class Address {
    @Override
    public String toString() {
        return "Address [street=" + street + ", city=" + city + ", state=" + state + ", zipCode=" + zipCode + "]";
    }

    private String street;
    private String city;
    private String state;
    private String zipCode;

    public Address(String street, String city, String state, String zipCode) {
        this.street = street;
        this.city = city;
        this.state = state;
        this.zipCode = zipCode;
    }

    // Getters and setters:
    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }
```



```java
    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getZipCode() {
        return zipCode;
    }

    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }

    public boolean validateAddress() {
        if (street.isEmpty() || city.isEmpty() || state.isEmpty() || zipCode.isEmpty()) {
            return false;
        }
        else
        {
            return true;
        }
    }
}
```

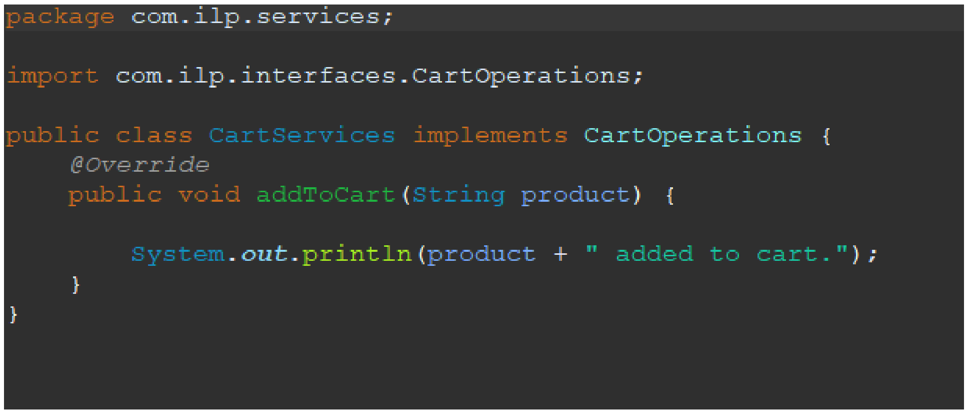
Here Product class is used to store the details related to a Product and hence follows Single Responsibility. Similarly, Address class is used to store data related to address and hence follows Single Responsibility.

Open - Closed Principle (OCP)

Statement: - Classes should be Open for Extension but Closed for modification.

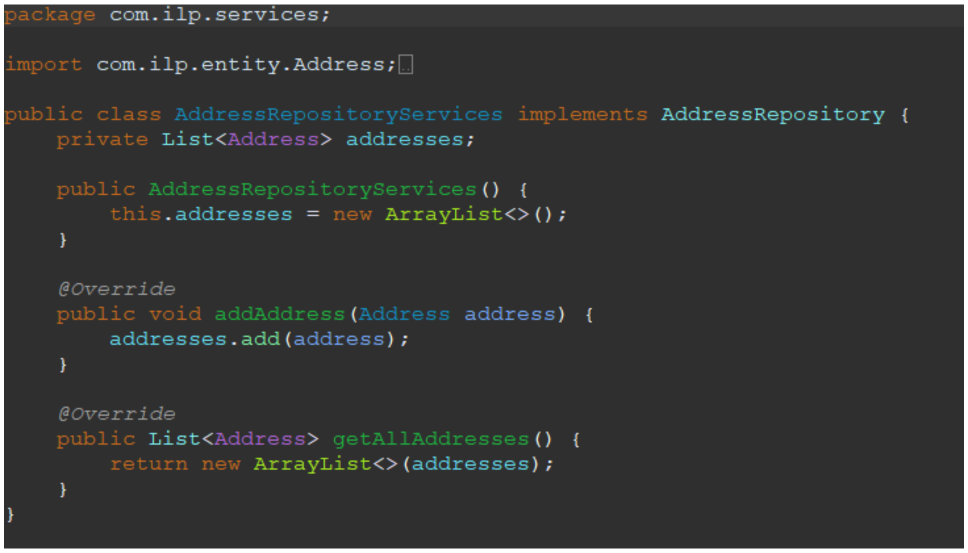
- AddressRepository - It defines an interface for managing addresses, allowing different implementations without modifying client code.
- Address - It's designed for extension through inheritance.
- CartOperations - They define interfaces for cart operations, enabling different implementations and future extensions.
- CartOperations and WishlistOperations - They define interfaces for wishlist operations, enabling different implementations and future extensions.
- AddressBook - It depends on the AddressRepository interface, allowing address storage to be swapped without changing AddressBook itself.

CartServices.java



```java
package com.ilp.services;

import com.ilp.interfaces.CartOperations;

public class CartServices implements CartOperations {
    @Override
    public void addToCart(String product) {

        System.out.println(product + " added to cart.");
    }
}
```

AddressRepositoryServices.java



```java
package com.ilp.services;

import com.ilp.entity.Address;

public class AddressRepositoryServices implements AddressRepository {
    private List<Address> addresses;

    public AddressRepositoryServices() {
        this.addresses = new ArrayList<>();
    }

    @Override
    public void addAddress(Address address) {
        addresses.add(address);
    }

    @Override
    public List<Address> getAllAddresses() {
        return new ArrayList<>(addresses);
    }
}
```

Both the classes follow open for extension but closed for modification.

Liskov's Substitution Principle (LSP)

Statement: - Objects of a superclass should be replaceable with objects of its subclasses without altering the correctness of any program that uses objects of that superclass.

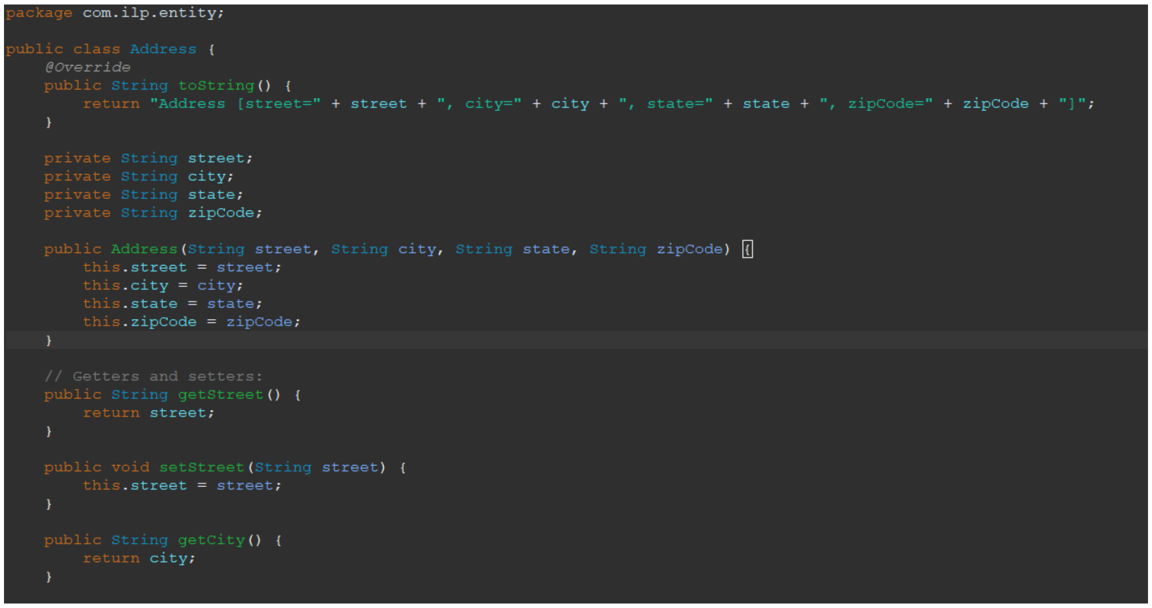
- Address, DeliveryAddress, InternationalAddress:

  They inherit the validateAddress() method without altering its behavior.

  They introduce new fields and methods, but they don't change the expected functionality of the base class.

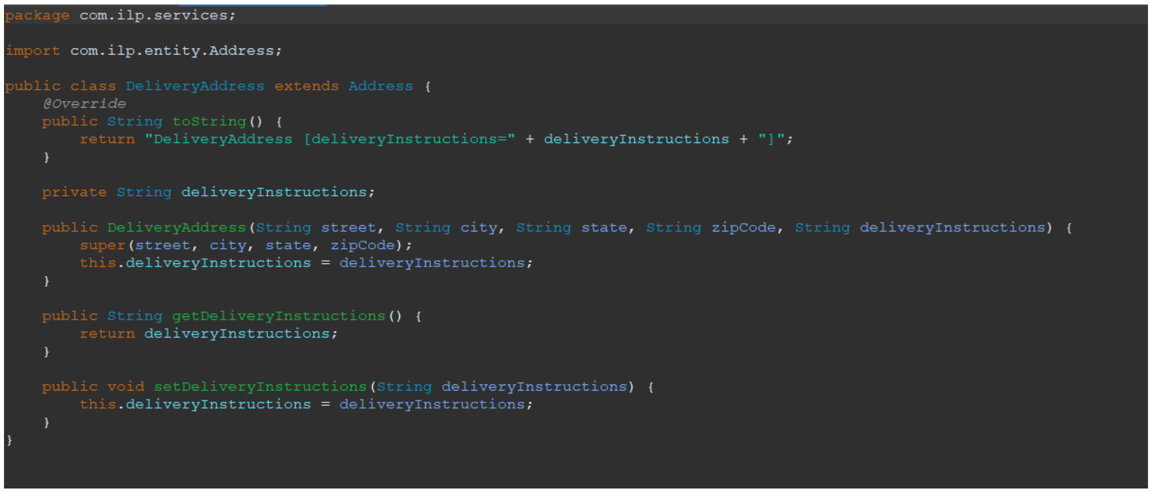
  We can substitute any of these subclasses for an Address object without unexpected issues.

  Address.java



```java
package com.ilp.entity;

public class Address {
    @Override
    public String toString() {
        return "Address [street=" + street + ", city=" + city + ", state=" + state + ", zipCode=" + zipCode + "]";
    }

    private String street;
    private String city;
    private String state;
    private String zipCode;

    public Address(String street, String city, String state, String zipCode) {
        this.street = street;
        this.city = city;
        this.state = state;
        this.zipCode = zipCode;
    }

    // Getters and setters:
    public String getStreet() {
        return street;
    }

    public void setStreet(String street) {
        this.street = street;
    }

    public String getCity() {
        return city;
    }
```

DeliveryAddress.java



```java
package com.ilp.services;

import com.ilp.entity.Address;

public class DeliveryAddress extends Address {
    @Override
    public String toString() {
        return "DeliveryAddress [deliveryInstructions=" + deliveryInstructions + "]";
    }

    private String deliveryInstructions;

    public DeliveryAddress(String street, String city, String state, String zipCode, String deliveryInstructions) {
        super(street, city, state, zipCode);
        this.deliveryInstructions = deliveryInstructions;
    }

    public String getDeliveryInstructions() {
        return deliveryInstructions;
    }

    public void setDeliveryInstructions(String deliveryInstructions) {
        this.deliveryInstructions = deliveryInstructions;
    }
}
```

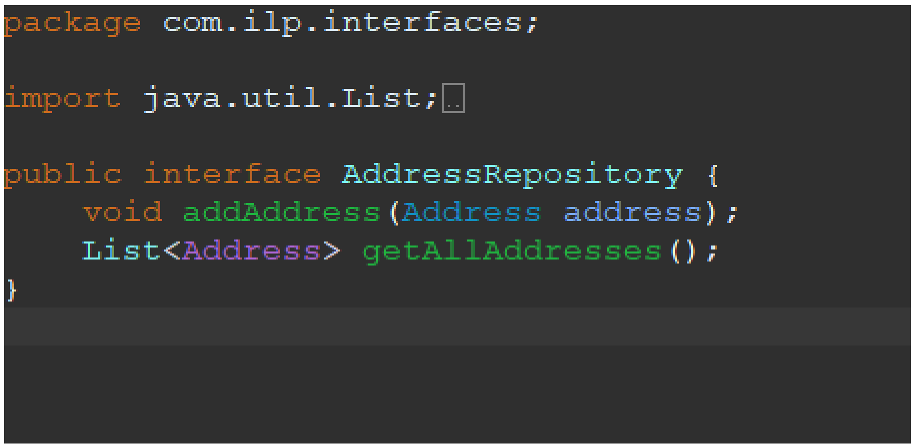
Interface Segregation Principle (ISP)

Statement: - No code should be forced to depend on methods it does not use.

- AddressRepository interface - It has a focused set of methods (addAddress, getAllAddresses) related to address management. Classes implementing it only need to implement these specific methods.

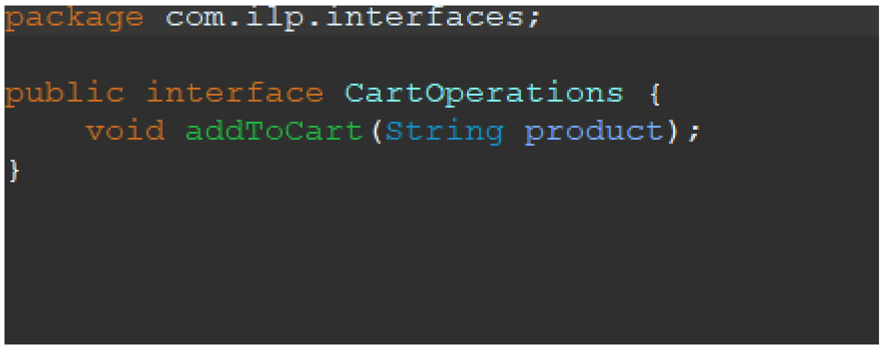
- CartOperations interface - It has a single method (addToCart), ensuring classes only depend on cart-related functionality.
- WishlistOperations interface- It has a single method (addToWishlist), ensuring classes only depend on wishlist-related functionality.

AddressRepositry.java



```
package com.ilp.interfaces;

import java.util.List;

public interface AddressRepository {
    void addAddress(Address address);
    List<Address> getAllAddresses();
}
```

CartOperations.java



```
package com.ilp.interfaces;

public interface CartOperations {
    void addToCart(String product);
}
```

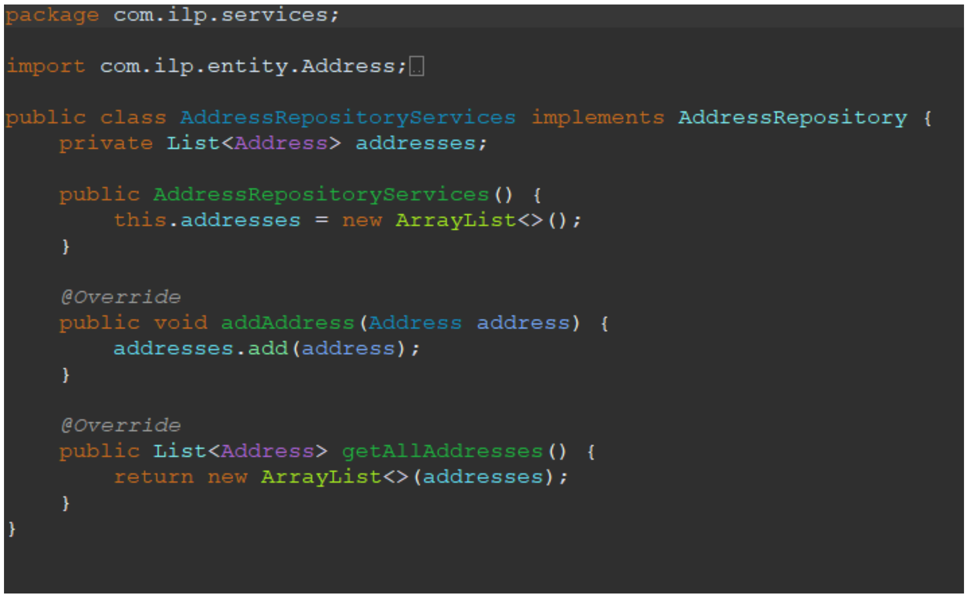
When it comes to the declared operations, these interfaces are focused and precise. The AddressRepository interface can be implemented by a class that requires address-related operations, while the CartOperations interface can be implemented by a class that needs cart-related operations.

Dependency Inversion Principle (DIP)

Statement: - High-level modules should not depend on low-level modules. Both should depend on abstractions.

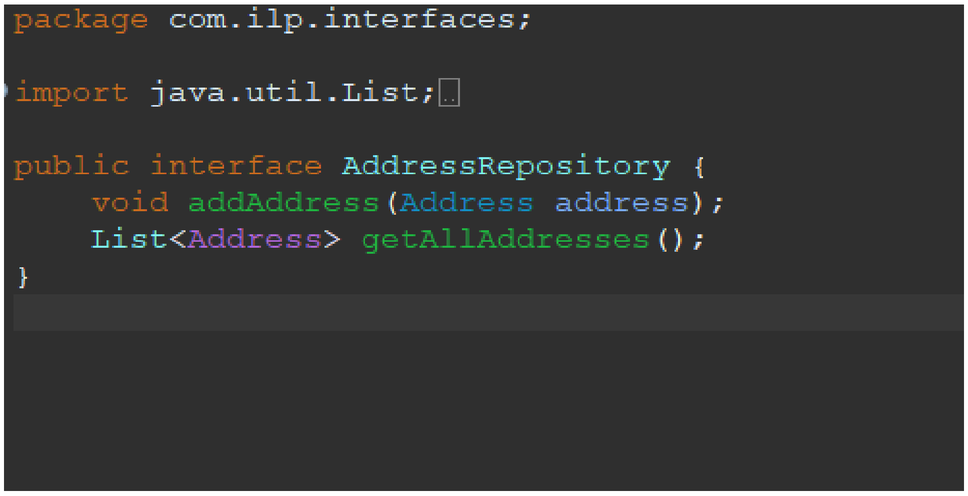
- AddressRepository: Defined for interacting with AddressRepositoryServices class, promoting abstraction and flexibility.
- CartOperations: Represents operations for CartServices class, enabling code to work with different cart implementations.
- WishlistOperations: Represents WishlistServices class, resulting in switching of wishlist implementations.

AddressRepositoryServices.java



```java
package com.ilp.services;

import com.ilp.entity.Address;

public class AddressRepositoryServices implements AddressRepository {
    private List<Address> addresses;

    public AddressRepositoryServices() {
        this.addresses = new ArrayList<>();
    }

    @Override
    public void addAddress(Address address) {
        addresses.add(address);
    }

    @Override
    public List<Address> getAllAddresses() {
        return new ArrayList<>(addresses);
    }
}
```

AddressRepository.java



```
package com.ilp.interfaces;

import java.util.List;

public interface AddressRepository {
    void addAddress(Address address);
    List<Address> getAllAddresses();
}
```