# SOLID PRINCIPLES ASSIGNMENT

## 1. Single Responsibility Principle (SRP):

- **AuthenticationServiceImpl:**
    - Responsible for authenticating a user based on the provided credentials.
    - Follows SRP by having a single responsibility: user authentication.
- **LoginService:**
    - Manages the authentication process using an Authenticator.
    - Follows SRP by handling user authentication without getting involved in the actual authentication logic.
- **SignUpValidator:**
    - Validates user email, password, and confirms sign-up email.
    - Follows SRP by handling validation concerns related to sign-up.
- Account**:**
    - **Has all attributes related to Account**
    - Follows SRP by handling all details related to account.

```
public class Account
{
    private String accountNo;
    private String accountName;
    public String getAccountNo() {
        return accountNo;
    }
    public void setAccountNo(String accountNo) {
        this.accountNo = accountNo;
    }
    public String getAccountName() {
        return accountName;
    }
    public void setAccountName(String accountName) {
        this.accountName = accountName;
    }

}
```

- **PlatinumAccountService, PremiumAccountService, SilverAccountService:**
    - Each service is responsible for creating a specific type of account.
    - Follow SRP by having a single responsibility: creating an account of a specific type.

## 2. Open/Closed Principle (OCP):

- **AccountType Interface:**
  - Defines the contract for creating an account.
  - Open for extension: New account types can be added by implementing this interface.
  - Closed for modification: Existing code using AccountType doesn't need to be modified to accommodate new account types.

```
package com.ilp.interfaces;

public interface AccountType {
    public void createAccount();
}
```

### 3. Liskov Substitution Principle (LSP):

- **Account, PlatinumAccountService, PremiumAccountService, SilverAccountService:**
  - Subtypes can be substituted for their base type (AccountType).
  - LSP is followed as each account service implements the createAccount method from the AccountType interface.

```
package com.ilp.service;

import com.ilp.entity.Account;

public class PlatinumAccountService extends Account implements AccountType
{


@Override
public void createAccount() {
    System.out.println("Platinum Account Created");
}
}
```

```
package com.ilp.service;

import com.ilp.entity.Account;

public class PremiumAccountService extends Account implements AccountType
{
    public void createAccount()
    {
        System.out.println("Premium account created");
    }
}
```

```
package com.ilp.service;

import com.ilp.entity.Account;

public class SilverAccountService extends Account implements AccountType
{
    public void createAccount()
    {
        System.out.println("Silver account created");
    }
}
```

### 4. Interface Segregation Principle (ISP):

- **AccountType Interface:**

- Contains a single method (createAccount) specific to its purpose.
- Follows ISP by not forcing implementing classes to provide methods they don't need.

### 5. Dependency Inversion Principle (DIP):

- **MainApplication:**
  - Depends on abstractions (interfaces: Authenticator, ValidateUserEmail, ValidatePassword, ValidateConfirmSignUpEmail, AccountType) rather than concrete implementations.
  - Allows for easy substitution of different implementations for these interfaces.

```
package com.ilp.interfaces;

public interface ValidatePassword
{
    void validatePassWord();
}
```

```
package com.ilp.interfaces;

public interface ValidateUserEmail
{
     void validateUserEmail();

}
```