✅ ISP (Interface Segregation Principle) – Explained with Case Study

> **Definition (ISP - from SOLID)**:
> *"Clients should not be forced to depend on interfaces they do not use."*

🏦 Real-World Case Study: Banking – Customer Services

🎯 Problem Statement:

A bank has multiple customer roles:

- **RegularCustomer** → Can view balance and withdraw.
- **LoanCustomer** → Can apply for a loan and check loan status.
- **CorporateCustomer** → Can view balance, get credit rating, and apply for bulk loans.

You initially create a **large interface** for all operations called `CustomerService`. But soon you'll see how this violates ISP.

❌ ISP Violation – Fat Interface

```
public interface CustomerService {
    void viewBalance();
    void withdraw();
    void applyLoan();
    void checkLoanStatus();
    void getCreditRating();
}
```

Now, every class must implement all methods, even if not needed:

```
public class RegularCustomer implements CustomerService {
    public void viewBalance() {
        System.out.println("Viewing balance");
    }

    public void withdraw() {
        System.out.println("Withdrawing funds");
    }

    public void applyLoan() {
        throw new UnsupportedOperationException("Not supported");
    }

    public void checkLoanStatus() {
        throw new UnsupportedOperationException("Not supported");
    }

    public void getCreditRating() {
        throw new UnsupportedOperationException("Not supported");
    }
```

```
}
```

This is bad because:

- Clients are forced to implement irrelevant methods.
- Future changes to the interface might break unrelated classes.

✅ ISP Compliant Design – Split Interfaces

Step 1: Split the fat interface

```
public interface BalanceService {
    void viewBalance();
}

public interface WithdrawService {
    void withdraw();
}

public interface LoanService {
    void applyLoan();
    void checkLoanStatus();
}

public interface CreditService {
    void getCreditRating();
}
```

Step 2: Implement only what you need

```
public class RegularCustomer implements BalanceService, WithdrawService {
    public void viewBalance() {
        System.out.println("Viewing balance");
    }

    public void withdraw() {
        System.out.println("Withdrawing funds");
    }
}

public class LoanCustomer implements LoanService {
    public void applyLoan() {
        System.out.println("Applying for loan");
    }

    public void checkLoanStatus() {
        System.out.println("Checking loan status");
    }
}

public class CorporateCustomer implements BalanceService, CreditService,
LoanService {
    public void viewBalance() {
        System.out.println("Corporate account balance");
    }

    public void getCreditRating() {
```

```
        System.out.println("Getting credit rating");
    }

    public void applyLoan() {
        System.out.println("Corporate loan applied");
    }

    public void checkLoanStatus() {
        System.out.println("Checking corporate loan status");
    }
}
```

✅ Benefits of ISP Applied:

Violation (Fat Interface)	ISP Compliant Version
Classes forced to override useless methods	Classes implement only what they need
`UnsupportedOperationException` risk	Strong contracts, cleaner design
Harder to maintain and extend	Easier to evolve with minimal impact

---

👩‍🏫 Teaching Tips:

- Show how maintenance becomes painful in the fat-interface version.
- Introduce a **new service (e.g., ForexService)** and demonstrate how it can be plugged in easily without breaking existing customers.
- Ask: "What happens if you remove a method from the fat interface vs. from a segregated one?"

🧠 Analogy:

Fat interface is like a buffet menu with 100 dishes, but you're forced to pay and pretend to eat all.
ISP is à la carte — choose only what you want.
