Dependency Injection

Usa Sammapun

Dependency Injection (DI)

- One of important software design principles and patterns
    - Loosely coupled code
    - Easy to change implementation
- Focus at composing objects (dependency)
    - A class doesn't instantiate depending objects within itself
        - No "new" keyword in class definition
    - But receive depending object via constructor or setter methods
        - —> "inject" dependency via constructor/setter
    - object instantiation now locates at only one place

Dependency Injection

- Dependency Injection (DI)
  - Specifically in the context of assembling dependencies between objects

inject

verb (SOMETHING NEW) UK US /ɪn'dʒekt/ [T]

Definition

to introduce something new that is necessary or helpful to a situation or process

*A large amount of money will have to be injected **into** the company if it is to survive.*

*I tried to inject a little humour **into** the meeting.*

(Definition of inject verb (SOMETHING NEW) from the Cambridge Advanced Learner's Dictionary)

Source: http://martinfowler.com/articles/injection.html
http://dictionary.cambridge.org/

Dependency Injection

- Dependency Injection is also known as Inversion of Control (IoC)
  - Instead of the object creates its own depending objects
  - It lets some other objects create and inject dependencies instead

ATM with no Dependency Injection

(create depending object within class)

https://github.com/ladyusa/atm

ATM with no Dependency Injection

```
public class BankAccount {

    private int id;
    private String name
    private double balance;

    // . . . code . . .
}
```

```
public class Customer {

    private int id;
    private String name
    private int pin;

    private BankAccount account;

    // . . . code . . .
}
```

ATM with no Dependency Injection

```
public class DataSourceDB {

    public Map<Integer,Customer> readCustomers() {

        // . . . code . . .

    }
}
```

ATM with no Dependency Injection

```
public class Bank {

    private String name
    private Map<Integer,Customer> customers;
    private DataSourceDB dataSource;

    public Bank(String name) {
        this.name = name;
        this.dataSource = new DataSourceDB();
        this.customers = dataSource.readCustomers();
    }

    // . . . code . . .

}
```

ATM with no Dependency Injection

```
public class ATM {
    private Bank bank;
    private Customer currentCustomer;

    public ATM() {
        this.bank = new Bank();
        this.currentCustomer = null;
    }

    // . . . code . . .
}
```

ATM with no Dependency Injection

```
public class AtmUI {

    private ATM atm;

    public AtmUI() {
        this.atm = new ATM();
    }

    // . . . code . . .
}
```

ATM with no Dependency Injection

```
public class Main {

    public static void main(String[] args) {

        AtmUI atmUI = new AtmUI();
        atmUI.run();
    }
}
```

ATM with Dependency Injection

(receive dependent object via constructor or setter)

https://github.com/ladyusa/atm-di

ATM with Dependency Injection

```
public class Bank {

    private String name
    private Map<Integer,Customer> customers;
    private DataSourceDB dataSource;

    public Bank(String name, DataSourceDB dataSource) {
        this.name = name;
        this.dataSource = dataSource;
        this.customers = dataSource.readCustomers();
    }

    // . . . code . . .

}
```

ATM with Dependency Injection

```
public class ATM {
    private Bank bank;
    private Customer currentCustomer;

    public ATM(Bank bank) {
        this.bank = bank;
        this.currentCustomer = null;
    }

    // . . . code . . .
}
```

ATM with Dependency Injection

```
public class AtmUI {

    private ATM atm;

    public AtmUI(ATM atm) {
        this.atm = atm;
    }

    // . . . code . . .
}
```

ATM with Dependency Injection

```
public class Main {

    public static void main(String[] args) {

        DataSourceDB dataSource = new DataSourceDB();
        Bank bank = new Bank("My Bank",dataSource);
        ATM atm = new ATM(bank);
        AtmUI atmUI = new AtmUI(atm);
        atmUI.run();

    }
}
```

Dependency Injection with Interface

(Change implementation easier --- Layer of Indirection)

https://github.com/ladyusa/atm-di-layer

No Layer of Indirection

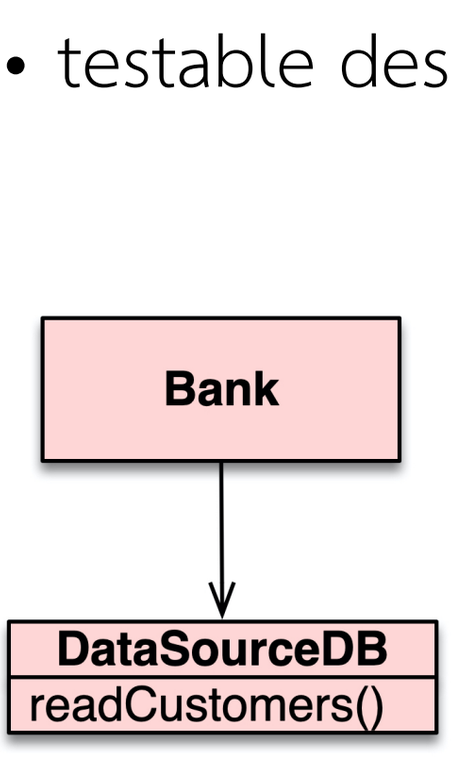
- If we want to change from reading database to files
  - Must change both class Bank and Main

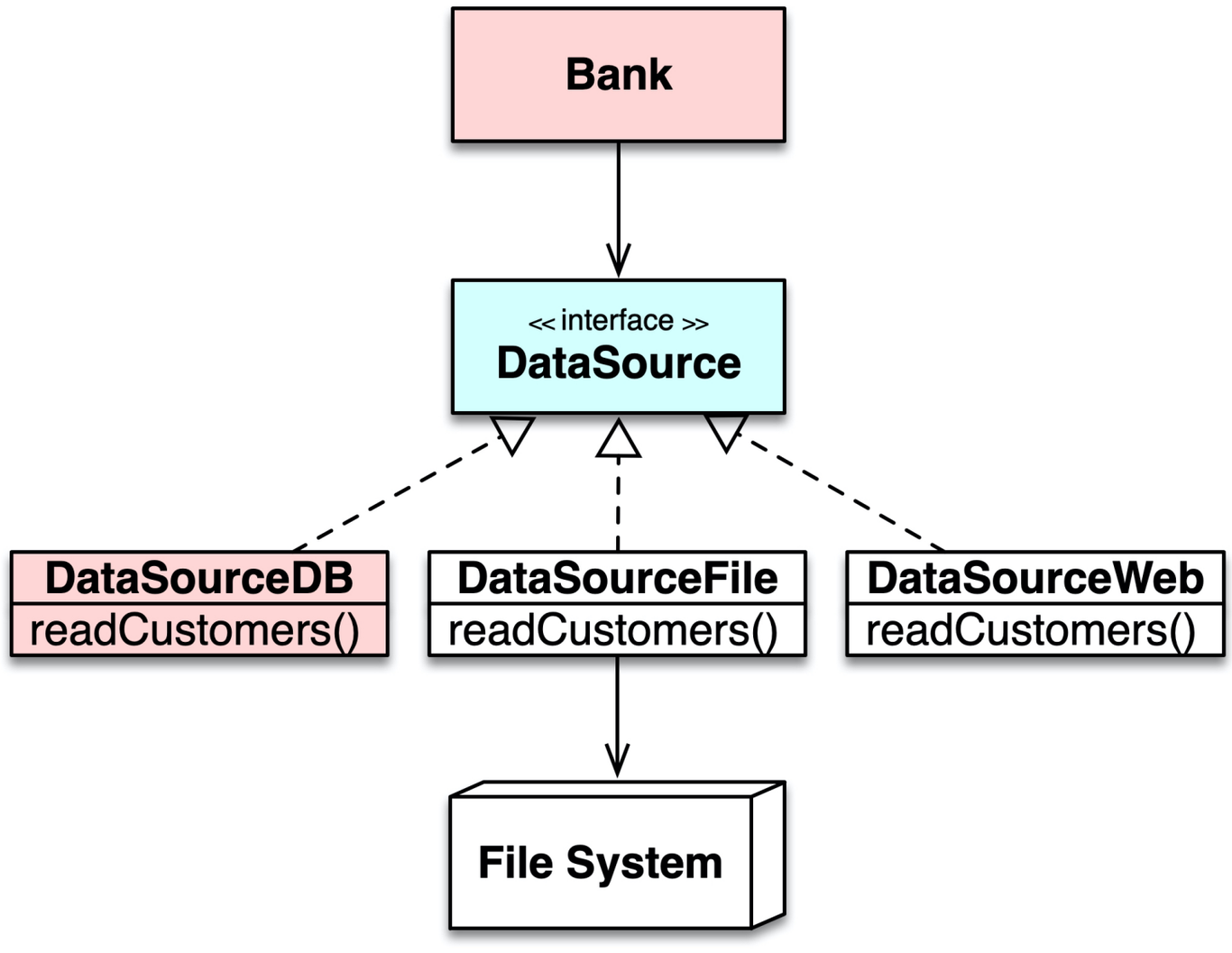
```
public class Bank {

   private String name
   private Map<Integer,Customer> customers;
   private DataSourceDB dataSource;
   private DataSourceFile dataSource;

   public Bank(String name, DataSourceFile dataSource DataSourceDB dataSource) {

    // . . . code . . .
```

```
public class Main {
   public static void main(String[] args) {
        DataSourceDB dataSource = new DataSourceDB();
        DataSourceFile dataSource = new DataSourceFile("customers.txt");
        Bank bank = new Bank("My Bank",dataSource);
        ATM atm = new ATM(bank);
        AtmUI atmUI = new AtmUI(atm);
        atmUI.run();
    }
}
```

Layer of Indirection

- insert a “layer” of interface between classes
  - help change implementation easier
  - testable design





ATM with DI and Layer of Indirection

```
public interface DataSource{

    Map<Integer,Customer> readCustomers();

}

public class DataSourceDB implements DataSource {

    public Map<Integer,Customer> readCustomers() {

        // . . . code . . .
    }
}

public class DataSourceFile implements DataSource {

    public Map<Integer,Customer> readCustomers() {

        // . . . code . . .
    }
}
```

ATM with DI and Layer of Indirection

```
public class Bank {

    private String name
    private Map<Integer,Customer> customers;
    private DataSourceDB dataSource;
    private DataSource dataSource;

    public Bank(String name, DataSource dataSource DataSourceDB dataSource) {
        this.name = name;
        this.dataSource = dataSource;
        this.customers = dataSource.readCustomers();
    }

    // . . . code . . .

}
```

Layer of Indirection

- If we want to change from reading database to files
  - only change class Main

```
public class Main {

    public static void main(String[] args) {
        DataSourceDB dataSourceDB = new DataSourceDB();
        DataSourceFile dataSourceFile = new DataSourceFile("customers.txt");
        Bank bank = new Bank("My Bank", dataSourceFile);
        ATM atm = new ATM(bank);
        AtmUI atmUI = new AtmUI(atm);
        atmUI.run();
    }
}
```

Layer of Indirection

- Very helpful principle
  - Very easy to change implementation
    - No need to touch domain classes, just a main method (which is very unstable already)
  - Flexible design and implementation