Tech Share

Object-oriented programming
&
SOLID Principles

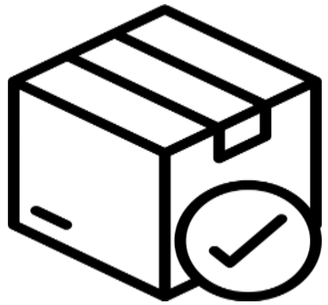
Speaker: 蔡榮峯

目錄

什麼是
Object-oriented?

OO 是什麼?

- OO (Object Oriented) 用現實物件的概念來設計程式
- 三大特性：封裝(Encapsulation)、繼承(Inheritance)、多型 (Polymorphism)

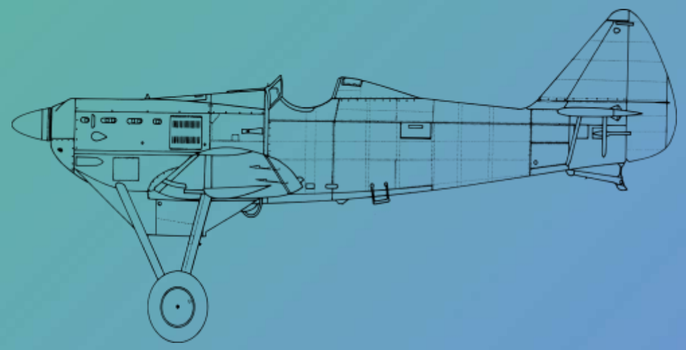


封裝

繼承

多型

我們如何用程式來描述現實世界發生的事物??

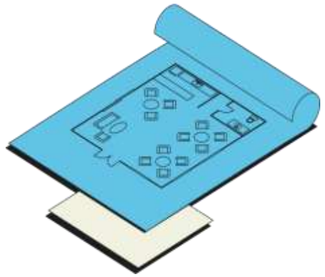


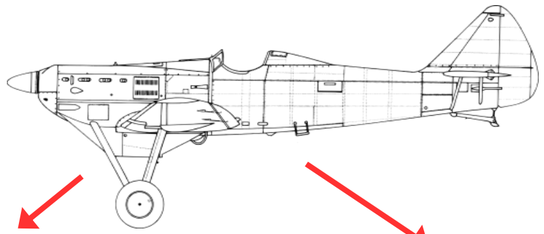
Class(類別)的定義概念

- Class：在程式中描述現實物件的一種方法
  - 組成：
    - 屬性：描述物件 特徵、品牌、顏色...名詞類
    - 方法：描述物件 動作、執行方法 ...動詞類
  - 定義：Class 類別名稱 {

      屬性；方法

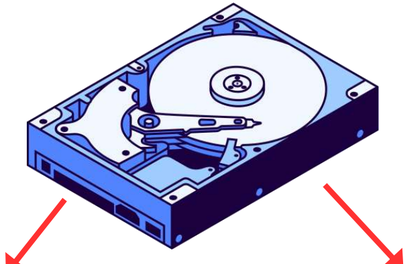
    }





屬性：
機翼
起落架
螺旋槳..

方法：
飛行
降落..

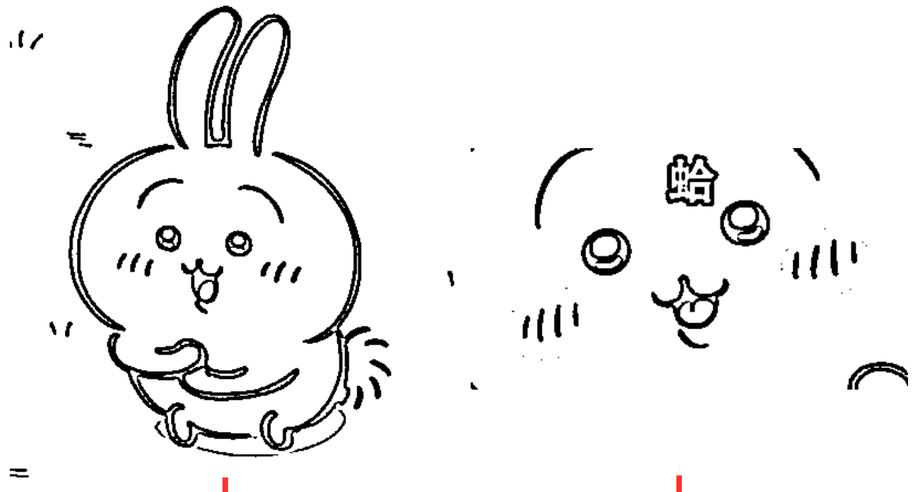


屬性：
廠牌
讀寫頭

方法：
儲存資料
讀取資料

Class的實作範例

- 找一個有印象的東西做聯想(動畫、生活例子...)



屬性：
兔耳朵
米黃色

方法：
轉手
出聲音：蛤?

類別名 吉伊卡哇

屬性 兔耳朵

屬性 米黃色

方法 轉手

方法 出聲音

```java
public class Chikawa {
    // 私有屬性：耳朵
    private String ear;

    // 私有屬性：皮膚顏色
    private String skinColor;

    // 動作方法：輸出[轉手]
    public void action() {
        System.out.println("轉手");
    }

    // 發聲方法：輸出[蛤？]
    public void voiced() {
        System.out.println("蛤？");
    }

    // Getter 方法：取得耳朵
    public String getEar() {
        return ear;
    }

    // Setter 方法：設置耳朵
    public void setEar(String ear) {
        this.ear = ear;
    }

    // Getter 方法：取得皮膚顏色
    public String getSkinColor() {
        return skinColor;
    }

    // Setter 方法：設置皮膚顏色
    public void setSkinColor(String skinColor) {
        this.skinColor = skinColor;
    }
}
```

物件(Object)

- 物件 ：Class的實例化，按照設計圖產生的物件
- 定義 ：類別名 物件名 = new 類別名( );

類別名 吉伊卡哇 烏薩奇 = new 吉伊卡哇( );

```
//Code Snippet//
// 建立 Chiikawa 類別的實例，物件名為 Usagi
Chiikawa usagi = new Chiikawa();

// 設置耳朵屬性
usagi.setEar("兔耳朵");

// 設置皮膚顏色屬性
usagi.setSkinColor("米黃色");

// 使用 Getter 方法取得並輸出耳朵屬性
System.out.println("Usagi 的耳朵: " + usagi.getEar());

// 使用 Getter 方法取得並輸出皮膚顏色屬性
System.out.println("Usagi 的皮膚顏色: " + usagi.getSkinColor());

// 呼叫 Usagi 的動作方法
usagi.action();

// 呼叫 Usagi 的發聲方法
usagi.voiced();
```

獲得 烏薩奇

什麼是封裝

封裝(Encapsulation)

- 概念 ：隱藏物件內部屬性和行為，只給特定方法與外界互動
  - 關鍵字： 私有的(private) 、此物件(this)
  - 情境： 機敏資訊、資料不被任意修改...

私有屬性：
帳號
密碼

私有方法：
密碼驗證

封裝(Encapsulation)

- 修飾符(Access Modifiers)：定義類別、方法與變數的訪問權限

Java語言

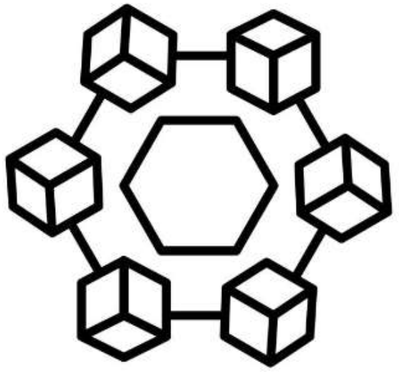
訪問修飾符	Class	package	SubClass
public	✓	✓	✓
protected	✓	✓	✓
default	✓	✓	
private	✓		

封裝(Encapsulation)

為什麼要使用封裝?

安全性

易於維護



模組化

什麼是繼承

繼承(Inheritance)

- 概念：允許子類別獲得父類別的屬性和方法，讓程式碼覆用或擴展功能
  - 關鍵字：extends、理解成is a 的概念，Dog is an Animal

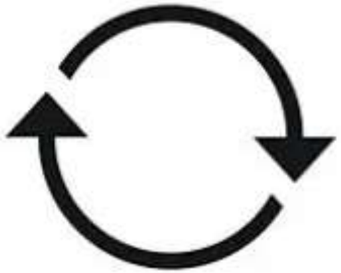
Black / Color

DNA

繼承(Inheritance)

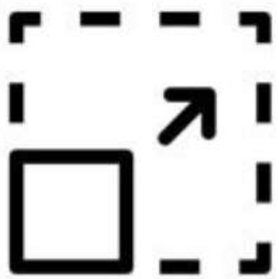
為什麼要使用繼承?

有限制條件的使用

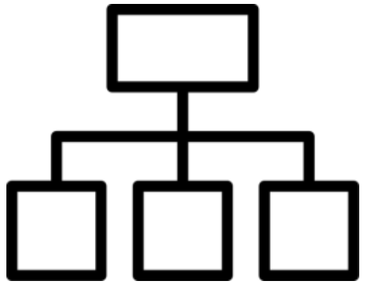


程式碼重用

易於維護



擴展性

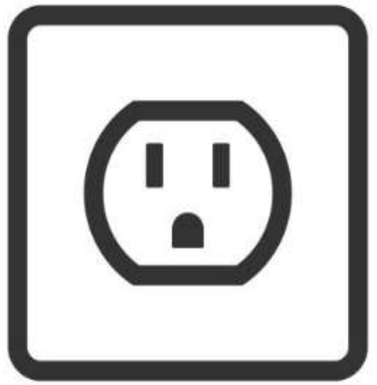


結構清晰

什麼是多型

多型(Polymorphism)

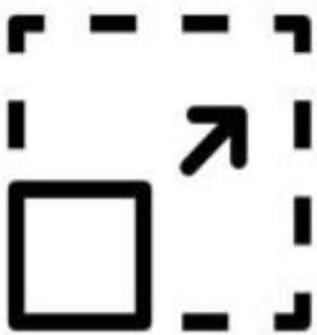
- 概念：提供一種介面方法，讓不同的類別實現
  - 關鍵字：介面(Interface)
  - 關聯字：Implements



多型(Polymorphism)

為什麼要使用多型?

易於維護



擴展性

靈活性

Spring Boot框架運用範例

User

- id: Long
- name: String

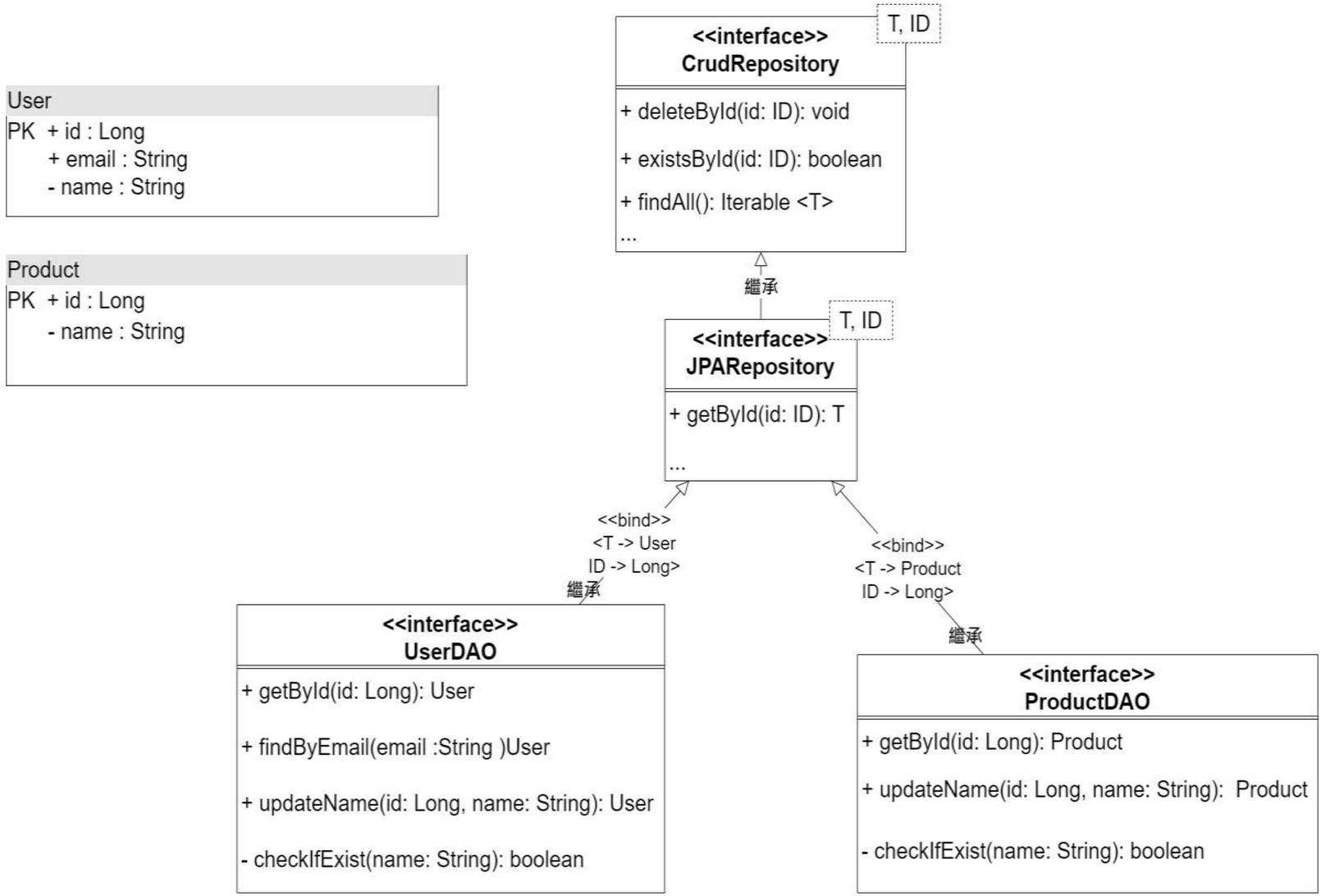
+ setId(id: Long): void
+ getId(): void
+ setName(name: string): void
+ getName(): void
+ User()
+ User(id: Long, name: String)

Product

- id: Long
- name: String

+ setId(id: Long): void
+ getId(): void
+ setName(name: string): void
+ getName(): void
+ Product()
+ Product(id: Long, name: String)

封裝



多型+繼承

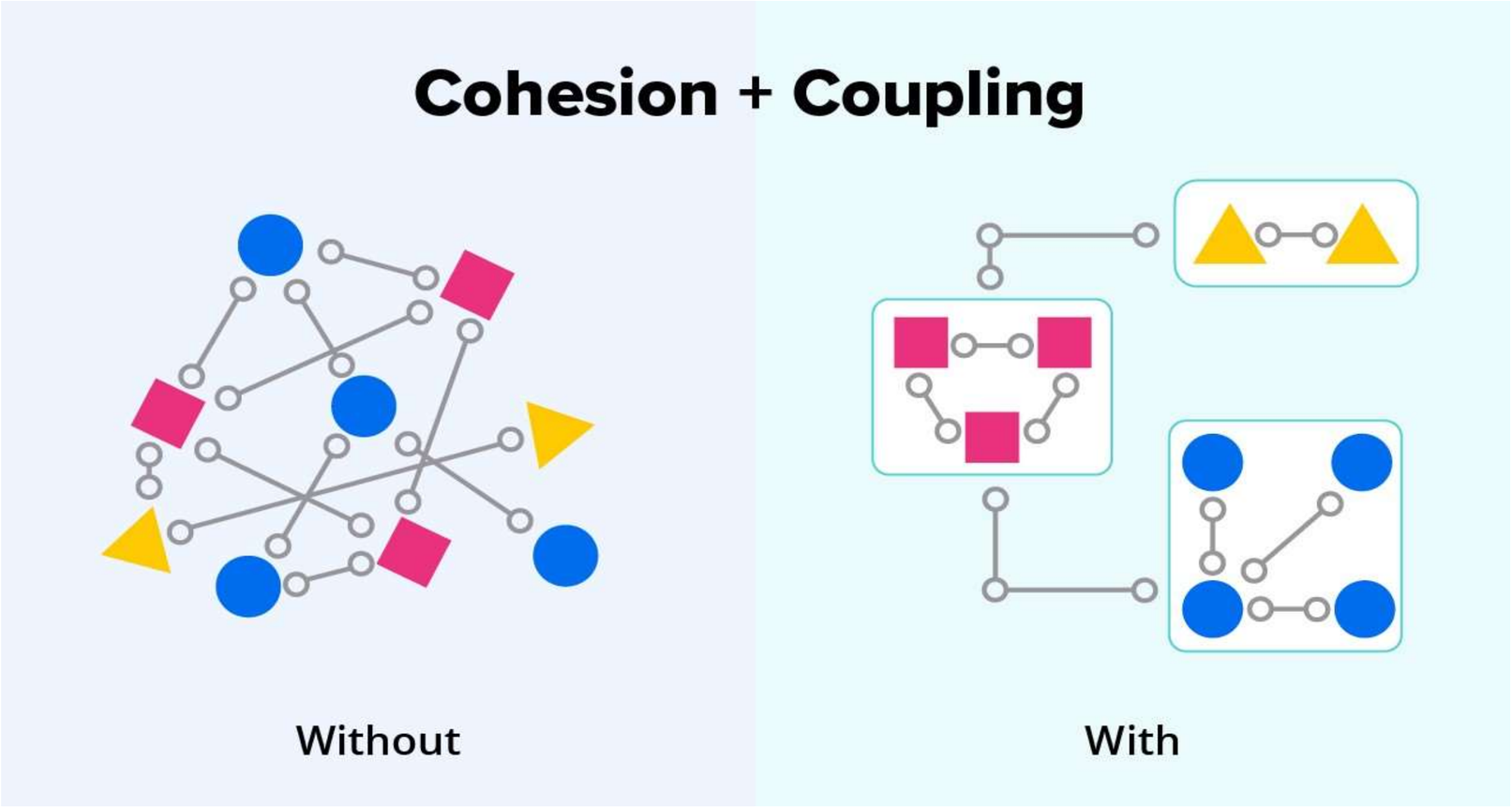
SOLID原則

SOLID原則

名稱	說明
S - 單一職責原則 (SRP) Single Responsibility Principle	每個類別只應該負責一項職責， 避免類的過度複雜和多職責
O - 開放封閉原則 (OCP) Open/Closed Principle	對擴展開放，對修改封閉， 通過繼承或介面來擴展功能
L - 里氏替換原則 (LSP) Liskov Substitution Principle	子類別必須能替換父類， 並且應保持程序行為的一致性
I - 介面隔離原則 (ISP) Interface Segregation Principle	按照功能或需求的介面拆成多個介面， 避免實現不需要的功能
D - 依賴反轉原則 (DIP) Dependency Inversion Principle	依賴於抽象而非具體實現， 讓高層和低層模組通過抽象解耦

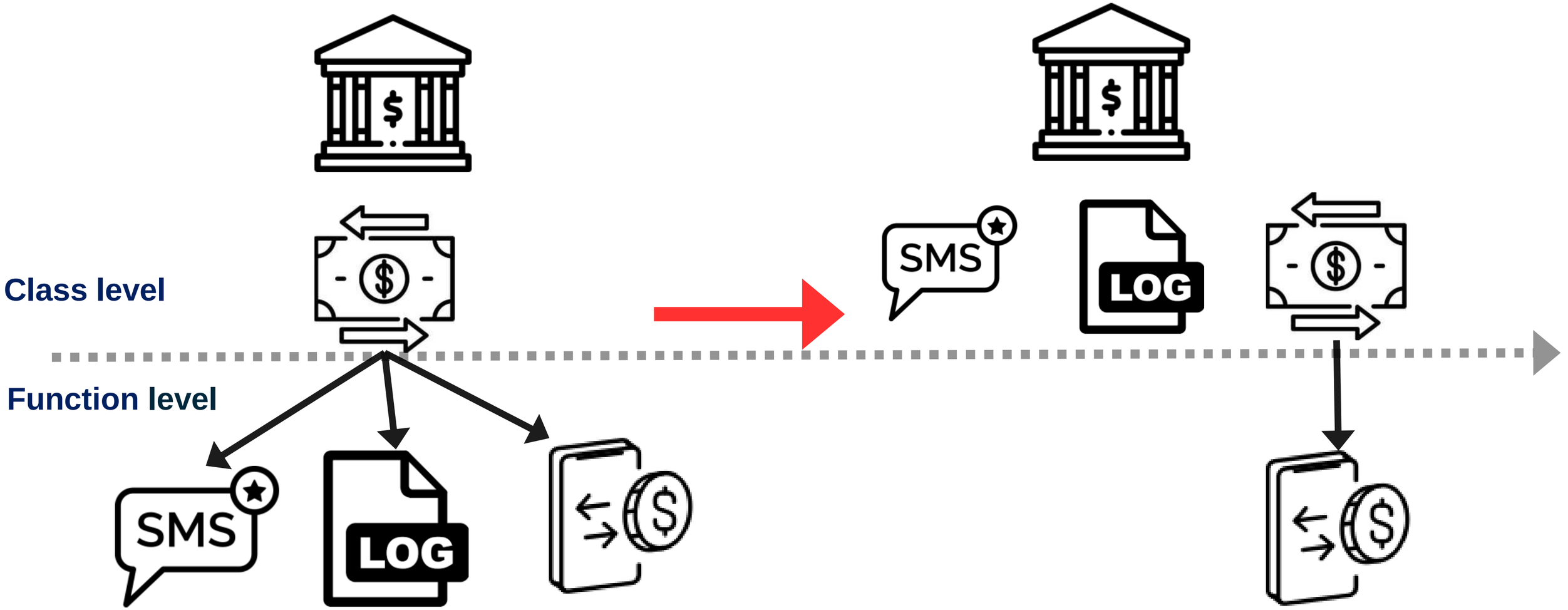
內聚(Cohesion)、耦合(Coupling)

- 內聚：模組或類別內部的功能彼此緊密相關的程度
- 耦合：模組之間資訊或參數的依賴程度越高，耦合度越高

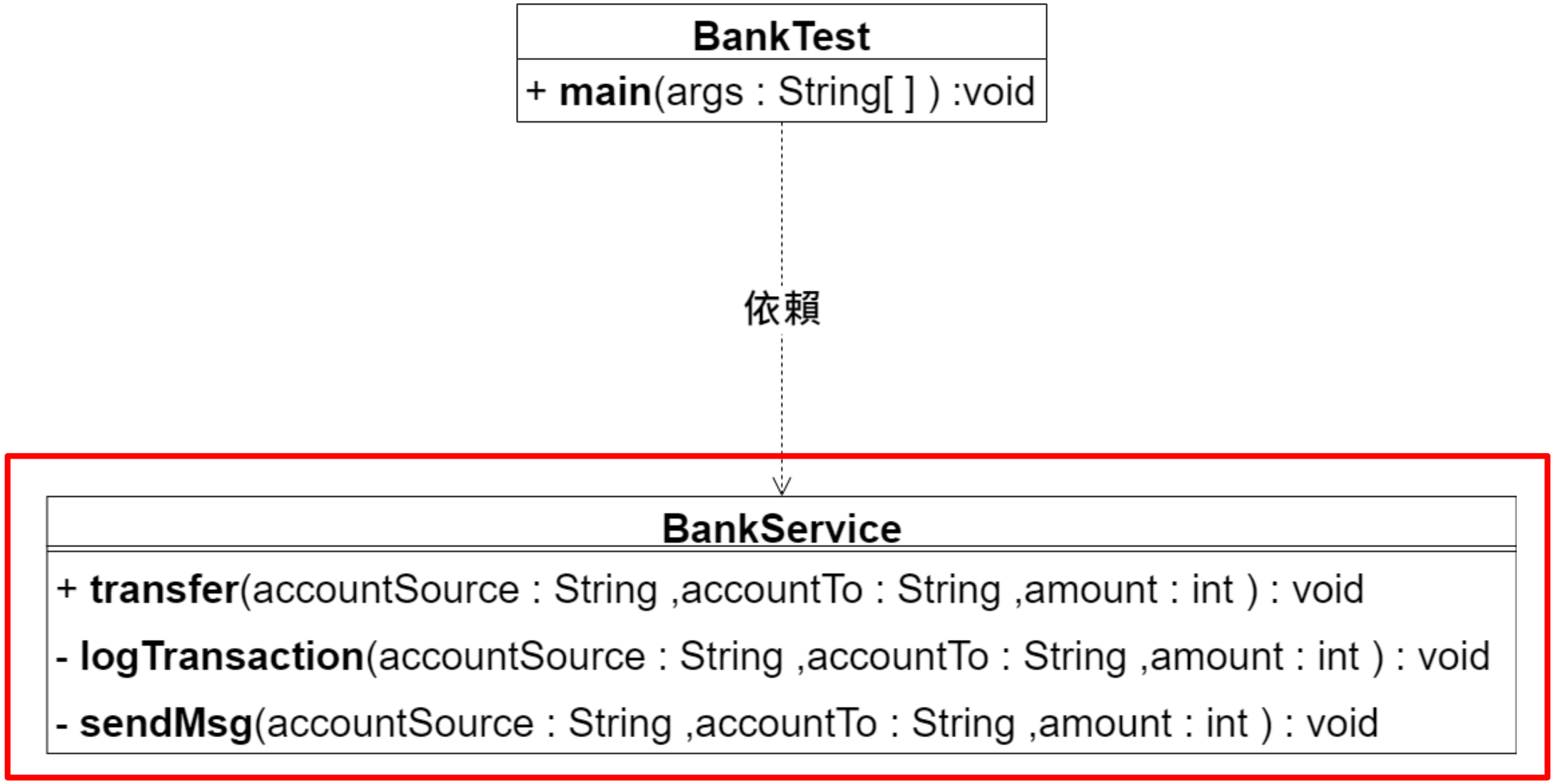


單一職責原則(SRP)

- 每個類別應該只有一個改變的理由，它應該只負責一件事



單一職責原則(SRP)

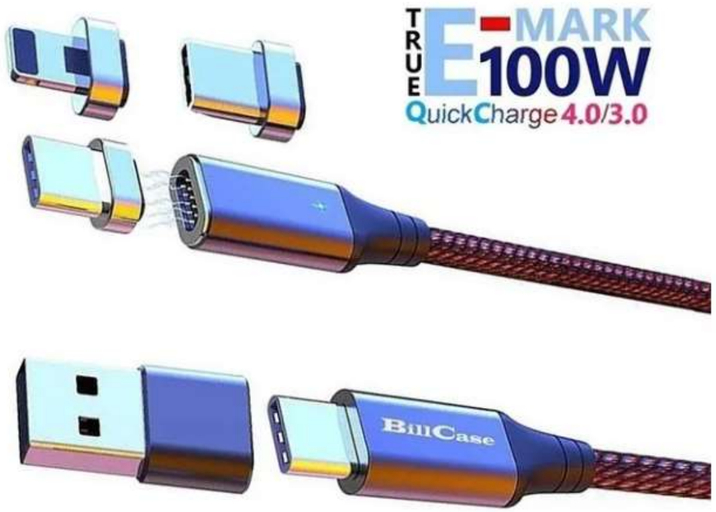

BankTest
+ main(args : String[ ]) :void
依賴
BankService
+ transfer(accountSource : String ,accountTo : String ,amount : int) : void
- logTransaction(accountSource : String ,accountTo : String ,amount : int) : void
- sendMsg(accountSource : String ,accountTo : String ,amount : int) : void

單一職責原則(SRP)

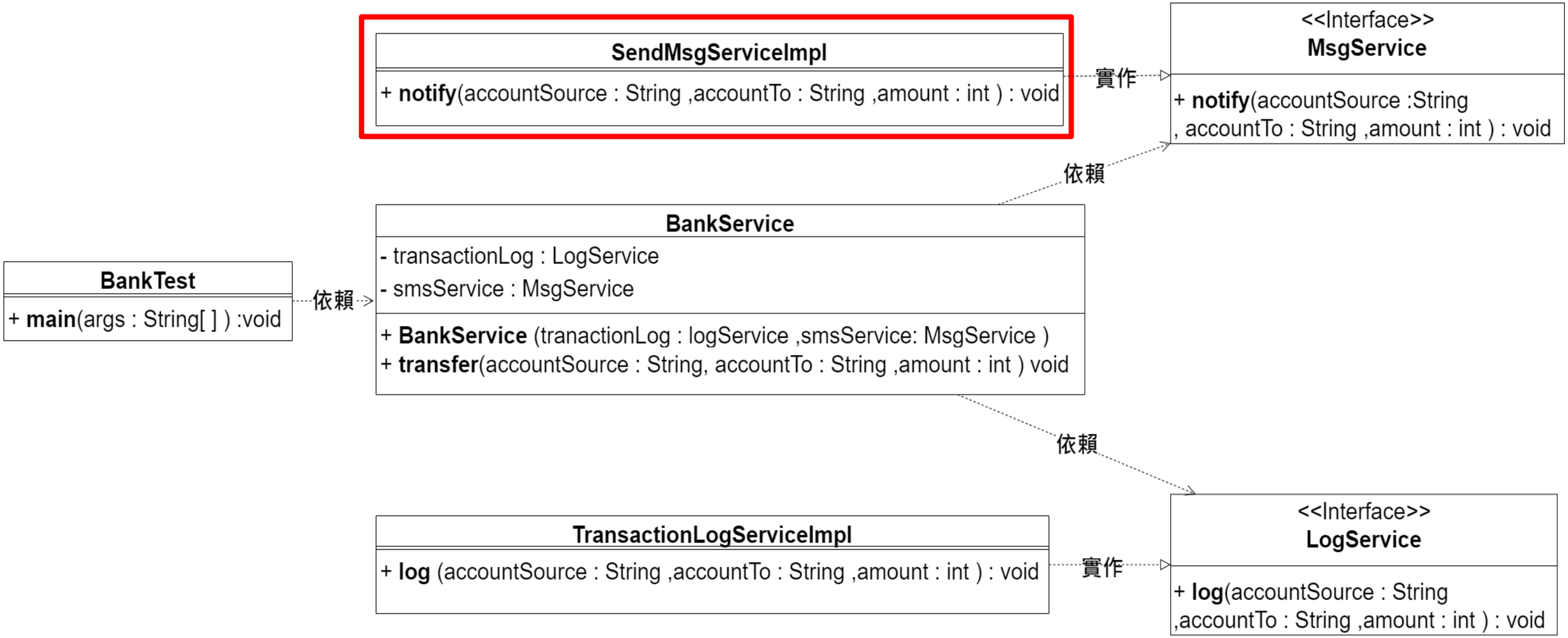

SendMsgService
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
依賴
BankService
- transferService : TransferService
- logService : TransactionLogService
- sendMsgService : SendMsgService
+ BankService (transferService : TransferService,logService : TransactionLogService,endMsgService : SendMsgService)
+ transfer(accountSource : String ,accountTo : String ,amount : int) : void
BankTest
+ main(args : String[ ]) : void
依賴
依賴
TransferService
+ transfer(accountSource : String , accountTo : String ,amount : int) : void
依賴
TransactionLogService
+ log (accountSource : String ,accountTo : String ,amount : int) : void

開放封閉原則(OCP)

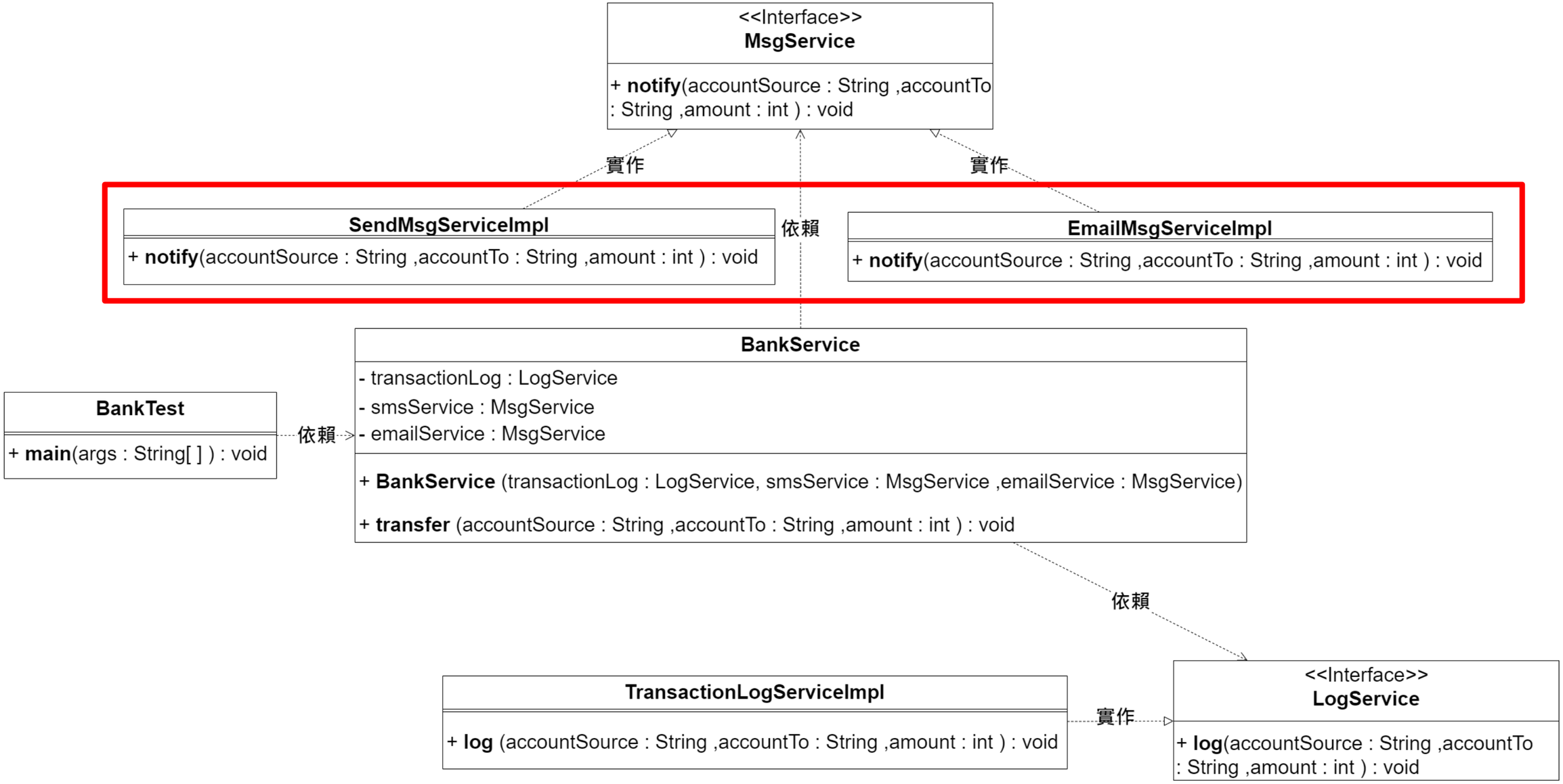
- 對拓展開放對修改封閉



開放封閉原則(OCP)


SendMsgServiceImpl
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
實作
<<Interface>>
MsgService
+ notify(accountSource :String
, accountTo : String ,amount : int) : void
依賴
BankService
- transactionLog : LogService
- smsService : MsgService
+ BankService (tranactionLog : logService ,smsService: MsgService)
+ transfer(accountSource : String, accountTo : String ,amount : int) void
BankTest
+ main(args : String[ ]) :void
依賴
依賴
TransactionLogServiceImpl
+ log (accountSource : String ,accountTo : String ,amount : int) : void
實作
<<Interface>>
LogService
+ log(accountSource : String
,accountTo : String ,amount : int) : void

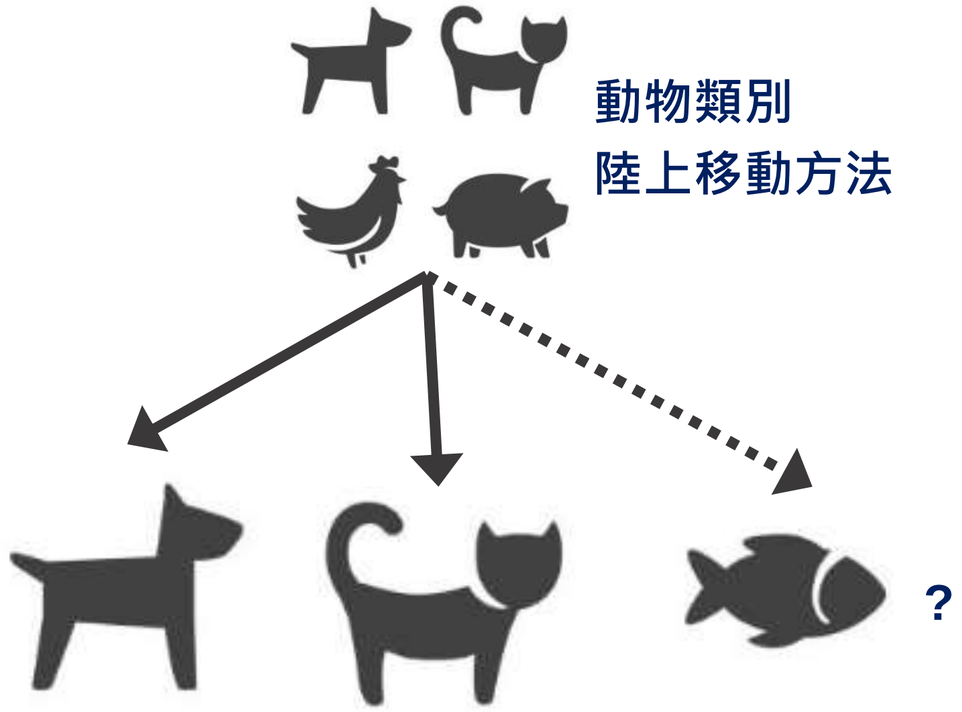
開放封閉原則(OCP)


<<Interface>>
MsgService
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
實作
實作
SendMsgServiceImpl
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
依賴
EmailMsgServiceImpl
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
BankService
- transactionLog : LogService
- smsService : MsgService
- emailService : MsgService
+ BankService (transactionLog : LogService, smsService : MsgService ,emailService : MsgService)
+ transfer (accountSource : String ,accountTo : String ,amount : int) : void
BankTest
+ main(args : String[ ]) : void
依賴
依賴
TransactionLogServiceImpl
+ log (accountSource : String ,accountTo : String ,amount : int) : void
實作
<<Interface>>
LogService
+ log(accountSource : String ,accountTo : String ,amount : int) : void

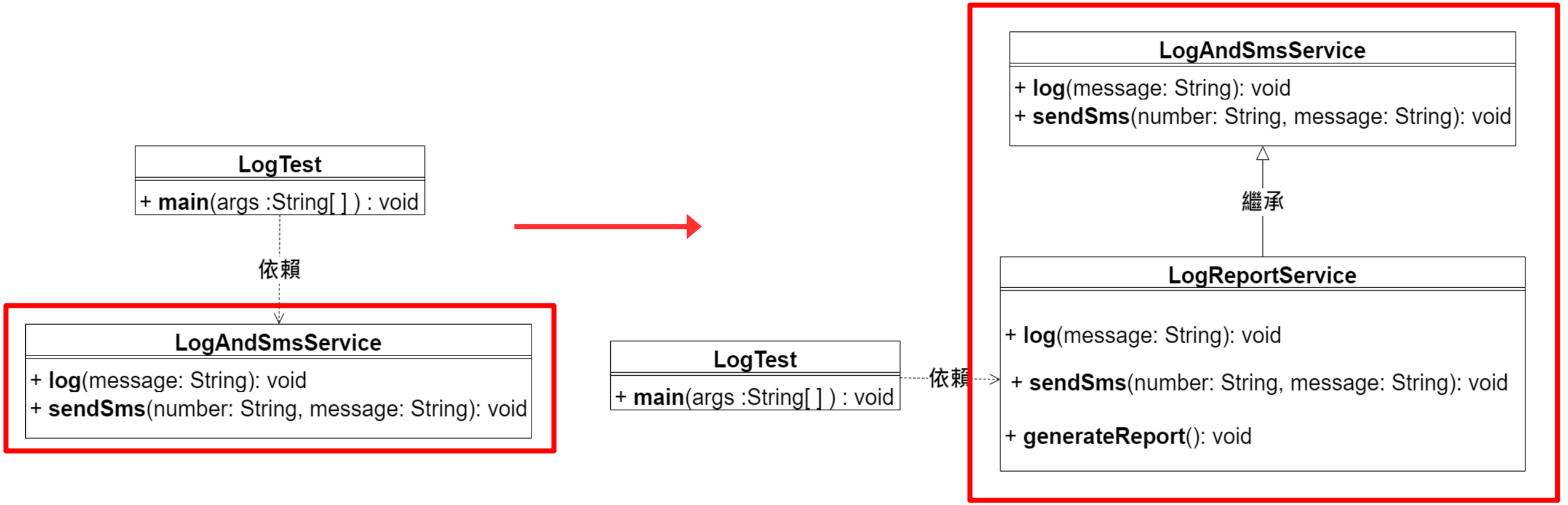
里氏替換原則(LSP)

- 子類別必須能夠替換父類別而不影響程式的正確性

plugin in 功能

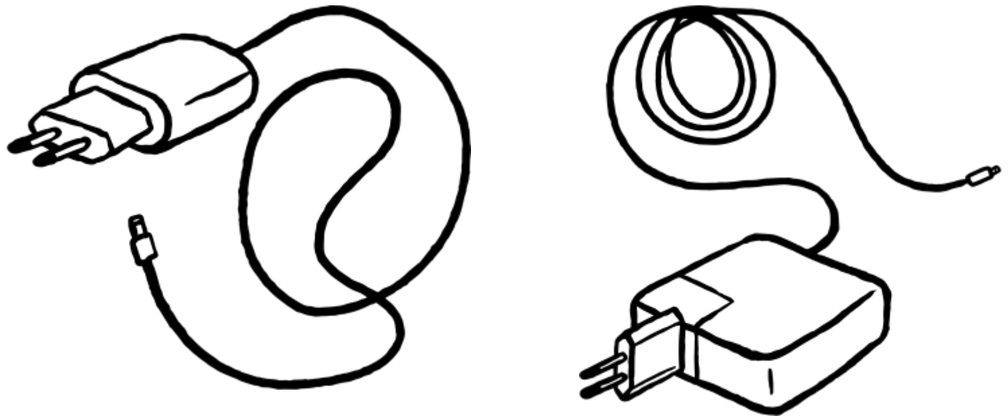


里氏替換原則(LSP)

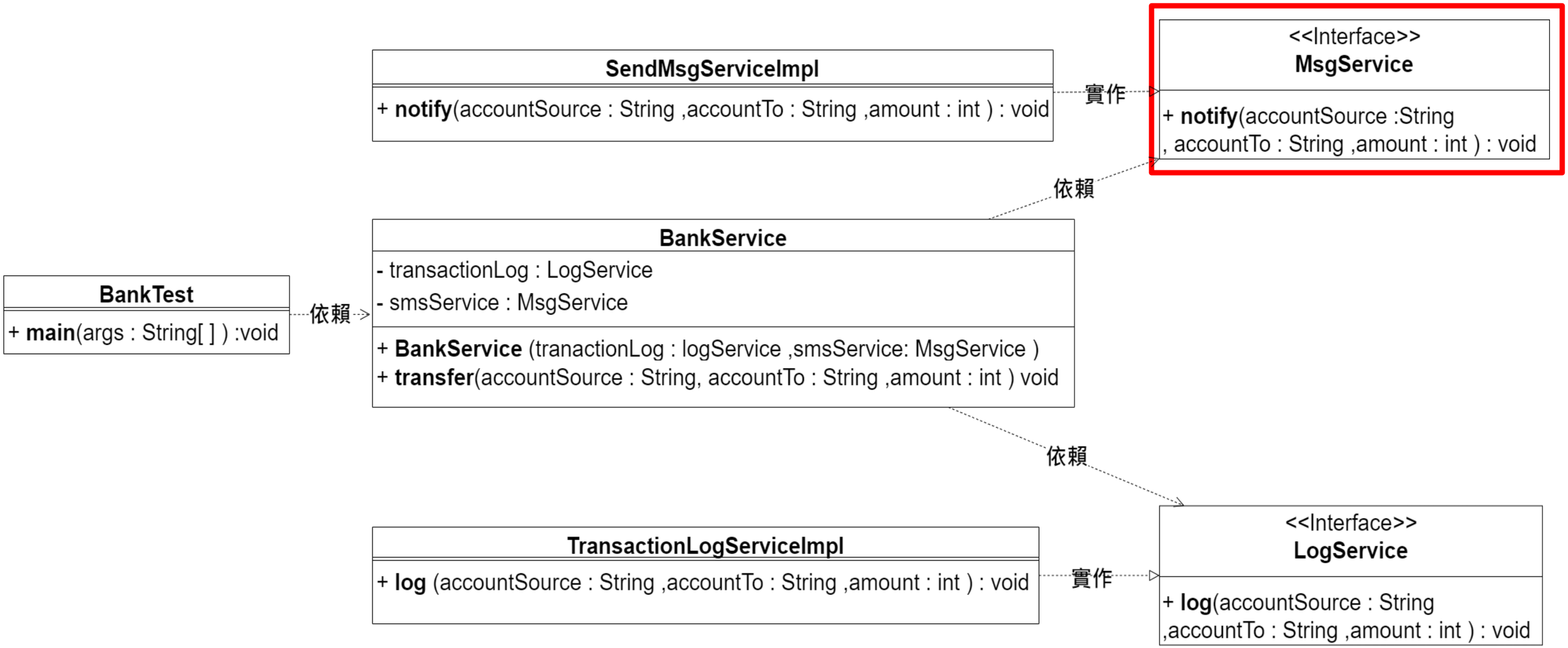

LogTest
+ main(args :String[ ]) : void
依賴
LogAndSmsService
+ log(message: String): void
+ sendSms(number: String, message: String): void
LogTest
+ main(args :String[ ]) : void
依賴
LogAndSmsService
+ log(message: String): void
+ sendSms(number: String, message: String): void
繼承
LogReportService
+ log(message: String): void
+ sendSms(number: String, message: String): void
+ generateReport(): void

介面隔離原則(ISP)

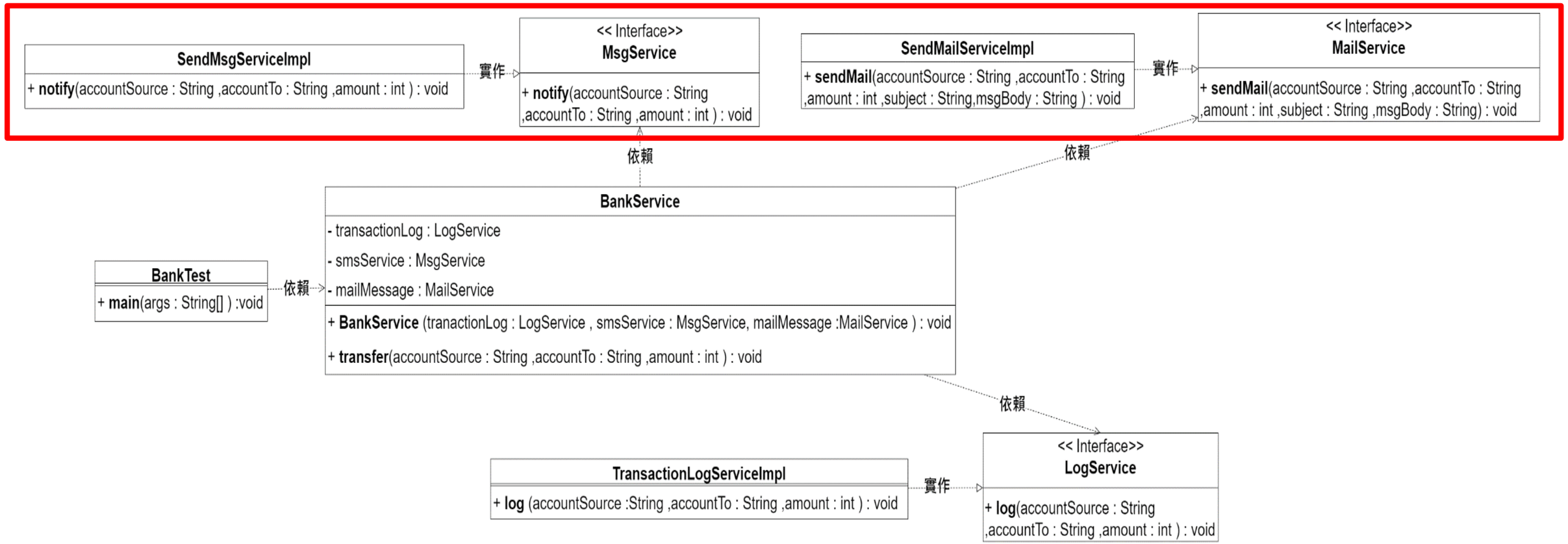
- 不應該讓客戶端依賴它們不需要的介面
- 細分多個介面功能



介面隔離原則(ISP)

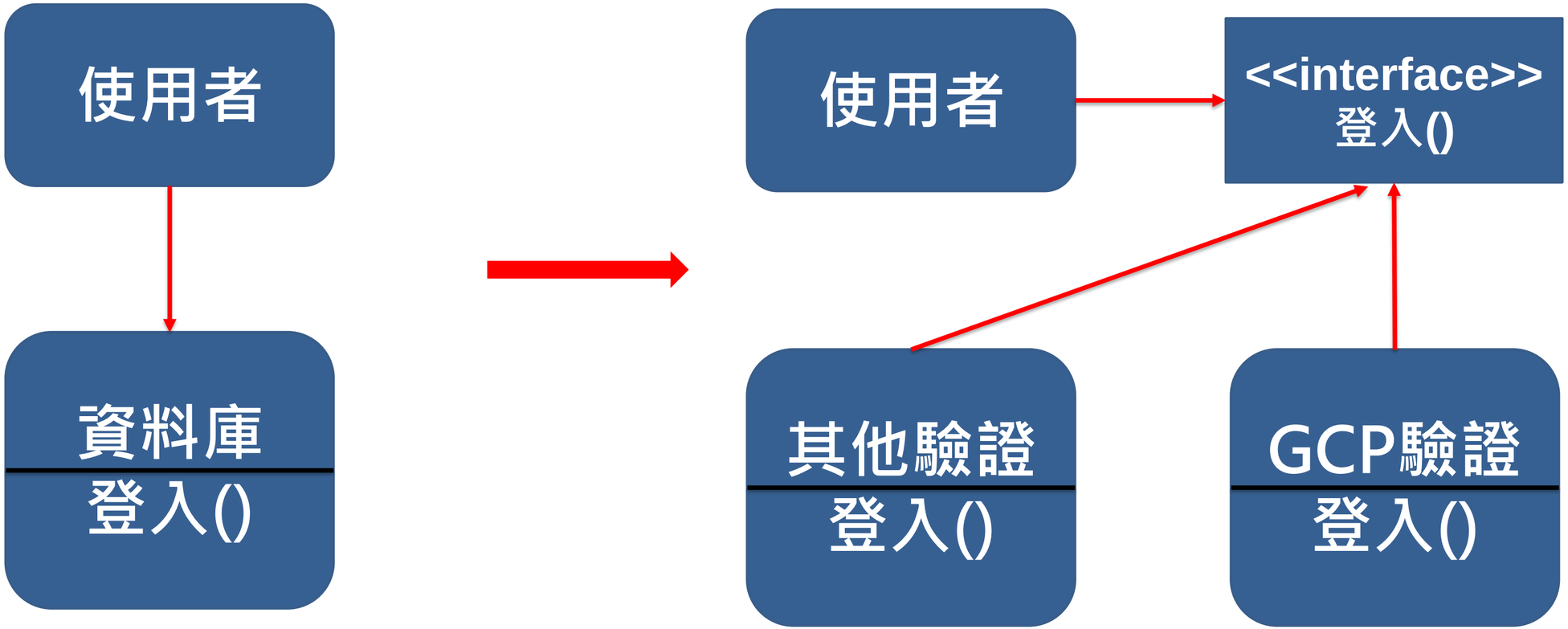

SendMsgServiceImpl
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
實作
<<Interface>>
MsgService
+ notify(accountSource :String
, accountTo : String ,amount : int) : void
依賴
BankService
- transactionLog : LogService
- smsService : MsgService
+ BankService (tranactionLog : logService ,smsService: MsgService)
+ transfer(accountSource : String, accountTo : String ,amount : int) void
BankTest
+ main(args : String[ ]) :void
依賴
依賴
TransactionLogServiceImpl
+ log (accountSource : String ,accountTo : String ,amount : int) : void
實作
<<Interface>>
LogService
+ log(accountSource : String
,accountTo : String ,amount : int) : void

介面隔離原則(ISP)


SendMsgServiceImpl
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
實作
<< Interface>>
MsgService
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
SendMailServiceImpl
+ sendMail(accountSource : String ,accountTo : String ,amount : int ,subject : String,msgBody : String) : void
實作
<< Interface>>
MailService
+ sendMail(accountSource : String ,accountTo : String ,amount : int ,subject : String ,msgBody : String) : void
依賴
依賴
BankService
- transactionLog : LogService
- smsService : MsgService
- mailMessage : MailService
+ BankService (tranactionLog : LogService , smsService : MsgService, mailMessage :MailService) : void
+ transfer(accountSource : String ,accountTo : String ,amount : int) : void
BankTest
+ main(args : String[]) :void
依賴
依賴
<< Interface>>
LogService
+ log(accountSource : String ,accountTo : String ,amount : int) : void
實作
TransactionLogServiceImpl
+ log (accountSource :String ,accountTo : String ,amount : int) : void

依賴翻轉原則(DIP)

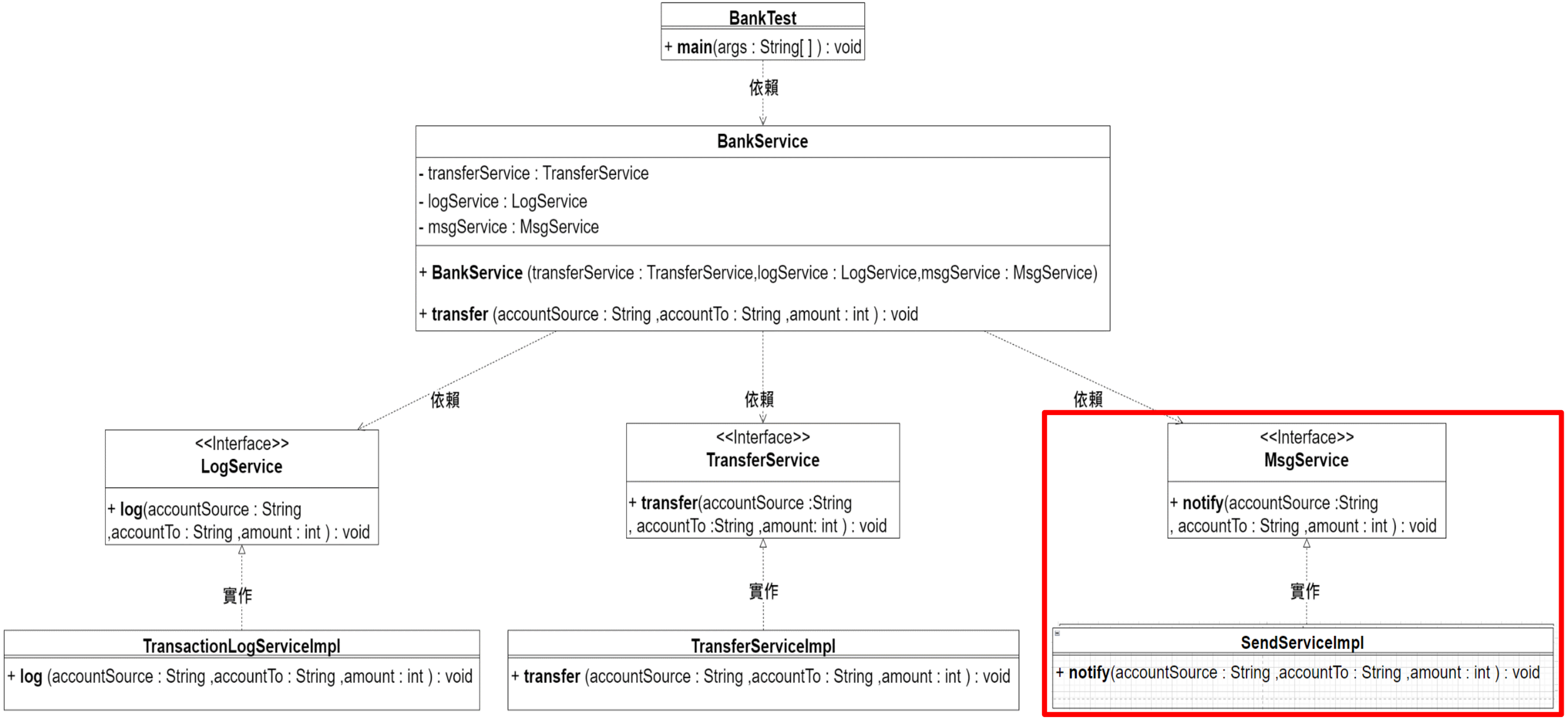
- 高層模組不應依賴低層模組，兩者都應該依賴抽象
- 抽象不應依賴具體實現，具體實現應依賴抽象



依賴翻轉原則(DIP)


SendMsgService
+ notify(accountSource : String ,accountTo : String ,amount : int) : void
依賴
BankService
- transferService : TransferService
- logService : TransactionLogService
- sendMsgService : SendMsgService
+ BankService (transferService : TransferService,logService : TransactionLogService,endMsgService : SendMsgService)
+ transfer(accountSource : String ,accountTo : String ,amount : int) : void
BankTest
+ main(args : String[ ]) : void
依賴
依賴
TransferService
+ transfer(accountSource : String , accountTo : String ,amount : int) : void
依賴
TransactionLogService
+ log (accountSource : String ,accountTo : String ,amount : int) : void

依賴翻轉原則(DIP)


BankTest
+ main(args : String[ ]) : void
依賴
BankService
- transferService : TransferService
- logService : LogService
- msgService : MsgService
+ BankService (transferService : TransferService,logService : LogService,msgService : MsgService)
+ transfer (accountSource : String ,accountTo : String ,amount : int) : void
依賴
依賴
依賴
<<Interface>>
LogService
+ log(accountSource : String
,accountTo : String ,amount : int) : void
<<Interface>>
TransferService
+ transfer(accountSource :String
, accountTo :String ,amount: int) : void
<<Interface>>
MsgService
+ notify(accountSource :String
, accountTo : String ,amount : int) : void
實作
實作
實作
TransactionLogServiceImpl
+ log (accountSource : String ,accountTo : String ,amount : int) : void
TransferServiceImpl
+ transfer (accountSource : String ,accountTo : String ,amount : int) : void
SendServiceImpl
+ notify(accountSource : String ,accountTo : String ,amount : int) : void

QA

Thank you!