Kyoto, Japan in 2010
by Usa Sammapun

06 Unit Testing with Stubs

Usa Sammapun

Outline

- **การทดสอบ unit testing ด้วย stub เบื้องต้น**
  - ใช้คลาส / ฟังก์ชัน / เมธอด แบบ hard code
  - เพื่อให้สามารถ unit test คลาสเดี่ยว ๆ โดยไม่ต้องใช้คลาส/เมธอดอื่น
- **การใช้ stub**
  - การใช้ stub แทน complex class
  - การใช้ stub แทน external resource

การทดสอบโปรแกรมเชิงวัตถุ

- **การโปรแกรมเชิงวัตถุโดยทั่วไป**
  - การทำงานร่วมกันระหว่าง class หรือ object ต่างๆ

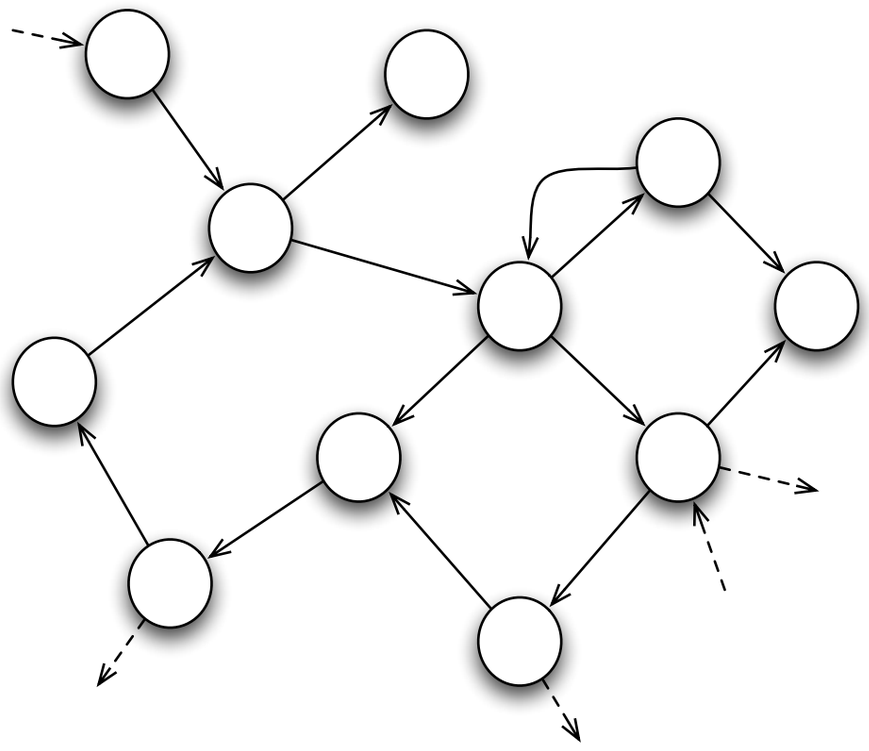


Image Source: Steve Freeman, Nat Pryce. “Growing Object-Oriented Software, Guided by Tests, “ Addison Wesley, 2010.

Unit testing โปรแกรมเชิงวัตถุ

- **หากต้องการ unit testing คลาสที่มีทำงานร่วมกับคลาสอื่น**
  - และทดสอบคลาสทั้งหมดร่วมกัน
  - หาก test fail จะไม่สามารถระบุได้ว่า คลาสใดทำงานผิดพลาด

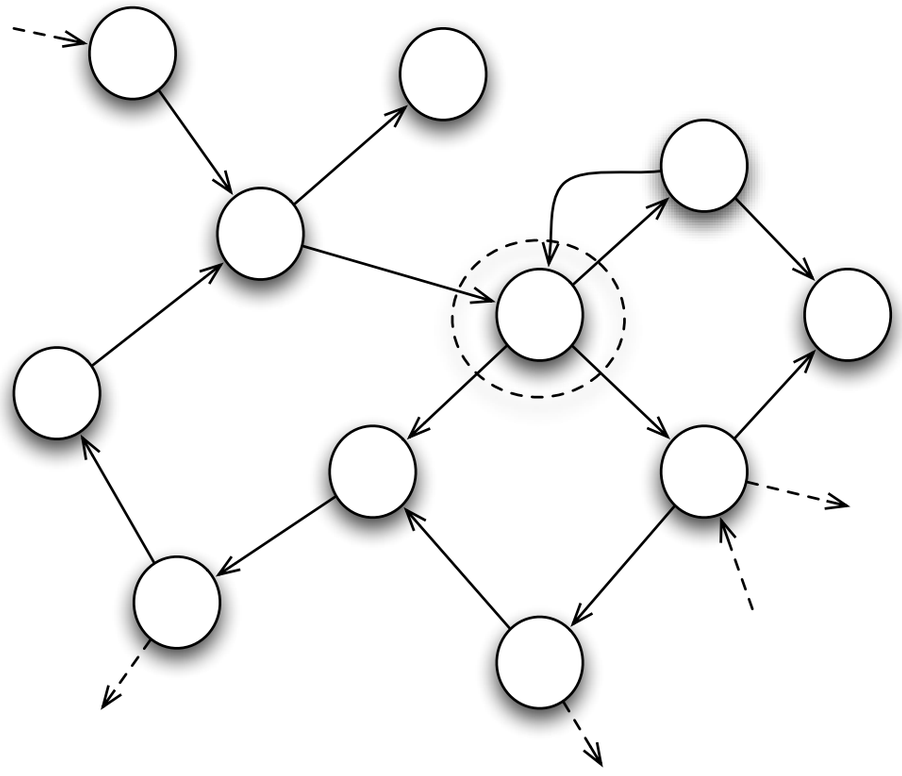
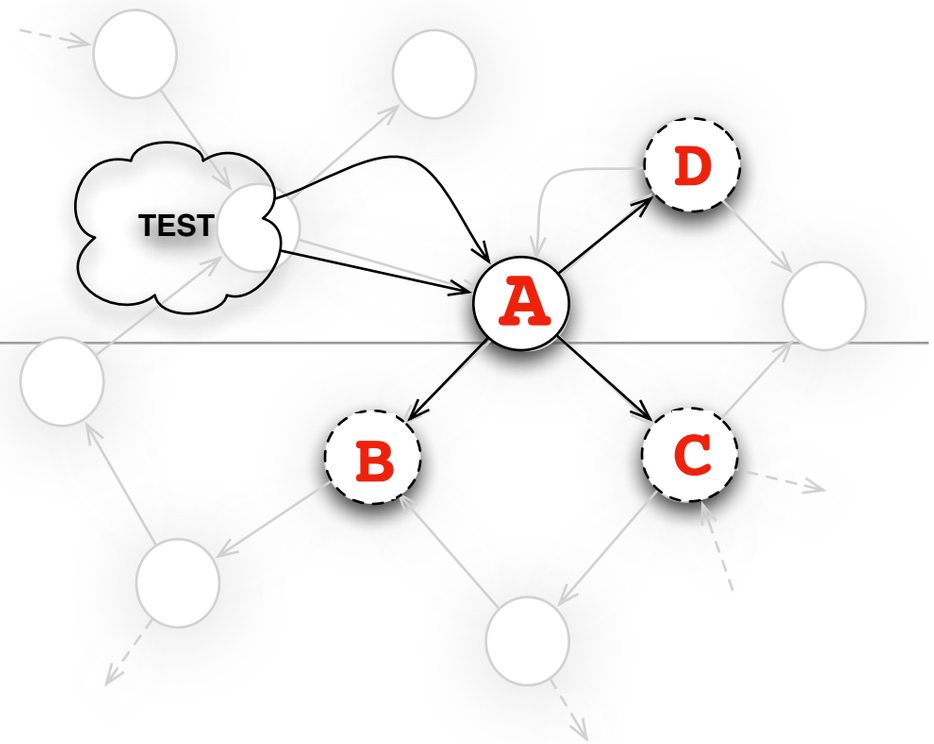


Image Source: Steve Freeman, Nat Pryce. “Growing Object-Oriented Software, Guided by Tests, “ Addison Wesley, 2010.

ตัวอย่าง



- **คลาส A ทำงานร่วมกับคลาส B, C, D**
  - ต้องการ unit test คลาส A คลาสเดียว
  - หาก test fail จะแน่ใจได้อย่างไรว่า A ผิด
  - B, C, D อาจมี bug ทำให้ A fail ก็ได้

```
class A {
    private B b;
    private C c;
    private D d;

    public int importantMethod() {
        return b.doB() + c.doC() + d.doD();
    }
}
```

```
class B {
    public int doB() {
        int b = 0;
        // ... complex code ...
        return b;
    }
}
```

```
class C {
    public int doC() {
        int c = 0;
        // ... complex code ...
        return c;
    }
}
```

```
class D {
    public int doD() {
        int d = 0;
        // ... complex code ...
        return d;
    }
}
```

Unit testing โปรแกรมเชิงวัตถุ

- **นำ stub มาใช้แทน object จริง**
  - Stub คือ คลาส / ฟังก์ชัน / เมธอด แบบ hard code
  - เพื่อให้สามารถ unit test คลาสเดี่ยว ๆ โดยไม่ต้องใช้คลาส/เมธอดจริงอื่น

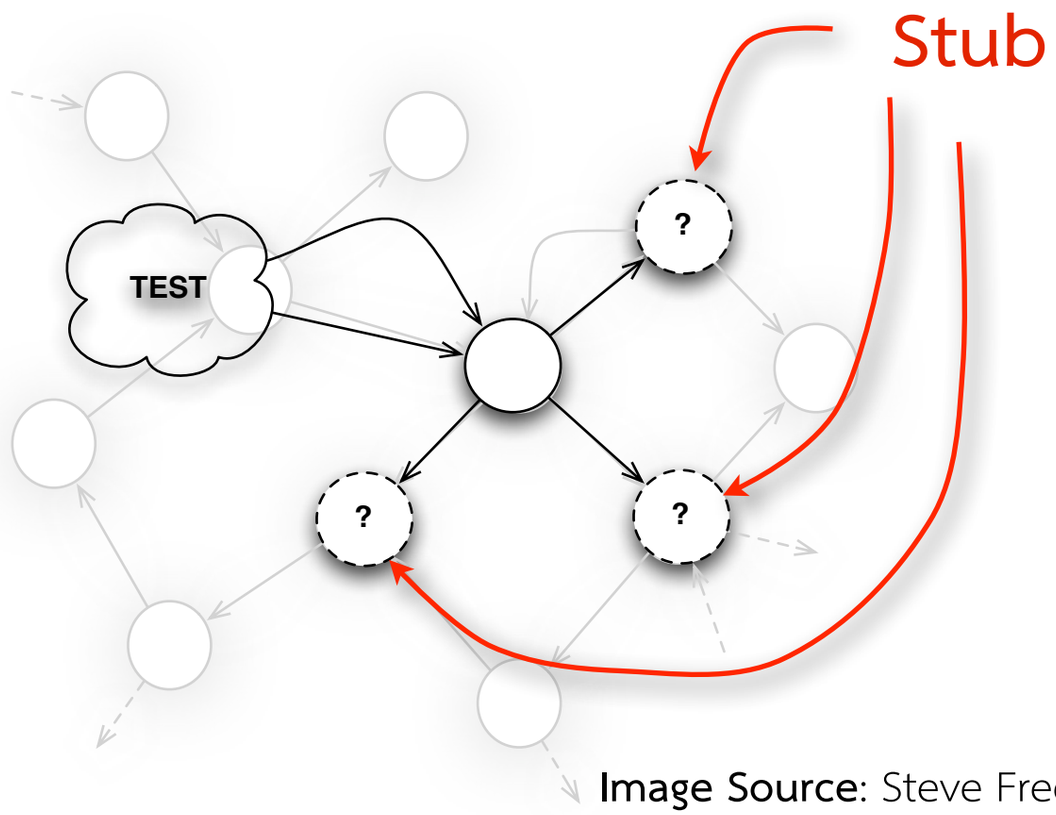


Image Source: Steve Freeman, Nat Pryce. “Growing Object-Oriented Software, Guided by Tests, “ Addison Wesley, 2010.

Stub

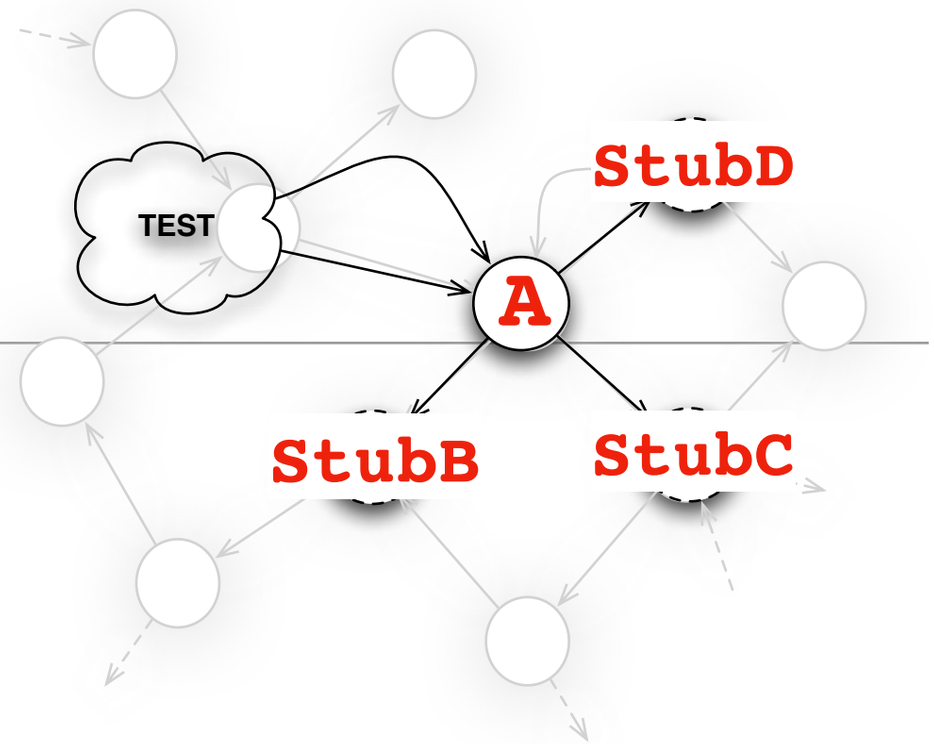
- “A stub is a piece of code that’s inserted at runtime in place of the real code, in order to isolate the caller from the real implementation. The intent is to replace a complex behavior with a simpler one that allows independent testing of some part of the real code.”

Source : P. Tahchiev, F. Leme, V. Massol, G. Gregory. JUnit in Action, 2nd ed. Manning, 2010.

Stub

- **stub คือ ตัวแทนของ object จริง (เป็น fake object ประเภทหนึ่ง)**
  - แยก caller จาก real implementation
  - ช่วย break dependency
- **การสร้าง stub**
  - สร้าง object / method / function แบบ hard code / simple
  - อาจซับซ้อนได้ ถ้า object จริงซับซ้อนมาก
  - ต่างสถานการณ์ อาจใช้ stub ต่างกัน
- **การวาง stub**
  - inner class หรือ separate class

ตัวอย่างการใช้ stub (1)



- สร้าง stub ของ B, C, D แบบ hard-code
  - นำหลักการ inheritance มาช่วย
  - และให้ A สามารถผลัดเปลี่ยน B, C, D ได้
  - โดยรับ B, C, D ผ่าน constructor

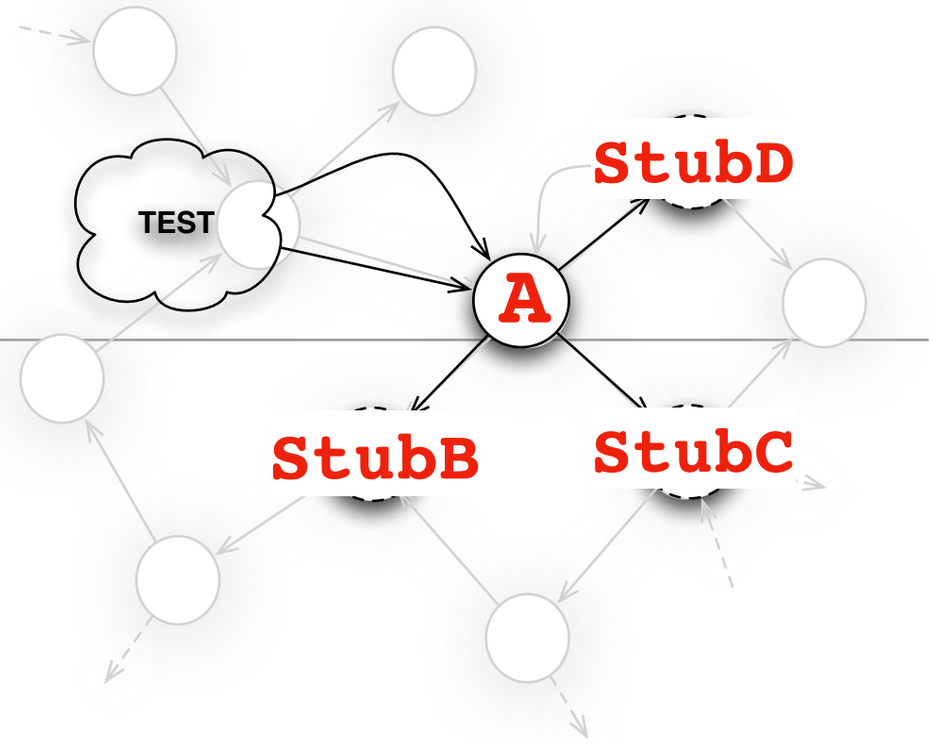
```
class A {
    private B b;
    private C c;
    private D d;

    public A(B b, C c, D d) {
        this.b = b;
        this.c = c;
        this.d = d;
    }

    public int importantMethod() {
        return b.doB() + c.doC() + d.doD();
    }
}
```

```
class StubB extends B {
    @Override
    public int doB() {
        return 10;    // hard code
    }
}
```

```
class StubC extends C {
    @Override
    public int doC() {
        return 0;    // hard code
    }
}
```

```
class StubD extends D {
    @Override
    public int doD() {
        return 50;    // hard code
    }
}
```

ตัวอย่างการใช้ stub (2)



- **เมื่อทดสอบใช้ stub แทน B, C, D จริง**
  - ส่ง stub ผ่าน constructor ไปให้ A
  - หาก test fail แน่ใจได้เลยว่า A มี bug
  - B, C, D ไม่มี bug แน่นอน เพราะเรา hard-code ให้ถูกต้องไปแล้ว

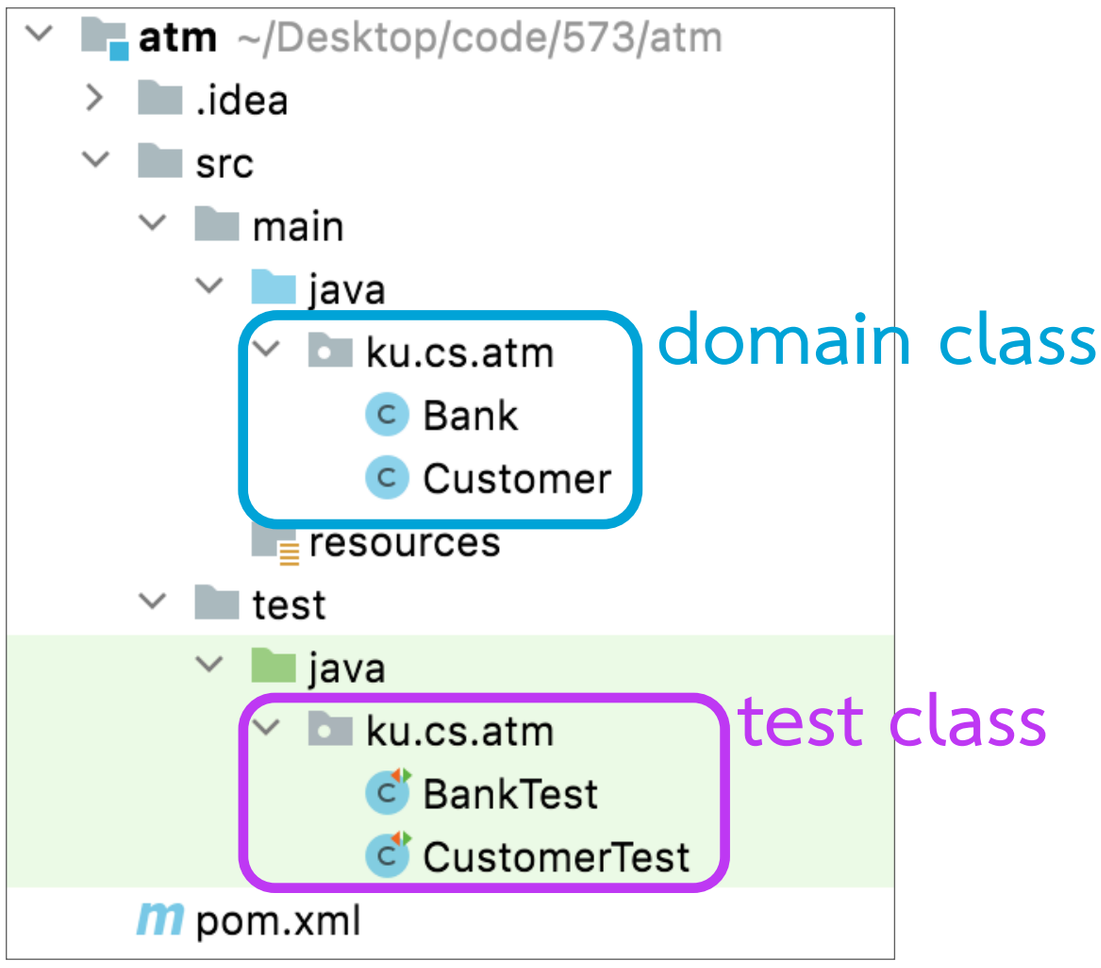
```
public class ATest {

    @Test
    void testA() {
        A a = new A(new StubB(), new StubC(), new StubD());
        assertEquals(60, a.importantMethod());
    }
}
```

การใช้ Stub

- **ใช้เมื่อ**
  - ต้องการ unit testing แค่คลาสเดียว
  - ใน incremental integration testing
  - ไม่สามารถแก้ existing system ได้
  - คลาสที่ทำงานด้วยยัง implement ไม่เสร็จ

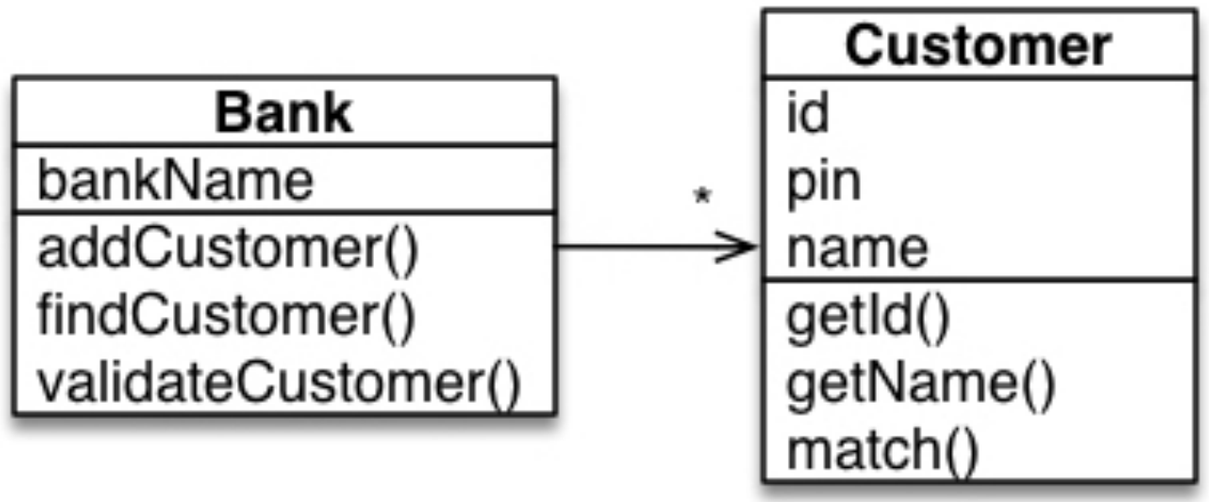
Object ในการทดสอบ

- Domain objects
  - object ในโปรแกรมเรา (จะถูกทดสอบ)
- Test objects
  - object จาก test class ต่างๆ
  - fake object
  - stub



Source : P. Tahchiev, F. Leme, V. Massol, G. Gregory. JUnit in Action, 2nd ed. Manning, 2010.

ตัวอย่าง ทดสอบโค้ดที่มี object มากกว่า 1



Implement Customer (Java)

```java
public class Customer {
    private int id;
    private int pin;
    private String name;

    public Customer(int id, int pin, String name) {
        this.id = id;
        this.pin = pin;
        this.name = name;
    }
    public int getId() {
        return id;
    }
    public boolean match(int pin) {
        return this.pin == pin;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Implement Bank (Java)

```java
public class Bank {
    private String bankName;
    private Map<Integer, Customer> customers;

    public Bank(String name) {
        this.bankName = name;
        this.customers = new HashMap<>();
    }
    public void addCustomer(Customer c) {
        customers.put(c.getId(), c);
    }
    public Customer findCustomerById(int custId) {
        return customers.get(custId);
    }

    public boolean validateCustomer(int custId, int pin) {
        Customer customer = findCustomerById(custId);
        if (customer != null && customer.match(pin)) {
            return true;
        }
        return false;
    }
}
```

ต้องการทดสอบ validateCustomer() แต่มีการใช้เมธอด match ของ Customer

จะทดสอบคลาส Bank เดี่ยว ๆ ได้อย่างไร

ตัวอย่างการทำงาน (Java)

```
public class AtmMain {

    public static void main(String[] args) {
        Bank kuBank = new Bank("KU Bank");
        Customer alice = new Customer(1, 1234, "Alice");
        Customer bob = new Customer(2, 2345, "Bob");

        kuBank.addCustomer(alice);
        kuBank.addCustomer(bob);

        System.out.println(  kuBank.validateCustomer(1, 1234)  );
        System.out.println(  kuBank.validateCustomer(5, 1234)  );
    }
}
```

```
true
false
```

Implement Customer (Python)

```python
class Customer:

    def __init__(self, id, pin, name):
        self.id = id
        self.pin = pin
        self.name = name

    def match(self, pin):
        return self.pin == pin
```

Implement Bank (Python)

```
from customer import Customer

class Bank:
    def __init__(self, name):
        self.bank_name = name
        self.customers = {}

    def add_customer(self, customer):
        self.customers[customer.id] = customer

    def find_customer_by_id(self, cust_id):
        if cust_id in self.customers:
            return self.customers[cust_id]
        return None

    def validate_customer(self, cust_id, pin):
        customer = self.find_customer_by_id(cust_id)
        if customer != None and customer.match(pin):
            return True
        return False
```

ต้องการทดสอบ validate_customer() แต่มีการใช้เมธอด match ของ Customer

จะทดสอบคลาส Bank เดี่ยว ๆ ได้อย่างไร

ตัวอย่างการทำงาน (Python)

```python
from customer import Customer
from bank import Bank

if __name__ == '__main__':
    ku_bank = Bank("KU Bank")
    alice = Customer(1, 1234, "Alice")
    bob = Customer(2, 2345, "Bob")

    ku_bank.add_customer(alice)
    ku_bank.add_customer(bob)

    print(  ku_bank.validate_customer(1, 1234)  )
    print(  ku_bank.validate_customer(5, 1234)  )
```

```
$ python atm.py
True
False
```

Unit test คลาส Customer และ Bank แยกกัน ?

- **ทดสอบคลาส Customer โดยไม่ใช้ Bank ได้ไหม ?**
  - ได้ เนื่องจาก Customer ไม่มีตัวแปร Bank ข้างใน
- **ทดสอบคลาส Bank โดยไม่ใช้ Customer ได้ไหม ?**
  - 2 กรณี
    - 1. Customer ไม่ซับซ้อน ใช้ Customer จริงได้
    - 2. Customer ซับซ้อน/ช้า ใช้ stub ของ Customer

ทดสอบคลาส Customer (Java) (1)

```java
public class CustomerTest {

    Customer customer;

    @BeforeEach
    public void setup() {
        customer = new Customer(1, 123, "Kwan");
    }
    @Test
    void testGetId() {
        assertEquals(1, customer.getId());
    }
    @Test
    void testGetName() {
        assertEquals("Kwan", customer.getName());
    }
    @Test
    void testSetName() {
        customer.setName("Noon");
        assertEquals("Noon", customer.getName());
    }
```

ทดสอบ Customer เดี่ยว ๆ ได้

ถ้าโค้ด get/set แบบ auto-generate, อาจไม่จำเป็นต้องเทส

ทดสอบคลาส Customer (Java) (2)

```
    @Test
    void testPinMatch() {
        assertTrue(customer.match(123));
    }
    @Test
    void testPinNotMatch() {
        assertFalse(customer.match(999));
    }

}
```

ทดสอบคลาส Customer (Python) (1)

ทดสอบ Customer เดี่ยว ๆ ได้

```
import unittest
from customer import Customer

class CustomerTest(unittest.TestCase):
    def setUp(self):
        self.customer = Customer(1, 123, "Kwan")

    def test_get_id(self):
        self.assertEqual(1, self.customer.id)

    def test_get_name(self):
        self.assertEqual("Kwan", self.customer.name)

    def test_set_name(self):
        self.customer.name = "Noon"
        self.assertEqual("Noon", self.customer.name)

    def test_pin_match(self):
        self.assertTrue(self.customer.match(123))

    def test_pin_not_match(self):
        self.assertFalse(self.customer.match(999))
```

Unit test คลาส Customer และ Bank แยกกัน ?

- **ทดสอบคลาส Customer โดยไม่ใช้ Bank ได้ไหม ?**
  - ได้ เนื่องจาก Customer ไม่มีตัวแปร Bank ข้างใน

- **ทดสอบคลาส Bank โดยไม่ใช้ Customer ได้ไหม ?**
  - 2 กรณี
    - 1. Customer ไม่ซับซ้อน ใช้ Customer จริงได้
    - 2. Customer ซับซ้อน/ช้า ใช้ stub ของ Customer

ทดสอบคลาส Bank กับ Customer จริง (Java)

```java
public class BankCustomerTest {
    Bank bank;
    Customer customer;

    @BeforeEach
    public void setup() {
        bank = new Bank("MyBank");
        customer = new Customer(1, 123, "Kwan");
        bank.addCustomer(customer);
    }

    @Test
    void testFindCustomer() {
        Customer found = bank.findCustomerById(1);
        assertNotNull(found);
        assertSame(customer, found);
    }

    @Test
    void testValidateCustomerValid() {
        assertTrue(bank.validateCustomer(1, 123));
    }

    @Test
    void testValidateCustomerNotValid() {
        assertFalse(bank.validateCustomer(1, 999));
    }
}
```

หากทดสอบกับ customer จริง จะเป็น **integration testing** ไม่ใช่ unit testing

ทดสอบคลาส Bank กับ Customer จริง (Python)

```
import unittest
from customer import Customer
from bank import Bank

class BankCustomerTest(unittest.TestCase):
    def setUp(self):
        self.bank = Bank("My Bank")
        self.customer = Customer(1, 123, "Kwan")
        self.bank.add_customer(self.customer)

    def test_find_customer(self):
        found = self.bank.find_customer_by_id(1)
        self.assertIsNotNone(found)
        self.assertIs(self.customer, found)

    def test_validate_customer_valid(self):
        self.assertTrue(self.bank.validate_customer(1,123))

    def test_validate_customer_not_valid(self):
        self.assertFalse(self.bank.validate_customer(1,999))
```

หากทดสอบกับ customer จริง จะเป็น integration testing ไม่ใช่ unit testing

Stub แทน complex object

Unit test คลาส Customer และ Bank แยกกัน ?

- **ทดสอบคลาส Customer โดยไม่ใช้ Bank ได้ไหม ?**
  - ได้ เนื่องจาก Customer ไม่มีตัวแปร Bank ข้างใน

- **ทดสอบคลาส Bank โดยไม่ใช้ Customer ได้ไหม ?**
  - 2 กรณี
    - 1. Customer ไม่ซับซ้อน ใช้ Customer จริงได้
    - 2. Customer ซับซ้อน/ช้า ใช้ stub ของ Customer

สมมติ Customer มีเมธอดซับซ้อน / ช้า

```java
public class Customer {
    . . .

    public boolean match(int pin) {
         // complex or slow code
         // ex. hash / encrypt / decrypt pin
    }
}
```

```python
class Customer:

    . . .

    def match(self, pin):
        # complex or slow code
        # ex. hash / encrypt / decrypt pin
```

การสร้าง stub อย่างง่าย

- ให้ stub เป็น subclass ของคลาสจริง
  - และ hardcode เมธอด/ฟังก์ชันที่ซับซ้อนหรือช้า

CustomerStub class (Java)

```
// ----- customer stub -----
class CustomerStub extends Customer {

    boolean hardCodeMatch = false;

    public CustomerStub(int id, int pin, String name) {
        super(id, pin, name);
    }

    // override complex / slow method
    public boolean match(int pin) {
        // hard code
        return hardCodeMatch;
    }
}
```

Stub เป็น subclass ของ Customer

มีตัวแปรสำหรับปรับค่า hardcode ได้

CustomerStub class (Python)

```
## -------- customer stub ---------
class CustomerStub(Customer):

    def __init__(self, id, pin, name):
        super().__init__(id, pin, name)

    # override complex / slow method
    def match(self, pin):
        # hard code
        return self.hard_code_match
```

Stub เป็น subclass ของ Customer

มีตัวแปรสำหรับปรับค่า hardcode ได้

ทดสอบคลาส Bank ด้วย CustomerStub (Java)

```java
public class BankTest {
    Bank bank;
    CustomerStub customer;

    @BeforeEach
    public void setup() {
        bank = new Bank("MyBank");
        customer = new CustomerStub(1, 123, "Kwan");
        bank.addCustomer(customer);
    }

    @Test
    void testFindCustomer() {
        Customer found = bank.findCustomerById(1);
        assertNotNull(found);
        assertSame(customer, found);
    }
    @Test
    void testValidateCustomerValid() {
        customer.hardCodeMatch = true;
        assertTrue(bank.validateCustomer(1, 123));
    }
    @Test
    void testValidateCustomerNotValid() {
        customer.hardCodeMatch = false;
        assertFalse(bank.validateCustomer(1, 999));
    }
}
```

ใช้ customer stub แทน
อ๊อบเจ็กต์ customer จริง

ปรับค่า hardcode ตามต้องการได้

ทดสอบคลาส Bank ด้วย CustomerStub (Python)

```
class BankTest(unittest.TestCase):

    def setUp(self):
        self.bank = Bank("My Bank")
        self.customer = CustomerStub(1, 123, "Kwan")
        self.bank.add_customer(self.customer)

    def test_find_customer(self):
        found = self.bank.find_customer_by_id(1)
        self.assertIsNotNone(found)
        self.assertIs(self.customer, found)

    def test_validate_customer_valid(self):
        self.customer.hard_code_match = True
        self.assertTrue(self.bank.validate_customer(1,123))

    def test_validate_customer_not_valid(self):
        self.customer.hard_code_match = False
        self.assertFalse(self.bank.validate_customer(1,999))
```

ใช้ customer stub แทน
อ๊อบเจ็กต์ customer จริง

ปรับค่า hardcode ตามต้องการได้

ทดสอบ 1 domain object ใน 1 test method

- ทดสอบ 1 domain object ในแต่ละเทสเท่านั้นใน unit testing
  - 1 class under test (CUT)
- ถ้าจำเป็นต้องใช้ object อื่น
  - ใช้ object ที่รู้ผลการทำงานอย่างแน่นอน เช่น stub, dummy object
  - การวาง class ของ object เหล่านี้
    - inner class ใน test class
    - public class

Source : P. Tahchiev, F. Leme, V. Massol, G. Gregory. JUnit in Action, 2nd ed. Manning, 2010.

ให้ stub เป็น inner class ของคลาสเทสได้ (Java)

```
public class BankTest {
    Bank bank;
    CustomerStub customer;

    @BeforeEach
    public void setup() {
        bank = new Bank("MyBank");
        customer = new CustomerStub(1, 123, "Kwan");
        bank.addCustomer(customer);
    }

    .... test methods ....

    // ----- customer stub -----
    class CustomerStub extends Customer {

        public CustomerStub(int id, int pin, String name) {
            super(id, pin, name);
        }
        // override complex / slow method
        public boolean match(int pin) {
            // hard code
       }
    }
}
```

เป็น subclass ของ Customer

วาง stub ในไฟล์เดียวกับคลาสเทส (Python)

```
class BankTest(unittest.TestCase):
    def setUp(self):
        self.bank = Bank("My Bank")
        self.customer = CustomerStub(1, 123, "Kwan")
        self.bank.add_customer(self.customer)

    def test_find_customer(self):
        found = self.bank.find_customer_by_id(1)
        self.assertIsNotNone(found)
        self.assertIs(self.customer, found)

    def test_validate_customer_valid(self):
        self.customer.hard_code_match = True
        self.assertTrue(self.bank.validate_customer(1,123))

    def test_validate_customer_not_valid(self):
        self.customer.hard_code_match = False
        self.assertFalse(self.bank.validate_customer(1,999))

## -------- customer stub ---------
class CustomerStub(Customer):

    def __init__(self, id, pin, name):
        super().__init__(id, pin, name)

    # override complex / slow method
    def match(self, pin):
        # hard code
        return self.hard_code_match
```

เป็น subclass ของ Customer

ดาวน์โหลด code และ test

- Java
  - https://github.com/ladyusa/atm-unit-test
- Python
  - https://github.com/ladyusa/atm-py-unit-test

Stub แทน external resource

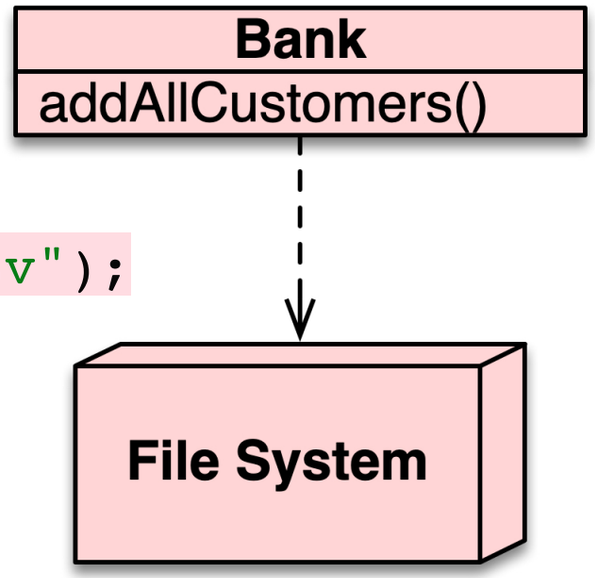
Unit testing กับ external resource

- **ทดสอบ application กับ external resource เช่น**
  - file system, connection to server, database, network, HTTP, SOAP
  - module ของเพื่อนที่ยังสร้างไม่เสร็จ หรือ server ที่ยังไม่ได้ setup
- **ทดสอบอย่างไรให้ดี**
  - ควบคุม environment ที่เรารันด้วยได้
  - เมื่อรันหลายครั้ง ต้องให้ผลเดียวกัน (reproducible)
- **วิธีการ**
  - ใช้ stub แทน environment จริง

ตัวอย่าง Bank ที่เชื่อมกับ file system โดยตรง (Java)

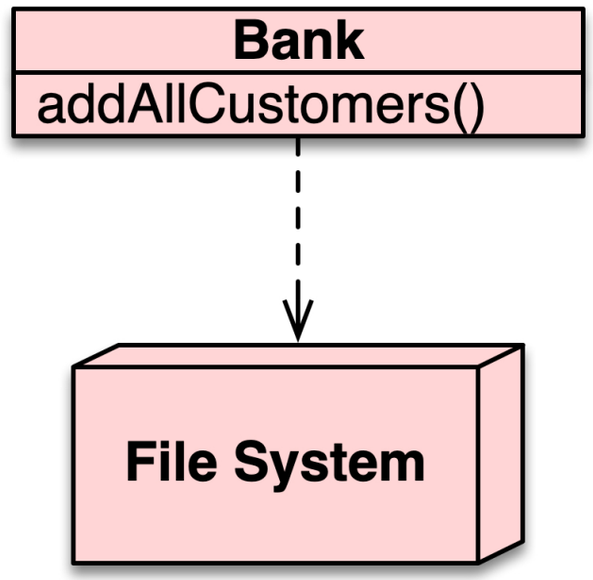
```
public class Bank {
    . . .

     public void addAllCustomers() {
          try {
               FileReader file = new FileReader("customers.csv");
               BufferedReader in = new BufferedReader(file);

               String line;
               while ((line = in.readLine()) != null) {
                    . . .
               }
          } catch (FileNotFoundException e) {
               System.err.println("File cannot be found");
          } catch (IOException e) {
               System.err.println("Error reading file");
               e.printStackTrace();
          }
     }
    . . .
}
```

เปลี่ยนเป็น stub ไม่ได้



ตัวอย่าง Bank ที่เชื่อมกับ file system โดยตรง (Python)

```
class Bank:
    def __init__(self, name):
        self.bank_name = name
        self.customers = {}

    def add_all_customers(self):
        file = open('customers.csv', 'r')
        lines = file.readlines()

        for line in lines:
            row = line.strip().split(",")
            customer = Customer(int(row[0]), int(row[1]), row[2])
            self.add_customer(customer)

def add_customer(self, customer):
    self.customers[customer.id] = customer

. . .
```



Unit testing กับ external resource

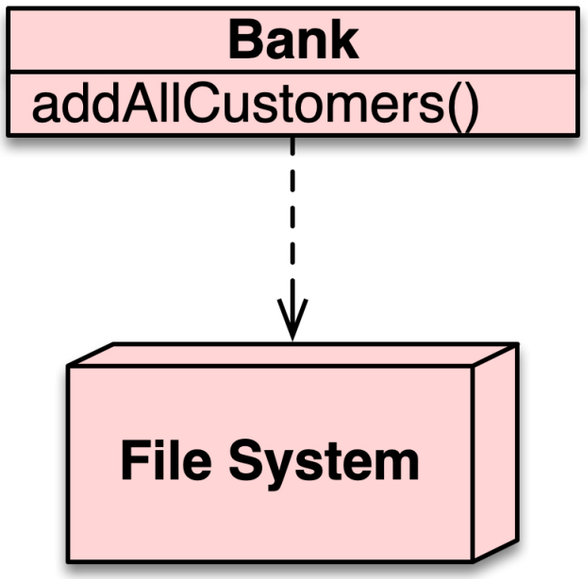
- **หากเราไม่ต้องการทดสอบกับ file system จริง**
  - เปลี่ยนไปใช้ stub แทน file system
- **แต่จะเปลี่ยน file system เป็น stub ได้อย่างไร**
  - ต้องแยก CUT กับ environment โดยการ break dependency

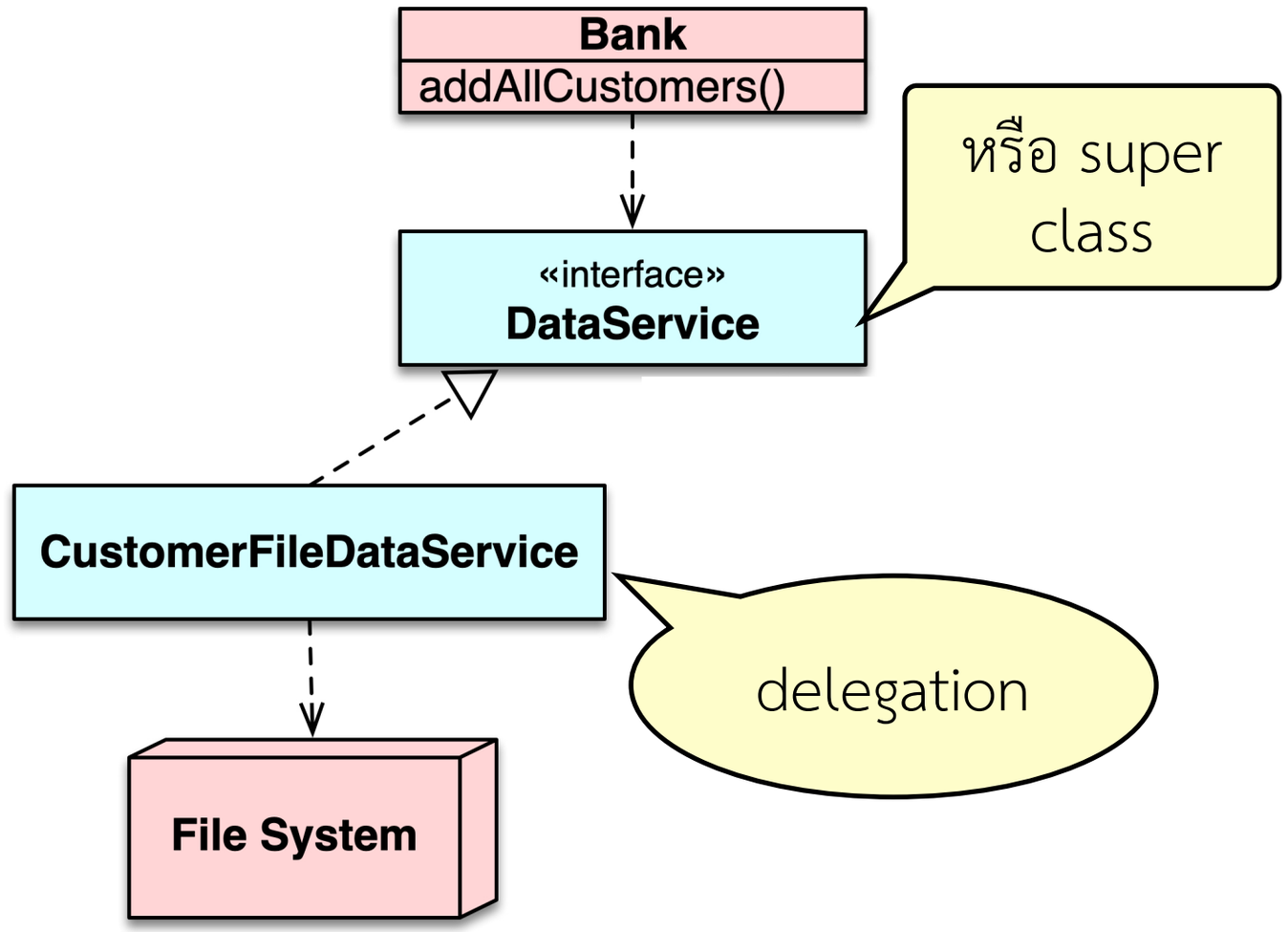
การ break dependency

- 1. หา interface / super class ของ external resource
  - ถ้าเชื่อมกันโดยตรง เพิ่ม layer of indirection และ delegation
- 2. เปลี่ยน implementation ของ interface นั้น ๆ ไปใช้ stub แทน

Layer of indirection

- เพิ่ม layer of indirect ให้กับ bank
  - ทำให้ code มีลักษณะ testable
  - design ดีขึ้น

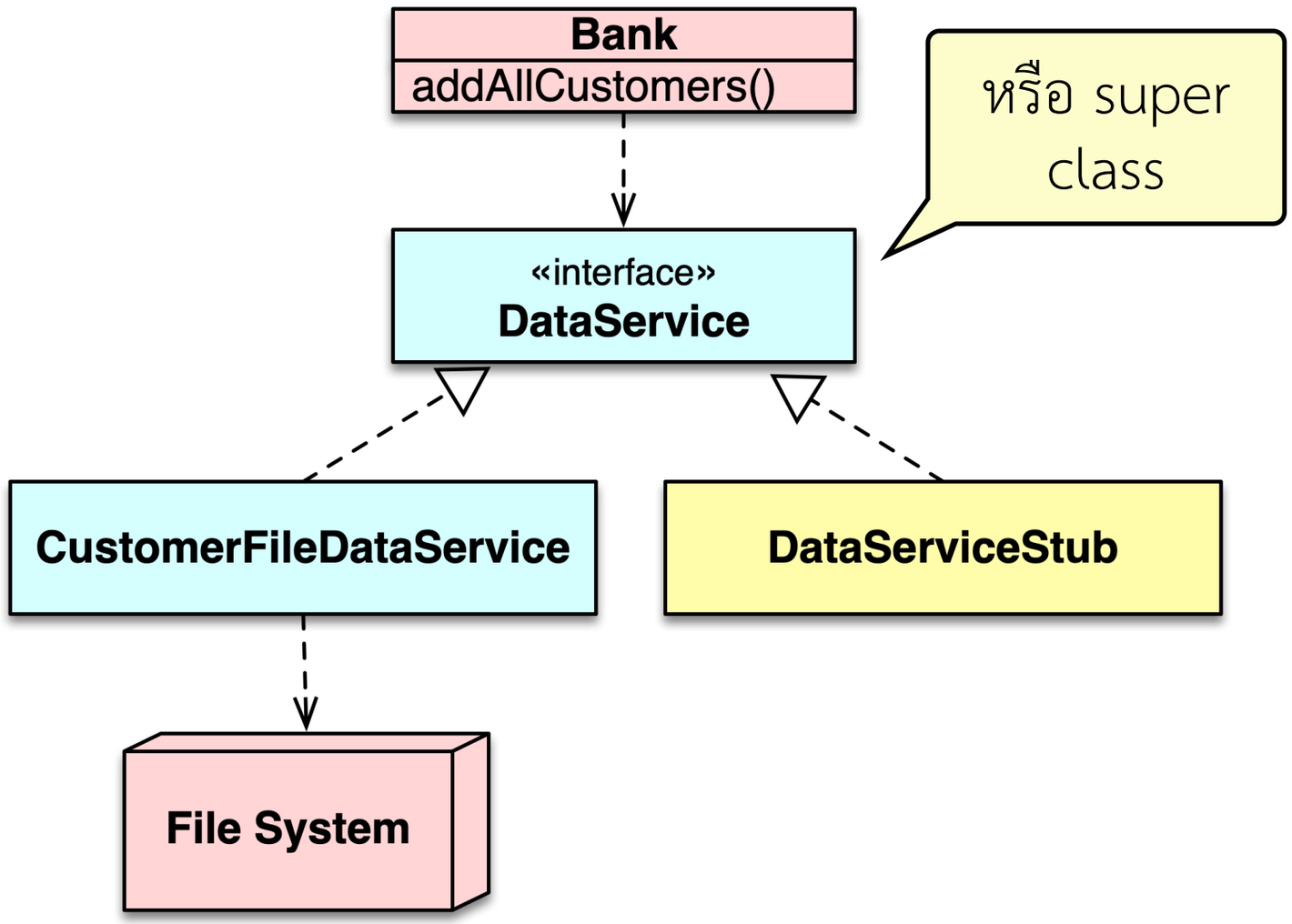




Example Source : R. Osherove. The Art of Unit Testing : with Examples in .NET. Manning, 2009.

เพิ่ม stub

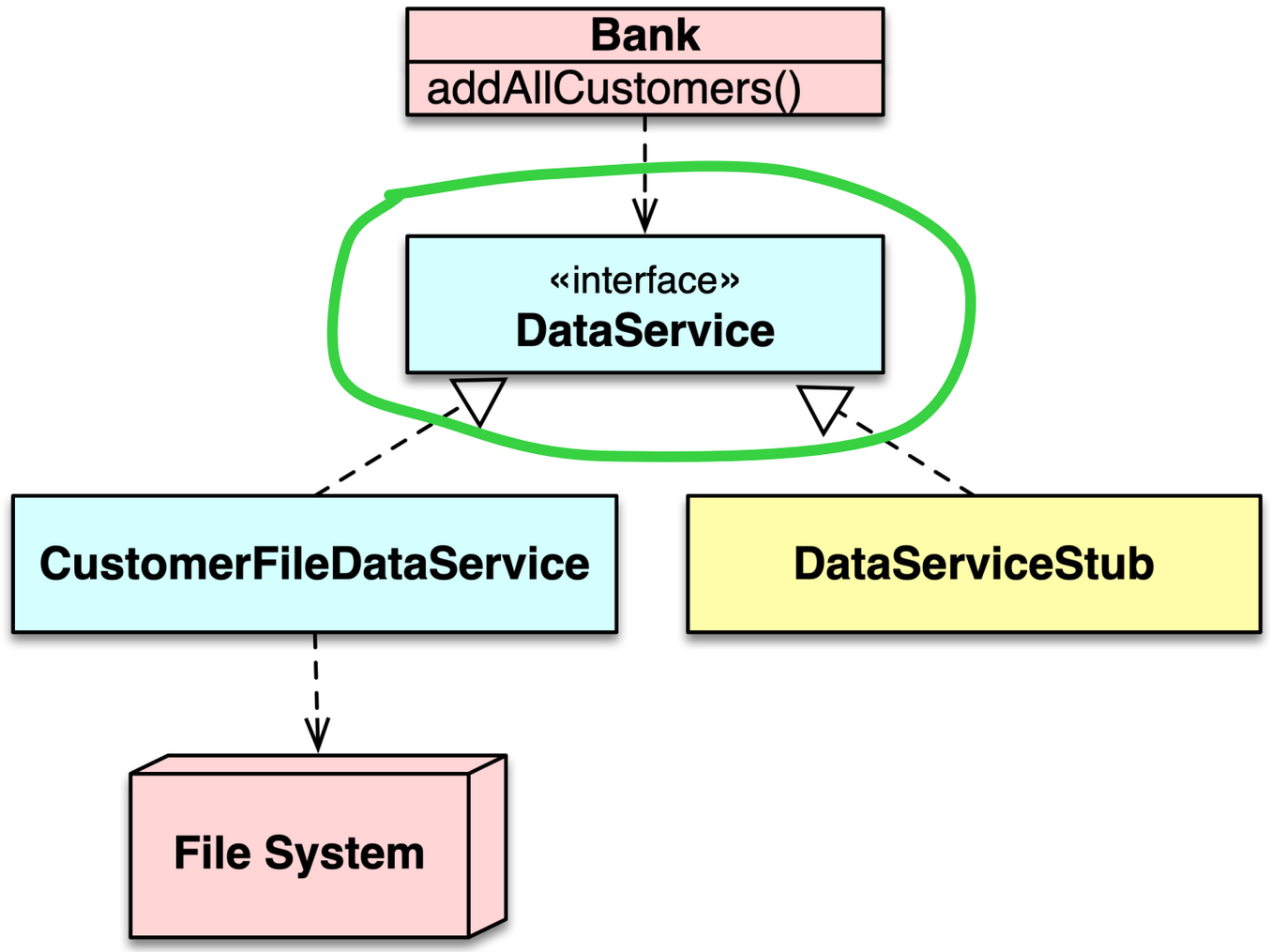
- เปลี่ยน actual implementation เป็น stub



Example Source : R. Osherove. The Art of Unit Testing : with Examples in .NET. Manning, 2009.

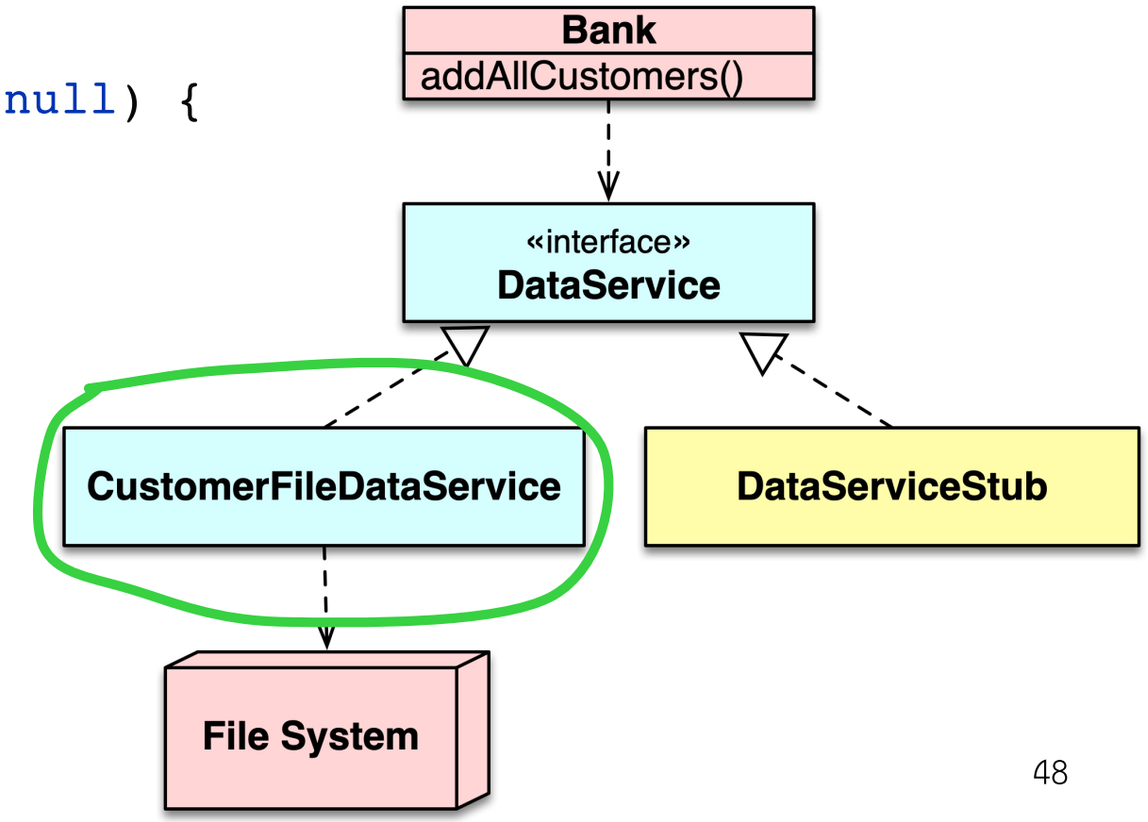
Layer of indirection (Java)

```
public interface DataService<E> {
    List<E> getAllData();
}
```



Layer of indirection (Java)

```
public class CustomerFileDataService implements DataService<Customer> {

    private String filename = "customers.csv";

    @Override
    public List<Customer> getAllData() {
        List<Customer> customers = new ArrayList<>();
        try {
            FileReader file = new FileReader(filename);
            BufferedReader in = new BufferedReader(file);

            String line;
            while ((line = in.readLine()) != null) {
                . . .
            }
        } catch (FileNotFoundException e) {
            . . .
        } catch (IOException e) {
            . . .
        }
        return customers;
    }
}
```



ปรับคลาส Bank โดยใช้ layer of indirection (Java)

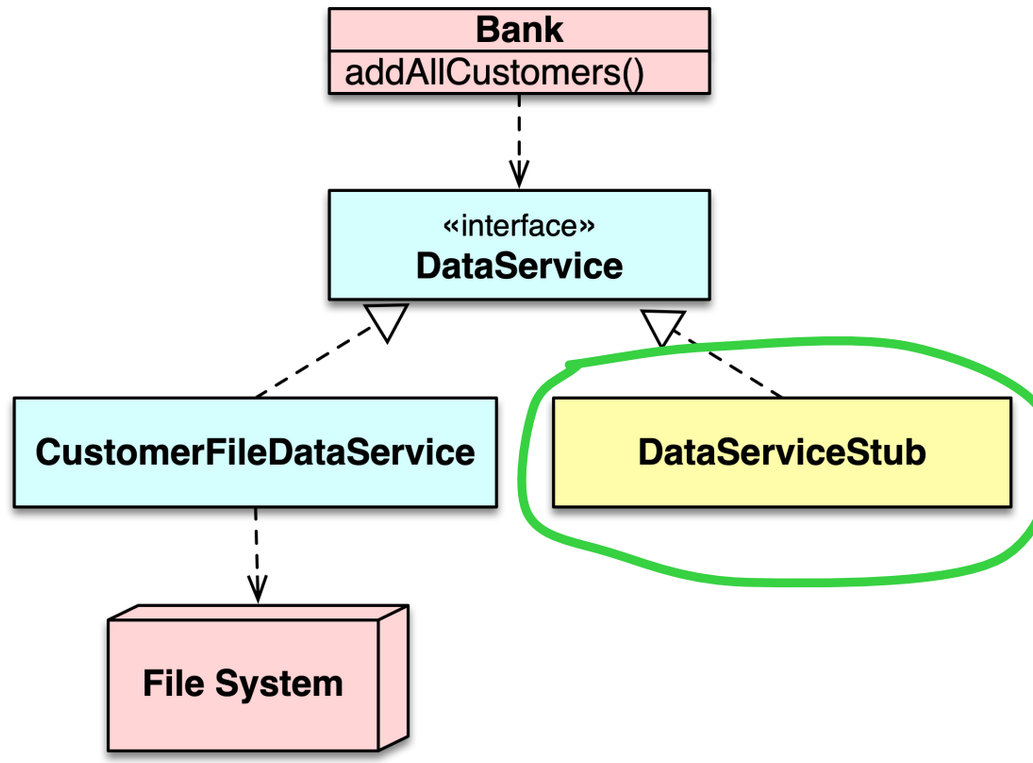
```
public class Bank {
    private String bankName;
    private Map<Integer, Customer> customers;

    private DataService<Customer> dataService;

    public Bank(String name, DataService dataService) {
        this.bankName = name;
        this.customers = new HashMap<>();
        this.dataService = dataService;
    }

    public Bank(String name) {
        this(name, new CustomerFileDataService());
    }

    public void addAllCustomers() {
        List<Customer> customerList = dataService.getAllData();
        for (Customer customer : customerList) {
            addCustomer(customer);
        }
    }
    . . .
}
```

รับ การเชื่อมกับ env / file system ผ่าน constructor

สามารถกำหนด env แบบ default ได้

Test Bank with Data Service Stub (Java)

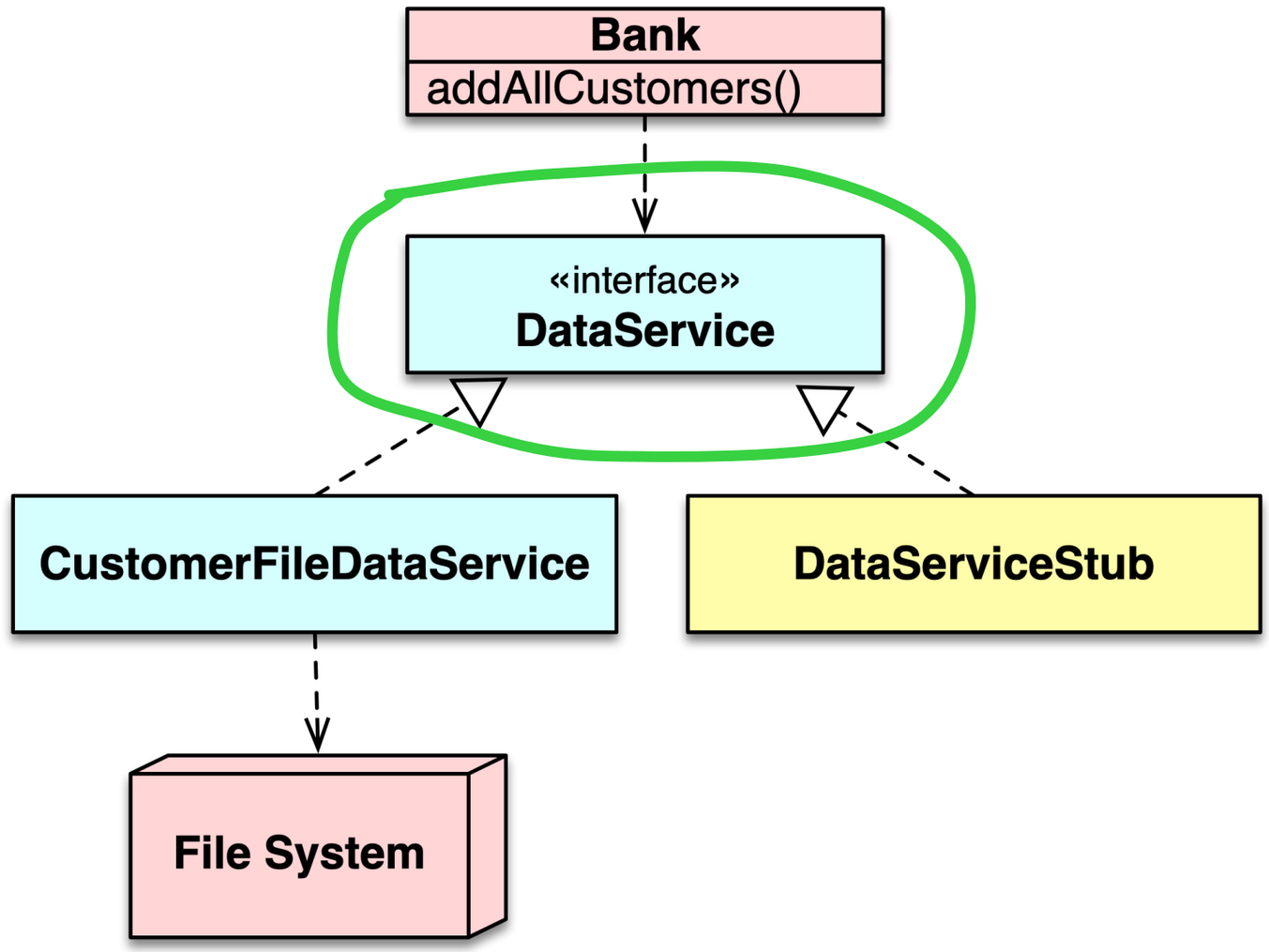
```
public class BankDataServiceStubTest {
    Bank bank;

    @BeforeEach
    void setup() {
        bank = new Bank("KU Bank", new DataServiceStub(
        bank.addAllCustomers();
    }

    @Test
    void testFindCustomerById() {
        Customer customer = bank.findCustomerById(1);
        assertEquals("Kwan", customer.getName());
    }

    . . .

    private class DataServiceStub implements DataService<Customer> {
        @Override
        public List<Customer> getAllData() {
            ArrayList<Customer> customerList = new ArrayList<>();
            customerList.add(new Customer(1, 123, "Kwan"));
            customerList.add(new Customer(2, 456, "Noon"));
            return customerList;
        }
    }
}
```



ไม่ได้อ่านจากไฟล์ แต่สร้าง customer แบบ hard-code ไปเลย

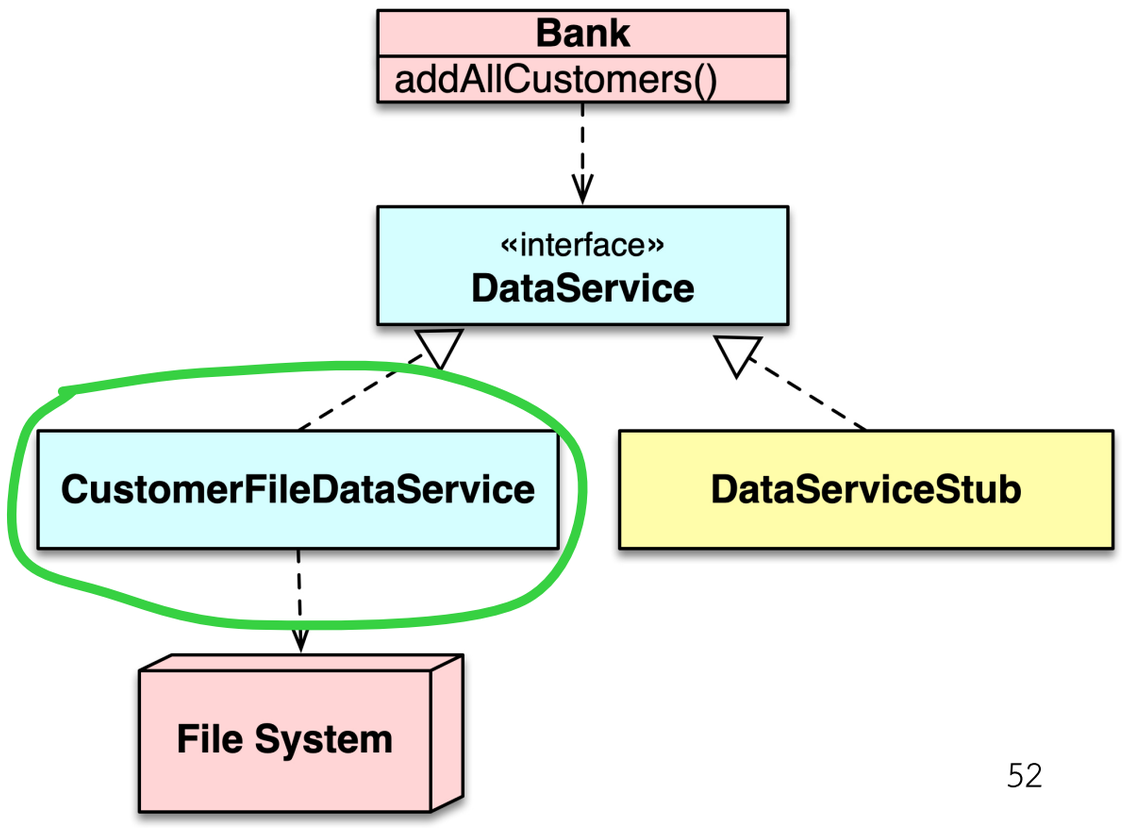
Layer of indirection (Python)

```
class DataService:
    def get_all_data(self):
        pass
```



Layer of indirection (Python)

```
from data_service import DataService
from customer import Customer

class CustomerFileDataService(DataService):
    def get_all_data(self):
        customers = []
        file = open('customers.csv', 'r')
        lines = file.readlines()

        for line in lines:
            row = line.strip().split(",")
            customer = Customer(int(row[0]), int(row[1]), row[2])
            customers.append(customer)
        return customers
```



ปรับคลาส Bank โดยใช้ layer of indirection (Python)

```python
from customer import Customer
from customer_file_data_service import CustomerFileDataService
```

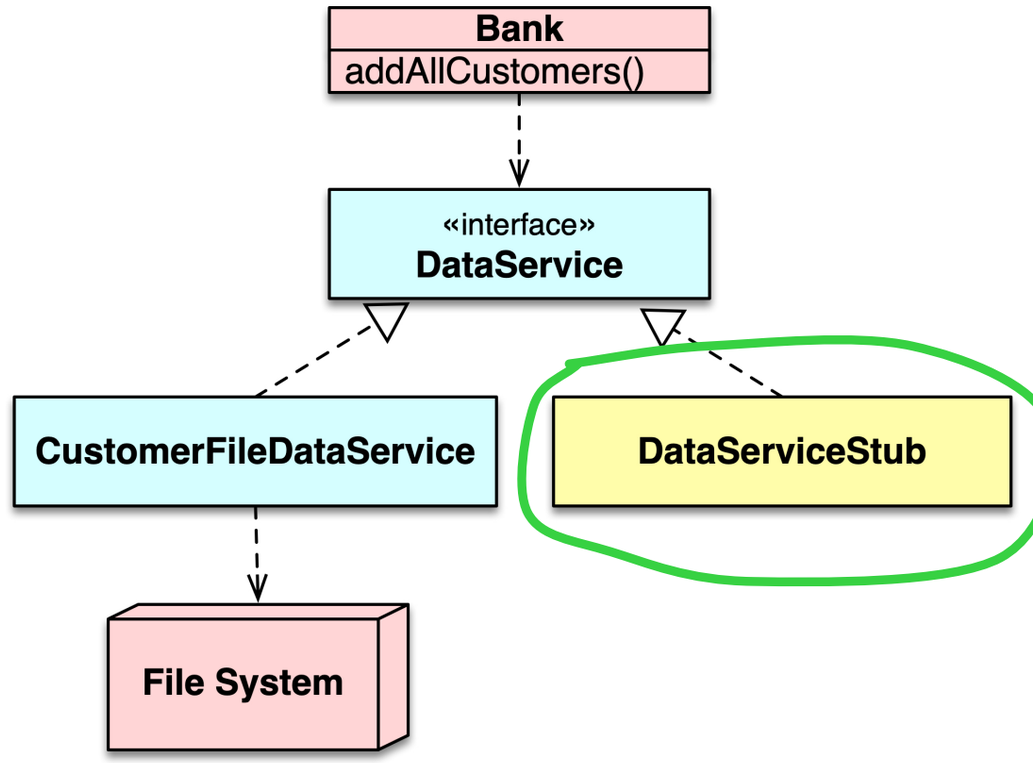
สามารถกำหนด env แบบ default ได้

```python
class Bank:
    def __init__(self, name, data_service = CustomerFileDataService()):
        self.bank_name = name
        self.customers = {}
        self.data_service = data_service

    def add_all_customers(self):
        customers = self.data_service.get_all_data()
        for customer in customers:
            self.add_customer(customer)

    def add_customer(self, customer):
        self.customers[customer.id] = customer
```

รับ การเชื่อมกับ env / file system ผ่าน constructor

Test Bank with Data Service Stub (Python)

```python
import unittest
from customer import Customer
from bank import Bank
from data_service import DataService

class BankDataServiceStubTest(unittest.TestCase):

    def setUp(self):
        self.bank = Bank("KU Bank", DataServiceStub())
        self.bank.add_all_customers()

    def test_find_customer(self):
        customer = self.bank.find_customer_by_id(1)
        self.assertEqual("Kwan", customer.name)

    # . . .

class DataServiceStub(DataService):
    def get_all_data(self):
        customers = []
        customers.append(Customer(1, 123, "Kwan"))
        customers.append(Customer(2, 456, "Noon"))
        return customers
```



ไม่ได้อ่านจากไฟล์ แต่สร้าง customer แบบ hard-code ไปเลย

การรับค่า stub

- รับผ่าน constructor parameter
- รับผ่าน setter (Java)
  - กำหนดค่าให้เป็น stub โดยตรง (Python)
- extend CUT ด้วย class ที่ใช้ stub
- อื่นๆ เช่น
  - ผ่าน factory method / abstract factory pattern

รับ stub ผ่านทาง setter method (Java 1)

```
public class Bank {
    private String bankName;
    private Map<Integer, Customer> customers;
    private DataService<Customer> dataService;

    // . . . constructors . . .

    protected DataService<Customer> getDataService() {
        return dataService;
    }

    protected void setDataService(DataService<Customer> dataService) {
        this.dataService = dataService;
    }
```

"protected" เพื่อให้ subclass หรือ class ใน package เดียวกันใช้ได้เท่านั้น

(test class อยู่ใน package เดียวกัน)

รับ stub ผ่านทาง setter method (Java 2)

- ในคลาสการทดสอบ

```
public class BankDataServiceStubTest {
    Bank bank;

    @BeforeEach
    void setup() {
        bank = new Bank("KU Bank");
        bank.setDataService(new DataServiceStub());
        bank.addAllCustomers();
    }

    . . .
}
```

รับ stub ผ่านทาง setter method

Example Source : R. Osherove. The Art of Unit Testing : with Examples in .NET. Manning, 2009.

รับ stub ผ่านการกำหนดค่าโดยตรง (Python)

- ในคลาสการทดสอบ

```
class BankDataServiceStubTest(unittest.TestCase):

    def setUp(self):
        self.bank = Bank("KU Bank")
        self.bank.data_service = DataServiceStub()
        self.bank.add_all_customers()

    . . .
```

กำหนดให้เป็น stub โดยตรง

Example Source : R. Osherove. The Art of Unit Testing : with Examples in .NET. Manning, 2009.

การเลือกใช้

- **การรับค่าผ่าน constructor parameter**
  - ใช้สำหรับ mandatory variable
  - อาจทำให้ต้องแก้โค้ดเพื่อเพิ่ม constructor parameter
- **การรับค่าผ่าน setter method**
  - ใช้สำหรับ optional variable
  - อาจทำให้เสีย information hiding (อาจแก้โดยใช้ protected)

ดาวน์โหลด code และ test

- Java
  - https://github.com/ladyusa/atm-unit-test-stub-env
- Python
  - https://github.com/ladyusa/atm-py-unit-test-stub-env

สรุป

- **การทดสอบ unit testing ด้วย stub เบื้องต้น**
  - ใช้คลาส / ฟังก์ชัน / เมธอด แบบ hard code ---> เรียกว่า stub
  - นำ stub มาใช้แทนคลาสที่ยังไม่ต้องการทดสอบ
  - เพื่อให้สามารถทดสอบ class under test โดยไม่ต้องใช้คลาส/เมธอดอื่น
- **การใช้ stub (1) แทน complex class**
  - โดยการสืบทอด complex class แล้ว hard code การทำงาน
- **การใช้ stub (2) แทน external resource (file, network, etc)**
  - สร้างคลาสใหม่มาครอบ external resource แล้วสร้าง stub ที่มี interface/super class เดียวกับคลาสใหม่นั้น

Reference

- P. Tahchiev, F. Leme, V. Massol, G. Gregory. JUnit in Action, 2nd ed. Manning, 2010.
- R. Osherove. The Art of Unit Testing : with Examples in .NET. Manning, 2009.