

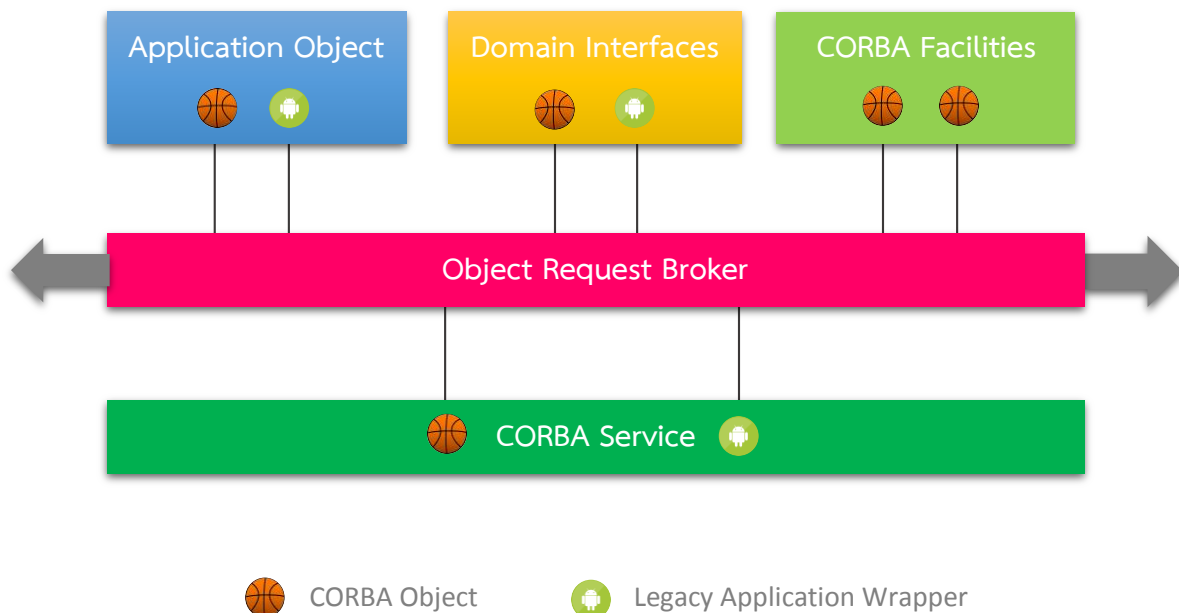
CORBA

คอร์บา เป็นรูปแบบมาตรฐานที่สร้างขึ้นโดยโอเอ็มจี (OMG-Object Management Group) ซึ่งเป็นองค์กรที่ไม่แสวงหาผลประโยชน์อยู่ที่สหรัฐอเมริกา ก่อตั้งเมื่อเมษายน 1989

โอเอ็มจีพัฒนาคอร์บาขึ้นมามีวัตถุประสงค์ดังนี้

- เพื่อสนับสนุนการทำงานอ็อบเจกต์แบบกระจาย (distributed object) และมีความแตกต่างกันทั้งแพลตฟอร์มผู้ใช้และผู้พัฒนาโปรแกรม
- ทำงานผสมผสานที่สร้างใหม่และเก่าที่มีอยู่แล้ว
- โดยไม่คิดค่าใช้จ่าย
- จะต้องอยู่ในส่วนที่เป็นมาตรฐานที่ได้ตกลงกันไว้แล้ว

สถาปัตยกรรมของการจัดการวัตถุ (Object Management Architecture) แสดงดังภาพ



ภาพที่ 1 แสดงสถาปัตยกรรมของคอร์บา

จากภาพที่ 1 แต่ละส่วนมีหน้าที่ดังนี้

- **Application Object** คือโปรแกรมที่นักพัฒนาเขียนขึ้น
- **Domain Interfaces** คือโดเมนที่สร้างขึ้นเพื่อให้ผู้อื่นเรียกใช้งาน
- **CORBA facilities** คือส่วนที่เตรียมไว้สำหรับอำนวยความสะดวก

- ORB คือตัวกลางที่จะหา object และทำการ activate ให้
- CORBA Service คือส่วนที่ให้บริการแก่ผู้ใช้งาน

โมเดลอ็อบเจกต์และไอดีแอล

ก่อนอื่นขออธิบายส่วนประกอบที่ใช้ในการพัฒนาโปรแกรมของ CORBA ตามหัวข้อต่อไปนี้

- Object
- Type
- Modules
- Attribute
- Operations
- Requests
- Exceptions
- Subtypes

รายละเอียดที่สำคัญดังกล่าวข้างบนจะบรรจุอยู่ในส่วน OMG Interface Definition Language คือ ภาษาสำหรับการใช้งานทุก ๆ แนวคิดของ CORBA Object Model ซึ่งมีรายละเอียดดังนี้

- เป็นโปรแกรมภาษาที่ไม่ขึ้นกับภาษาโปรแกรมใด ๆ
- รูปแบบของภาษามาจาก C++
- ยังไม่สมบูรณ์จะต้องมีการเขียนเพิ่มเติมภายหลังจากการคอมไพล์ ซึ่งจะมีเมธอดให้ใช้งานแต่จะไม่บอกว่าจะทำอย่างไรบ้าง
- ในรายละเอียดของอ็อบเจกต์โมเดลแต่ละอย่างนี้จะอธิบายโดยใช้ตัวอย่างการใช้ ไอดีแอลไฟล์ เพื่อให้เห็นส่วนประกอบได้ชัดเจนยิ่งขึ้น

CORBA Object Model: Objects

- แต่ละ object จะมีเพียงชื่อเดียวที่ไม่ซ้ำกันภายใต้ ORB
- สามารถอ้างอิงไปยัง Object ที่มากกว่า 1 ได้ (Multiple reference to objects)
- Object reference จะอยู่อย่างถาวร แต่จะทำการ Deactivate ถ้าไม่ได้ใช้งานนาน ๆ แต่ก็ยังไม่ใช้งานหมายเลขเดิมได้อีก

CORBA Object Model: Type

กลุ่มของ Type มีอยู่ในส่วนของ CORBA จะสามารถแยกออกได้ดังนี้

1. Constructed type ที่เป็นลักษณะของโครงสร้างคล้ายกับ struct ของภาษาซีนั่นเอง
2. Atomic type เป็น type ที่เล็กที่สุดโดยปกติแล้วจะคล้ายกับ type มาตรฐานของข้อมูลของโปรแกรมภาษาต่างๆ เช่น int, string เป็นต้น
3. Object type คือ type ที่เป็น Object

CORBA Object Model: Modules

เนื่องจากใน ORB นั้นไม่สามารถที่จะมีชื่อของ Module ซ้ำกันได้ ดังนั้น Module นั้นจะต้องสร้างเพื่อให้เป็นไปตาม ORB และป้องกันการซ้ำกันของ type ต่าง ๆ ซึ่งแสดงให้เห็นดังตัวอย่าง ข้อสังเกตจะพบว่าชื่อโมดูลจะจำแนก type ได้ จะช่วยจำแนกรายละเอียดต่าง ๆ หากเกิดการซ้ำกันและมีการเรียกใช้ type นั้น ๆ

CORBA Object Model: Attribute

ใน Middleware* นั้นยอมให้ประกาศ Attribute และประเภท type โดยทางด้านของไคลเอนต์จะไม่สามารถเปลี่ยนแปลง Attribute ได้ ตัวอย่างที่แสดงถึง Attribute ดังนี้

- Attribute เป็น public เพราะจะต้องเปิดเผยให้คนอื่นรู้ เพื่อนำไปใช้จึงจะสามารถใช้งานได้

CORBA Object Model: Operations

ในส่วนของ Operations คือการกำหนดให้ method อะไรบ้าง เพื่อให้ผู้ใช้เรียกใช้งาน ตัวอย่างดังนี้

CORBA Object Model: Request

Request จะถูกกำหนดโดย Client Object และหากเป็น public นั้นจะสร้าง setter method และ getter method มาให้ แต่ถ้าหากเป็น read only จะไม่มี setter method ให้ แต่จะมีชื่อเดียวกับ attribute โดย request จะประกอบด้วย

- Reference of Server object
- Name of requested Operation
- Actual request parameter

* Middleware คือ software computer ที่คอยช่วยเหลือดูแล application ที่รันอยู่บน OS หรือจะเรียกว่าตัวเชื่อมระหว่าง APP และ OS ก็ได้

- Context information

Request จะประมวลผลหรือเรียกใช้งาน Executed โดยการ synchronously การกำหนด request สามารถที่จะกำหนดได้ถ้าตายตัว (statically) โดย Client จะต้องรู้ว่ามี operation ไตบ้างให้ใช้ และแบบยืดหยุ่น(dynamically)

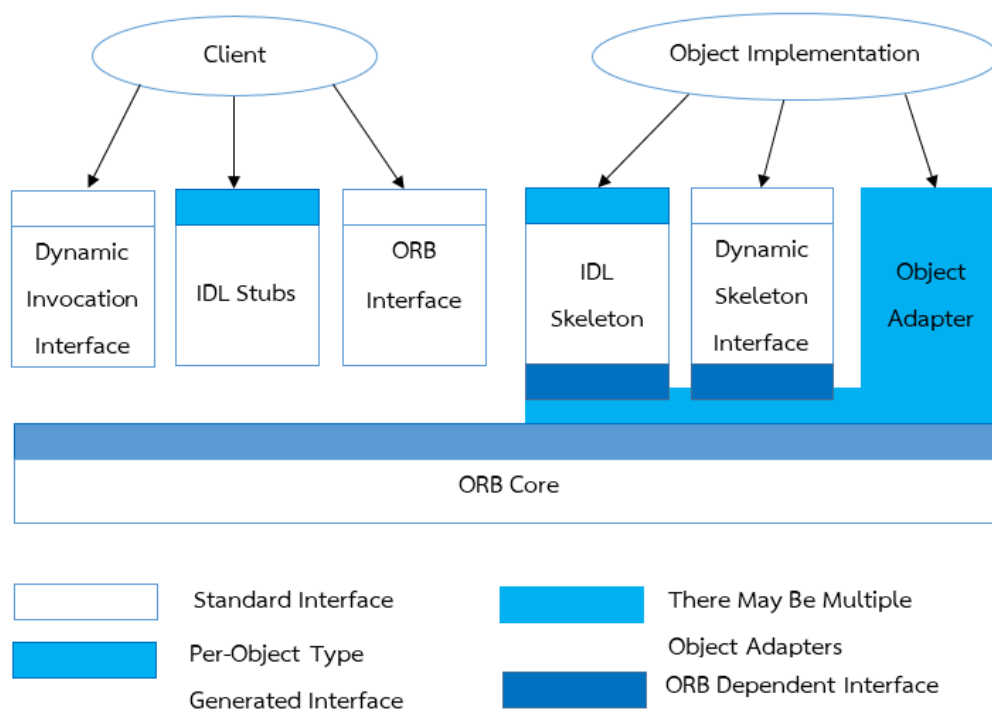
CORBA Object Model: Exception

โดยปกติแล้ว Exception มี 2 ประเภท คือ

1. Generic Exception ซึ่งเป็นส่วนที่รายงานออกมาให้ทราบเองเช่น Network down, invalid object หรือ out of memory
2. Type-specific Exception ประเภทนี้ผู้เขียนโปรแกรมต้องเขียนจับ Exception เอง ในการใช้งาน CORBA เองก็สามารถที่จะเขียนส่วนจับ Exception เองได้

CORBA Object Model: Subtype

การกล่าวถึงการสืบทอดจะใช้ subtype ในการสืบทอดคุณสมบัติของ Object ที่ประกาศขึ้นมาในส่วนนี้จะแสดงดังตัวอย่างสถาปัตยกรรมและเครื่องมือของ Java IDL (Architecture & Java IDL Tools) โครงสร้างของการติดต่อระหว่างไคลเอนต์และซอฟต์แวร์นั้นจะต้องผ่านตัวกลางที่ใช้คือ ORB Core ซึ่งเป็นตัวกลางเชื่อมต่อในการส่งข้อมูลถึงกัน โครงสร้างการทำงาน



ภาพที่ 2 แสดงสถาปัตยกรรมของ Java IDL

การทำงานจะมี ORB Core เป็นตัวเชื่อมต่อ

Dynamic Invocation และ client stubs จะเป็นส่วนที่ client ใช้ในการติดต่อลงไปที่ ORB Core นอกจากนั้น ต้องทำหน้าที่ในส่วนที่เป็น ORB Interface เพื่อลงทะเบียนในฝั่งเซิร์ฟเวอร์และฝั่งไคลเอนต์จะใช้ในการค้นหา ในการทำ จะติดต่อกับอ็อบเจกต์นั้นใช้ skeletons (Implement) เพื่อเข้าใช้อ็อบเจกต์นั้น ๆ โดยตัว Object Adapter จะเป็นส่วนที่ ควบคุมการ activate/deactivate การที่จะมีแต่ละส่วนในการใช้งานอ็อบเจกต์นั้นจะอธิบายตามหมายเลขที่ให้ไว้

1. มีเพียง 1-Interface และเป็นมาตรฐานเดียวกัน
2. มีเพียง 1-Interface เดียวสำหรับ Object Operation
3. จะมีเพียง 1-Interface สำหรับ Object Adapter
4. จะเป็นส่วนของ ORB ที่เป็นอิสระจาก Interface

Java IDL Tools

ในส่วนของ Java จะมี idlj ที่ใช้ในการ compile IDL ให้เป็นภาษาจาวา

- Orbd จะเป็นส่วนของ server process ที่เตรียม Naming Series และบริการอื่น ๆ ที่จัดการเกี่ยวกับการจัดการ ORB
- Server tool จัดเตรียมคำสั่งสำหรับการจัดการโปรแกรมทั้งส่วนที่ลงทะเบียน ยกเลิกการลงทะเบียน รวมถึงการ Start และ shutdown server ในการให้บริการ

การเขียนโปรแกรมผ่านระบบเครือข่ายด้วยจาวาคอร์บา

นอกเหนือจากความรู้เชิงวัตถุ ความรู้เกี่ยวกับขั้นตอนการพัฒนาซอฟต์แวร์โดยอาศัย Object Oriented Middleware ความรู้เกี่ยวกับภาษาไอดีแอล และการพัฒนาโปรแกรมสำหรับใช้งานจริง ดังนั้นในส่วนนี้จึงกล่าวถึงขั้นตอน ดังกล่าวมีดังนี้

1. เขียน IDL file
2. ใช้ idlj คอมไพล์
3. เขียนส่วน Implement ให้สมบูรณ์
4. คอมไพล์แต่ละส่วน(Client & Server)
5. แยกส่วน client server
6. Start ORB และ Register(ลงทะเบียน)

จากตัวอย่างที่ 1 กำหนดให้บันทึกแฟ้มในชื่อ hello.idl หลังจากนั้นจะมีขั้นตอนการสร้างส่วนต่าง ๆ เพื่อให้โปรแกรมทำงานดังนี้

1. ขั้นตอนที่ 1 สร้างแฟ้ม hello.idl

```
module HelloApp
{
    interface Hello
    {
        string sayHello();
        oneway void shutdown();
    };
};
```

2. คอมไพล์ hello.idl ด้วย idlj ดังนี้

```
Idlj -f all hello.idl
```

หลังจากนั้นจะได้แฟ้มทั้งหมดภายใต้ subdirectory HelloApp ซึ่งเป็นชื่อของ Module idlj แฟ้มดังนี้

- HelloOperations.java
- Hello.java
- HelloHelper.java
- HelloHolder.java
- HelloStub.java
- HelloPOA.java

HelloOperations.java เป็นส่วนที่เป็น interface ของ Operation โดยจะเป็นส่วนที่แปลงจาก IDL interface

```
public interface HelloOperations
```

```
{
```

```
    String sayHello ();
```

```
    void shutdown ();
```

- Hello.java แฟ้มนี้จะเป็นส่วนที่เป็นคุณสมบัติที่ประกาศให้ไอดีแอลและส่วนที่เป็นคอร์บารวมกัน แล้วกำหนดส่วนการแปลงจาก Interface เข้ารวมกันดังตัวอย่างต่อไปนี้

```
public interface Hello extends HelloOperations, org.omg.CORBA.Object,
```

```
org.omg.CORBA.portable.IDLEntity
```

```
{
```

```
,
```

- HelloHelper.java เป็นฟังก์ชันช่วยให้เราเรียกใช้อ็อบเจกต์ของคอร์บาผ่านทางภาษาได้
- HelloHolder.java ใช้โดยการ Map in, out พารามิเตอร์ของไอดีแอลให้อยู่ในรูปของภาษาจาวาให้ถูกต้อง
- HelloStub.java จะเป็นส่วนที่เป็น implement Hello.java interface
- HelloPOA.java เป็นส่วนเซิร์ฟเวอร์ stub หรือ server skeleton และ POA (Portable Object Adapter)

ฝั่งของเซิร์ฟเวอร์ (Server side classes)

ส่วนของ server จะมี 2 คลาสที่จำเป็นต้องเตรียมสำหรับการให้บริการ การร้องขอของไคลเอนต์ โดยจะแยกออกดังนี้

1. Implement ของ IDL ในที่นี้คือ HelloImpl นิยามแบ่งออกเป็น 2 คลาสคือ HelloImpl.java, HelloServer.java
2. Object ของแต่ละคลาส ซึ่งจะเป็น instant ของคลาสแต่ละคลาสในที่นี้คือ Object Hello

จึงเขียนโปรแกรมแสดง Hello โดยใช้จาวาคอร์บา

1. สร้างโมดูลจากเท็กซ์ติเตอร์ดังนี้

```
module HelloApp
{
    interface Hello
    {
        string sayHello();

        oneway void shutdown();
    };
};
```

แล้วบันทึกไว้ในตำแหน่งที่ต้องการเป็นชื่อ HelloApp.idl

2. คอมไพล์ด้วย idlj -f all hello.idl

The screenshot shows a Windows command prompt window titled "Administrator: C:\Windows\system32\cmd.exe". The user is in the directory D:\Code\Test. The command 'dir' is executed, showing a directory listing with files and subdirectories. The output shows a file named 'hello.idl' with a size of 98 bytes. The command 'idlj -f all hello.idl' is then executed, and the prompt returns to D:\Code\Test>.

```
Administrator: C:\Windows\system32\cmd.exe

D:\Code\Test>dir
Volume in drive D has no label.
Volume Serial Number is 2859-9C12

Directory of D:\Code\Test

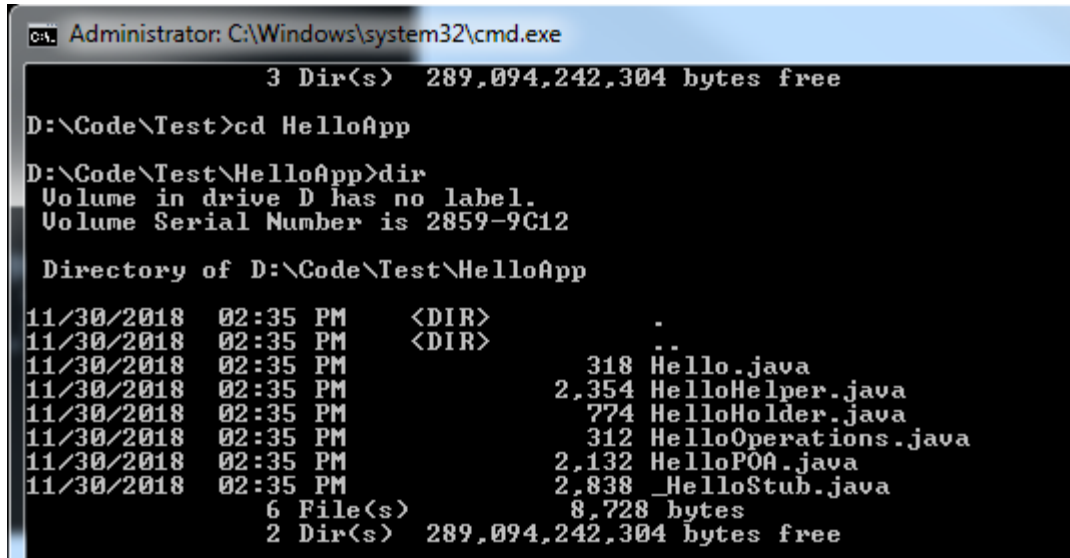
11/30/2018  02:34 PM    <DIR>          .
11/30/2018  02:34 PM    <DIR>          ..
11/29/2018  10:16 AM                98 hello.idl
               1 File(s)                98 bytes
               2 Dir(s)  289,094,262,784 bytes free

D:\Code\Test>idlj -f all hello.idl

D:\Code\Test>
```

ภาพที่ 8 แสดงการคอมไพล์เพิ่ม IDL

3. จะได้โฟลเดอร์ขึ้นมาใหม่ตามชื่อโมดูล HelloApp และภายในจะบรรจุแฟ้มดังนี้



```

Administrator: C:\Windows\system32\cmd.exe

3 Dir(s) 289,094,242,304 bytes free

D:\Code\Test>cd HelloApp
D:\Code\Test\HelloApp>dir
Volume in drive D has no label.
Volume Serial Number is 2859-9C12

Directory of D:\Code\Test\HelloApp

11/30/2018  02:35 PM    <DIR>          .
11/30/2018  02:35 PM    <DIR>          ..
11/30/2018  02:35 PM                318 Hello.java
11/30/2018  02:35 PM            2,354 HelloHelper.java
11/30/2018  02:35 PM                774 HelloHolder.java
11/30/2018  02:35 PM                312 HelloOperations.java
11/30/2018  02:35 PM            2,132 HelloPOA.java
11/30/2018  02:35 PM            2,838 _HelloStub.java
               6 File(s)              8,728 bytes
               2 Dir(s) 289,094,242,304 bytes free
  
```

ภาพที่ 9 แสดงรายละเอียดของแฟ้มจาวาหลังการคอมไพล์

4. หลังจากนั้นให้เขียนโปรแกรมส่วนการอิมพลีเมนต์เซิร์ฟเวอร์ โดยสร้างคลาสคือ HelloServer.java

```

import HelloApp.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import org.omg.PortableServer.POA;

import java.util.Properties;

public class HelloServer{

    public static void main(String args[]){

        try{
  
```

```
ORB orb = ORB.init(args, null);

//get rootpoa address and call POA Manager

POA rootpoa =
POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

rootpoa.the_POAManager().activate();

//create usage part and register with ORB

HelloImpl helloImpl = new HelloImpl();

helloImpl.setORB(orb);

//get address from usage

org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);

Hello href = HelloHelper.narrow(ref);

//get root naming

//NameService call usage name

org.omg.CORBA.Object objRef =
orb.resolve_initial_references("NameService");

//define service part

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

//combine object and name

String name = "Hello";

NameComponent path[] = ncRef.to_name( name );

ncRef.rebind(path, href);
```

```

        System.out.println("HelloServer ready and waiting ...");

        //wait request from client

        orb.run();

    }    catch (Exception e) {

        System.err.println("ERROR: " + e);

        e.printStackTrace(System.out);

    }

    System.out.println("HelloServer Exiting ...");

}

}

```

คำอธิบาย

1. สร้างส่วนการติดตั้ง ORB
 2. รับส่วนการอ้างอิงไปยัง root POA และสั่งการทำงานของ POAManager
 3. สร้างส่วนการทำงานหรือการสร้างอ็อบเจกต์ CORBA Hello object และเป็นการบอก ORB ให้รับรู้
 4. รับชื่ออ้างอิง(CORBA object reference for a naming context) ส่วนที่ได้ลงทะเบียนคอร์บาใหม่ไว้
 5. รับส่วนชื่อ root naming context
 6. ลงทะเบียนอ็อบเจกต์ใหม่ใน naming context ภายใต้อ้างอิงชื่อ “Hello”
 7. รอกการสั่งทำงานจากไคลเอนต์
-
5. สร้างโปรแกรมอิมพลีเมนต์ส่วนของไคลเอนต์ โดยทำตามขั้นตอนดังนี้
 - สร้างส่วนการติดตั้ง ORB
 - แสดงชื่ออ้างอิงไปยัง root naming context
 - ค้นหาชื่อ “Hello” ใน naming context และรับชื่ออ้างอิงนั้นมาใช้อ้างอิง

- เรียกการทำงานเมธอดสำหรับการแสดงข้อความ sayHello() และ shutdown()

สร้างโปรแกรมชื่อ HelloClient.java ตามนี้

```
import HelloApp.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

public class HelloClient

{

    static Hello helloImpl;

    public static void main(String args[])

    {

        try {

            //create ORB Installation

            ORB orb = ORB.init(args, null);

            //get root naming context

            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");

            //use NamingContextExt replace NamingContext is part of service

            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            //search object that refered

            String name = "Hello";
```

```

helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));

System.out.println("Obtained a handle on server object: " + helloImpl);

System.out.println(helloImpl.sayHello());

helloImpl.shutdown();

} catch(Exception e){

    System.out.println("ERROR : " + e);

    e.printStackTrace(System.out);

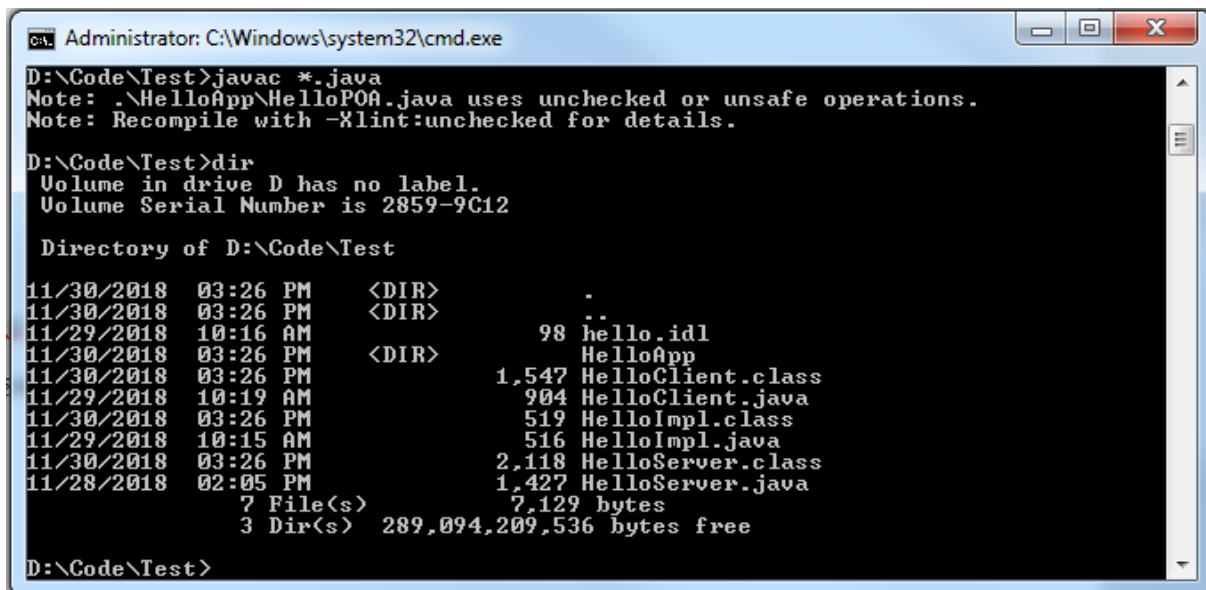
}

}

}

```

6. บันทึกแฟ้มจะต้องบันทึกเอาไว้ในโฟลเดอร์ HelloApp
7. คอมไพล์แฟ้มทั้งสองแฟ้ม โดยใช้คำสั่ง `javac *.java` เมื่อคอมไพล์เสร็จแล้วจะได้แฟ้มดังนี้



```

Administrator: C:\Windows\system32\cmd.exe

D:\Code\Test>javac *.java
Note: .\HelloApp\HelloPOA.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

D:\Code\Test>dir
Volume in drive D has no label.
Volume Serial Number is 2859-9C12

Directory of D:\Code\Test

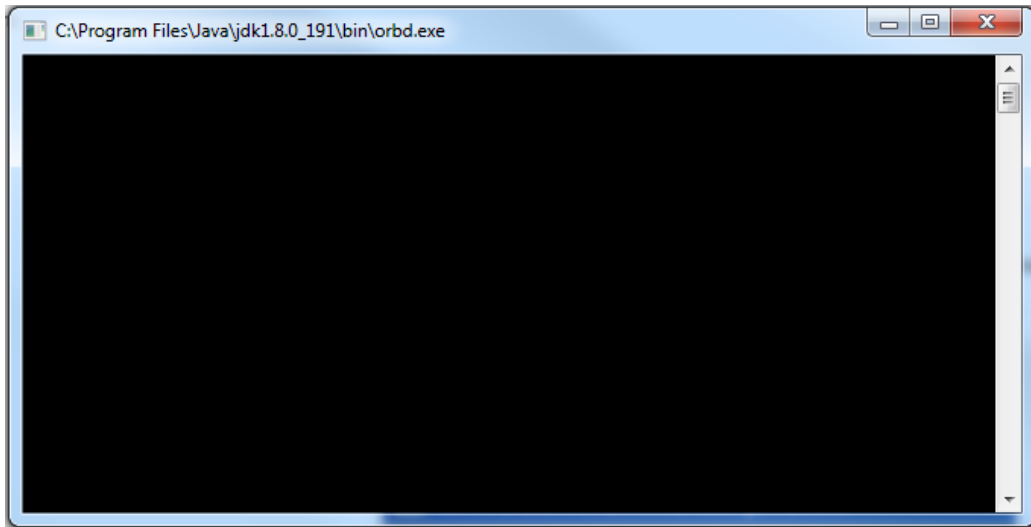
11/30/2018  03:26 PM    <DIR>          .
11/30/2018  03:26 PM    <DIR>          ..
11/29/2018  10:16 AM             98 hello.idl
11/30/2018  03:26 PM    <DIR>          HelloApp
11/30/2018  03:26 PM             1,547 HelloClient.class
11/29/2018  10:19 AM             904 HelloClient.java
11/30/2018  03:26 PM             519 HelloImpl.class
11/29/2018  10:15 AM             516 HelloImpl.java
11/30/2018  03:26 PM             2,118 HelloServer.class
11/28/2018  02:05 PM             1,427 HelloServer.java
               7 File(s)              7,129 bytes
               3 Dir(s)  289,094,209,536 bytes free

D:\Code\Test>

```

ภาพที่ 10 แสดงรายชื่อแฟ้มส่วนของการอิมพลิเมนต์หลังการคอมไพล์

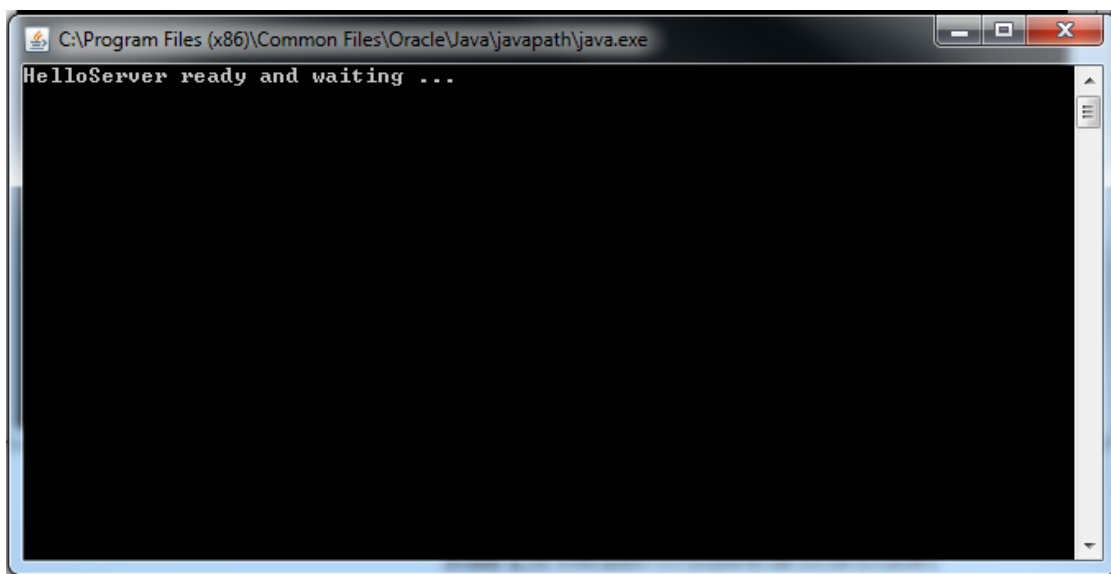
หลังจากนั้นใช้คำสั่ง `start orbd -ORBInitialPort 1500` จะปรากฏหน้าต่างของการทำงานของ orb ขึ้นมา เป็นหน้าจอสีดำขึ้นมาเท่านั้น(ไม่ต้องปิดหน้าต่าง)



8. สั่งให้เซิร์ฟเวอร์ทำงานโดยใช้คำสั่งไปที่พอร์ตที่รัน orb ไว้ดังนี้

```
start java HelloServer -ORBInitialPort 1500 -ORBInitialHost localhost
```

จะได้หน้าต่างใหม่ดังภาพด้านล่าง



ภาพ 11 แสดงผลการรันเมื่อเซิร์ฟเวอร์ทำงานแล้ว

9. หลังจากนั้นรันส่วนของไคลเอนต์เพื่อทดสอบโปรแกรมดังนี้

```
java HelloClient -ORBInitialPort 1500 -ORBInitialHost localhost
```

จะได้ผลลัพธ์ดังภาพ

```
Administrator: C:\Windows\system32\cmd.exe
11/29/2018 10:19 AM          904 HelloClient.java
11/30/2018 03:26 PM          519 HelloImpl.class
11/29/2018 10:15 AM          516 HelloImpl.java
11/30/2018 03:26 PM       2,118 HelloServer.class
11/28/2018 02:05 PM       1,427 HelloServer.java
7 File(s)              7,129 bytes
3 Dir(s)      289,094,209,536 bytes free

D:\Code\Test>start orbd -ORBInitialPort 1500

D:\Code\Test>start java HelloServer -ORBInitialPort 1500 -ORBInitialHost localhost

D:\Code\Test>java HelloClient -ORBInitialPort 1500 -ORBInitialHost localhost
Obtained a handle on server object: IOR:00000000000000001749444c3a48656c6c6f417070
2f48656c6c6f3a312e300000000000010000000000000086000102000000000d3137322e31362e31
382e36380000ef8500000031afabcb000000002063bf47b0000000010000000000000001000000008
526f6f74504f4100000000080000000100000000140000000000002000000010000002000000000
00010001000000002050100010001002000010109000000010001010000000026000000020002

Hello world!!

D:\Code\Test>
```

ภาพที่ 12 แสดงผลการรันที่ส่งไปยังไคลเอนต์

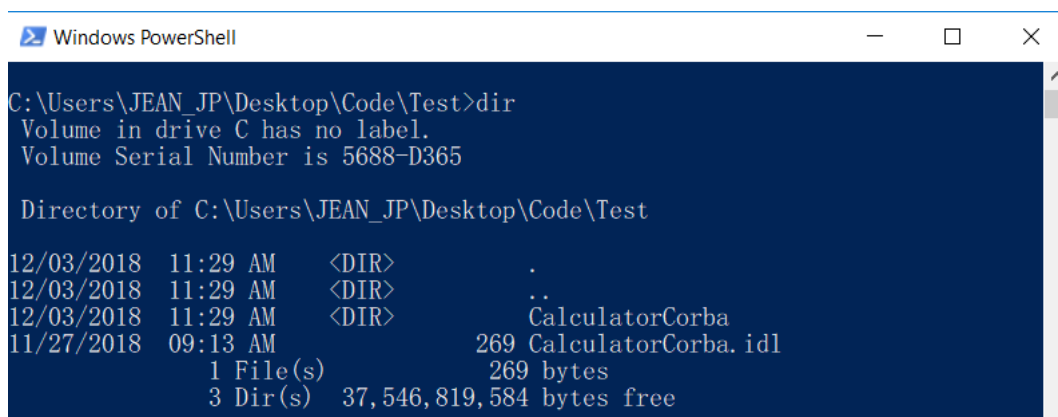
หากต้องการนำไปรันต่างเครื่องก็สามารถแยกส่วนไปรันได้ โดยส่วนของเซิร์ฟเวอร์ จะรันในเครื่องแม่ข่าย และ ส่วนของไคลเอนต์จะระบุชื่อของเครื่องแทน localhost ได้

ตัวอย่างที่ 2 จงเขียนโปรแกรมเครื่องคิดเลขอย่างง่ายด้วยคอร์บา โดยกำหนดให้มีเมธอดการบวก ลบ คูณ และหารเลขแบบ double

1. สร้าง idl ไฟล์ชื่อ CalculatorCorba.idl ดังนี้

```
module CalculatorCorba
{
    interface CalculatorEngine
    {
        double plus(in double val1, in double val2);
        double subtract(in double val1, in double val2);
        double multiply(in double val1, in double val2);
        double divide(in double val1, in double val2);
    };
}
```

2. คอมไพล์แฟ้มในข้อ 1 ด้วยคำสั่ง idlj -f all CalculatorCorba.idl จะได้แฟ้มดังนี้



```
Windows PowerShell
C:\Users\JEAN_JP\Desktop\Code\Test>dir
Volume in drive C has no label.
Volume Serial Number is 5688-D365

Directory of C:\Users\JEAN_JP\Desktop\Code\Test

12/03/2018  11:29 AM    <DIR>          .
12/03/2018  11:29 AM    <DIR>          ..
12/03/2018  11:29 AM    <DIR>          CalculatorCorba
11/27/2018  09:13 AM                269 CalculatorCorba.idl
               1 File(s)                269 bytes
               3 Dir(s)  37,546,819,584 bytes free
```

ภาพที่ 14 แสดงแพ็คเกจหลังจากการคอมไพล์


```

C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba>cd CalculatorCorba
C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba\CalculatorCorba>dir
Volume in drive C has no label.
Volume Serial Number is 5688-D365

Directory of C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba\CalculatorCorba

11/27/2018  03:25 PM    <DIR>          .
11/27/2018  03:25 PM    <DIR>          ..
11/27/2018  09:56 AM             247 CalculatorEngine.class
11/27/2018  09:13 AM             387 CalculatorEngine.java
11/27/2018  09:59 AM        2,623 CalculatorEngineHelper.class
11/27/2018  09:13 AM        2,711 CalculatorEngineHelper.java
11/27/2018  09:13 AM             933 CalculatorEngineHolder.java
11/27/2018  09:56 AM             227 CalculatorEngineOperations.class
11/27/2018  09:13 AM             508 CalculatorEngineOperations.java
11/27/2018  09:59 AM        2,662 CalculatorEnginePOA.class
11/27/2018  09:13 AM        3,381 CalculatorEnginePOA.java
11/27/2018  09:59 AM        3,632 _CalculatorEngineStub.class
11/27/2018  09:13 AM        4,978 _CalculatorEngineStub.java
                11 File(s)          22,289 bytes
                2 Dir(s)  37,545,865,216 bytes free

```

ภาพที่ 15 รายชื่อแฟ้มหลังคอมไพล์ภายในแพ็คเกจ CalculatorCorba

3. เขียนส่วนอิมพลีเมนต์ของเซิร์ฟเวอร์ โดยแบ่งเป็นสองแฟ้มดังนี้

3.1 แฟ้ม CalculatorImpl.java เพื่อเขียนส่วนการอิมพลีเมนต์ดังนี้

```

import CalculatorCorba.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import org.omg.PortableServer.POA;

import java.util.Properties;

class CalculatorImpl extends CalculatorEnginePOA {

    private ORB;

    public void setORB(ORB orb_value){

```

```

        orb = orb_value;
    }

    public double plus(double val1, double val2){

        return val1+val2;
    }

    public double subtract(double val1, double val2){

        return val1-val2;
    }

    public double multiply(double val1, double val2){

```

3.2 เขียนส่วนโปรแกรมหลัก ดังนี้

```

import CalculatorCorba.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import org.omg.PortableServer.POA;

import java.util.Properties;

public class CalculatorCorbaServer{

    public static void main(String args[]){

        try{

            //create and initialize the ORB

```

```
ORB = ORB.init(args, null);

//get reference to rootpoa & activate the POA Manager

POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

rootpoa.the_POAManager().activate();


//create servant and register it with the ORB

CalculatorImpl casio = new CalculatorImpl();

casio.setORB(orb);


//get object reference from the servant

org.omg.CORBA.Object ref = rootpoa.servant_to_reference(casio);

CalculatorEngine href = CalculatorEngineHelper.narrow(ref);


//get the root naming context

//NameService invokes the name service

org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");


//Use NamingContextExt which is part of the Interoperable

//Naming Service (INS) specification.

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
```

```
//binding the object Reference in Naming

String name = "casioCalculator";

NameComponent path[] = ncRef.to_name(name);

ncRef.rebind(path, href);

System.out.println("Easy Calculator by CORBA Server ready and waiting ...");


//waiting for invocations from clients

orb.run();

}

catch(Exception e) {

System.err.println("ERROR: " + e);

e.printStackTrace(System.out);

}

System.out.println("Calculator Server Exiting ...");

}

}
```

4. สร้างโปรแกรมอิมพลีเมนต์ส่วนของไคลเอนต์ดังนี้

```
import CalculatorCorba.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

public class CalculatorCorbaClient

static CalculatorEngine casioImpl;

    public static void main(String args[])

    {

        try {

            //create and initialize the ORB

            ORB orb = ORB.init(args, null);

            //get the root naming context

            org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");

{

            //Use NamingContextExt instead of NamingContext. This is part of the Interoperable
            naming Service.

            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

            //resolve the object refernece in Naming

            String name = "casioCalculator";
```

```
casiImpl = CalculatorEngineHelper.narrow(ncRef.resolve_str(name));

System.out.println("****Testing casio Calculator by CORBA with the real number
567.89 and 123.45****");

System.out.println("567.89 + 123.45 = " + casiImpl.plus(567.89, 123.45));

System.out.println("567.89 - 123.45 = " + casiImpl.subtract(567.89, 123.45));

System.out.println("567.89 * 123.45 = " + casiImpl.multiply(567.89, 123.45));

System.out.println("567.89 / 123.45 = " + casiImpl.divide(567.89, 123.45));

} catch(Exception e){

System.out.println("ERROR : " + e);

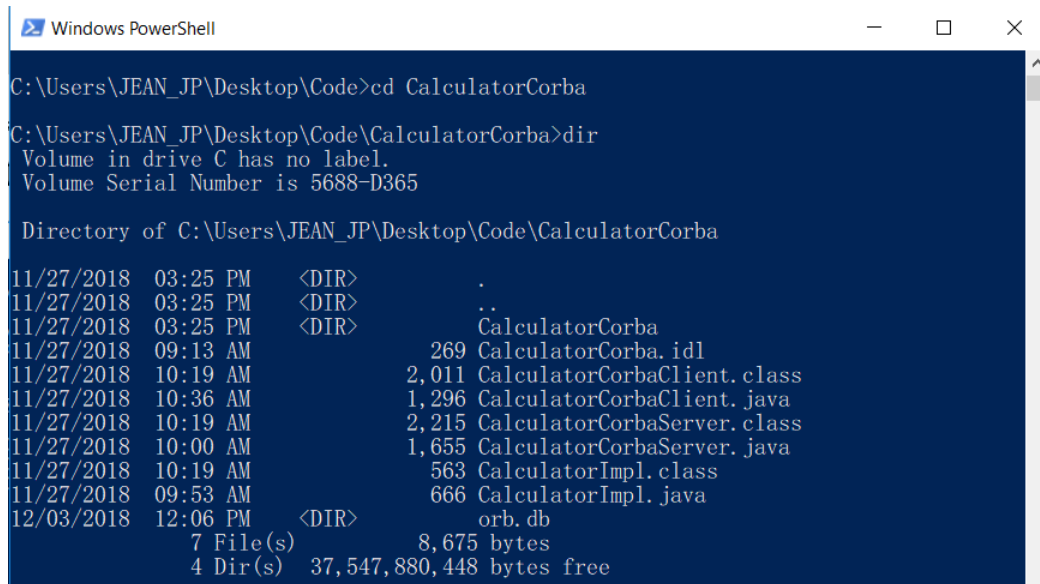
    e.printStackTrace(System.out);

}

}

}
```

5. บันทึกแฟ้มจะต้องบันทึกเอาไว้บนไดรฟ์คอมพิวเตอร์ CalculatorCorba
6. คอมไพล์แฟ้มทั้งสองแฟ้ม โดยใช้คำสั่ง `javac *.java` เมื่อคอมไพล์เสร็จแล้วจะได้แฟ้มดังนี้



```

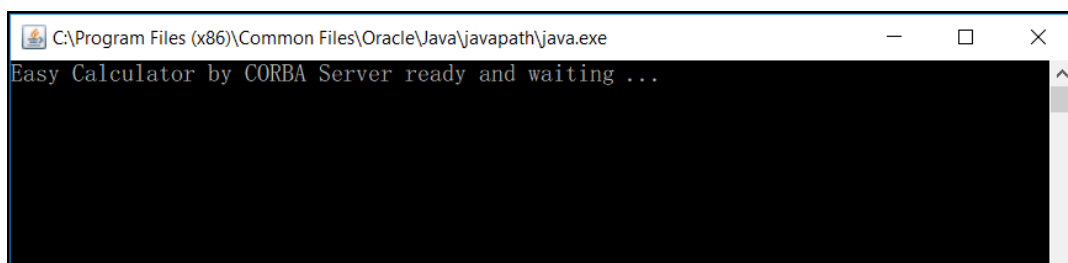
C:\Users\JEAN_JP\Desktop\Code>cd CalculatorCorba
C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba>dir
Volume in drive C has no label.
Volume Serial Number is 5688-D365

Directory of C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba

11/27/2018  03:25 PM    <DIR>          .
11/27/2018  03:25 PM    <DIR>          ..
11/27/2018  03:25 PM    <DIR>          CalculatorCorba
11/27/2018  09:13 AM             269 CalculatorCorba.idl
11/27/2018  10:19 AM           2,011 CalculatorCorbaClient.class
11/27/2018  10:36 AM           1,296 CalculatorCorbaClient.java
11/27/2018  10:19 AM           2,215 CalculatorCorbaServer.class
11/27/2018  10:00 AM           1,655 CalculatorCorbaServer.java
11/27/2018  10:19 AM             563 CalculatorImpl.class
11/27/2018  09:53 AM             666 CalculatorImpl.java
12/03/2018  12:06 PM    <DIR>          orb.db
              7 File(s)            8,675 bytes
              4 Dir(s)       37,547,880,448 bytes free
  
```

ภาพที่ 19 แสดงรายชื่อแฟ้มของการอิมพลีเมนต์ CalculatorCorba

7. สั่งให้ ORB ทำงานโดยใช้คำสั่ง `start orbd - ORBInitialPort 1500` โดยการเปิดพอร์ตสำหรับติดต่อใช้งานที่พอร์ต 1500
8. สั่งให้เซิร์ฟเวอร์ทำงานดังนี้ `start java CalculatorCorba -ORBInitialPort 1500 -ORBInitialHost localhost` จะได้ผลดังนี้



```

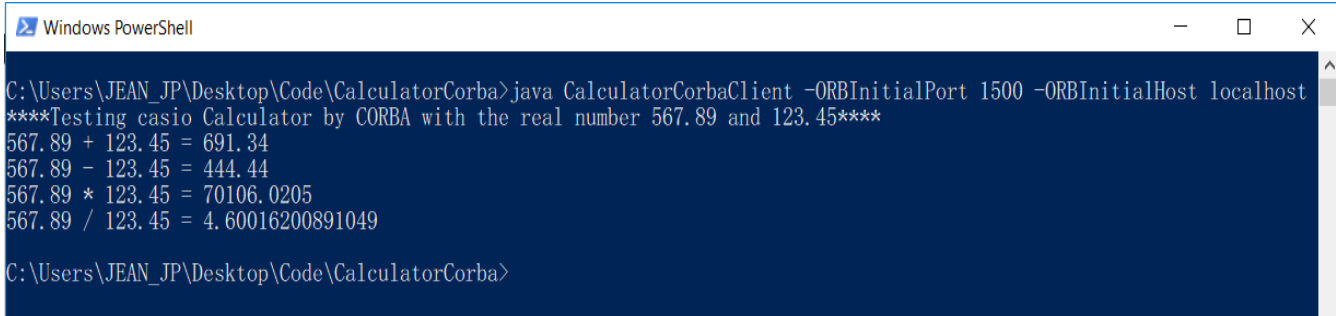
C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe
Easy Calculator by CORBA Server ready and waiting ...
  
```

ภาพที่ 20 แสดงหน้าต่างการรันของเซิร์ฟเวอร์เครื่องคิดเลข

9. สั่งให้ส่วนไคลเอนต์ติดต่อเพื่อทำงานดังนี้

```
java CalculatorCorbaClient -ORBInitialPort 1500 -ORBInitialHost localhost
```

ผลการทำงานของโปรแกรมจะแสดงดังนี้



```
Windows PowerShell
C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba>java CalculatorCorbaClient -ORBInitialPort 1500 -ORBInitialHost localhost
****Testing casio Calculator by CORBA with the real number 567.89 and 123.45****
567.89 + 123.45 = 691.34
567.89 - 123.45 = 444.44
567.89 * 123.45 = 70106.0205
567.89 / 123.45 = 4.60016200891049
C:\Users\JEAN_JP\Desktop\Code\CalculatorCorba>
```

ภาพ 21 แสดงผลการรันเมื่อมีการคำนวณที่ส่งไปยังไคลเอนต์

ตัวอย่างที่ 3 จงเขียนโปรแกรมจำลองฐานข้อมูลเอทีเอ็ม โดยสมมติว่าในฐานข้อมูลมีข้อมูลลูกค้าดังนี้

```
String Name[5] = {"Jakkarin", "Sasipa", "Kanyarat", "Piranan"};
```

```
Double Salary[5] = {70000, 50000, 30000, 40000};
```

1. สร้าง idl ไฟล์ชื่อ easyDBCorba.idl ดังนี้

```
module easyDBCorba
{
    interface dbEngine
    {
        string getName(in long no);

        double getSalary(in long no);

        void setSalary(in long no, in double newSalary);
    };
}
```

2. คอมไพล์แฟ้มในข้อ 1 ด้วยคำสั่ง idlj -f all easyDBCorba.idl จะได้แฟ้มดังนี้

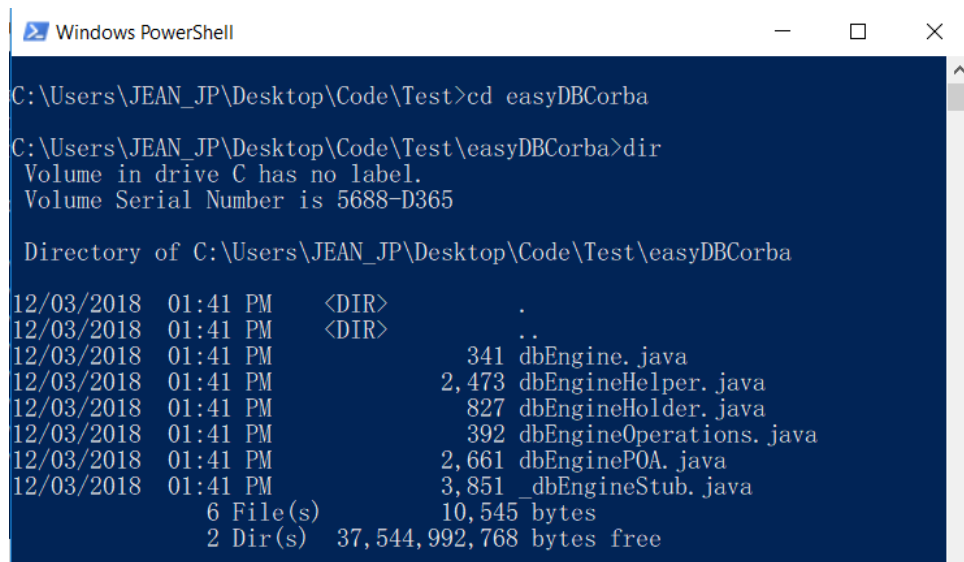


```
Windows PowerShell
C:\Users\JEAN_JP\Desktop\Code\Test>dir
Volume in drive C has no label.
Volume Serial Number is 5688-D365

Directory of C:\Users\JEAN_JP\Desktop\Code\Test

12/03/2018  01:41 PM    <DIR>          .
12/03/2018  01:41 PM    <DIR>          ..
12/03/2018  01:41 PM    <DIR>          easyDBCorba
11/27/2018  11:28 AM                173 easyDBCorba.idl
               1 File(s)                173 bytes
               3 Dir(s)  37,538,283,520 bytes free
```

ภาพที่ 23 แสดงแพคเกจหลังจากการคอมไพล์



```

Windows PowerShell

C:\Users\JEAN_JP\Desktop\Code\Test>cd easyDBCorba

C:\Users\JEAN_JP\Desktop\Code\Test\easyDBCorba>dir
Volume in drive C has no label.
Volume Serial Number is 5688-D365

Directory of C:\Users\JEAN_JP\Desktop\Code\Test\easyDBCorba

12/03/2018  01:41 PM    <DIR>          .
12/03/2018  01:41 PM    <DIR>          ..
12/03/2018  01:41 PM                341 dbEngine.java
12/03/2018  01:41 PM            2,473 dbEngineHelper.java
12/03/2018  01:41 PM            827 dbEngineHolder.java
12/03/2018  01:41 PM            392 dbEngineOperations.java
12/03/2018  01:41 PM            2,661 dbEnginePOA.java
12/03/2018  01:41 PM            3,851 dbEngineStub.java
               6 File(s)            10,545 bytes
               2 Dir(s)  37,544,992,768 bytes free

```

ภาพที่ 24 รายชื่อแฟ้มหลังคอมไพล์ภายในแพคเกจ easyDBCorba

3. เขียนส่วนอิมพลิเมนต์ของเซิร์ฟเวอร์ โดยแบ่งเป็นสองแฟ้มดังนี้

3.1 แฟ้ม easyDBImpl.java เพื่อเขียนส่วนการอิมพลิเมนต์ดังนี้

```

import easyDBCorba.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import org.omg.PortableServer.POA;

import java.util.Properties;

class easyDBImpl extends dbEnginePOA {

    private ORB orb;

    String Name[] = {"Jakkarin", "Sasipa", "Kanyarat", "Piranan"};

    double Salary[] = {70000, 50000, 30000, 40000};

    public void setORB(ORB orb_value){

        orb = orb_value;
    }
}

```

```
}

public String getName(int no)
{
    return Name[no-1];
}

public double getSalary(int no)
{
    return Salary[no-1];
}

public void setSalary(int no, double newSalary)
{
    Salary[no-1] = newSalary;
}

}
```

3.2 เขียนส่วนโปรแกรมหลัก ดังนี้

```
import easyDBCorba.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import org.omg.PortableServer.*;

import org.omg.PortableServer.POA;

import java.util.Properties;

public class easyDBCorbaServer{

    public static void main(String args[]){

        try{

            //create and initialize the ORB

            ORB orb = ORB.init(args, null);

            //get reference to rootpoa & activate the POAManager

            POA rootpoa = POAHelper.narrow(orb.resolve_initial_references("RootPOA"));

            rootpoa.the_POAManager().activate();

            //create servant and register it with the ORB

            easyDBImpl myDB = new easyDBImpl();

            myDB.setORB(orb);
```

```
//get object reference from the servant

org.omg.CORBA.Object ref = rootpoa.servant_to_reference(myDB);

dbEngine href = dbEngineHelper.narrow(ref);


//get the root naming context

//NameService invokes the name service

org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");

//Use NamingContextExt which is part of the Interoperable

//Naming Service(INS) specification.

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

//bind the Object Reference in Naming

String name = "easyDB";

NameComponent path[] = ncRef.to_name( name );

ncRef.rebind(path, href);


System.out.println("Easy Database By CORBA Server ready and waiting ...");

//wait for invocations from clients

orb.run();

}

catch(Exception e){
```

```

        System.out.println("ERROR: " + e);

        e.printStackTrace(System.out);
    }

    System.out.println("Calculator Server Exiting ...");
}
}

```

4. สร้างโปรแกรมอิมพลีเมนต์ส่วนของไคลเอนต์ดังนี้

```

import easyDBCORBA.*;

import org.omg.CosNaming.*;

import org.omg.CosNaming.NamingContextPackage.*;

import org.omg.CORBA.*;

import java.io.*;

public class easyDBCORBAClient
{
    static dbEngine myDBImpl;

    public static void main(String args[])
    {
        InputStreamReader reader = new InputStreamReader(System.in);
    }
}

```

```
BufferedReader stdin = new BufferedReader(reader);

try{

//create and initialize the ORB

    ORB orb = ORB.init(args, null);

//get the root naming context

org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");

//Use NamingContextExt instead of NamingContext. This is part of the
//Interoperable naming Service.

NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);

//resolve the Object Reference in Naming

String name = "easyDB";

myDBImpl = dbEngineHelper.narrow(ncRef.resolve_str(name));

//***** main responding *****

String choice = "0", number = "", salary = "";

while(Integer.parseInt(choice)!=9){

    System.out.println("*****Please choose Menu 1=show, 2=set Salary,
    9=Exit*****");

    System.out.println("Choice:");

    choice = stdin.readLine();

    switch(Integer.parseInt(choice)){

        case 1:{
```

```
        System.out.println("Input Number of Person: ");

        number = stdin.readLine();

        System.out.println("Show Data Number: " +
            number + " Name: " +
            myDBImpl.getName(Integer.parseInt(number)) + "
            Salary: " +
            myDBImpl.getSalary(Integer.parseInt(number)));

        break;
    }

    case 2:{

        System.out.println("Input Number of Person: ");

        number = stdin.readLine();

        System.out.println("Input Salary: ");

        salary = stdin.readLine();

        myDBImpl.setSalary(Integer.parseInt(number),
            Double.parseDouble(salary));

        System.out.println("Set new salary OK.");

        break;
    }

    default:{System.out.println("Input is not valid");}

}

//end switch
```



```

        System.out.println("5 6 7.89 * 123.45 = " +
        casioImpl.multiply(567.89, 123.45));

        System.out.println("5 6 7.89 / 123.45 = " +
        casioImpl.divide(567.89, 123.45));

    } catch (Exception e){

        System.out.println("ERROR : " + e);

        e.printStackTrace(System.out);

    }

}

```

5. บันทึกแฟ้มจะต้องบันทึกเอาไว้บนไดรฟ์ easyDBCorba
6. คอมไพล์แฟ้มทั้งสองแฟ้ม โดยใช้คำสั่ง `javac *.java` เมื่อคอมไพล์เสร็จแล้วจะได้แฟ้มดังนี้

```

C:\Users\JEAN_JP\Desktop\Code>cd easyDBCorba

C:\Users\JEAN_JP\Desktop\Code\easyDBCorba>dir
Volume in drive C has no label.
Volume Serial Number is 5688-D365

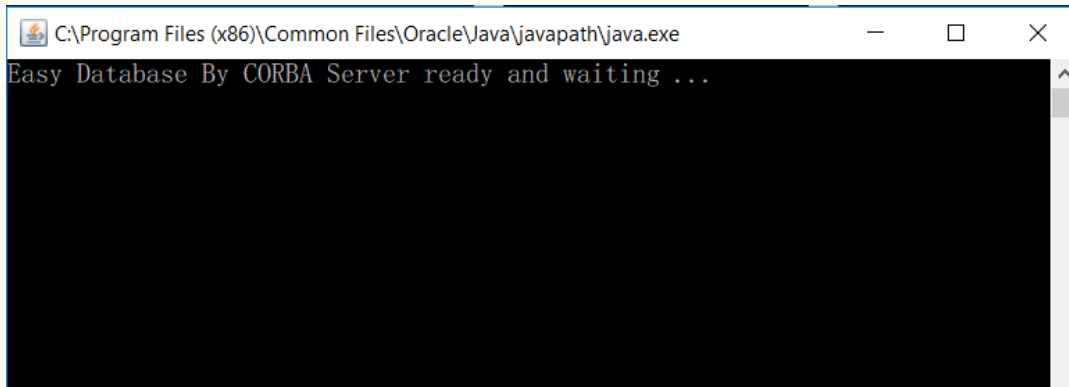
Directory of C:\Users\JEAN_JP\Desktop\Code\easyDBCorba

12/03/2018  03:49 PM    <DIR>          .
12/03/2018  03:49 PM    <DIR>          ..
11/29/2018  04:35 PM    <DIR>          easyDBCorba
11/27/2018  11:28 AM             173 easyDBCorba.idl
11/28/2018  09:39 AM          2,629 easyDBCorbaClient.class
11/27/2018  04:58 PM          2,074 easyDBCorbaClient.java
11/28/2018  09:39 AM          2,152 easyDBCorbaServer.class
11/28/2018  09:39 AM          1,607 easyDBCorbaServer.java
11/28/2018  09:39 AM           805 easyDBImpl.class
11/27/2018  04:45 PM           678 easyDBImpl.java
11/29/2018  04:35 PM    <DIR>          orb.db
              7 File(s)          10,118 bytes
              4 Dir(s)      37,526,544,384 bytes free

```

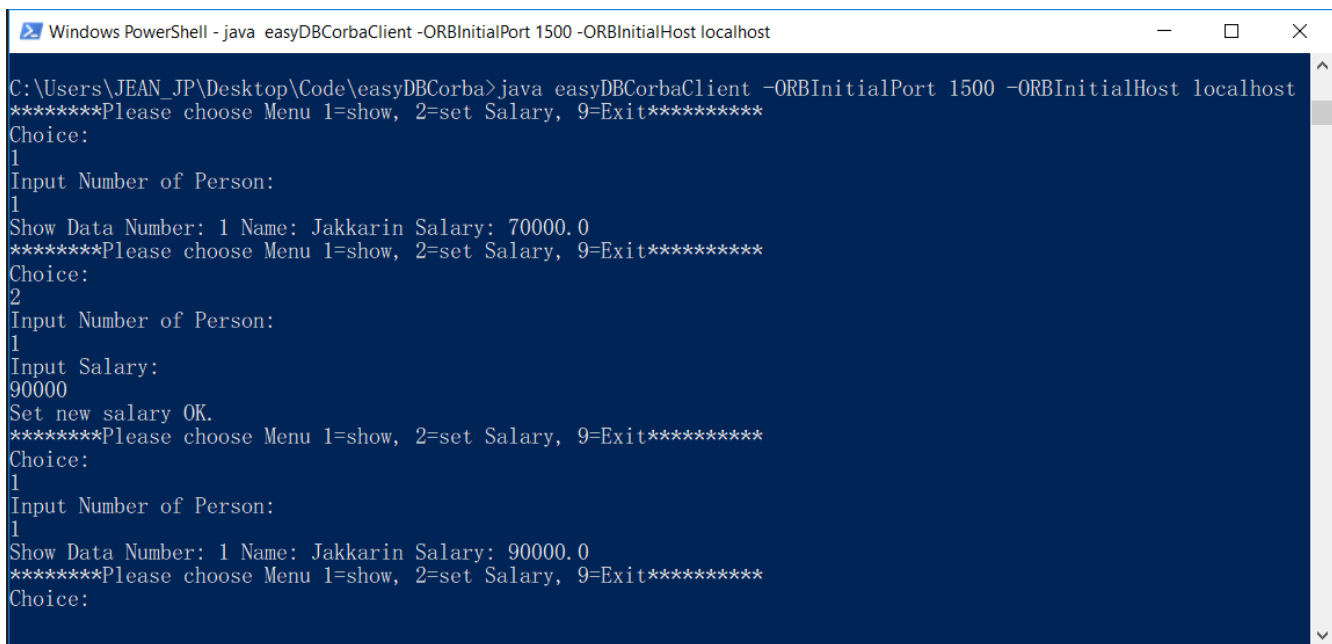
ภาพที่ 28 แสดงรายชื่อแฟ้มของการอิมพลีเมนต์ easyDBCorba

7. สั่งให้ ORB ทำงานโดยใช้คำสั่ง `start orbd -ORBInitialPort 1500` โดยการเปิดพอร์ตสำหรับติดต่อใช้งานที่พอร์ต 1500
8. สั่งให้เซิร์ฟเวอร์ทำงานดังนี้ `start java easyDBCorbaServer -ORBInitialPort 1500 -ORBInitialHost localhost` จะได้ผลตามภาพด้านล่าง



ภาพที่ 29 แสดงหน้าต่างการรันของเซิร์ฟเวอร์

9. สั่งให้ส่วนไคลเอนต์ติดต่อเพื่อทำงานดังนี้
`java easyDBCorbaClient -ORBInitialPort 1500 -ORBInitialHost localhost`
 ผลการทำงานของโปรแกรมจะแสดงดังนี้



ภาพ 30 แสดงผลการรันตัวอย่างที่ 3

สรุป

เนื่องจากการเขียนโปรแกรมในการสร้างเซิร์ฟเวอร์ให้บริการด้วยการส่งเป็นไบนารีมีปัญหาหลายประการดังนี้ จึงมีแนวคิดสำหรับสร้างมิดเดิลแวร์เพื่อใช้เป็นซอฟต์แวร์ตัวกลางเพื่อขจัดปัญหาต่าง ๆ ของการจัดการข้อมูลเครือข่ายขึ้น อีกทั้งสามารถสร้างโปรแกรมแบบกระจายได้ และคอร์บาเป็นมาตรฐานของการส่งข้อมูลที่กำหนดโดยโอเอ็มจี ที่สามารถนำไปสร้างเพื่อใช้งานจริงได้ในหลากหลายภาษาคอมพิวเตอร์ โดยผ่านภาษาไอดีแอล และคอมไพล์ด้วยภาษาที่ต้องการเขียนโปรแกรม การใช้งานจะผ่านอ็อบเจ็กต์ซึ่งเป็นส่วนของการสร้างการทำงานให้สามารถใช้งานได้โดยไม่เกิดข้อจำกัดขึ้น การทำงานอยู่ในระดับของแอปพลิเคชันเลเยอร์ของโอเอสไอโมเดล ดังนั้นจึงมีข้อดีมาก และการส่งทำงานสามารถกำหนดได้ในระดับการรันแอปพลิเคชัน

ตัวอย่างที่ 4 ทดสอบการทำงานบนเครื่องเซิร์ฟเวอร์จริง

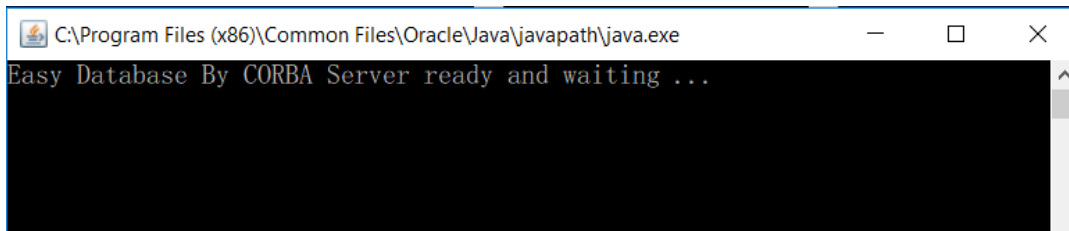
จากตัวอย่างที่ 3 ให้ทำตามขั้นตอนที่ 1-3, 5-6 บนเครื่องเซิร์ฟเวอร์ และทำตามขั้นตอนที่ 1,2,4-6 บนเครื่องไคลเอนต์ หลังจากนั้นให้ทำตามขั้นตอนดังนี้

7. (ฝั่ง Server) สั่งให้ ORB ทำงานโดยใช้คำสั่ง `start orbd -ORBInitialPort 1500` โดยการเปิดพอร์ตสำหรับติดต่อใช้งานที่พอร์ต 1500

8. (ฝั่ง Server) สั่งให้เซิร์ฟเวอร์ทำงานดังนี้

`start java easyDBCorbaServer -ORBInitialPort 1500 -ORBInitialHost 172.16.19.134`

จะได้ผลตามภาพด้านล่าง

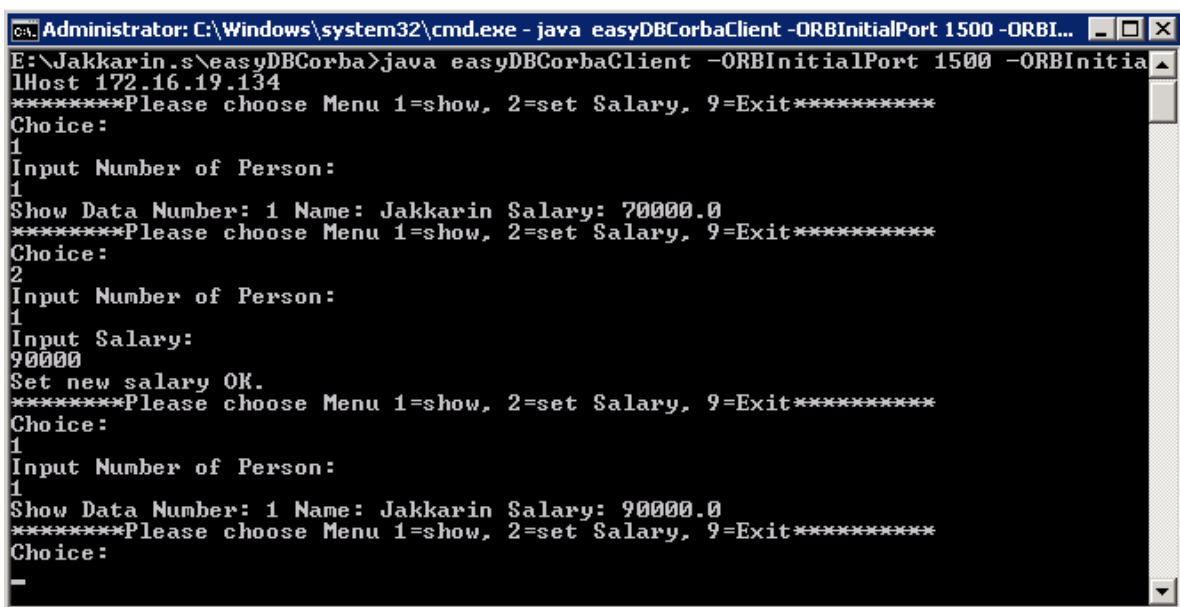


ภาพที่ 31 แสดงหน้าต่างการรันของเซิร์ฟเวอร์

9. (ฝั่งไคลเอนต์) สั่งให้ส่วนไคลเอนต์ติดต่อเพื่อทำงานดังนี้

`java easyDBCorbaClient -ORBInitialPort 1500 -ORBInitialHost 172.16.19.134`

ผลการทำงานของโปรแกรมจะแสดงดังนี้



ภาพ 32 แสดงผลการรันตัวอย่างที่ 4