Part 7

Java Programming Language
Mr.Rungrote Phonkam
rungrote@it.kmitl.ac.th

Contents

1. Method Member #2
2. Constructor
3. Finalize
4. Mutation and Assessor Method
5. Overloading
6. Static Scope
7. Inner Class

1. Method Member #2

- สำหรับกำหนดพฤติกรรมการทำงานของออปเจ็ค
- รับข้อมูลผ่านทางพารามิเตอร์
  - Copy Value Parameter
  - Reference Parameter
- รับข้อมูลมาประมวลผล โดยอาจเปลี่ยนแปลงค่า Data ภายใน หรือผลทางด้านอื่น เช่น ออปเจ็คที่เกี่ยวกับการแสดงภาพ อาจมีเมธรอดเพื่อการปรับสีของภาพให้สว่างขึ้น
- ถ้าเมธรอดต้องการส่งผลลัทธ์คืน ส่งกลับโดยการระบุด้วยคำสั่ง return

1. Method Member #2

รูปแบบการกำหนด

```
Access_Level  final  static Return_Type  Method_Name ( Argument_List )
{
        Statements
        return Expression
}
```

1. Method Member #2

ตัวอย่าง 1

```
void  methodA ( )
{ … }
```

ตัวอย่าง 2

```
void  methodB (int a)
{ … }
```

1. Method Member #2

ตัวอย่าง 3

```
void  methodC (char c, Double obj)
{ … }
```

ตัวอย่าง 4

```
int  methodD (Float f)
{ …
   return  15;
}
```

1. Method Member #2

ตัวอย่าง 5

```
Double  methodE (Double obj)
{ …
   return new Double(12.45);
}
```

ตัวอย่าง 6

```
String  methodG ()
{ … ..
   return new Double(12.45);
  ……
}
```

2. Constructors

คอนสตรักเตอร์

- ทำงานอัตโนมัติ เมื่อมีการสร้างอินสแตนซ์ขึ้น
- เพื่อใช้กำหนดค่าเริ่มต้นให้กับ Data ภายในอินสแตนซ์
- ชื่อคอนสตรักเตอร์ เป็นชื่อเดียวกับชื่อคลาสเสมอ
- คลาสหนึ่งคลาส สามารถกำหนดคอนสตรักเตอร์ได้หลายรูปแบบ
- ในกรณีมีหลายคอนสตรักเตอร์ การสร้างอินสแตนซ์ต้องเลือกใช้คอนสตรักเตอร์รูปแบบใด รูปแบบหนึ่ง
- คอนสตรักเตอร์ ไม่มีการกำหนดการคืนค่า เหมือนเมธรอดทั่วไป

2. Constructors

```
class Pen
{   private String color;
      Pen(String c)       {          color = c;          }
      public void printPenColor( )
      {          System.out.println("Pen is " + color);          }
}
class MyPen
{   public static void main(String args[])
      {   Pen Pen1 = new Pen("RED");
          Pen Pen2 = new Pen("BLUE");
          Pen1.printPenColor( );  Pen2.printPenColor( );
      }
}
```

2. Constructors

```
java MyPen
Pen is RED
pen is BLUE
```

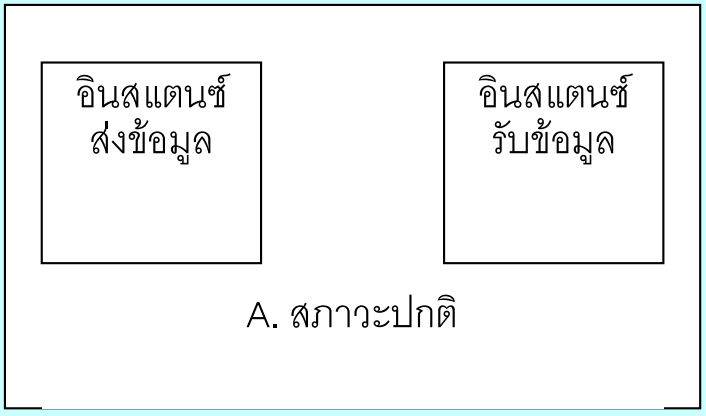
3. Finalize

ไฟนัลไลซ์

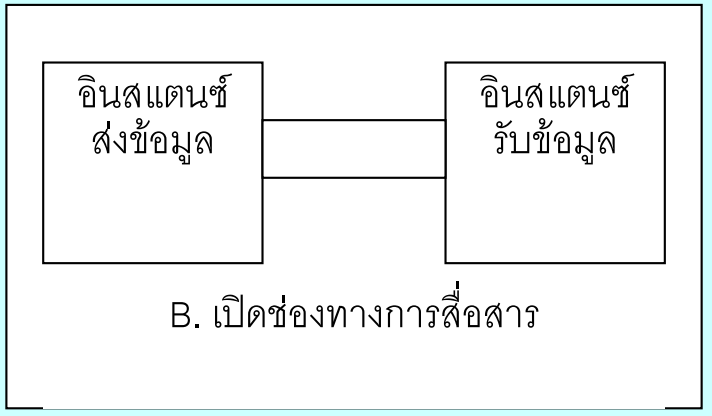
- ทำหน้าที่ตรงกันข้ามกับคอนสตรักเตอร์
- ใช้สำหรับลบ หรือทำลายค่าบางค่าอย่างอัตโนมัติ
- การทำงานเกิดขึ้นเมื่ออินสแตนซ์ ถูกทำลายลง จากกลไก Garbage Collector
- ในภาษาอื่น เช่น C++ เมื่อต้องการใช้งานหลักการเหมือนไฟนัลไลซ์ คือการสร้างเมธรอดจำพวก Deconstructor ซึ่งจะทำงานเมื่อมีการใช้คำสั่ง free กับอินสแตนซ์

3. Finalize

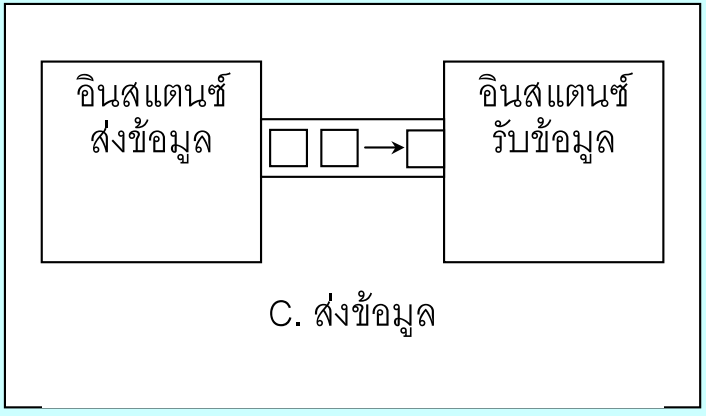
ภาพจำลองการดำเนินงานที่ไม่เรียบร้อย



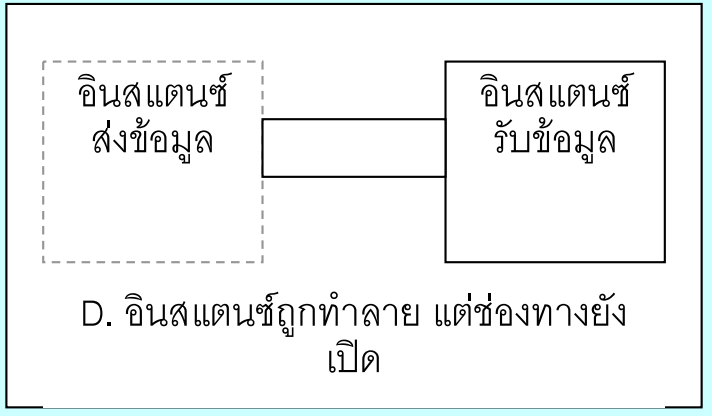
A. สภาวะปกติ



B. เปิดช่องทางการสื่อสาร



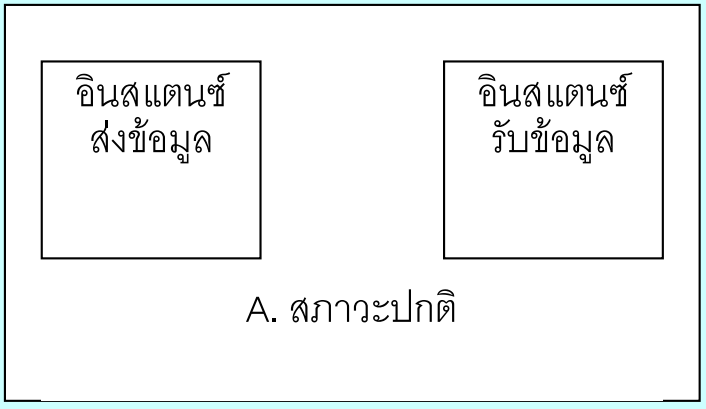
C. ส่งข้อมูล



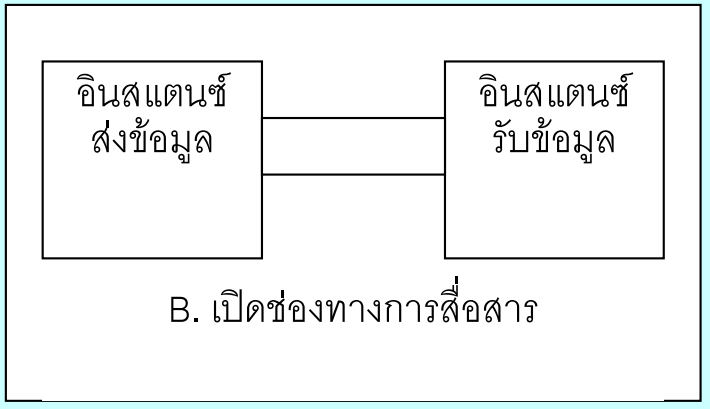
D. อินสแตนซ์ถูกทำลาย แต่ช่องทางยังเปิด

3. Finalize

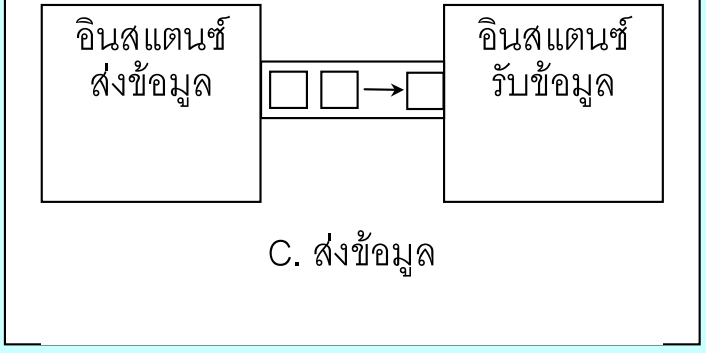
ภาพจำลองการดำเนินงานที่เรียบร้อย (มีการกำจัดส่วนที่เกี่ยวข้อง)



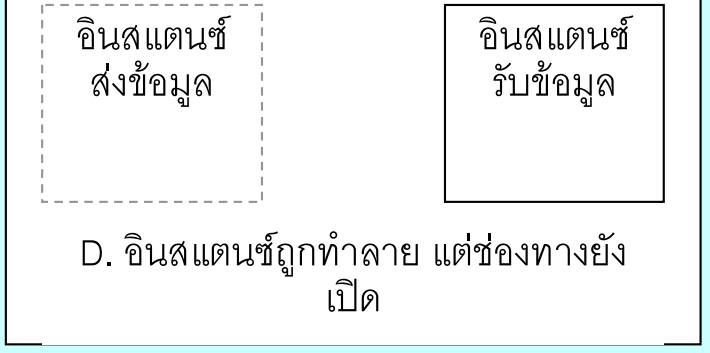
A. สภาวะปกติ



B. เปิดช่องทางการสื่อสาร



C. ส่งข้อมูล



D. อินสแตนซ์ถูกทำลาย แต่ช่องทางยังเปิด

3. Finalize

```
class Connection
{   Connection( )
        {           System.out.print(" Connecting\n");   }
        void SendData( )
        {           System.out.print(" Sending\n");      }
        protected void finalize( ) throws Throwable
        {           System.out.print(" Closing\n");                  }
}
class MyConnection
{   public static void main(String args[])
        {           Connection conn = new Connection( );
                    conn.SendData( );
                    System.runFinalizersOnExit(true);
        }
}
```

3. Finalize

```
javac –deprecation FileName.java
java MyConnection
```

4. Mutation and Assessor

เมธรอดตั้งค่า (Mutation)

- เมธรอดสำหรับการกระทำในการ เพื่อเปลี่ยนแปลงค่าของ Data ภายในคลาส
- เนื่องจากไม่ต้องการให้มีการเปลี่ยนแปลงค่า Data โดยตรง
- นำหน้าด้วย set เช่น setData( )

เมธรอดอ่านค่า (Assessor)

- เมธรอดสำหรับการกระทำในการ เพื่อดึงหรืออ่านค่าของ Data ภายในคลาส
- เนื่องจากไม่ต้องการให้มีการอ่านค่า Data โดยตรง
- นำหน้าด้วย get เช่น getData( )

4. Mutation and Assessor

```
class Baby
{   private String name;
      private char gender;
      void setName(String n)    {       name = n;       }
      String getName ( )        {       return name;    }
      void setGender(char g)    {       gender = g;     }
      char getGender ( )        {       return gender;  }
}
class MyBaby
{   public static void main(String args[])
      {         Baby Kid = new Baby( );
                Kid.setName(args[0]);
                Kid.setGender(args[1].charAt(0));
                if (Kid.getGender( )=='M')
                        System.out.print("My Son is ");
                else System.out.print("My Daughter is ");
                System.out.print(Kid.getName( ) );   }
}
```

4. Mutation and Assessor

```
java MyBaby Somchai Male
My Son is Somchai
java MyBaby Somsrt Female
My Daughter is Somsri
```

5. Overloading

โอเวอร์โหลดดิ่ง

- คือกฎเกณฑ์พิเศษในทางการเขียนโปรแกรมเชิงวัตถุ
- สามารถกำหนดให้มีเมธรอดชื่อเหมือนกันได้ แต่ต้องมีพารามิเตอร์ต่างกัน
- ความแตกต่างทางด้านพารามิเตอร์หมายถึง
  - ต่างชนิด เช่น methodA(int a) กับ methodB(char ch)
  - ต่างจำนวน เช่น methodA(int a) กับ methodB(int a, int b)
- สามารถนำไปใช้กับเมธรอดทั่วไป หรือคอนสตรักเตอร์ได้

5. Overloading

```
class Flight1
{    String captain_name; int qty_hostess, qty_seat;
     void FlightTeam(String n, int h, int s)
     {       captain_name = n; qty_hostess = h; qty_seat = s;       }
     void FlightTeam(String n)          {          captain_name = n;          }
     void FlightTeam(int s, int h, String n)
     {       captain_name = n; qty_hostess = h; qty_seat = s;       }
     void printInfo( )
     {       System.out.println("Captain Name  \t\t" + captain_name);
             System.out.println("Amount of Seat \t\t" + qty_seat);
             System.out.println("Amount of Air-Hostess\t" + qty_hostess);
     }
}
```

5. Overloading

```
class AirLine
{    public static void main(String args[])
     {       Flight1 f1 = new Flight1();
             Flight1 f2 = new Flight1();
             Flight1 f3 = new Flight1();
             f1.FlightTeam("Peter Anderson", 5, 50);
             f2.FlightTeam("Sompong N. Lumpang");
             f3.FlightTeam(120, 10, "Alen Tangee");
             f1.printInfo(); f2.printInfo(); f3.printInfo();
     }
}
```

5. Overloading

```
java AirLine
Captain Name          Peter Anderson
Amount of Seat        50
Amount of Air-Hostess 5
Captain Name          Sompong N. Lumpang
```

5. Overloading

```
class Flight2
{    String captain_name;
     int qty_hostess, qty_seat;
     Flight2(String n, int h, int s)
     {       captain_name = n; qty_hostess = h; qty_seat = s;       }
     Flight2(String n)
     {       captain_name = n;            }
     Flight2(int s, int h, String n)
     {       captain_name = n; qty_hostess = h; qty_seat = s;       }
     void printInfo( )
     {       System.out.println("Captain Name  \t\t" + captain_name);
             System.out.println("Amount of Seat \t\t" + qty_seat);
             System.out.println("Amount of Air-Hostess\t" + qty_hostess);
     }
}
```

5. Overloading

```
class AirLine
{    public static void main(String a[])
     {       Flight2 f1 = new Flight2("Peter Anderson", 5, 50);
             Flight2 f2 = new Flight2("Sompong N. Lumpang");
             Flight2 f3 = new Flight2(120, 10, "Alen Tangee");
             f1.printInfo(); f2.printInfo(); f3.printInfo();
     }
}
```

6. Static Block

สเตติกบล็อก (Static Block)

- สโคปหรือบล็อกที่นำหน้าด้วยคีย์เวิร์ด static
- ต้องถูกสร้างอยู่ภายในคลาส และนอกเมธรอด
- ถูกใช้สำหรับการกำหนดค่าเริ่มต้น (Initialization)
- ถูกทำงานครั้งแรกทันทีที่คลาสมีการโหลดจากไฟล์ Bytecodes เข้าสู่ JVM

รูปแบบ

```
class Class_Name {
        static { … }
}
```

6. Static Block

```
class Employee
{   String name;
    long salary;
    short employee_number;

    static int total_payroll;

    static
    {     System.out.println("Calculating payroll total ");
          salary = 3000;
    }
}
```

7. Static Class

สแตติกคลาส (Static Class)

- สำหรับกำหนดให้ทั้งคลาส ซึ่งหมายถึง Constructors, Method, Data Member ที่อยู่ภายใน ให้มีคุณสมบัติเป็นสเตติกทั้งหมด
- ถูกกำหนดขึ้นภายในคลาส นำหน้าด้วยคีย์เวิร์ด static
- ใช้งานร่วมกับคีย์เวิร์ด this ไม่ได้

รูปแบบ

```
class Outer_Class_Name {
        static class Inner_Class_Name{ … }
        …..
}
```

8. Inner Class

อินเนอร์คลาส (Inner Class)

- แบ่งเป็น 3 แบบ
  - Member Class
    คลาสที่ถูกกำหนดไว้ภายในคลาส (ระดับเดียวกับ Data และ Method)
  - Local Class
    คลาสที่สร้างไว้ภายใน Method
  - Anonymous Class
    คลาสซึ่งสร้างด้วยคำสั่งสร้างอินสแตนซ์

8. Inner Class

- ไม่สามารถอ้างถึงได้ภายนอกตัวคลาสที่เป็น Parent
- เหมาะกับการสร้างคลาสที่มีคำสั่งไม่มาก
- ถูกใช้งานในการใช้งานระบบ GUI เพื่อรองรับ Event Handler

8.1. Member Class

```
class top {
    int I=33;

    class myNested {
        int k=I;
        void foo() { }
    }

    void bar() {
        myNested mn1 = new myNested();
        myNested mn2 = new myNested();
        mn1.k = 564 * mn2.k;
    }
}
```

8.2. Local Class

```
//<applet code=f.class height=100 width=200> </applet>
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class f extends Applet {
    Button apple = new Button("press me");
    public void init() {
        class MyinnerBHClass implements java.awt.event.ActionListener
        {     int i=1;
              public void actionPerformed(ActionEvent e)
              {   System.out.println("button pressed "+ i++ +" times");     }
        }
        add(apple);
        apple.addActionListener( new MyinnerBHClass() );
    }
}
```

8.2. Local Class

หลังการคอมไพล์

- f.class
- f1MyinnerBHClass.class

8.3. Anonymous Class

```
//<applet code=f2.class height=100 width=200> </applet>
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class f2 extends Applet {
    Button apple = new Button("press me");
    public void init() {
        add(apple);
        apple.addActionListener(
          new ActionListener()
          {  public void actionPerformed(ActionEvent e)
            {  System.out.println( e.paramString() + " pressed");    }
          } // end anon class
        ); // end method call
    }
}
```

8.3. Anonymous Class

หลังการคอมไพล์

- f.class
- f$1.class