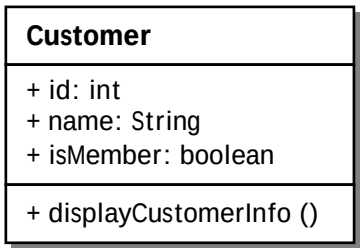
บทที่ 1
เริ่มต้นการเขียนโปรแกรม

Project Direct Clothing#1

กิจกรรมที่ 1: Customer.java



1. ประกาศคลาสใหม่ชื่อ **`Customer`**
2. ประกาศ Attribute เพื่อเก็บรายละเอียดของลูกค้า *รหัสลูกค้า, ชื่อลูกค้า* และ *สถานะของลูกค้าว่าเป็นสมาชิกหรือไม่* ดังต่อไปนี้

```
public int customerId = 1234;
public String name = “Robert”;
public boolean isMember = true;
```

3. ประกาศ Method ชื่อ **`displayCustomerInfo`** เพื่อแสดงผลลัพธ์ออกทางจอภาพ ดังตัวอย่างต่อไปนี้

```
===== CUSTOMER INFORMATION =====
 ID             : <value>
 Name           : <value>
 Status         : <value>
--------------------------------
```

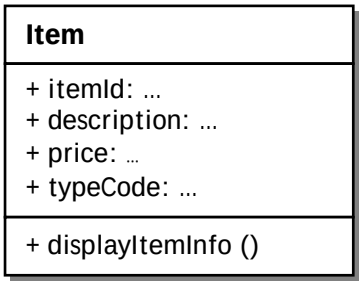
กิจกรรมที่ 2: TestDirect1.java

1. ประกาศคลาสชื่อ **`TestDirect1`**
2. ประกาศ Main Method ที่ภายในมีการเรียกใช้ Method สำหรับแสดงรายละเอียดของลูกค้าออกทางจอภาพ

บทที่ 2
Primitive Type

Project Direct Clothing #2

กิจกรรมที่ 1: Item.java



1. สร้างคลาสใหม่ชื่อ **Item**
2. ประกาศ Attribute ดังรายละเอียดต่อไปนี้ โดยพิจารณาประเภทของตัวแปร และกำหนดค่าเริ่มต้นตามความเหมาะสม
   - รหัสสินค้า
   - รายละเอียดของสินค้า
   - ราคาสินค้า
   - รหัสแสดงประเภทสินค้า ('B' แทนเครื่องดื่ม, 'C' แทนเครื่องสำอางค์, 'S' แทนขนมขบเคี้ยว)
3. ประกาศ Method ชื่อ **`displayItemInfo`** เพื่อใช้แสดงรายละเอียดของสินค้าออกทางจอภาพ โดยใช้ตัวอย่างของผลลัพธ์ดังต่อไปนี้

```
===== ITEM INFORMATION =====
 Item-Id     : <value>
 Description: <value>
 Price       : <value>
 Type        : <value>
----------------------------
```

กิจกรรมที่ 2: TesDirect2.java

1. ประกาศคลาสชื่อ **`TestDirect2`**
2. ประกาศ Main Method เพื่อเรียกใช้ Method สำหรับแสดงรายละเอียดของสินค้า ออกทางจอภาพ

บทที่ 3
Reference Type (Object Type)

Project Direct Clothing #3

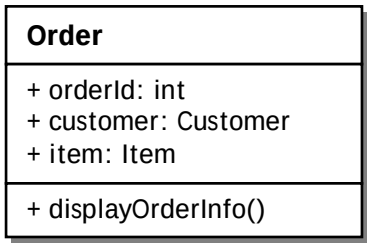
กิจกรรมที่ 1: Item.java

1. ในคลาส **Item** ให้ลบข้อมูลที่กำหนดเป็นค่าเริ่มต้นของแต่ละ Attribute

กิจกรรมที่ 2: Customer.java

1. ในคลาส **Customer** ให้ลบข้อมูลที่กำหนดเป็นค่าเริ่มต้นของแต่ละ Attribute

กิจกรรมที่ 3: Order.java



1. ประกาศคลาสใหม่ชื่อ **`Order`**
2. ประกาศ Attribute สำหรับจัดเก็บข้อมูลต่อไปนี้ รหัสใบสั่งซื้อ, ลูกค้าที่สั่ง, สินค้าที่สั่งซื้อ (ใช้ประเภทของ Attribute ตามที่กำหนดใน Class Diagram)
3. ประกาศ Method ชื่อ **`displayOrderInfo`** เพื่อใช้แสดงรายละเอียดของใบสั่งซื้อทางจอภาพ ดังตัวอย่างต่อไปนี้

```
===== ORDER INFORMATION =====
 Order Id   : <value>
 Customer   : <customer-name>
-----------------------------
 ITEM       : <item-desc> - <item-price>
=============================
```

หมายเหตุ: ข้อมูลของลูกค้า ให้แสดงเฉพาะ*ชื่อลูกค้า*เท่านั้น
ข้อมูลของสินค้าที่สั่ง ให้แสดงเฉพาะ*รายละเอียดสินค้า* และ*ราคาสินค้า*เท่านั้น

กิจกรรมที่ 4: TestDirect3.java

1. ประกาศคลาสใหม่ชื่อ **`TestDirect3`**
2. ภายใน Main Method ให้ปฏิบัติดังนี้

   2.1. สร้างตัวแปรที่อ้างอิงไปยัง Object ของคลาส Customer

   - กำหนดค่าให้กับ Attribute ของ Object: Customer ตามความเหมาะสม (อาจเลือกกำหนดเฉพาะ ชื่อลูกค้า: name)

   2.2. สร้างตัวแปรที่อ้างอิงไปยัง Object ของคลาส Item

   - กำหนดค่าให้กับ Attribute ของ Object: Item ตามความเหมาะสม (อาจเลือกกำหนดเฉพาะรายละเอียดของสินค้า: `description` และราคาสินค้า: `price`)

   2.3. สร้างตัวแปรที่อ้างอิงไปยัง Object ของคลาส Order

   - กำหนดค่าให้กับข้อมูลรหัสใบสั่งซื้อเป็น 10001.
   - กำหนดค่าให้กับข้อมูลลูกค้าในใบสั่งซื้อ โดยการอ้างอิงไปยังตัวแปร Customer ที่ได้สร้างไว้ในข้อ 2.1 ด้วยวิธีการ Assigning Reference
   - กำหนดค่าให้กับข้อมูลสินค้าในใบสั่งซื้อ โดยการอ้างอิงไปยังตัวแปร Item ที่ได้สร้างไว้ในข้อ 2.2 ด้วยวิธีการ Assigning Reference
   - เรียกใช้ Method: `displayOrderInfo` เพื่อแสดงรายละเอียดของใบสั่งซื้อ

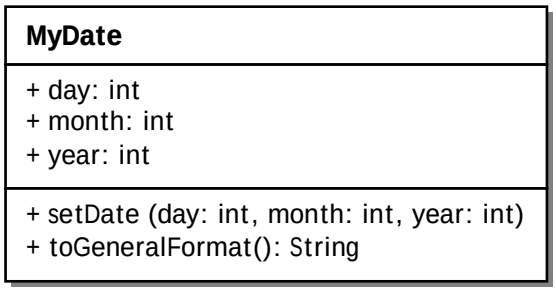
3. คอมไพล์และรันโปรแกรม

บทที่ 4
การสร้าง Methods

แบบฝึกหัดที่ 1: การสร้าง Method

รายละเอียด: เขียนโปรแกรมเพื่อสร้าง Method สำหรับกำหนดค่าวันที่และส่งข้อมูลกลับในรูปแบบวันที่ทั่วไป

กิจกรรมที่ 1: MyDate.java



1. ประกาศคลาสใหม่ชื่อ **`MyDate`**.
2. ประกาศ Attribute เพื่อใช้สำหรับเก็บข้อมูล วัน, เดือน และปี
3. ประกาศ Method ชื่อ **`setDate`** เพื่อใช้กำหนดข้อมูลให้กับแต่ละ Attribute โดยมีรายละเอียดดังนี้
   - มีการรับ Arguments 3 ตัวคือ ข้อมูลวัน, เดือน และปีที่ต้องการกำหนดให้กับ Object
   - กำหนดค่าให้กับ Attribute แต่ละตัว ด้วยข้อมูลที่ส่งผ่าน Arguments มา
4. ประกาศ Method ชื่อ **`toGeneralFormat`** เพื่อส่งข้อมูลในรูปของข้อความกลับในรูปแบบของวันที่ โดยมีรายละเอียด ดังนี้
   - ใช้ส่งข้อความกลับเป็นข้อมูลใน Attribute แต่ละตัวในรูปแบบของวันที่ เช่น *3/8/2004* (การใช้เครื่องหมาย + ระหว่างตัวเลขและข้อความ ผลลัพธ์ที่ได้จะถูกแปลงเป็นข้อความให้อัตโนมัติ)

กิจกรรมที่ 2: TestMyDate.java

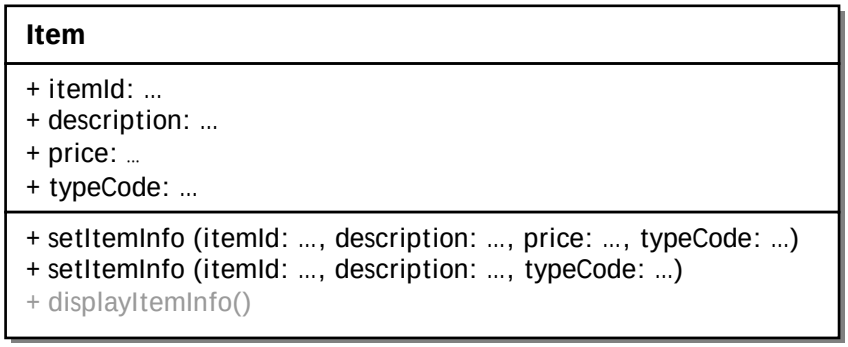
1. ประกาศคลาสใหม่ชื่อ **`TestMyDate`**
2. ภายใน Main Method ให้ปฏิบัติดังนี้
   2.1. สร้างตัวแปรที่อ้างอิงไปยัง Object ของคลาส `MyDate`
   2.2. เรียกใช้ Method: `setDate` เพื่อกำหนดเป็นวันที่ 12/5/2004
   2.3. เรียกใช้ Method: `toGeneralFormat` เพื่อรับข้อความในรูปแบบของวันที่ แล้วแสดงวันที่นั้นออกทางจอภาพ
3. คอมไพล์และรันโปรแกรม

```
Date: 12/5/2004
```

4. ทดสอบโปรแกรมโดยกำหนดค่าให้กับแต่ละ Attribute ด้วยวิธีการ Command Line Input

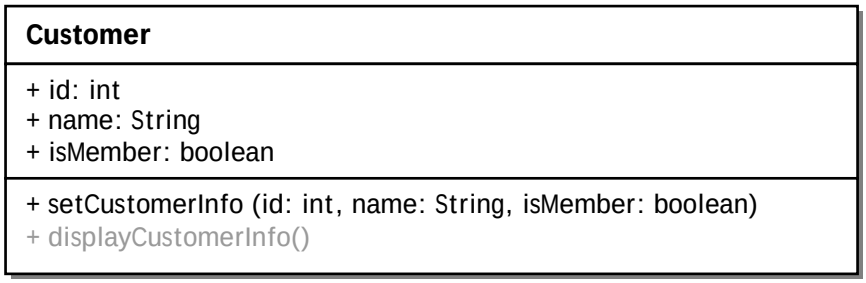
Project Direct Clothing #4

กิจกรรมที่ 1: Item.java



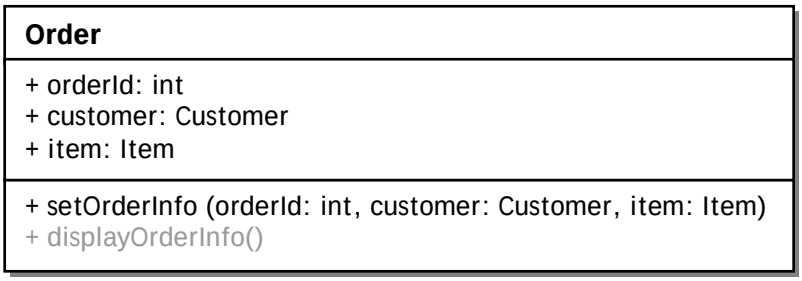
1. ในคลาส **`Item`** ให้ประกาศ Method ชื่อ **`setItemInfo`** เพื่อใช้กำหนดข้อมูลของสินค้า ที่มีรายละเอียดดังนี้
   - รับค่า Arguments ที่เป็นข้อมูลของ รหัสสินค้า, รายละเอียดสินค้า, ราคา และประเภทสินค้า
   - กำหนดค่าให้กับ Attribute แต่ละตัวด้วยข้อมูลที่ส่งผ่าน Arguments มา
2. ประกาศ Method ใหม่ชื่อ **`setItemInfo`** (Overloading Method) ที่มีรายละเอียดดังนี้
   - รับค่า Arguments ที่เป็นข้อมูลของ รหัสสินค้า, รายละเอียดสินค้าและประเภทสินค้า
   - กำหนดค่าให้กับ Attribute แต่ละตัวด้วยข้อมูลที่ส่งผ่าน Arguments มา โดยกำหนดราคาของสินค้าเป็น *100.0*

กิจกรรมที่ 2: Customer.java



1. ในคลาส **`Customer`** ให้ประกาศ Method ชื่อ **`setCustomerInfo`** ที่มีรายละเอียดดังนี้
   - รับค่า Arguments ที่เป็นข้อมูลของ รหัสลูกค้า ชื่อลูกค้า และสถานะลูกค้า
   - กำหนดค่าให้กับ Attribute แต่ละตัวด้วยข้อมูลที่ส่งผ่าน Arguments มา

กิจกรรมที่ 3: Order.java



1. ในคลาส **Order** ให้ประกาศ Method ชื่อ **`setOrderInfo`** ที่มีรายละเอียดดังนี้
   - รับค่า Arguments ที่เป็นข้อมูลของ รหัสใบสั่งซื้อ, ลูกค้าที่สั่ง และสินค้าที่สั่ง
   - กำหนดค่าให้กับ Attribute แต่ละตัวด้วยข้อมูลที่ส่งผ่าน Arguments มา

กิจกรรมที่ 4: TestDirect4.java

1. ประกาศคลาสใหม่ชื่อ **`TestDirect4`**
2. ภายใน Main Method ให้ปฏิบัติดังต่อไปนี้

   2.1. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Customer
   - เรียกใช้ Method: `setCustomerInfo` เพื่อกำหนดค่า Attribute ของ Customer

   2.2. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Item
   - เรียกใช้ Method: `setItemInfo` เพื่อกำหนดค่า Attribute ของ Item

   2.3. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Order
   - เรียกใช้ Method: `setOrderInfo` เพื่อกำหนดค่า Attribute ของ Order โดยใช้ตัวแปร Object: Customer และ Item ที่สร้างไว้ในข้อ 2.1 และ 2.2 สำหรับกำหนดค่าให้กับข้อมูล *ลูกค้า* และ*สินค้าที่สั่ง* ในใบสั่งซื้อ
   - เรียกใช้ Method: `displayOrderInfo` เพื่อแสดงรายละเอียดของใบสั่งซื้อ
3. คอมไพล์และรันโปรแกรม

```
===== ORDER INFORMATION =====
 Order Id    : <value>
 Customer    : <customer-name>
-----------------------------
 ITEM        : <item-desc> - <item-price>
=============================
```

บทที่ 5
นิพจน์และคำสั่งควบคุม

แบบฝึกหัดที่ 1: การใช้คำสั่งเงื่อนไข

รายละเอียด: เขียนโปรแกรมเพื่อสร้าง Method สำหรับการกำหนดหาค่าภาษีเงินได้

กิจกรรมที่ 1: IncomeFunction.java

1. ประกาศคลาสชื่อ **`IncomeFunction`**
2. ประกาศ Method ชื่อ **`calculateTax`** เพื่อใช้คำนวณภาษีที่ต้องจ่ายในแต่ละเดือนโดยมีการรับข้อมูล **เงินเดือน**เป็น Argument และส่งค่ากลับเป็นภาษีที่คำนวณได้
3. สำหรับวิธีการคำนวณภาษี จะมีสูตรที่ใช้ดังนี้

   3.1. รายได้ต่อปี = เงินเดือน x 12

   3.2. เงินได้ = รายได้ต่อปี – ค่าใช้จ่าย 40% ของรายได้ต่อปี (ไม่เกิน 60,000)

   3.3. อัตราภาษีที่ใช้คำนวณ มีดังนี้

   - เงินได้ 0 – 80,000 ยกเว้นภาษี
   - เงินได้ 80,001 - 100,000 ภาษี = เงินได้ส่วนที่เกิน 80,000 * 5%
   - เงินได้ 100,001 - 500,000 ภาษี = 1,000 + เงินได้ส่วนที่เกิน 100,000 * 10%
   - เงินได้ 500,001 – 1,000,000 ภาษี = 41,000 + เงินได้ส่วนที่เกิน 500,000 * 20%
   - เงินได้1,000,001 ขึ้นไป ภาษี = 141,000 + เงินได้ส่วนที่เกิน 1,000,000 * 30%

   3.4. ภาษีที่ต้องจ่าย = ภาษี ÷ 12
4. ข้อมูลที่ใช้ส่งกลับคือ ภาษีที่ต้องจ่าย

กิจกรรมที่ 2: TestIncome.java

1. สร้างคลาสชื่อ **`TestIncome`**
2. ภายใน Main Method เรียกใช้ Method: **`calculateTax`** เพื่อคำนวณหาภาษีที่ต้องจ่ายด้วยข้อมูลที่ป้อนผ่านทาง Command Line

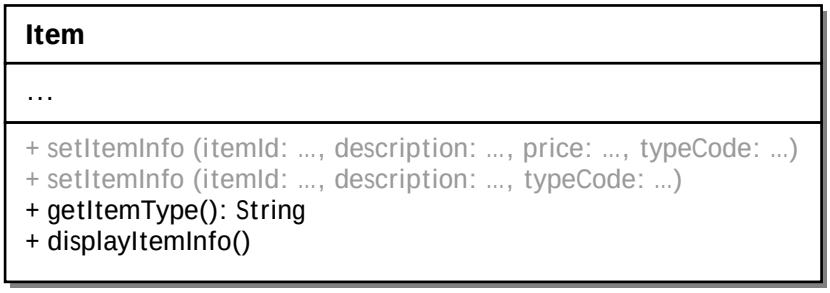
   หมายเหตุ การแปลงข้อมูลจาก String ให้เป็น double สามารถเรียกใช้ Method ได้ดังนี้

   ```
   double x = Double.parseDouble(str);
   ```
3. คอมไพล์และรันโปรแกรมที่มีการป้อนข้อมูลรายได้สุทธิด้วยวิธีการ Command Line Input

```
> java TestIncome 25000
Tax = 1250.0
```

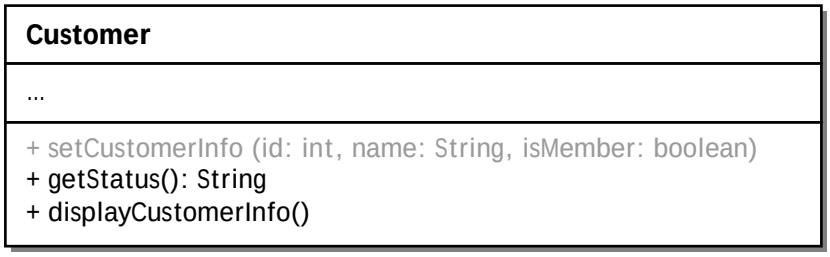
Project Direct Clothing #5

กิจกรรมที่ 1: Item.java



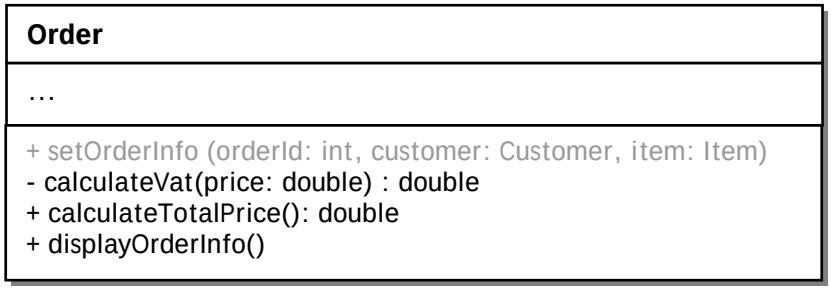
1. ในคลาส **`Item`** ให้ประกาศ Method ชื่อ **`getItemType`** ที่มีรายละเอียดดังนี้
   - ส่งข้อความกลับเป็น *ชื่อของประเภทสินค้า* ตามข้อมูลที่อยู่ใน Attribute: `typeCode` (‘B’=Beverage, ‘C’=Cosmetic, ‘S’=Snack)
2. แก้ไข Method: **`displayItemInfo`** เพื่อเรียกใช้ Method: `getItemType` แทนการอ้างอิงชื่อ Attribute

กิจกรรมที่ 2: Customer.java



1. ในคลาส **`Customer`** ให้ประกาศ Method ชื่อ **`getStatus`** ที่มีรายละเอียดดังนี้
   - ส่งข้อความกลับตามข้อมูลใน Attribute: `status` เพื่อแสดงว่าเป็นสมาชิกหรือไม่ โดยถ้าเป็นสมาชิก ให้ส่งค่ากลับเป็นคำว่า “Member” แต่ถ้าไม่ใช่ให้ส่งค่ากลับเป็นคำว่า “Not Member”
2. แก้ไข Method: **`displayCustomerInfo`** เพื่อเรียกใช้ Method: `getStatus` แทนการอ้างอิงชื่อ Attribute

กิจกรรมที่ 3: Order.java



1. ในคลาส **Order** ให้ประกาศ Private Method ชื่อ **calculateVat** ที่มีรายละเอียดดังนี้
   - รับค่า Arguments 1 ตัวที่เป็นข้อมูลราคาสินค้าที่ต้องการคำนวณหาภาษี
   - ส่งค่ากลับเป็นภาษีมูลค่าเพิ่มที่คำนวณได้
2. ประกาศ Method ชื่อ **calculateTotalPrice** ที่มีรายละเอียดดังต่อไปนี้
   - คำนวณหาราคาสุทธิ ซึ่งเป็นผลลัพธ์ของราคาสินค้าในใบสั่งซื้อที่รวมภาษีมูลค่าเพิ่มแล้ว (เรียกใช้ Method: `calculateVat` เพื่อการคำนวณหาภาษีมูลค่าเพิ่ม)
   - ส่งค่ากลับเป็นราคาสุทธิที่คำนวณได้
3. แก้ไข Method: **displayOrderInfo** เพื่อเพิ่มการแสดงผลราคาสุทธิของใบสั่งซื้อ ดังตัวอย่างต่อไปนี้ (อาจใช้วิธีการปรับแต่งการแสดงผลตัวเลขราคาสุทธิเป็นทศนิยม 2 ตำแหน่ง)

```
===== ORDER INFORMATION =====
 Order Id    : <value>
 Customer    : <customer-name>
-----------------------------
 ITEM        : <item-desc> - <item-price>
-----------------------------
 NET PRICE   : <value>
=============================
```

กิจกรรมที่ 3: TestDirect5.java

1. ประกาศคลาสใหม่ชื่อ **TestDirect5**
2. ภายใน Main Method ให้ปฏิบัติดังต่อไปนี้

   2.1. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Customer
   - เรียกใช้ Method: `setCustomerInfo` เพื่อกำหนดค่า Attribute ของ Customer
   - เรียกใช้ Method: `displayCustomerInfo` เพื่อแสดงรายละเอียดของลูกค้า

   2.2. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Item
   - เรียกใช้ Method: `setItemInfo` เพื่อกำหนดค่า Attribute ของ Item
   - เรียกใช้ Method: `displayItemInfo` เพื่อแสดงรายละเอียดของสินค้า

   2.3. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Order
   - เรียกใช้ Method: `setOrderInfo` เพื่อกำหนดค่า Attribute ของ Order
   - เรียกใช้ Method: `displayOrderInfo` เพื่อแสดงรายละเอียดของใบสั่งซื้อ
3. คอมไพล์และรันโปรแกรม

บทที่ 6
Arrays

แบบฝึกหัดที่ 1 การสร้าง Array แบบ Primitive Type

รายละเอียด: เขียนโปรแกรมเพื่อสร้าง Method สำหรับการหาตัวเลขที่มีค่ามากที่สุดจากชุดของตัวเลขใดๆ

กิจกรรมที่ 1: NumberUtil.java

1. ประกาศคลาสชื่อ **`NumberUtil`**
2. ประกาศ Method ชื่อ **`findMaximum`** ที่มีรายละเอียด ดังนี้
   - รับ Arguments เป็นชุดข้อมูลเลขจำนวนเต็ม
   - นำตัวเลขเหล่านั้นมาคำนวณหาตัวเลขที่มีค่ามากที่สุด
   - ส่งค่ากลับเป็นตัวเลขที่คำนวณได้

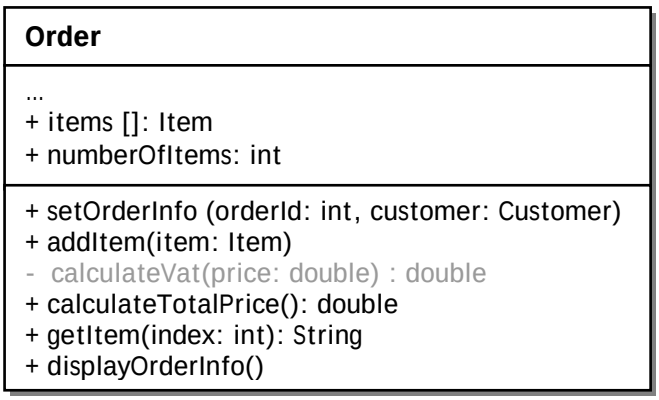
กิจกรรมที่ 2: TestNumberUtil.java

1. ประกาศคลาสชื่อ **`TestNumberUtil`**
2. ภายใน Main Method :
   2.1. รับชุดข้อมูลผ่านทาง Command Line Input แล้วใช้คำสั่งเพื่อแปลงชุดข้อมูลนั้นให้เป็นตัวเลข แล้วจัดเก็บไว้ใน Array ของ int
   2.2. เรียกใช้ Method: **`findMaximum`** โดยส่งตัวแปร Array ในข้อ 2 เป็น Argument เพื่อให้ได้ข้อมูลตัวเลขที่มีค่ามากที่สุด
   2.3. แสดงผลลัพธ์ตัวเลขสูงสุดออกทางจอภาพ
3. คอมไพล์และรันโปรแกรมด้วยการป้อนข้อมูลชุดตัวเลข ดังตัวอย่าง

```
> java ArrayInput 12 5 8 19 6
Maximum number is 19
```

Project Direct Clothing #6

กิจกรรมที่ 1: Order.java



1. แก้ไขคลาส **`Order`** ด้วยการเปลี่ยนแปลง Attribute ต่างๆ ดังนี้
   - เปลี่ยนแปลงตัวแปร **`item`** ให้เป็นแบบ Array เพื่อให้สามารถเก็บข้อมูลสินค้าได้มากกว่า 1 รายการ (ในที่นี้จะเปลี่ยนชื่อตัวแปรเป็น **`items`**) โดยกำหนดให้มีสินค้าได้ไม่เกิน 10 รายการ
   - ประกาศตัวแปรชื่อ **`numberOfItems`** สำหรับเก็บจำนวนสินค้าที่มีอยู่จริงในใบสั่งซื้อ
2. แก้ไข Method **`setOrderInfo`** ดังนี้
   - ยกเลิก Argument และการกำหนดค่าให้กับ Attribute ข้อมูลที่เป็นสินค้าในใบสั่งซื้อ
3. ประกาศ Method ชื่อ **`addItem`** ซึ่งมีรายละเอียดดังนี้
   - รับค่า Argument 1 ตัวเป็น Object ของสินค้าที่ต้องการสั่งซื้อ
   - กำหนดค่าให้กับ Attribute: `items` ด้วย Argument ที่รับมาในตำแหน่งของ Array ที่เหมาะสม
   - เพิ่มค่าในตัวแปร `numberOfItems`
4. แก้ไข Method: **`calculateTotalPrice`** ที่ใช้คำนวณหาราคาสุทธิ ดังนี้
   - คำนวณหาราคาสุทธิที่เป็นผลรวมของราคาสินค้าทั้งหมดที่รวมภาษีมูลค่าเพิ่มแล้ว
   - ส่งค่ากลับเป็นราคาสุทธิที่คำนวณได้
5. ประกาศ Method: **`getItem`** ดังรายละเอียดต่อไปนี้
   - รับค่า Argument 1 ตัวเป็นลำดับของสินค้าใน Array (index) ที่ต้องการข้อมูล
   - ส่งค่ากลับเป็น รายละเอียดสินค้า และราคาของสินค้า ตามลำดับของสินค้าที่ส่งเข้ามาโดยใช้รูปแบบของข้อความเป็น *<item-desc> - <item-price>*
6. แก้ไข Method: **`displayOrderInfo`** เพื่อแสดงรายละเอียดของสินค้าทั้งหมดที่อยู่ใน Array ออกทางจอภาพ (ใช้ Method: `getItem()` เพื่อช่วยในการแสดงผล)

```
-----------------------------
 ITEM#1   : <item-desc> - <item-price>
 ITEM#2   : <item-desc> - <item-price>
  …
-----------------------------
```

กิจกรรมที่ 2: TestDirect6.java

1. ประกาศคลาสใหม่ชื่อ **`TestDirect6`**
2. ภายใน Main Method ให้ปฏิบัติดังนี้

   2.1. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Customer

   - เรียกใช้ Method: `setCustomerInfo()` เพื่อกำหนดค่าให้กับ Attribute ของ Customer

   2.2. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ Item 3 ตัวแปร (หรือมากกว่า)

   - เรียกใช้ Method: `setItemInfo()` เพื่อกำหนดค่าให้กับ Attribute ของแต่ละ Item

   2.3. สร้างตัวแปรที่อ้างอิงไปยัง Object ของ `Order`

   - เรียกใช้ Method: `setOrderInfo()` เพื่อกำหนดค่าให้กับ Attribute ของ Order
   - เรียกใช้ Method: `addItem()` เพื่อเพิ่มสินค้าที่ได้สร้างไว้ในข้อ 2.2 ลงในใบสั่งซื้อ
   - เรียกใช้ Method: `displayOrderInfo()` เพื่อแสดงรายละเอียดของใบสั่งซื้อ

3. คอมไพล์และรันโปรแกรม

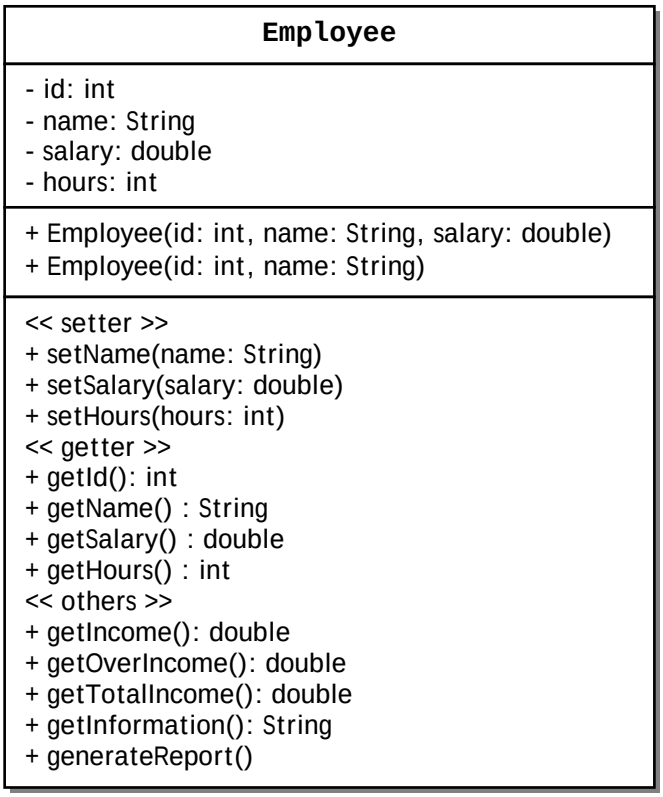
```
===== ORDER INFORMATION =====
 Order Id    : <value>
 Customer    : <customer-name>
-----------------------------
 ITEM#1      : <item-desc> - <item-price>
 ITEM#2      : <item-desc> - <item-price>
 ITEM#3      : <item-desc> - <item-price>
  …
-----------------------------
 NET PRICE   : <value>
=============================
```

บทที่ 7

Encapsulation และ Constructor

Project Personnel#1: Encapsulation และ Constructor

รายละเอียด: เขียนโปรแกรมเพื่อสร้างคลาส Employee ที่มีการใช้คุณสมบัติ Encapsulation และการสร้าง Constructor



กิจกรรมที่ 1: Employee

บริษัท ABC ที่เปิดให้บริการด้านการฝึกอบรม ต้องการสร้างโปรแกรมสำหรับคิดรายได้ของพนักงานทั่วไป มีรายละเอียดดังนี้

1. พนักงานแต่ละคนจะมีข้อมูลส่วนตัวคือ รหัสพนักงาน: **id**, ชื่อพนักงาน: **name**, อัตราเงินเดือน : **salary** และจำนวนชั่วโมงล่วงเวลา: **hours**
2. กำหนดให้มีการใช้คุณสมบัติ **Encapsulation** กับคลาสนี้
3. การสร้างพนักงานแต่ละคนจะต้องมีการรับข้อมูล *รหัสพนักงาน ชื่อพนักงาน* และ*อัตราเงินเดือน* ถ้าไม่มีข้อมูลอัตราเงินเดือนให้กำหนดเป็นค่าปกติคือ **10,000** บาท
4. การคิดรายได้ต่อเดือน (**getIncome**) จะต้องมีการหักเงินประกันสังคม **3%** ของอัตราเงินเดือน

   รายได้ต่อเดือน = อัตราเงินเดือน - เงินประกันสังคม

5. การคิดรายได้พิเศษ (**getOverIncome**) จะเป็นการคำนวณหาค่าล่วงเวลา โดยให้ในอัตราชั่วโมงละ 40 บาท

   รายได้พิเศษ = จำนวนชั่วโมงล่วงเวลา x 40

6. การคิดรายได้รวม (**getTotalIncome**) เป็นการหาผลรวมระหว่างรายได้ต่อเดือนและรายได้พิเศษ

   รายได้รวม = รายได้ต่อเดือน + รายได้พิเศษ

7. ข้อความที่ส่งกลับเพื่อแสดงรายละเอียดของพนักงาน (**getInformation**) มีรูปแบบดังตัวอย่างต่อไปนี้

```
ID: <emp-id> NAME: <emp-name>
```

8. การแสดงรายงาน (**generateReport**) ออกทางจอภาพ จะมีรูปแบบการแสดงผล ดังนี้

```
======================
ID: <emp-id> NAME: <emp-name>
----------------------
TOTAL: <total-income>
======================
```

กิจกรรมที่ 2 TestEmployee

1. ถ้าในบริษัทแห่งนี้มีพนักงาน 2 คน คือ

รหัสพนักงาน	ชื่อพนักงาน	อัตราเงินเดือน
1001	Mark	12000
1002	Susan	10000

2. ในเดือนมิถุนายน Susan ทำงานล่วงเวลาจำนวน 30 ชั่วโมง จงคำนวณหารายได้รวมที่แต่ละคนจะได้รับ

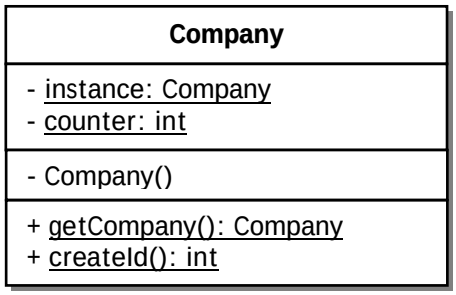
```
======================
ID: 1001 NAME: Mark
----------------------
TOTAL: 11640.0
======================
======================
ID: 1002 NAME: Susan
----------------------
TOTAL: 10900.0
======================
```

บทที่ 8

Static Member และ Deployment

Project Personnel#2: Singleton Design Pattern, Running Number และ Package

รายละเอียด: เขียนโปรแกรมเพื่อสร้างคลาส Company ที่มีคุณสมบัติในแบบของ Singleton Object, Static Member และการประกาศ Package



กิจกรรมที่ 1: personnel.data.Company

1. ประกาศคลาสชื่อ **Company** (กำหนดให้อยู่ใน Package personnel.data)
2. กำหนดให้มีการใช้ **Singleton Design Pattern** สำหรับ Object ของ Company
3. กิจกรรมสำหรับการส่งค่ารหัสพนักงานกลับ (**createId**) ในวิธีการแบบ Running Number โดยเริ่มต้นที่รหัส 1001 และกำหนดให้สามารถเรียกใช้กิจกรรมนี้ได้โดยไม่ต้องผ่านทาง Instance

กิจกรรมที่ 2: personnel.data.Employee

1. แก้ไขคลาส `Employee` ให้อยู่ใน Package personnel.data

กิจกรรมที่ 3: TestEmployee

1. แก้ไขคำสั่งสร้างพนักงานแต่ละคนโดยให้เรียกใช้ Method: `createId` ในคลาส Company เพื่อช่วยกำหนดข้อมูลรหัสพนักงานแทน (ใช้ข้อมูลของพนักงานคนเดิม)

```
====================
ID: 1001 NAME: Mark
--------------------
TOTAL: 11640.0
====================
====================
ID: 1002 NAME: Susan
--------------------
TOTAL: 10900.0
====================
```

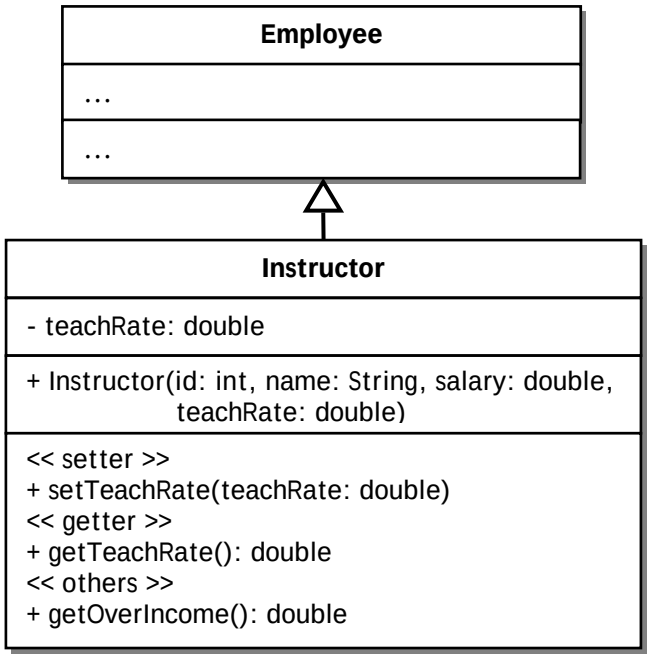
2. ทดสอบการสร้าง Executable Jar File โดยใช้คลาสที่สร้างไว้ใน Project ทั้งหมด

บทที่ 9
Inheritance

Project Personnel#3: คุณสมบัติ Inheritance

รายละเอียด: เขียนโปรแกรมเพื่อสร้างคลาส Instructor โดยใช้คุณสมบัติ Inheritance (ใช้คลาส Employee จากแบบฝึกหัดก่อนหน้า)

กิจกรรมที่ 1: personnel.data.Instructor



จากโปรแกรมการคิดรายได้ของบริษัท ABC ถ้ามีการว่าจ้างวิทยากรเพิ่มขึ้นอีก 2 คน ซึ่งจะมีกฎเกณฑ์การทำงานเหมือนกับพนักงานทั่วไป แต่มีรายละเอียดเพิ่มขึ้น ดังนี้

1. วิทยากรแต่ละคนจะมีข้อมูลเพิ่มเติมคือ อัตราค่าสอน: **`teachRate`**
2. กำหนดให้มีการใช้คุณสมบัติ **Encapsulation** กับคลาสนี้
3. การสร้างวิทยากรจะต้องมีการรับข้อมูล *รหัสพนักงาน ชื่อพนักงาน อัตราเงินเดือน* และ*อัตราค่าสอน*
4. การคำนวณหารายได้พิเศษของวิทยากร (**`getOverIncome`**) จะไม่มีการคิดค่าล่วงเวลา แต่จะใช้จำนวนชั่วโมงล่วงเวลาเป็นจำนวนชั่วโมงสอนแทน การคิดรายได้พิเศษจึงเป็นการคำนวณค่าสอนแทน *รายได้พิเศษ= จำนวนชั่วโมงสอน* x *อัตราค่าสอน*
5. กำหนดให้คลาสนี้อยู่ใน Package personnel.data

กิจกรรมที่ 2: TestEmployee

1. แก้ไขคลาส TestEmployee เพื่อเพิ่มวิทยากร 2 คนซึ่งมีข้อมูลส่วนตัว ดังนี้

รหัสพนักงาน	ชื่อพนักงาน	อัตราเงินเดือน	อัตราค่าสอน
1003	Linda	13000	150
1004	Antony	10000	120

2. ในเดือนมิถุนายน Antony มีสอน **54** ชั่วโมง จงคำนวณหารายได้ของวิทยากรแต่ละคน

```
======================
ID: 1001 NAME: Mark
----------------------
TOTAL: 11640.0
======================
======================
ID: 1002 NAME: Susan
----------------------
TOTAL: 10900.0
======================
======================
ID: 1003 NAME: Linda
----------------------
TOTAL: 12610.0
======================
======================
ID: 1004 NAME: Antony
----------------------
TOTAL: 16180.0
======================
```

บทที่ 10
Polymorphism

Project Personnel#4: คุณสมบัติ Polymorphism

รายละเอียด: เขียนโปรแกรมประยุกต์ใช้คุณสมบัติ Polymorphism เพื่อออกรายงานแสดงรายได้สุทธิของพนักงานแต่ละคน (ใช้คลาส Employee และ Instructor จากแบบฝึกหัดก่อนหน้า)

ถ้าต้องการเปลี่ยนแปลงรูปแบบของการออกรายงานใหม่ โดยสร้างคลาสใหม่ที่มี Method สำหรับการออกรายงานของพนักงานแต่ละคนโดยเฉพาะ

กิจกรรมที่ 1: personnel.data.Employee

1. ลบกิจกรรมสำหรับการแสดงรายงานออกทางจอภาพ: `generateReport`

กิจกรรมที่ 2: personnel.reports.Report

1. สร้างคลาสใหม่ชื่อ **`Report`** เพื่อใช้สำหรับการออกรายงานเพื่อแสดงรายละเอียดการคิดรายได้สุทธิของพนักงานแต่ละคนทางจอภาพ โดยกำหนดให้คลาสนี้อยู่ใน Package personnel.reports
2. สร้างกิจกรรมสำหรับการแสดงข้อมูลรายได้สุทธิของพนักงาน (**generateReport**) (กำหนดให้สามารถเรียกใช้กิจกรรมได้โดยผ่านทางชื่อคลาส) ซึ่งมีรายละเอียดดังนี้

```
public static void generateReport(Employee person) {
  String position = "";
  if (person instanceof Instructor) {
    position = "Instructor";
  } else if (person instanceof Employee) {
    position = "Employee";
  }

  double income     = person.getIncome();
  int    hours      = person.getHours();
  double overIncome = person.getOverIncome();
  double total      = person.getTotalIncome();

  System.out.println("=======================");
  System.out.println(person.getInformation());
  System.out.println("-----------------------");
  System.out.printf ("POSITION   : %s%n", position);
  System.out.println("-----------------------");
  System.out.printf ("INCOME     : %10.2f%n", income);
  System.out.printf ("OT/TEACH   : %10d%n", hours);
  System.out.printf ("OVER INCOME: %10.2f%n", overIncome);
  System.out.printf ("TOTAL      : %10.2f%n", total);
  System.out.println("=======================\n");
}
```

กิจกรรมที่ 2: TestEmployee

1. จัดเก็บข้อมูลพนักงานทุกคนในแบบของ Heterogeneous Collection
2. กำหนดจำนวนชั่วโมงล่วงเวลาให้กับพนักงานแต่ละคน ดังนี้

ชื่อพนักงาน	จำนวนชั่วโมงล่วงเวลา
Susan	30
Antony	54

3. เรียกใช้ Method: **`generateReport`** ในคลาส `Report` เพื่อแสดงรายละเอียดการคิดรายได้สุทธิของพนักงานแต่ละคนออกทางจอภาพ

```
======================
ID: 1001 NAME: Mark
----------------------
POSITION   : Employee
----------------------
INCOME     :   11640.00
OT/TEACH   :          0
OVER INCOME:       0.00
TOTAL      :   11640.00
======================
======================
ID: 1002 NAME: Susan
----------------------
POSITION   : Employee
----------------------
INCOME     :    9700.00
OT/TEACH   :         30
OVER INCOME:    1200.00
TOTAL      :   10900.00
======================
```

```
======================
ID: 1003 NAME: Linda
----------------------
POSITION   : Instructor
----------------------
INCOME     :   12610.00
OT/TEACH   :          0
OVER INCOME:       0.00
TOTAL      :   12610.00
======================
======================
ID: 1004 NAME: Antony
----------------------
POSITION   : Instructor
----------------------
INCOME     :    9700.00
OT/TEACH   :         54
OVER INCOME:    6480.00
TOTAL      :   16180.00
======================
```

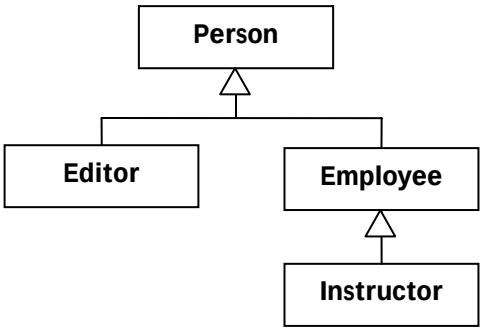
บทที่ 12

Abstract Class และ Interface

Project Personnel#5: การสร้าง Abstract Class

รายละเอียด: เขียนโปรแกรมเพื่อสร้างและเรียกใช้ Abstract Class และการประยุกต์ใช้คุณสมบัติ Polymorphism (ใช้คลาส Employee และ Instructor จากแบบฝึกหัดก่อนหน้า)

ถ้าบริษัท ABC ต้องการคิดรายได้ให้กับพนักงานที่เป็นเจ้าหน้าที่ผลิตเอกสาร (**Editor**) ซึ่งทั้งเจ้าหน้าที่ผลิตเอกสารและพนักงานทั่วไปนั้น จะมีคุณสมบัติบางอย่างที่เหมือนกันคือ *ชื่อพนักงาน* และ*จำนวนชั่วโมงล่วงเวลา* *การคำนวณหารายได้พิเศษ* และ*รายได้รวม* และบางกิจกรรมที่มีรายละเอียดการทำงานที่ต่างกัน เช่น การคิดรายได้ต่อเดือน การคิดค่าเบี้ยเลี้ยงพิเศษ ข้อความที่ใช้แสดงรายละเอียดของพนักงาน เป็นต้น จึงต้องมีการสร้างคลาสที่ใช้เป็นต้นแบบสำหรับพนักงานและเจ้าหน้าที่ผลิตเอกสารชื่อ **Person** ดังนี้

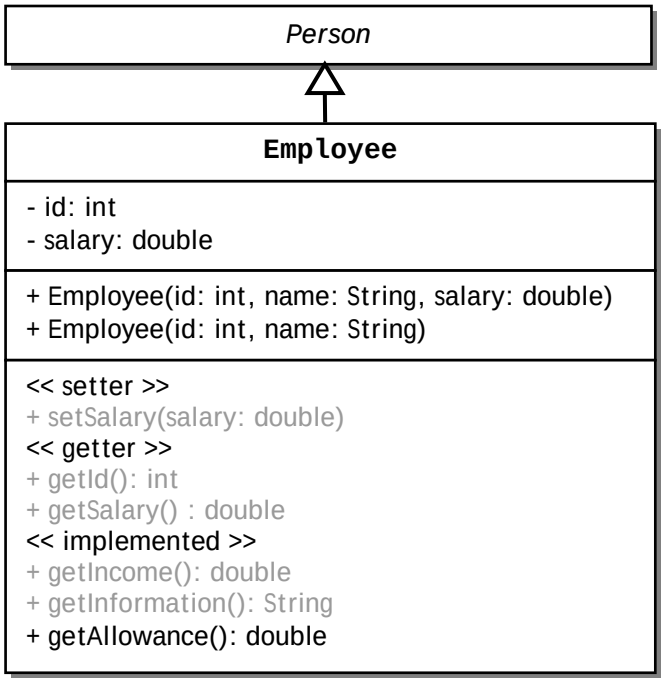


กิจกรรมที่ 1: personnel.data.Person

Person
- OT_RATE: int - name: String - hours: int
Person(name: String)
<< setter >> + setName(name: String) + setHours(hours: int) << getter >> + getName() : String + getHours() : int << abstract >> + *getIncome(): double* + *getInformation(): String* + *getAllowance(): double* << others >> + getOverIncome(): double + getTotalIncome(): double + getTax(salary: double): double

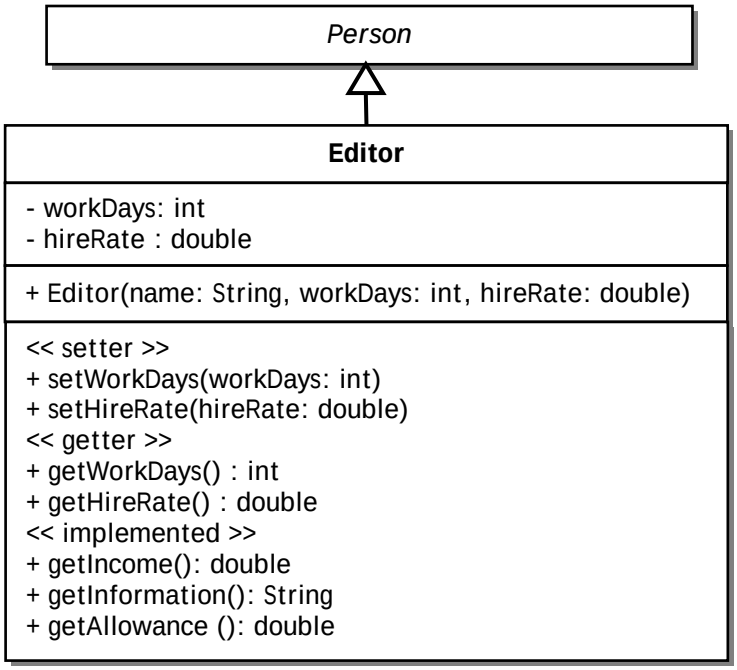
1. กำหนดให้อยู่ใน Package personnel.data
2. กำหนดให้คลาส Person เป็น **Abstract Class** ที่ประกอบไปด้วยข้อมูล ดังต่อไปนี้
   - ค่าคงที่แสดงอัตราค่าล่วงเวลา (OT_RATE) เท่ากับ 40 บาทต่อชั่วโมง
   - ข้อมูลทั่วไปสำหรับทุกๆ Subclass (Common Attribute) ได้แก่ ชื่อพนักงาน: **name** และจำนวนชั่วโมงล่วงเวลา/ชั่วโมงสอน: **hours**
3. กำหนดให้มีการใช้คุณสมบัติ **Encapsulation** กับคลาสนี้
4. Constructor ของ **Person** จะมีการรับข้อมูล*ชื่อพนักงาน*เพื่อใช้กำหนดค่าให้กับ Attribute: name โดยกำหนดให้ Constructor สามารถเรียกใช้ได้เฉพาะจาก Subclass ของคลาสนี้เท่านั้น
5. คลาส **Person** ประกอบไปด้วยกิจกรรมที่มีรายละเอียดแตกต่างกันไปใน Subclass แต่ละตัว ดังนี้
   - กิจกรรมสำหรับการคิดรายได้ต่อเดือน
     `public abstract double getIncome();`
   - กิจกรรมสำหรับการส่งข้อความแสดงรายละเอียดของพนักงานหรือเจ้าหน้าที่
     `public abstract String getInformation();`
   - กิจกรรมสำหรับการคิดค่าเบี้ยเลี้ยงพิเศษ
     `public abstract double getAllowance();`
6. คลาส **Person** จะมีกิจกรรมที่สามารถเรียกใช้งานได้จากทุกๆ Subclass คือ
   - กิจกรรมสำหรับการคำนวณหารายได้พิเศษ
     `public double getOverIncome()`
     (สามารถคัดลอกมาจากคลาส Employee ได้)
   - กิจกรรมสำหรับการคิดรายได้รวม (รวมค่าล่วงเวลา, ค่าสอน)
     `public double getTotalIncome()`
     (สามารถคัดลอกมาจากคลาส Employee ได้)
   - กิจกรรมสำหรับการคิดภาษีเงินได้ (เมื่อ salary คือรายได้ที่ใช้คำนวณภาษี)
     `public double getTax(double salary)`
     (ให้ใช้สูตรในการคำนวณภาษีตามแบบฝึกหัดในบทที่ 6)

กิจกรรมที่ 2: personnel.data.Employee



1. กำหนดให้คลาส **Employee** มีการสืบทอดคุณสมบัติจาก **Person**
2. แก้ไขรายละเอียดในคลาส Employee ดังรายละเอียดต่อไปนี้
   2.1. ลบข้อมูลชื่อพนักงาน: name และจำนวนชั่วโมงล่วงเวลา: hours (ใช้จากคลาส Person)
   2.2. ลบ Setter และ Getter Method ที่ใช้จัดการกับข้อมูลชื่อพนักงานและจำนวนชั่วโมงล่วงเวลา (ใช้จากคลาส Person) ถ้ามีการอ้างอิง Attribute: name หรือ hours ใน Method อื่นๆ ให้เลี่ยงไปใช้ Getter Method แทนการอ้างอิงชื่อตัวแปร
   2.3. แก้ไขการสร้างพนักงาน ด้วยการส่งข้อมูลชื่อพนักงานที่ได้รับมาไปยัง Constructor ของคลาส Person เพื่อทำหน้าที่กำหนดค่าให้
   2.4. ลบกิจกรรมสำหรับการคิดรายได้พิเศษ: getOverIncome และรายได้รวม: getTotalIncome (ให้ใช้ตามสูตรของ Person)
3. พนักงานแต่ละคนจะได้รับค่าเบี้ยเลี้ยงพิเศษต่อเดือน (**getAllowance**)เป็นจำนวนเงิน 10% ของอัตราเงินเดือน

กิจกรรมที่ 3: personnel.data.Editor



เจ้าหน้าที่ผลิตเอกสาร (Editor) จะมีการคิดรายได้ต่อเดือนแบบอัตราจ้างรายวัน โดยเจ้าหน้าที่แต่ละคนจะมีจำนวนวันทำงานในแต่ละเดือนไม่เท่ากัน และสามารถที่จะทำงานล่วงเวลาได้โดยจะได้รับค่าล่วงเวลาเท่ากับพนักงานทั่วไป

1. กำหนดให้คลาส **`Editor`** มีการสืบทอดคุณสมบัติจาก **`Person`** (อยู่ใน Package personnel.data)
2. เจ้าหน้าที่แต่ละคนจะมีข้อมูลส่วนตัวที่เพิ่มมาคือ จำนวนวันทำงาน: **`workDays`** และอัตราค่าจ้างรายวัน: **`hireRate`**
3. กำหนดให้มีการใช้คุณสมบัติ **Encapsulation** กับคลาสนี้
4. การสร้างเจ้าหน้าที่ผลิตเอกสาร จะต้องมีการรับข้อมูล *ชื่อเจ้าหน้าที่ จำนวนวันทำงาน*และ*อัตราค่าจ้างรายวัน*
5. การคิดรายได้ต่อเดือนของเจ้าหน้าที่ผลิตเอกสารแต่ละคน (**`getIncome`**) จะเป็นแบบอัตราจ้างรายวัน มีสูตรคำนวณคือ

   รายได้ต่อเดือน = (อัตราค่าจ้าง x *จำนวนวันทำงาน)*
6. ข้อมูลที่ส่งกลับเพื่อแสดงรายละเอียดของเจ้าหน้าที่ (**`getInformation`**) มีรูปแบบดังตัวอย่างต่อไปนี้

```
NAME: <editor-name>
DAYS: <workDays> RATES: <hireRate>
```

7. เจ้าหน้าที่ผลิตเอกสารจะได้เบี้ยเลี้ยงพิเศษ (**`getAllowance`**)เป็นเงิน 50 บาทต่อวัน

กิจกรรมที่ 4: personnel.reports.Report

1. แก้ไขกิจกรรมแสดงรายงาน (**generateReport**) เพื่อให้สามารถออกรายงานสำหรับเจ้าหน้าที่ได้ และเพิ่มการแสดงผล ค่าเบี้ยเลี้ยงพิเศษ ภาษี และ รายได้สุทธิ ดังนี้

   1.1. เปลี่ยนชนิดของ Argument ให้สามารถรับได้ทั้งพนักงาน (Employee) และเจ้าหน้าที่ (Editor)

   1.2. เพิ่มการตรวจสอบชนิดของพนักงานที่ส่งผ่านเข้ามา เพื่อกำหนดข้อความแสดงตำแหน่งของเจ้าหน้าที่ (Editor)

   1.3. รายได้สุทธิ (Net Income) มาจากรายได้รวม (Total) ที่รวมค่าเบี้ยเลี้ยงพิเศษ (Allowance) และหักภาษี (Tax) เรียบร้อยแล้ว (การคำนวณภาษีให้คิดจากรายได้รวม: Total)

   1.4. เพิ่มข้อความแสดงผลค่าเบี้ยเลี้ยงพิเศษ ภาษี และรายได้สุทธิ ในรายงาน ดังตัวอย่างต่อไปนี้

```
=======================
<< Information >>
-----------------------
POSITION   : <<position>>
-----------------------
INCOME     :   #####.##
OT/TEACH   :         ##
OVER INCOME:   #####.##
TOTAL      :   #####.##
ALLOWANCE  :   #####.##
TAX        :   #####.##
NET INCOME :   #####.##
=======================
```

กิจกรรมที่ 5: TestEmployee

1. แก้ไขชนิดของตัวแปร Heterogeneous Collection เพื่อให้สามารถเพิ่มเจ้าหน้าที่ผลิตเอกสาร 2 คนได้ ดังรายละเอียดต่อไปนี้

ชื่อเจ้าหน้าที่	จำนวนวัน	อัตราค่าจ้างรายวัน
John	20	220
Tom	23	200

2. เจ้าหน้าที่ Tom มีการทำงานล่วงเวลา 18 ชั่วโมง
3. เรียกใช้ Method: **`generateReport`** ในคลาส `Report` เพื่อแสดงรายละเอียดการคิดรายได้สุทธิของพนักงานและเจ้าหน้าที่ทุกคนออกทางจอภาพ

```
========================
ID: 1001 NAME: Mark
------------------------
POSITION    : Employee
------------------------
INCOME      :   11640.00
OT/TEACH    :          0
OVER INCOME:       0.00
TOTAL       :   11640.00
ALLOWANCE   :    1200.00
TAX         :      15.87
NET INCOME  :   12824.13
========================
========================
ID: 1002 NAME: Susan
------------------------
POSITION    : Employee
------------------------
INCOME      :    9700.00
OT/TEACH    :         30
OVER INCOME:    1200.00
TOTAL       :   10900.00
ALLOWANCE   :    1000.00
TAX         :       0.00
NET INCOME  :   11900.00
========================
```

```
========================
ID: 1003 NAME: Linda
------------------------
POSITION    : Instructor
------------------------
INCOME      :   12610.00
OT/TEACH    :          0
OVER INCOME:       0.00
TOTAL       :   12610.00
ALLOWANCE   :    1300.00
TAX         :      47.17
NET INCOME  :   13862.83
========================
========================
ID: 1004 NAME: Antony
------------------------
POSITION    : Instructor
------------------------
INCOME      :    9700.00
OT/TEACH    :         54
OVER INCOME:    6480.00
TOTAL       :   16180.00
ALLOWANCE   :    1000.00
TAX         :     368.00
NET INCOME  :   16812.00
========================
```

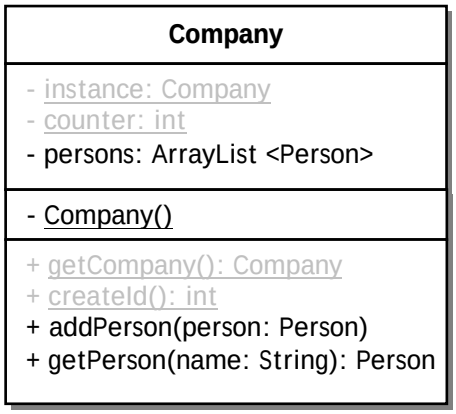
```
========================
NAME: John
DAYS: 20 RATES: 220.0
------------------------
POSITION    : Editor
------------------------
INCOME      :    4400.00
OT/TEACH    :          0
OVER INCOME:       0.00
TOTAL       :    4400.00
ALLOWANCE   :    1000.00
TAX         :       0.00
NET INCOME  :    5400.00
========================
========================
NAME: Tom
DAYS: 23 RATES: 200.0
------------------------
POSITION    : Editor
------------------------
INCOME      :    4600.00
OT/TEACH    :         18
OVER INCOME:     720.00
TOTAL       :    5320.00
ALLOWANCE   :    1150.00
TAX         :       0.00
NET INCOME  :    6470.00
========================
```

บทที่ 12
API Classes

Project Personnel#6: การสร้าง Collection (ArrayList)

รายละเอียด: เขียนโปรแกรมเพื่อใช้ Collection เพื่อเก็บข้อมูลของพนักงานและเจ้าหน้าที่ในคลาส Company

กิจกรรมที่ 1: personnel.data.Company



1. ประกาศตัวแปรเพื่อใช้เก็บข้อมูลพนักงานและเจ้าหน้าที่ เป็นชนิด **`ArrayList`** ที่อนุญาตให้เก็บได้เฉพาะ Object ที่เป็นชนิด **`Person`** เท่านั้น
2. ภายในคลาส Company ประกอบไปด้วยกิจกรรมเพิ่มเติม ดังนี้
   - กิจกรรมสำหรับการเพิ่มพนักงานหรือเจ้าหน้าที่ที่ส่งเข้ามา (person) ลงใน ArrayList
     `public void addPerson(Person person)`
   - กิจกรรมสำหรับส่งกลับเป็นพนักงานตาม*ชื่อของพนักงานหรือเจ้าหน้าที่*ที่ต้องการ
     `public Person getPerson(String name)`
     การเปรียบเทียบชื่อพนักงาน หรือเจ้าหน้าที่สามารถใช้พิมพ์เล็กหรือพิมพ์ใหญ่ก็ได้ (เรียกใช้ Method: `equalsIgnoreCase` เพื่อเปรียบเทียบข้อความโดยไม่สนใจตัวพิมพ์เล็กใหญ่) และส่งค่ากลับเป็น null ในกรณีที่ไม่พบชื่อพนักงานหรือเจ้าหน้าที่ที่ต้องการ

กิจกรรมที่ 2: TestEmployee

การทำงานของโปรแกรมจะเปลี่ยนไป โดยจะสร้างกิจกรรมสำหรับการสร้างบริษัทที่มีข้อมูลของพนักงานและเจ้าหน้าที่ทั้งหมดจัดเก็บไว้ และใช้วิธีการรับข้อมูลของพนักงานหรือเจ้าหน้าที่ผ่านทางแป้นพิมพ์ แล้วแสดงรายงานการคิดรายได้สุทธิของพนักงานหรือเจ้าหน้าที่คนนั้นออกทางจอภาพ

1. สร้างกิจกรรมสำหรับการสร้างบริษัท (Object: Company) ที่มีการเพิ่มพนักงานและเจ้าหน้าที่ทั้งหมดลงในบริษัท (กำหนดให้เป็น Static Method)

   `public static Company createCompany()`

   1.1. เรียกใช้ Method: `getCompany` สำหรับการสร้าง Object: Company (Singleton Object)

   1.2. เรียกใช้ Method: `addPerson` สำหรับการเพิ่มพนักงานและเจ้าหน้าที่ทุกคนลงใน Company (ใช้ข้อมูลจากแบบฝึกหัดก่อนหน้า)

   1.3. ส่งค่ากลับเป็น Object ของ Company (บริษัทที่มีข้อมูลของพนักงานทั้งหมดแล้ว)

2. แก้ไข Main Method ดังนี้

   2.1. เรียกใช้ Method: `createCompany` เพื่อสร้างบริษัทที่มีข้อมูลของพนักงานแล้ว

   2.2. รับข้อมูลผ่านทางแป้นพิมพ์โดยใช้ Scanner เป็นชื่อพนักงาน และจำนวนชั่วโมงล่วงเวลา

   2.3. เรียกใช้ Method: `getPerson(name)` เพื่อค้นหาพนักงานหรือเจ้าหน้าที่ตามข้อมูลที่ส่งผ่านมา

   2.4. ตรวจสอบข้อมูลว่าพบพนักงานที่ต้องการหรือไม่ ถ้าพบให้กำหนดจำนวนชั่วโมงล่วงเวลาตามข้อมูลที่ได้รับมา แล้วเรียกใช้ Method: `generateReport` เพื่อแสดงรายละเอียดการคิดรายได้สุทธิของพนักงานหรือเจ้าหน้าที่คนนั้นออกทางจอภาพ หากไม่พบให้แจ้งข้อความผิดพลาด

```
Enter person name   : Susan
Enter OT/Teach hours: 30
=======================
ID: 1002 NAME: Susan
-----------------------
POSITION  : Employee
-----------------------
INCOME      :    9700.00
OT/TEACH    :         30
OVER INCOME:    1200.00
TOTAL       :   10900.00
ALLOWANCE   :    1000.00
TAX         :       0.00
NET INCOME  :   11900.00
=======================
```

```
Enter person name: Robert
Enter OT/Teach hours: 30

ERROR: Invalid person name: Robert
```

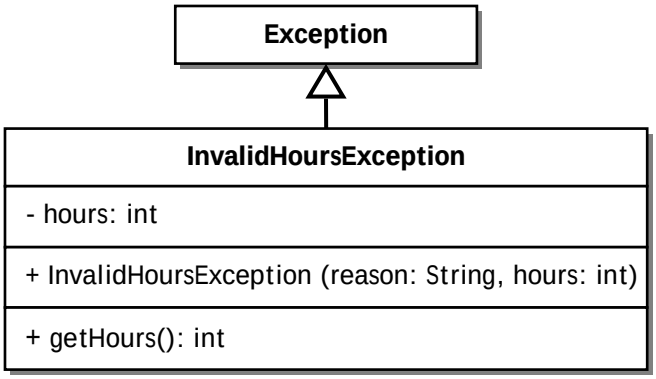
บทที่ 13
Exceptions

Project Personnel#7: การสร้าง Own Exception

รายละเอียด: สร้าง Exception Class เพื่อรองรับความผิดพลาดที่อาจเกิดขึ้นในการคิดรายได้สุทธิ

กิจกรรมที่ 1: personnel.data.InvalidHoursException

ในการกำหนดค่าจำนวนชั่วโมงล่วงเวลาจะมีกฎเกณฑ์หรือ ต้องไม่เกิน 80 ชั่วโมง ถ้ามีการส่งค่าตัวเลขจำนวนชั่วโมงล่วงเวลาผิดพลาดกำหนดให้เกิด Exception ขึ้น ดังนี้

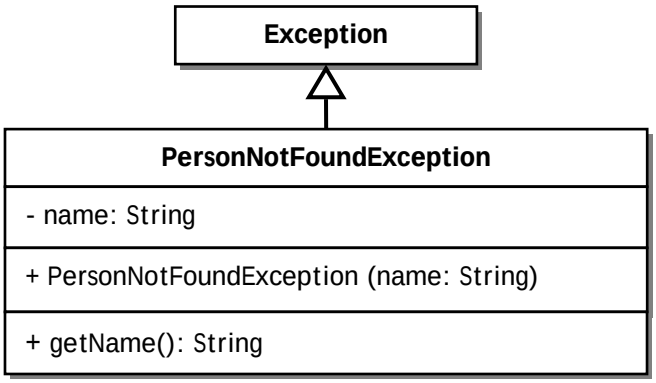


1. ประกาศตัวแปรเพื่อใช้เก็บตัวเลขชั่วโมงที่เกิดความผิดปกติ
2. ใน Constructor กำหนดให้มีการรับข้อมูล เหตุผลของความผิดปกติ และจำนวนชั่วโมงที่ผิดปกติ
3. กิจกรรมสำหรับการส่งค่ากลับเป็นจำนวนชั่วโมงที่ผิดปกติ

```
public int getHours()
```

กิจกรรมที่ 2: personnel.data.PersonNotFoundException

ในการค้นหาข้อมูลของพนักงานในบริษัทตามชื่อของพนักงานหรือเจ้าหน้าที่ที่กำหนด ถ้าไม่สามารถค้นหาข้อมูลได้ กำหนดให้เกิด Exception ขึ้น ดังนี้



1. ประกาศตัวแปรเพื่อใช้เก็บชื่อของพนักงานหรือเจ้าหน้าที่ที่ค้นหาไม่พบ
2. ใน Constructor กำหนดให้มีการรับเฉพาะข้อมูล ชื่อของพนักงาน ส่วนเหตุผลให้กำหนดเป็นข้อความตามความเหมาะสม
3. กิจกรรมสำหรับการส่งค่ากลับชื่อของพนักงานหรือเจ้าหน้าที่ที่ค้นหาไม่พบ
   `public String getName()`

กิจกรรมที่ 3: personnel.data.Person

1. แก้ไข Method: **`setHours()`** ดังนี้
   - กำหนดให้มีโอกาสเกิด **`InvalidHoursException`**
   - Exception จะเกิดขึ้นเมื่อมีความผิดปกติของตัวเลขจำนวนชั่วโมงล่วงเวลาใน 2 กรณี คือเป็นตัวเลขติดลบ หรือเกินกว่า **80** ชั่วโมง โดยใช้ข้อความเป็นเหตุผลตามความเหมาะสม

กิจกรรมที่ 4: personnel.data.Company

1. แก้ไข Method: **`getPerson(name)`** ดังนี้
   - กำหนดให้มีโอกาสเกิด **`PersonNotFoundException`**
   - Exception จะเกิดขึ้นเมื่อไม่สามารถค้นหาข้อมูลของพนักงานหรือเจ้าหน้าที่ในบริษัทตามชื่อที่ต้องการได้ (เปลี่ยนการส่งค่ากลับเป็น null เป็นการโยน Exception แทน)

กิจกรรมที่ 5: TestEmployee

1. แก้ไข Main Method เพื่อตรวจสอบความผิดพลาดต่างๆ โดยแสดงข้อความเหตุผลตามที่กำหนดไว้

   1.1. ความผิดพลาดจากการป้อนข้อมูลผิดประเภท (InputMismatchException)

   1.2. ความผิดพลาดจากการค้นหาพนักงานหรือเจ้าหน้าที่ไม่พบ (PersonNotFoundException)

   1.3. ความผิดพลาดจากการส่งข้อมูลจำนวนชั่วโมงล่วงเวลาที่ผิดปกติ (InvalidHoursException)

2. การรันโปรแกรมให้ทดลองป้อนชื่อของพนักงาน และจำนวนชั่วโมงที่ผิดปกติ

```
Enter person name   : Susan
Enter OT/Teach hours: xxx

ERROR: Invalid data format
```

กรณีป้อนข้อมูลผิดประเภท

```
Enter person name   : Robert
Enter OT/Teach hours: 30

ERROR: Invalid person name: Robert
```

กรณีค้นหาพนักงานหรือเจ้าหน้าที่ไม่พบ

```
Enter person name   : Susan
Enter OT/Teach hours: 90

ERROR: Over-time hours over limit: 90
```

กรณีส่งจำนวนชั่วโมงล่วงเวลาผิดพลาด

บทที่ 14
การติดต่อฐานข้อมูลเบื้องต้น

Project Personnel#8: ติดต่อไปยังฐานข้อมูล

รายละเอียด: สร้างคลาสเพื่อใช้ติดต่อไปยังฐานข้อมูล

กิจกรรมที่ 1: สร้างไฟล์ฐานข้อมูล

1. สร้างไฟล์ฐานข้อมูลชื่อ **Personnal**
2. สร้างตารางภายในไฟล์ฐานข้อมูลชื่อ **Employee** ที่ประกอบไปด้วยฟิลด์ต่างๆ คือ รหัสพนักงาน, ชื่อพนักงานและเงินเดือน
3. ใช้คำสั่งเพื่อเพิ่มข้อมูลของพนักงาน (อาจกำหนดขึ้นเอง หรือใช้ข้อมูลพนักงานจากแบบฝึกหัดก่อนหน้า) 4-6 เรคคอร์ด

กิจกรรมที่ 2: personnel.database.Database

1. ประกาศตัวแปร Connection เพื่อใช้ติดต่อไปยังไฟล์ฐานข้อมูล
2. ภายใน Constructor ของคลาส มีรายละเอียดดังนี้
   2.1. กำหนดให้มีการรับข้อมูล *ชื่อผู้ใช้* และ*รหัสผ่าน* เพื่อใช้สำหรับการ Login เข้าสู่ไฟล์ฐานข้อมูล
   2.2. โหลด JDBC Driver สำหรับไฟล์ฐานข้อมูลที่ใช้ (ในกรณีที่ใช้เป็น JDBC-ODBC Driver จะต้องสร้าง DSN เพื่อใช้ติดต่อไปยังไฟล์ฐานข้อมูลด้วย)
   2.3. สร้าง Connection เพื่อเชื่อมการติดต่อไปยังไฟล์ฐานข้อมูล
3. ภายในคลาส Database ประกอบไปด้วยกิจกรรมเพิ่มเติม ดังนี้
   - กิจกรรมสำหรับการตัดการเชื่อมต่อกับไฟล์ฐานข้อมูล
     `public void closeDatabase() throws SQLException`
   - กิจกรรมสำหรับการส่งค่ากลับเป็นข้อมูลของพนักงานทั้งหมดที่อยู่ในตาราง Employee
     `public Employee[] getAllEmployees() throws SQLException`

กิจกรรมที่ 2: TestEmployee

1. แก้ไขกิจกรรมสำหรับการสร้างบริษัท ดังนี้
   - เรียกใช้ Method: `getAllEmployees()` เพื่อให้ได้ข้อมูลพนักงานทั้งหมดจากไฟล์ฐานข้อมูล
   - เรียกใช้ Method: `addPerson(Person person)` เพื่อเพิ่มข้อมูลพนักงานที่อ่านจากไฟล์ฐานข้อมูลเข้าในบริษัท (Company)
   - กำหนดให้กิจกรรมนี้มีโอกาสเกิด Exception ด้วย