# บทที่ 8 การรับทอดและอินเทอร์เฟซ

การเขียนโปรแกรมเชิงวัตถุ ทำให้สามารถเขียนโปรแกรมให้มีการรับทอดจากคลาสแม่สู่คลาสลูกได้ ซึ่งจัดเป็นการใช้โปรแกรมเดิมซ้ำโดยผู้พัฒนาโปรแกรมรุ่นหลังไม่ต้องเริ่มเขียนโปรแกรมเองใหม่ทั้งหมด สามารถใช้คลาสเดิมที่มีอยู่มาขยายโปรแกรมเพิ่มเติมตามที่ต้องการ และสำหรับการรับทอดในภาษาจาวานี้ จาวาจะให้มีคลาสแม่เพียงคลาสเดียวเท่านั้น หากต้องการเขียนโปรแกรมให้มีการรับทอดจากคลาสแม่มากกว่าหนึ่งคลาส ในภาษาจาวาจะใช้อินเทอร์เฟซ

## 8.1 การรับทอด (Inheritance)

คือ การกำหนดคลาสใหม่ให้ขยายจากคลาสเดิมที่มีอยู่แล้ว โดยจะเรียกคลาสเดิมที่มีอยู่แล้วว่าเป็น คลาสแม่ (superclass, base class, parent class) ส่วนคลาสใหม่ที่ขยายมานั้นจะเรียกว่า คลาสลูก (subclass, derived class, extended class, child class) คลาสลูกจะได้รับการถ่ายทอดคุณสมบัติและพฤติกรรมจากคลาสแม่ โดยคลาสลูกสามารถเพิ่มตัวแปรหรือเมท็อดที่จำเป็นสำหรับคลาสลูก รวมทั้งสามารถปรับเปลี่ยนเมท็อดของคลาสแม่ได้ การรับทอดนี้เป็นการส่งเสริมให้มีการใช้โปรแกรมที่มีอยู่แล้ว จัดเป็นการใช้โค้ดซ้ำ (code reuse) ซึ่งช่วยลดความซ้ำซ้อนในการเขียนโปรแกรมได้ และทำให้สามารถพัฒนาโปรแกรมต่อยอดจากโปรแกรมเดิมที่มีอยู่ได้โดยไม่ต้องเข้าไปแก้ไขโปรแกรมเดิม คลาสทุกคลาสที่สร้างขึ้นในภาษาจาวานี้ จะเป็นคลาสลูกของคลาส Object เสมอ

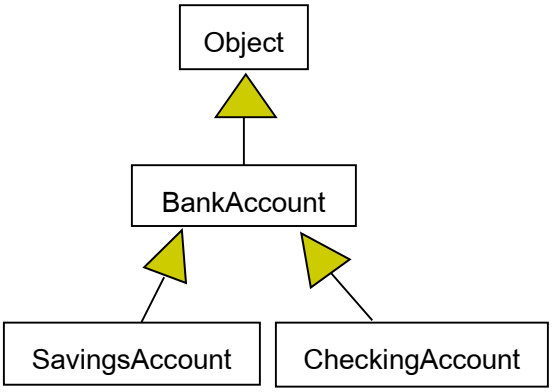
การสร้างคลาสใหม่ขยายจากคลาสแม่ ทำให้คลาสเหล่านี้มีความสัมพันธ์กันแบบ IS-A (IS-A relationship) เช่น มียานพาหนะเป็นคลาสแม่ และขยายเป็นคลาสลูกคือ รถยนต์ ความสัมพันธ์ของทั้งสองคลาสนี้คือแบบ IS-A relationship ถือว่ารถยนต์คือยานพาหนะ รถยนต์ก็จะมีคุณสมบัติและพฤติกรรมเหมือนของยานพาหนะ เช่น มีข้อมูลหรือคุณสมบัติคือหมายเลขทะเบียน รุ่น ยี่ห้อ สี เป็นต้น มีพฤติกรรมคือการเคลื่อนที่ เป็นต้น สำหรับรถยนต์อาจมีการเพิ่มข้อมูลอื่นได้ เช่น ความจุของถังน้ำมัน เป็นต้น ตัวอย่างของคลาสแม่และคลาสลูกอื่น ๆ แสดงดังตารางที่ 8.1

ตารางที่ 8.1 ตัวอย่างคลาสแม่และคลาสลูก

| **Superclass** | **Subclass** |
|---|---|
| Object | Other classes in Java |
| Student | GradStudent, UnderGradStudent |
| BankAccount | SavingsAccount, CheckingAccount, TimeDepositAccount |
| Shape | Triangle, Rectangle, Circle |
| Employee | SalariedEmployee, HourlyEmployee, CommissionEmployee |

### 8.1.1 ลำดับชั้นของการรับทอด (Inheritance hierarchy)

เราสามารถเขียนภาพแสดงความสัมพันธ์แบบ IS-A relationship ระหว่างคลาสได้ โดยใช้แผนภาพที่มีลักษณะโครงสร้างคล้ายต้นไม้ที่เรียกว่า inheritance hierarchy รูปที่ 8.1 แสดง inheritance hierarchy ของคลาส บัญชีเงินฝาก (BankAccount) บัญชีเงินฝากแบบออมทรัพย์ (SavingsAccount) และ บัญชีเงินฝากแบบใช้เช็ค (CheckingAccount)




รูปที่ 8.1 Inheritance hierarchy ระหว่างคลาส BankAccount, SavingsAccount และ CheckingAccount

### 8.1.2 ไวยากรณ์ของการรับทอด (Syntax of Inheritance)

จากรูปที่ 8.1 สามารถสร้างเป็นคลาส BankAccount ได้ดังนี้

ตัวอย่าง ไฟล์โปรแกรม BankAccount.java

```
1  public class BankAccount {
2     private double balance;
3     public BankAccount() {
4        balance = 0;
5     }
6     public BankAccount(double initialBalance) {
7        balance = initialBalance;
8     }
9     public void deposit(double amount) {
10       balance = balance + amount;
11    }
12    public void withdraw(double amount) {
13       balance = balance - amount;
14    }
15    public double getBalance() {
16       return balance;
17    }
18 }
```

คำอธิบายโปรแกรม

- โปรแกรมนี้เป็นคลาสแม่แบบของคลาสบัญชีเงินฝาก ซึ่งมีตัวแปร คือ balance เพื่อเก็บยอดเงินฝาก มีตัวสร้าง 2 แบบ แบบที่ไม่มีตัวแปรพารามิเตอร์ จะกำหนดค่าข้อมูลยอดเงินฝากเป็น 0 และแบบที่มีตัวแปรพารามิเตอร์หนึ่งตัว เพื่อกำหนดให้ค่าข้อมูลยอดเงินฝากเท่ากับค่าตัวแปรพารามิเตอร์ที่ส่งเข้ามา และมีเมท็อดคือ deposit() เพื่อฝากเงิน, withdraw() เพื่อถอนเงินและ getBalance() เพื่อเรียกรับค่ายอดเงินฝาก

จากคลาส BankAccount ที่กำหนด จะขยายคลาสเป็นคลาสลูกชื่อ SavingsAccount โดยคลาส SavingsAccount จะเป็นบัญชีเงินฝากแบบที่มีดอกเบี้ย เมท็อดทุกเมท็อดจากคลาส BankAccount จะถูกถ่ายทอดมายังคลาส SavingsAccount นั่นคือในคลาส SavingsAccount สามารถเรียกใช้เมท็อด deposit(), withdraw() และ getBalance() ได้

การสร้างคลาสลูกในการรับทอด มีรูปแบบของไวยากรณ์ คือ

class *SubclassName* extends *SuperclassName* {

*instance fields* *// เพิ่มตัวแปรใหม่สำหรับคลาสลูก*

*methods* *// เพิ่มเมท็อดใหม่สำหรับคลาสลูก หรือปรับเปลี่ยนเมท็อดของคลาสแม่ได้*

}

ในการสร้างคลาสลูกนั้น คลาสลูกจะได้รับการถ่ายทอดสมาชิกจากคลาสแม่ โดยคลาสลูกสามารถกำหนดตัวสร้างตัวแปรประจำอ็อบเจกต์ และเมท็อดใหม่เพิ่มเติมได้ รวมทั้งสามารถปรับเปลี่ยนเมท็อดของคลาสแม่ได้ (redefine or override methods of superclass) นั่นคือ หัวเมท็อดจะเหมือนกับที่ปรากฏในคลาสแม่

ตัวอย่าง ไฟล์โปรแกรม SavingsAccount.java

```
1  public class SavingsAccount extends BankAccount {
2        private double interestRate;
3        public SavingsAccount(double rate) {
4              interestRate = rate;
5        }
6        public void addInterest() {
7                    double interest = getBalance() * interestRate / 100;
8                    deposit(interest);
9        }
10 }
```

คำอธิบายโปรแกรม

- คลาส SavingsAccount เป็นคลาสลูกที่ขยายต่อจากคลาส BankAccount มีการประกาศตัวแปร ประจำอ็อบเจกต์ เพิ่มหนึ่งตัวคือ อัตราดอกเบี้ยเงินฝาก (interestRate) มีตัวสร้างเพื่อกำหนดค่าตัวแปร interestRate ให้เท่ากับค่าของตัวแปรพารามิเตอร์ และมีการกำหนดเมท็อดใหม่เพิ่มเติมอีกหนึ่งเมท็อดชื่อ addInterest() เพื่อคำนวณค่าดอกเบี้ยและนำดอกเบี้ยบวกเพิ่มในยอดเงินฝาก

### 8.1.3 ตัวสร้างคลาสลูก (Subclass constructor)

สำหรับตัวสร้างของคลาสแม่ มันจะไม่ถูกถ่ายทอดไปสู่คลาสลูก แต่คลาสลูกสามารถเรียกใช้ตัวสร้างของคลาสแม่ได้ รูปแบบของการเขียนตัวสร้าง มีดังนี้

```
public ClassName(parameters) {
   super(parameters);
   . . .
}
```

เมื่อมีการสร้างอ็อบเจกต์จากคลาสลูก ตัวสร้างของคลาสลูกจะถูกประมวลผล โดยหากผู้เขียนโปรแกรมต้องการเรียกตัวสร้างของคลาสแม่ จะต้องระบุคำสั่ง super(parameters); ***เป็นบรรทัดแรก***ในตัวสร้างของคลาสลูก แต่หากผู้เขียนโปรแกรมไม่ได้ระบุคำสั่งดังกล่าว คอมไพเลอร์จะเรียกตัวสร้างปริยายหรือตัวสร้างแบบไม่มีพารามิเตอร์ของคลาสแม่มาประมวลผลให้เอง

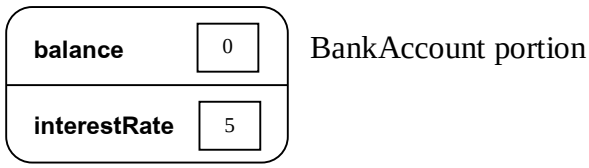
นั่นคือ เมื่อมีการสร้างอ็อบเจกต์ของคลาสลูก ลำดับการเรียกใช้ตัวสร้าง คือ เริ่มต้นที่ตัวสร้างของคลาสลูกก่อนเป็นลำดับแรก แล้วจึงเรียกใช้ตัวสร้างของคลาสแม่ ซึ่งอาจเป็นไปอย่าง implicit หรือ explicit ก็แล้วแต่ว่า ผู้เขียนโปรแกรมระบุคำสั่ง super(parameters); ไว้เองหรือไม่

ตัวอย่าง ตัวสร้างของคลาส SavingsAccount

```
public class SavingsAccount extends BankAccount {
      ...
      public SavingsAccount(double rate) {
            interestRate = rate;
      }
      ...
}
```

สำหรับคลาส SavingsAccount นี้ ภายในตัวสร้างไม่มีคำสั่งเรียกใช้ตัวสร้างของคลาสแม่ ดังนั้นเมื่อมีการสร้างอ็อบเจกต์ของคลาส SavingsAccount คอมไพเลอร์จะเรียกตัวสร้างแบบที่ไม่มีพารามิเตอร์ของคลาสแม่ ซึ่งคือ BankAccount ให้เอง นั่นคือจะทำให้ได้บัญชีเงินฝากแบบออมทรัพย์ซึ่งมียอดเงินฝากเริ่มต้นเป็น ศูนย์ และมีอัตราดอกเบี้ยตามที่ระบุในคำสั่ง new object ของคลาสลูก

การถ่ายทอดคุณสมบัติจากคลาสแม่สู่คลาสลูกนั้น คลาสลูกจะได้รับทั้งตัวแปรและเมท็อดของคลาสแม่ที่สิทธิ์ในการเข้าถึงตัวแปรไม่เป็น private มาทั้งหมด นั่นคือการเขียนโปรแกรมในคลาสลูกจะสามารถเรียกใช้หรืออ้างอิงถึงสมาชิกเหล่านั้นได้โดยไม่ต้องกำหนดสมาชิกเหล่านั้นขึ้นเอง แต่สำหรับตัวแปรของคลาสแม่ที่มีสิทธิ์ในการเข้าถึงเป็น private นั้น ถึงแม้ในคลาสลูกจะไม่สามารถเขียนโปรแกรมเพื่อเรียกใช้หรืออ้างอิงถึงได้โดยตรง แต่ตัวแปรเหล่านั้นก็ยังถือว่าเป็นส่วนประกอบของอ็อบเจกต์ของคลาสลูกด้วย ดังเช่นตัวแปร balance ในคลาส BankAccount ซึ่งเป็นคลาสแม่ได้ประกาศสิทธิ์ไว้เป็น private เมื่อขยายคลาสต่อออกมาเป็นคลาสลูก SavingsAccount นั้น ภายในคลาส SavingsAccount ไม่สามารถเรียกใช้ตัวแปร balance ได้โดยตรง แต่ก็ยังถือว่าตัวแปร balance เป็นส่วนหนึ่งของอ็อบเจกต์จากคลาส SavingsAccount ด้วย โดยจะสามารถเรียกค่าหรือปรับปรุงค่าของยอดเงินฝากผ่าน public method ต่าง ๆ ของ supercalss ได้ รูปที่ 8.2 จำลองภาพในหน่วยความจำของ อ็อบเจกต์จากคลาส SavingsAccount




รูปที่ 8.2 อ็อบเจกต์จากคลาส SavingsAccount

### 8.1.4 การรับทอด และสมาชิกภายในคลาส (Inheritance and fields / methods)

เมื่อมีการสร้างคลาสลูก เราอาจเพิ่มสมาชิกใหม่ หรือปรับเปลี่ยนสมาชิกบางตัวได้ โดยสำหรับตัวแปรประจำอ็อบเจกต์นั้น สามารถสรุปได้ดังนี้

1. inherit field คือ ถ่ายทอดตัวแปรทุกตัวที่ไม่เป็น private จากคลาสแม่มาสู่คลาสลูก ในคลาสลูกสามารถเรียกใช้หรืออ้างถึงชื่อตัวแปรนั้นได้โดยตรง โดยไม่ต้องกำหนดตัวแปรนั้นในคลาสลูก
2. add field คือ สามารถเพิ่มตัวแปรใหม่ได้
3. ไม่สามารถ override field ได้ นั่นคือ หากในคลาสลูกมีการประกาศตัวแปรที่มีชื่อซ้ำกับตัวแปรที่เคยประกาศในคลาสแม่ จะถือว่าตัวแปรที่ประกาศในคลาสลูกเป็นตัวใหม่ซึ่งเป็นคนละตัวกับตัวแปรที่ประกาศในคลาสแม่ การประกาศตัวแปรในคลาสลูกให้มีชื่อซ้ำกับตัวแปรในคลาสแม่นี้เรียกว่า shadowing ตัวที่เห็นและถูกเรียกใช้ในคลาสลูกก็คือตัวที่ประกาศใหม่ในคลาสลูกนั่นเอง

และสำหรับเมท็อดประจำอ็อบเจกต์ สามารถสรุปได้ดังนี้

1. inherit method คือ ถ่ายทอดเมท็อดทุกตัวที่ไม่เป็น private จากคลาสแม่มาสู่คลาสลูก ในคลาสลูกสามารถเรียกใช้เมท็อดเหล่านั้นได้โดยตรง โดยไม่ต้องเขียนเมท็อดเหล่านั้นซ้ำภายในคลาสลูก
2. add method คือ สามารถเพิ่มเมท็อดใหม่ได้ เพื่อให้ทำงานตามวัตถุประสงค์ของคลาสลูก
3. override method คือ สามารถปรับเปลี่ยนการทำงานของเมท็อดที่มีในคลาสแม่ได้ คือชื่อเมท็อดเหมือนกันแต่การทำงานในรายละเอียดอาจแตกต่างจากที่เคยทำในคลาสแม่ โดยในการเขียนโปรแกรม ผู้เขียนโปรแกรมจะเขียนประกาศเมท็อดไว้ภายในคลาสลูกโดยให้มีหัวเมท็อดเหมือนกับหัวเมท็อดที่ประกาศในคลาสแม่ทุกประการ เรียกว่า เมท็อดทั้งสองจะต้องมี same signature นั่นคือ ชื่อเมท็อด จำนวนและชนิดของตัวแปรพารามิเตอร์ ทั้งในคลาสแม่และคลาสลูกจะต้องเหมือนกัน แล้วปรับเปลี่ยนคำสั่งในเมท็อดให้ตรงตามวัตถุประสงค์ของคลาสลูก

สำหรับคลาส SavingsAccount สามารถสรุปตัวแปรและเมท็อดได้ดังนี้

- ตัวแปร – interestRate จัดเป็นตัวแปรที่เพิ่มใหม่
- เมท็อด – getBalance(), deposit(), withdraw() จัดเป็นเมท็อดที่ inherit มาจากคลาสแม่ และเมท็อด addInterest() เป็นเมท็อดที่เพิ่มใหม่

### 8.1.5 การ override เมท็อดของคลาสแม่ (Overriding base class method)

จากคลาส BankAccount และ SavingsAccount จะสร้างคลาสลูกของ BankAccount เพิ่มอีกหนึ่งคลาสคือ CheckingAccount โดยมีข้อกำหนดดังนี้คือ

- ไม่คิดค่าธรรมเนียมของการทำรายการ 3 รายการแรก
- คิดค่าธรรมเนียมของการทำรายการครั้งต่อไป ครั้งละ $2
- ต้อง override เมท็อด deposit() และ withdraw() เพื่อเพิ่มค่าตัวนับจำนวนรายการที่ทำ
- เพิ่มเมท็อดใหม่ชื่อ deductFees เพื่อหักค่าธรรมเนียมในการทำรายการจากยอดเงินฝาก และเปลี่ยนค่าตัวนับจำนวนรายการที่ทำไปแล้วเป็น 0

ตัวอย่าง ไฟล์โปรแกรม CheckingAccount.java

```
1   public class CheckingAccount extends BankAccount {
2      private int transactionCount;
3      private static final int FREE_TRANSACTIONS = 3;
4      private static final double TRANSACTION_FEE = 2.0;
5      public CheckingAccount(int initialBalance)   {
6         // construct superclass
7         super(initialBalance);
8         transactionCount = 0;
9      }
10     public void deposit(double amount)     {
11        transactionCount++;
12        // now add amount to balance
13        super.deposit(amount);
14     }
15
16     public void withdraw(double amount)     {
17        transactionCount++;
```

```
18          // now subtract amount from balance
19          super.withdraw(amount);
20       }
21       /*Deducts the accumulated fees and resets the transaction count.*/
22       public void deductFees()   {
23          if (transactionCount > FREE_TRANSACTIONS)  {
24             double fees = TRANSACTION_FEE *
                  (transactionCount - FREE_TRANSACTIONS);
25             super.withdraw(fees);
26          }
27          transactionCount = 0;
28       }
29    }
```

- คลาส CheckingAccount ขยายต่อจากคลาส BankAccount โดยมีการประกาศตัวแปรประจำ อ็อบเจกต์เพิ่มคือ transactionCount และตัวแปรประจำคลาสที่เป็นค่าคงที่ (static final variable) เพิ่มคือ FREE_TRANSACTIONS และ TRANSACTION_FEE
- บรรทัดที่ 5 ถึง 9 คือ ตัวสร้าง โดยมีการเรียกตัวสร้างของคลาสแม่เพื่อกำหนดค่ายอดเงินฝากเริ่มต้น และกำหนดค่าเริ่มต้นให้กับตัวแปร transactionCount ให้เป็น 0
- บรรทัดที่ 10 ถึง 20 เป็นการ **override เมท็อด deposit() และ withdraw()** โดยภายในเมท็อดมีคำสั่งเพื่อเพิ่มค่า transactionCount ขึ้นอีก 1 เมื่อมีการทำรายการฝากและถอน แล้วจึงทำการฝากและถอนตามปกติ สำหรับคำสั่งเพื่อฝากและถอนเงินนั้น จะใช้ **super.deposit() และ super.withdraw() คือ ใช้ super reference** โดยจะต้องระบุคำว่า super ไว้อย่างชัดเจน เพื่อเรียกใช้เมท็อดจากคลาสแม่ เนื่องจากหากเรียกเมท็อดโดยไม่ได้ระบุ super นั่นคือเขียนคำสั่งเป็น deposit() หรือ withdraw() ซึ่งเปรียบเหมือนกับการเขียนคำสั่งด้วย this.deposit() หรือ this.withdraw() จะเป็นการเรียกเมท็อดตัวเองซ้ำแบบ recursive สำหรับในการฝากและถอนนั้น จำเป็นจะต้องเรียกใช้เมท็อดจากคลาสแม่ เนื่องจากตัวแปร balance ถูกประกาศเป็น private ในคลาสแม่ ดังนั้น ภายในคลาสลูกจึงไม่สามารถเขียนคำสั่งเพื่อปรับยอดเงินฝากผ่านตัวแปร balance ได้โดยตรง จะต้องกระทำผ่านเมท็อด deposit() และ withdraw() ของคลาสแม่เท่านั้น
- ตั้งแต่บรรทัดที่ 22 เป็นต้นไป เป็นการกำหนดเมท็อดใหม่เพิ่มคือ deductFees() ซึ่งจะตรวจสอบว่า transactionCount เกินจำนวน FREE_TRANSACTION หรือไม่ หากเกินจะหักค่าธรรมเนียมในการทำรายการ โดยคำนวณค่าธรรมเนียมแล้วนำไปหักออกจากยอดเงินฝากด้วยการเรียกใช้ super.withdraw() หากไม่เกิน จะไม่หักค่าธรรมเนียม และตั้งค่า transactionCount ให้เป็น 0

สำหรับคลาส CheckingAccount สามารถสรุปตัวแปรและเมท็อดได้ดังนี้

- ตัวแปร – transactionCount จัดเป็นตัวแปรที่เพิ่มใหม่
- เมท็อด – getBalance() เป็นเมท็อดที่ inherit มาจากคลาสแม่ เมท็อด deposit() และ withdraw() เป็นการ override method ของคลาสแม่ และเมท็อด deductFees() เป็นเมท็อดที่เพิ่มใหม่

รูปแบบของคำสั่งในการเรียกใช้เมท็อดของคลาสแม่คือ

super.*methodName*(*parameters*)

เช่น `super.deposit(amount);`

เมื่อสร้างคลาส BankAccount, SavingsAccount และ CheckingAccount แล้ว จะสร้างคลาส AccountTest เพื่อทดสอบคลาสต่าง ๆ ที่กำหนดขึ้น ดังนี้

ตัวอย่าง ไฟล์โปรแกรม AccountTest.java

```
1  public class AccountTest {
2     public static void main(String[] args) {
3        SavingsAccount mySavings = new SavingsAccount(5);
4        CheckingAccount BoomChecking = new CheckingAccount(10000);
5        mySavings.deposit(10000);
6        BoomChecking.withdraw(1500);
7        BoomChecking.deposit(1500);
8        BoomChecking.withdraw(1500);
9        BoomChecking.withdraw(400);
10       // simulate end of month
11       mySavings.addInterest();
12       BoomChecking.deductFees();
13       System.out.println("My savings balance = $" +
                                mySavings.getBalance());
14       System.out.println("Boom's checking balance = $" +
                                BoomChecking.getBalance());
15    }
16 }
```

คำอธิบายโปรแกรม

- โปรแกรมเพื่อทดสอบคลาสบัญชีเงินฝากประเภทต่าง ๆ ที่สร้างขึ้นนี้ จะทดลองสร้างอ็อบเจกต์จากคลาส SavingsAccount และ CheckingAccount ขึ้นอย่างละ 1 อ็อบเจกต์ โดยมีตัวแปรอ้างอิงถึงอ็อบเจกต์ของ SavingsAccount ชื่อ mySavings ยอดเงินฝากเริ่มต้นเป็น 0 และอัตราดอกเบี้ย คือ 5% และตัวแปรอ้างอิงถึงอ็อบเจกต์ของ CheckingAccount ชื่อ BoomChecking มียอดเงินฝากเริ่มต้นเป็น 10,000
- จากนั้น ทดลองเรียกใช้เมท็อดต่าง ๆ ของทั้งสองอ็อบเจกต์ และสุดท้ายเรียกใช้เมท็อด getBalance() เพื่อแสดงค่ายอดเงินฝากคงเหลือของบัญชีเป็นผลลัพธ์

ผลลัพธ์ที่ได้จากโปรแกรมคือ

My savings balance = $10500.0

Boom's checking balance = $8098.0

### 8.1.6 การประกาศตัวแปรประจำอ็อบเจกต์ให้มีชื่อซ้ำกับที่ประกาศในคลาสแม่ (Shadowing instance variable)

จากตัวอย่างของคลาส CheckingAccount ซึ่งมีการ override เมท็อด deposit() และ withdraw() และภายในเมท็อดมีการเรียกใช้ super.deposit() และ super.withdraw() หากผู้เขียนโปรแกรมไม่เข้าใจในหลักการของการรับทอด อาจคิดว่าหากประกาศตัวแปร balance ไว้ภายในคลาส CheckingAccount เอง จะสามารถเขียนคำสั่งเพื่อจัดการกับยอดเงินฝาก balance ได้เอง โดยคิดว่าตัวแปร balance ที่กำหนดขึ้นใหม่ก็คือตัวแปรเดียวกันกับตัวแปร balance ที่ประกาศในคลาสแม่ ดังตัวอย่างต่อไปนี้

ตัวอย่าง ไฟล์โปรแกรม CheckingAccount.java แบบที่ใช้ shadowing instance variable (balance)

```
1  public class CheckingAccount extends BankAccount  {
2     private int transactionCount;
3     private int balance;     // shadowing instance variable
4     private static final int FREE_TRANSACTIONS = 3;
5     private static final double TRANSACTION_FEE = 2.0;
6     public CheckingAccount (int initialBalance) {
7        //construct superclass
8        super (initialBalance);
9        transactionCount = 0;
10    }
11    public void deposit (double amount) {
12       transactionCount++;
13       //now add amount to balance
14       balance += amount;
15    }
16    public void withdraw(double amount)   {
17       transactionCount++;
18       // now subtract amount from balance
19       balance -= amount;
20    }
21    /*Deducts the accumulated fees and resets the transaction count.*/
22    public void deductFees()   {
23       if (transactionCount > FREE_TRANSACTIONS)  {
24          double fees = TRANSACTION_FEE *
                          (transactionCount - FREE_TRANSACTIONS);
25          balance -= fees;
26       }
27       transactionCount = 0;
28    }
29 }
```

คำอธิบายโปรแกรม

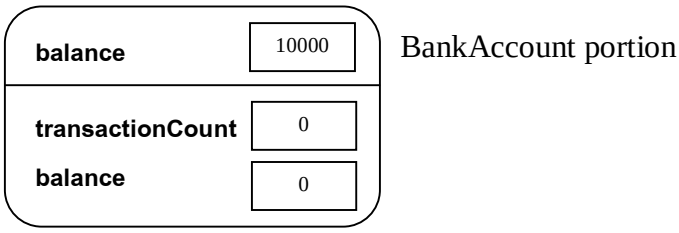
- คลาสนี้ ประกาศตัวแปร balance ขึ้นใหม่ ซึ่งจัดเป็นการทำ shadowing นั่นคือ ตัวแปร balance ที่กำหนดขึ้นใหม่นี้ จะเป็นคนละตัวกับ balance ของคลาส BankAccount ที่เมื่อสร้างอ็อบเจกต์แล้วbalance ของ BankAccount จะเป็นส่วนหนึ่งในอ็อบเจกต์ของ CheckingAccount อยู่แล้ว
- บรรทัดที่ 6 ถึง 10 คือ ตัวสร้าง ซึ่งมีการเรียกตัวสร้างของคลาสแม่เพื่อกำหนดค่าตัวแปร balance ที่เป็นส่วนหนึ่งของอ็อบเจกต์ให้มีค่าเริ่มต้นเป็น 10,000 จากนั้นกำหนดค่าให้ transactionCount เป็น 0 และสำหรับตัวแปร balance ที่เพิ่งประกาศขึ้นใหม่ในคลาส CheckingAccount นี้ จะมีค่าเริ่มต้นเป็นค่าปริยาย คือมีค่าเป็น 0
- เมท็อดทั้งหมดภายในคลาส ได้ปรับเปลี่ยนให้มีการเขียนคำสั่งเพื่อจัดการกับตัวแปร balance ที่ประกาศขึ้นใหม่เอง โดยไม่ได้เรียกใช้ super.deposit() หรือ super.withdraw() ซึ่งการทำเช่นนี้จะเป็นการกระทำกับตัวแปร balance ซึ่งกำหนดขึ้นใหม่ในคลาส CheckingAccount นี้ จะไม่ใช่การกระทำกับตัวแปร balance ของ BankAccount ซึ่งทำให้การทำงานของโปรแกรมให้ผลลัพธ์ที่ผิดพลาด

หากกำหนดให้ใช้โปรแกรม AccountTest ตัวเดิมที่เคยทดสอบคลาสทั้งสามก่อนหน้านี้ มาทดสอบกับคลาส CheckingAccount ที่เพิ่งแก้ไขนี้ จะได้ผลลัพธ์ดังนี้

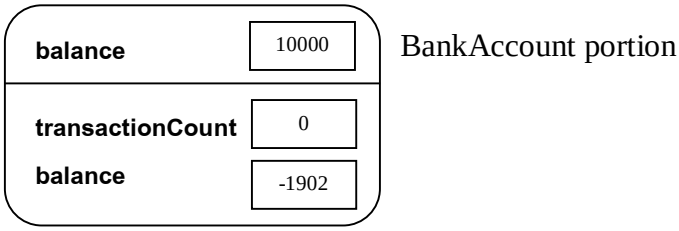
My savings balance = $10500.0

Boom's checking balance = $10000.0

ซึ่งจะเห็นว่าค่าผลลัพธ์ที่แสดงไม่ถูกต้อง เนื่องจากการทำรายการต่าง ๆ นั้น โปรแกรมจะปรับปรุงค่าให้กับตัวแปร balance ที่เพิ่งประกาศใน CheckingAccount แต่คำสั่งแสดงผลลัพธ์นั้นเรียกใช้เมท็อด getBalance() ซึ่งเป็นการเรียกค่าตัวแปร balance ของ BankAccount ซึ่งยังไม่เคยได้รับการปรับปรุงค่าใด ๆ หลังจากที่กำหนดค่าเริ่มต้นให้เป็น 10,000 เมื่อตอนสร้างอ็อบเจกต์ ภาพจำลองอ็อบเจกต์ในหน่วยความจำเมื่อ new CheckingAccount object เป็นดังรูปที่ 8.3 และภาพจำลองอ็อบเจกต์ของ CheckingAccount ในหน่วยความจำเมื่อจบโปรแกรม AccountTest เป็นดังรูปที่ 8.4




รูปที่ 8.3 อ็อบเจกต์จากคลาส CheckingAccount เมื่อ new object




รูปที่ 8.4 อ็อบเจกต์จากคลาส CheckingAccount ก่อนจบโปรแกรม AccountTest

**8.1.7 การประกาศตัวแปรประจำอ็อบเจกต์ให้มีสิทธิ์การเข้าถึงแบบ protected (Protected instance variable)**

จากตัวอย่างของคลาส CheckingAccount ก่อนหน้านี้ที่ผู้เขียนโปรแกรมใช้ shadowing instance variable การทำงานของโปรแกรมนั้นจะเห็นเพียงตัวแปร balance ตัวล่าสุดที่ประกาศในคลาส CheckingAccount หากผู้เขียนโปรแกรมต้องการ override เมท็อด deposit() และ withdraw() โดยภายในเมท็อดจะไม่เรียกใช้ super.deposit() และ super.withdraw() แต่ต้องการจัดการกับตัวแปร balance เอง ผู้เขียนโปรแกรมสามารถทำได้ โดยการประกาศตัวแปร balance ในคลาส BankAccount ให้มีสิทธิ์การเข้าถึงเป็น protected

จะขอทบทวนเรื่องสิทธิ์การเข้าถึงตัวแปร ดังนี้

- public ไม่ว่าคลาสใดก็สามารถเรียกใช้ตัวแปรนั้นได้
- private ตัวแปรจะถูกใช้ได้ภายในคลาสที่ประกาศตัวแปรนั้นเท่านั้น
- protected คลาสลูกและคลาสอื่นภายในแพ็กเกจเดียวกันสามารถเรียกใช้ตัวแปรได้
- package access (default, no modifier) คลาสอื่นภายในแพ็กเกจเดียวกันสามารถเรียกใช้ตัวแปรได้

สามารถสรุปเรื่องสิทธิ์การเข้าถึงกับการรับทอดได้ดังนี้

- ตัวแปรและเมท็อดที่ถ่ายทอดจากคลาสแม่สู่คลาสลูก คือตัวแปรและเมท็อดที่มีสิทธิ์การเข้าถึงเป็น public, protected และ package acess นั่นคือในคลาสลูกสามารถเขียนโปรแกรมเพื่อเรียกใช้ตัวแปรและเมท็อดเหล่านั้นได้โดยไม่ต้องมีการประกาศตัวแปรหรือกำหนดเมท็อดนั้นซ้ำภายในคลาสลูกอีกแล้ว

- สำหรับตัวแปรที่ประกาศเป็น private ในคลาสแม่นั้น มันจะเป็นส่วนประกอบหนึ่งของคลาสลูก เพียงแต่คลาสลูกจะไม่สามารถเรียกใช้มันได้โดยตรง การจัดการกับตัวแปรที่เป็น private นั้น จะต้องผ่าน public method ที่คลาสแม่ได้จัดเตรียมไว้ให้

จากตัวอย่างของคลาส BankAccount, SavingsAccount, CheckingAccount และ AccountTest จะแสดงการใช้ protected instance variable โดยปรับเปลี่ยนคลาส BankAccount และ CheckingAccount ดังนี้

ตัวอย่าง ไฟล์โปรแกรม BankAccount.java แบบใช้ protected instance variable (balance)

```
1  public class BankAccount {
2     protected double balance;     // using protected instance field
3     public BankAccount() {
4        balance = 0;
5     }
6        ...
7  }
```

คำอธิบายโปรแกรม

- คลาส BankAccount นี้ ปรับปรุงใหม่ โดยกำหนดตัวแปร balance ให้มีสิทธิ์การเข้าถึงเป็น protected ส่วนอื่นของโปรแกรมยังเป็นไปตามเดิม

ตัวอย่าง ไฟล์โปรแกรม CheckingAccountg.java

```
1  public class CheckingAccount extends BankAccount  {
2     private int transactionCount;
3     private static final int FREE_TRANSACTIONS = 3;
4     private static final double TRANSACTION_FEE = 2.0;
5     public CheckingAccount (int initialBalance) {
6     //construct superclass
7        super (initialBalance);
8        transactionCount = 0;
9     }
10    public void deposit (double amount) {
11       transactionCount++;
12       //now add amount to balance
13       balance += amount;
14    }
15    public void withdraw (double amount)   {
16       transactionCount++;
17       //now subtract amount from balance
18       balance -= amount;
19    }
20    /*Deducts the accumulated fees and resets the transaction count.*/
21    public void deductFees()   {
22       if (transactionCount > FREE_TRANSACTIONS)  {
23          double fees = TRANSACTION_FEE *
24                        (transactionCount - FREE_TRANSACTIONS);
25          balance -= fees;
26       }
27       transactionCount = 0;
28    }
29 }
```

คำอธิบายโปรแกรม

- คลาส CheckingAccount นี้ ไม่ได้มีการประกาศตัวแปร balance เป็น shadowing instance variable
- เมท็อดทั้งหมดภายในคลาสไม่ได้ใช้คำสั่ง super.deposit() หรือ super.withdraw() แต่เขียนโดยการปรับปรุงค่ายอดเงินฝากผ่านตัวแปร balance โดยตรง ซึ่งทำได้ และตัวแปรนี้ก็เป็นตัวแปรเดียวกันกับ balance ของคลาส BankAccount เนื่องจากในคลาสแม่ประกาศให้ balance เป็น protected ดังนั้น คลาสลูกก็จะมองเห็นตัวแปรนี้และสามารถเขียนโปรแกรมจัดการกับตัวแปรนี้ได้

เมื่อลองทดสอบโปรแกรมด้วยคลาส AccountTest ตัวเดิม ผลลัพธ์ที่ได้คือ

```
My savings balance = $10500.0
Boom's checking balance = $8098.0
```

ซึ่งจะเห็นว่าได้ผลลัพธ์ถูกต้อง เนื่องจากตัวแปร balance ที่ใช้ในเมท็อด deposit(), withdraw() และ deductFees() คือ balance ตัวเดียวกันกับ balance ในคลาส BankAccount

ข้อดีของการใช้ตัวแปรแบบ protected

- คลาสลูกสามารถแก้ไขค่าข้อมูลได้โดยตรง
- ช่วยเพิ่มความเร็วในการประมวลผลเล็กน้อย เนื่องจากสามารถเรียกใช้ตัวแปรได้โดยตรง ไม่ต้องผ่านการเรียกใช้ public method ที่สร้างไว้สำหรับเรียกค่าหรือปรับปรุงค่าของตัวแปร ทำให้ไม่ต้องเสีย overhead ในการทำ function call

ข้อด้อยของการใช้ตัวแปรแบบ protected

- คลาสลูกสามารถปรับเปลี่ยนค่าของตัวแปรดังกล่าวได้เอง โดยอาจกำหนดค่าที่คลาสแม่ไม่ต้องการ เช่น ถ้าคลาสแม่กำหนดว่าค่าของยอดเงินฝากห้ามติดลบ มีการเขียนเงื่อนไขตรวจสอบไว้เรียบร้อยในคลาสแม่ แต่เมื่อมีการขยายต่อไปยังคลาสลูก ผู้เขียนโปรแกรมอาจลืมเขียนเงื่อนไขเพื่อตรวจสอบค่าข้อมูลดังกล่าว ก็จะสามารถกำหนดค่าตัวแปรดังกล่าวเป็นติดลบได้ ซึ่งทำให้ความหมายของเนื้องานผิดพลาดไป
- ทำให้การเขียนโปรแกรมไม่เป็นอิสระ และจัดเป็นการพัฒนาซอฟต์แวร์แบบ fragile softwareเช่น หากคลาสแม่แก้ไขชื่อตัวแปร คลาสลูกก็ต้องแก้ไขโปรแกรมตามด้วย ไม่เช่นนั้นจะไม่สามารถประมวลผลได้

#### 8.1.8 Encapsulation or information hiding

การเขียนโปรแกรมให้มีการรับทอดจากคลาสแม่สู่คลาสลูกนั้น มีข้อแนะนำสำหรับผู้เขียนโปรแกรมคือ ควรยึดหลักของ encapsulation หรือ information hiding นั่นคือ จะไม่เปิดโอกาสให้ผู้อื่นมาปรับปรุงแก้ไขข้อมูลที่สำคัญ หากผู้อื่นต้องการจัดการกับข้อมูลจะต้องเป็นไปตามที่เจ้าของโปรแกรมอนุญาตเท่านั้น นั่นคือ จะกำหนดตัวแปรประจำอ็อบเจกต์เป็นแบบ private ทำให้คลาสอื่นไม่สามารถมองเห็นและเรียกใช้ตัวแปรได้ แต่ถ้าผู้เขียนโปรแกรมอนุญาตให้ผู้อื่นจัดการกับข้อมูลของตัวแปรนั้นได้ ก็จะสร้าง public method เพื่อให้คลาสอื่นสามารถเรียกใช้งานเมท็อดเหล่านั้นสำหรับการจัดการกับข้อมูลได้ ซึ่งถือว่าเป็นการอนุญาตให้คลาสอื่นจัดการกับข้อมูลโดยผ่านการควบคุมของผู้เขียนโปรแกรมเท่านั้น

ตัวแปรค่าคงที่แบบ static มักนิยมให้เป็น public เพื่อเปิดโอกาสให้คลาสอื่นเรียกใช้ได้ ส่วนเมท็อดในคลาสนั้นอาจกำหนดให้เป็น public หรือ private ก็แล้วแต่จุดประสงค์ในการทำงาน สำหรับคลาส โดยทั่วไปก็จะกำหนดให้เป็น public หรือ package access

### 8.2 การ override เมท็อด toString()

คลาส Object เป็นคลาสที่จาวากำหนดขึ้น โดยอยู่ในแพ็กเกจ ชื่อ java.lang คลาสอื่น ๆ ที่ผู้เขียนโปรแกรมสร้างขึ้นมาในภายหลัง ล้วนแต่เป็นคลาสลูกของคลาส Object ทั้งสิ้น แต่ในการสร้างคลาสที่ผ่านมา เราไม่จำเป็นต้องเขียนที่หัวคลาสว่า extends Object อย่างชัดเจน คอมไพเลอร์ก็เข้าใจว่าเป็นการขยายต่อจากคลาส Object หากผู้เขียนโปรแกรมระบุไว้อย่างชัดเจนก็ไม่ผิด ในคลาส Object นี้ จาวาได้กำหนดเมท็อดต่าง ๆ ไว้ให้ใช้งาน โดยในที่นี้จะขอกล่าวถึงเฉพาะเมท็อด toString() และ equals()

เมท็อด toString() เป็นเมท็อดที่ส่งค่าสายอักขระ ซึ่งแสดงข้อมูลเกี่ยวกับอ็อบเจกต์คืนกลับมาให้ผู้เรียกใช้เมท็อด

- Object.toString() จะให้ค่าคืนออกจากเมท็อดเป็นชื่อคลาสและที่อยู่ในหน่วยความจำของอ็อบเจกต์ เช่น myAccount.toString() จะให้ค่าคืนออกจากเมท็อดเป็น
  BankAccount@d2460bf
- Rectangle.toString() จะให้ค่าคืนออกจากเมท็อดเป็น
  java.awt.Rectangle[x=5,y=10,width=20,height=30]

จากจุดมุ่งหมายของเมท็อด toString() ที่มีไว้เพื่อแสดงข้อมูลเกี่ยวกับอ็อบเจกต์ โดยในระดับชั้น Object นั้น จะให้ค่าคืนมาเป็นชื่อคลาสและที่อยู่ในหน่วยความจำ ซึ่งจะเห็นว่า ไม่เป็นประโยชน์ในระดับการใช้งานทั่วไป ดังนั้น ผู้เขียนโปรแกรมจึงควร override เมท็อด toString() เพื่อให้แสดงค่าข้อมูลของอ็อบเจกต์ตามต้องการ ซึ่งโดยปกติจะเขียนให้มันแสดงชื่อคลาสและค่าของตัวแปร instance variable ทั้งหมด ดังเช่นตัวอย่างของคลาส Rectangle ข้างต้น การ override toString() ในคลาส BankAccount ทำได้ดังนี้

```
public class BankAccount {
   public String toString() {
     return "BankAccount[balance=" + balance + "]";
   }
   . . .
}
```

การ override toString() ในคลาส SavingsAccount ทำได้ดังนี้

```
public class SavingsAccount {
   public String toString() {
      return super.toString() + "\n[interestRate = " +
            interestRate + "]";
   }
   . . .
}
```

การ override toString() ในคลาส CheckingAccount ทำได้ดังนี้

```
public class CheckingAccount {
   public String toString() {
      return super.toString() + "\n[transactionCount = " +
                  transactionCount + "]";
   }
   . . .
}
```

ในคลาส AccountTest ให้เพิ่มคำสั่งเพื่อเรียกใช้เมท็อด toString() ดังตัวอย่างต่อไปนี้

ตัวอย่าง ไฟล์โปรแกรม AccountTest.java

```
1  public class AccountTest {
2     public static void main(String[] args) {
3        SavingsAccount mySavings = new SavingsAccount(5);
4        CheckingAccount BoomChecking = new CheckingAccount(10000);
5        mySavings.deposit(10000);
6        BoomChecking.withdraw(1500);
7        BoomChecking.deposit(1500);
8        BoomChecking.withdraw(1500);
9        BoomChecking.withdraw(400);
10       System.out.println(mySavings.toString());   // or mySavings
11       System.out.println();
12       System.out.println(BoomChecking.toString());   // or BoomChecking
13       System.out.println();
14       // simulate end of month
15       mySavings.addInterest();
16       BoomChecking.deductFees();
17       System.out.println("My savings balance = $" +
                            mySavings.getBalance());
18       System.out.println("Boom's checking balance = $" +
                            BoomChecking.getBalance());
19    }
20 }
```

คำอธิบายโปรแกรม

- คลาส AccountTest นี้เพิ่มบรรทัด 10-13 เพื่อให้แสดงผลลัพธ์เป็นค่าที่ได้จากเมท็อด toString() โดยในการเรียกใช้เมท็อดนี้ ทำได้สองทาง คือ เรียกโดยตรงผ่านชื่ออ็อบเจกต์.ชื่อเมท็อด เช่น
  `System.out.println(mySavings.toString());`
  หรือผ่านชื่ออ็อบเจกต์เท่านั้น เช่น
  `System.out.println(mySavings);`

ผลลัพธ์ที่ได้จากโปรแกรมคือ

```
BankAccount[balance = 10000.0]
[interestRate = 5.0]

BankAccount[balance = 8100.0]
[transactionCount = 4]

My savings balance = $10500.0
Boom's checking balance = $8098.0
```

จากผลลัพธ์ที่ได้ จะเห็นว่าชื่อคลาสที่แสดงนั้นเป็น BankAccount ทั้งคู่ ทั้ง ๆ ที่ก้อนอ็อบเจกต์จริงคือ SavingsAccount และ CheckingAccount ที่เป็นเช่นนี้เนื่องจากเมท็อด toString() ของคลาส SavingsAccount และ CheckingAccount ที่ override นั้น มีการใช้คำสั่ง super.toString() ซึ่งมันจะไปเรียกเมท็อด toString() ของคลาสแม่ โดยใน

คลาสแม่ได้สั่งให้ส่งค่าคืนเป็นคำว่า BankAccount ดังนั้น เพื่อให้ได้ข้อมูลที่ถูกต้องตรงตามความเป็นจริงที่สุด จึงควรปรับปรุงคำสั่งในเมท็อด toString() ของคลาส BankAccount โดยให้เรียกใช้เมท็อดเพื่อเอาชื่อคลาสออกมาแทนที่การพิมพ์ข้อความว่า BankAccount ลงไปในโปรแกรม ดังนี้

- getClass() คือเมท็อดในคลาส Object ซึ่งจะทำให้ได้คลาสของอ็อบเจกต์ออกมา
- getName()คือเมท็อดในคลาส Class ซึ่งจะทำให้ได้ชื่อของคลาสออกมา

ปรับปรุงโปรแกรมในคลาส BankAccount อีกครั้งเพื่อให้เมท็อด toString() เรียกใช้เมท็อดแทนการพิมพ์ข้อความดังนี้

```
public class BankAccount {
   public String toString() {
     return getClass().getName() + "[balance=" + balance + "]";
   }
   . . .
}
```

เมื่อลองรันโปรแกรม AccountTest อีกครั้ง จะได้ผลลัพธ์ซึ่งตรงตามความเป็นจริงดังนี้

```
SavingsAccount[balance = 10000.0]
[interestRate = 5.0]

CheckingAccount[balance = 8100.0]
[transactionCount = 4]

Mom's savings balance = $10500.0
Harry's checking balance = $8098.0
```

## 8.3 Casting objects

การสร้างอ็อบเจกต์ขึ้นมาแต่ละอ็อบเจกต์นั้น จะมีตัวแปรอ้างอิงเพื่ออ้างอิงถึงก้อนอ็อบเจกต์ โดยในการประกาศตัวแปรอ้างอิงนี้ เราสามารถประกาศตัวแปรอ้างอิงให้มีชนิดเป็นระดับคลาสแม่ เพื่ออ้างอิงไปยังก้อนอ็อบเจกต์ที่สร้างจากคลาสแม่ หรือประกาศตัวแปรอ้างอิงให้มีชนิดเป็นระดับคลาสลูก เพื่ออ้างอิงไปยังก้อนอ็อบเจกต์ที่สร้างจากคลาสลูกนั้นได้ ซึ่งการกระทำทั้งสองแบบที่กล่าวนี้ถือว่าเป็นการกระทำที่ตรงไปตรงมา ดังตัวอย่าง

```
BankAccount anAccount = new BankAccount(500);
BankAccount one = anAccount;
SavingsAccount saving = new SavingsAccount(5);
SavingsAccount two = saving;
```

### 8.3.1 Up-casting

ในทำนองเดียวกันกับการเปลี่ยนชนิดข้อมูล (type casting) ที่ได้ศึกษาไปแล้ว เช่น

double x = 19.988;

int y = (int) x;

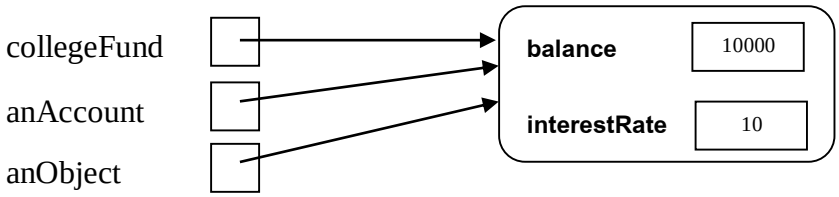
ทำให้ y มีค่าเป็น 19

เราสามารถเปลี่ยนชนิดตัวแปรอ้างอิงของอ็อบเจกต์จากคลาสชนิดหนึ่งไปเป็นคลาสชนิดอื่นได้ ภายใต้เงื่อนไขว่าชนิดตัวแปรอ้างอิงของอ็อบเจกต์กับชนิดของคลาสใหม่ที่เรากำหนดให้นั้นจะต้องอยู่ในลำดับชั้นความสัมพันธ์ตระกูล

เดียวกัน (same hierarchy) โดยคลาสหนึ่งเป็นแม่และอีกคลาสหนึ่งเป็นลูก เราสามารถเปลี่ยนชนิดตัวแปรอ้างอิงถึงอ็อบเจกต์จากคลาสชนิดหนึ่งไปเป็นคลาสอีกชนิดหนึ่งที่อยู่ในระดับชั้นที่สูงกว่าได้ เรียกว่าเป็นการทำ up-casting โดยในการกำหนดให้ตัวแปรอ้างอิงถึงอ็อบเจกต์ไปเป็นตัวแปรในชนิดที่เป็นคลาสระดับที่สูงกว่านั้น ไม่จำเป็นต้องใส่วงเล็บเพื่อทำการ casting ตัวอย่างเช่น

```
SavingsAccount collegeFund = new SavingsAccount(10);
BankAccount anAccount = collegeFund;
Object anObject = collegeFund;
```

เป็นการกำหนดตัวแปรอ้างอิงระดับคลาสแม่อย่าง BankAccount และ Object ให้อ้างอิงไปยังก้อน อ็อบเจกต์จากคลาส SavingsAccount ร่วมกัน ภาพแสดงตัวแปรอ้างอิงและก้อนอ็อบเจกต์จากคลาส SavingsAccount ภายในหน่วยความจำเป็นดังรูปที่ 8.5




รูปที่ 8.5 การใช้ตัวแปรอ้างอิงชนิดต่างกันอ้างไปยังอ็อบเจกต์ก้อนเดียวกันจากคลาส SavingsAccount

จากคำอธิบายข้างต้น สามารถพูดอีกแบบได้ว่า เป็นการกำหนด (assign) ค่าของตัวแปรอ้างอิงระดับคลาสลูกให้กับตัวแปรอ้างอิงชนิดคลาสแม่ คือทำให้ตัวแปรอ้างอิงทั้งสองเก็บค่าที่อยู่ในหน่วยความจำค่าเดียวกัน การที่เราสามารถกำหนดค่าให้เช่นนี้ได้ก็เนื่องจากคลาสเหล่านี้มีความสัมพันธ์กันแบบ IS-A relationship

เมื่อเราใช้ตัวแปรอ้างอิงระดับคลาสแม่อ้างอิงไปยังก้อนอ็อบเจกต์ระดับคลาสลูกแล้ว หากเราเรียกใช้เมท็อดของคลาสลูกผ่านชื่อตัวแปรอ้างอิงระดับคลาสแม่ จะเกิดข้อผิดพลาดในขณะคอมไพล์ เนื่องจากเมท็อดของคลาสลูกไม่ใช่เมท็อดของคลาสแม่ เช่น การเรียกเมท็อด

```
anAccount.addInterest(); // เกิดข้อผิดพลาด
```

ถึงแม้ว่าก้อนอ็อบเจกต์จริงจะเป็นอ็อบเจกต์ระดับคลาสลูกก็ตาม แต่ตัวที่ใช้อ้างอิงเป็นของระดับคลาสแม่ ดังนั้น anAccount ซึ่งมีชนิดเป็น BankAccount มันไม่รู้จักเมท็อด addInterest()

### 8.3.2 Down-casting

ในหัวข้อที่ผ่านมากล่าวถึง การกำหนด (assign) ค่าตัวแปรอ้างอิงของอ็อบเจกต์ให้กับตัวแปรอ้างอิงชนิดเดียวกันหรือชนิดที่เป็นคลาสระดับคลาสแม่ (up-casting) ซึ่งสามารถทำได้ ในหัวข้อนี้จะกล่าวถึงการกำหนด (assign) ค่าตัวแปรอ้างอิงของอ็อบเจกต์ให้กับตัวแปรอ้างอิงชนิดที่เป็นคลาสระดับคลาสลูก (down-casting)

หากเรากำหนด (assign) ค่าตัวแปรอ้างอิงของอ็อบเจกต์ในระดับคลาสแม่ไปให้กับตัวแปรอ้างอิงระดับคลาสลูก เช่น

```
savings = anAccount;
```

จะเกิดข้อผิดพลาด คอมไพล์ไม่ผ่าน เนื่องจากมันไม่มีความสัมพันธ์แบบ IS-A relationship ต่อกัน นั่นคือ BankAccount ไม่ใช่ SavingsAccount คลาส SavingsAccount มีตัวแปร (interrestRate) และเมท็อด (addInterest()) เพิ่มเติมที่ไม่มีใน BankAccount

แต่หากในการกำหนดค่านั้น อ็อบเจกต์ที่ตัวแปรอ้างอิงระดับคลาสแม่กำลังชี้อยู่นั้นเป็นอ็อบเจกต์ระดับคลาสลูก จะสามารถทำได้ โดยการทำ down-casting เพื่อเปลี่ยนชนิดของคลาสแม่ให้เป็นชนิดของคลาสลูก ดังคำสั่งต่อไปนี้

savings = (savingsAccount)anAccount;

อย่างไรก็ตาม ในการทำ down-casting นั้นควรมีการตรวจสอบให้ชัดเจนก่อนว่า ก้อนอ็อบเจกต์จริงเป็นก้อนอ็อบเจกต์ระดับคลาสลูกจริงหรือไม่ โดยการใช้ operator ที่เรียกว่า instanceOf ดังรูปแบบ

*Object* instanceOf *ClassName*

ซึ่งผลลัพธ์ที่ได้จาก operator นี้คือ ค่า boolean ว่า จริงหรือเท็จ

ตัวอย่าง

```
SavingsAccount savings=new SavingsAccount(5);
BankAccount anAccount = savings;
if (anAccount instanceOf SavingsAccount)
    SavingsAccount newSav =(SavingsAccount)anAccount;
```

## 8.4 การ override เมท็อด equals()

เมท็อด equals() เป็นอีกเมท็อดหนึ่งที่เราจะทำการ override มัน เพื่อให้ตรวจสอบว่าอ็อบเจกต์สองอ็อบเจกต์เหมือนกันหรือไม่ โดยเมท็อดจะคืนค่าเป็นชนิด boolean ว่าเป็นจริงหรือเท็จ คำว่าเหมือนกันคือค่าตัวแปรประจำอ็อบเจกต์ของทั้งสองอ็อบเจกต์เท่ากัน ไม่ใช่เป็นการเปรียบเทียบว่าค่าของตัวแปรอ้างอิงของสองอ็อบเจกต์เท่ากัน คือไม่ได้เทียบว่า ที่อยู่ของอ็อบเจกต์ในหน่วยความจำของตัวแปรอ้างอิงทั้งสองอ็อบเจกต์เท่ากัน การเปรียบเทียบค่าที่อยู่ของอ็อบเจกต์ในหน่วยความจำจะใช้เครื่องหมาย == ในการเปรียบเทียบ ซึ่งถ้ามันเท่ากัน ก็แสดงว่าตัวแปรทั้งสองชี้ไปยังก้อนอ็อบเจกต์เดียวกันนั่นเอง

เมท็อด equals() มีตัวแปรพารามิเตอร์หนึ่งตัวซึ่งมีชนิดเป็น Object ดังนั้นเมื่อต้องการใช้ในการเปรียบเทียบค่า instance variable ภายในอ็อบเจกต์ว่าเท่ากันหรือไม่ จำเป็นต้องทำการแปลงชนิดของตัวแปรพารามิเตอร์ให้มีชนิดเป็นคลาสระดับคลาสลูกของ Object ที่ต้องการเปรียบเทียบก่อน นั่นคือต้องทำ down-casting จากตัวแปรอ้างอิงชนิดคลาสระดับ Object เป็นตัวแปรอ้างอิงชนิดของคลาสที่ต้องการเปรียบเทียบ ดังตัวอย่าง

```
public class BankAccount {
   public boolean equals(Object otherObject) {
     BankAccount other = (BankAccount)otherObject;
     return balance == other.balance;
   }
   ...
}
```

ตัวอย่างต่อไปเป็นตัวอย่างของคลาส Coin เพื่อกำหนดเป็นคลาสของเหรียญ และคลาส Test เพื่อทดสอบคลาส Coin ที่สร้างขึ้น โดยจะตรวจสอบว่า อ็อบเจกต์ของเหรียญที่สร้างขึ้นมาเหมือนกันหรือไม่

ตัวอย่าง ไฟล์โปรแกรม Coin.java

```
1  public class Coin {
2     private double value;
3     private String name;
4
5     public Coin(double aValue, String aName) {
6        value = aValue;
7        name = aName;
8     }
9
```

```
10      public double getValue() {
11         return value;
12      }
13
14      public String getName() {
15         return name;
16      }
17
18      public boolean equals(Object otherObject)  {
19         Coin other = (Coin)otherObject;
20         return name.equals(other.name) && value==other.value;
21      }
22  }
```

คำอธิบายโปรแกรม

- คลาส Coin มีตัวแปรประจำอ็อบเจกต์ เพื่อเก็บค่า ชื่อของเหรียญ และมูลค่าของเหรียญ มีตัวสร้างเพื่อกำหนดค่าตัวแปรทั้งสอง มีเมท็อด getValue() และ getName() เพื่อเรียกค่าตัวแปรดังกล่าว และมีการ override equals เพื่อทดสอบว่าเหรียญทั้งสองอ็อบเจกต์เหมือนกันหรือไม่ โดยตัวแปรพารามิเตอร์ของเมท็อดมีชนิดเป็น Object เมื่อเข้ามาทำงานภายในเมท็อด จะต้องมีการแปลงชนิดตัวแปรอ้างอิงจาก Object ให้เป็นชนิด Coin เสียก่อน จึงนำค่าข้อมูลของเหรียญทั้งสองมาเปรียบเทียบกันได้ และส่งค่าคืนออกจากเมท็อดเป็นจริงหรือเท็จ

ตัวอย่าง ไฟล์โปรแกรม Test.java

```
1   public class Test {
2       public static void main(String[] args) {
3           Coin c1 = new Coin(10, "Gold");
4           Coin c2 = new Coin(10, "Gold");
5           Coin c3 = new Coin(5, "Silver");
6           if (c1.equals(c2))
7               System.out.println(c1.getName() + " equals " + c2.getName());
8           else
9               System.out.println("Unequal");
10          if (c2.equals(c3))
11              System.out.println(c2.getName() + " equals " + c3.getName());
12          else
13              System.out.println("Unequal");
14      }
15  }
```

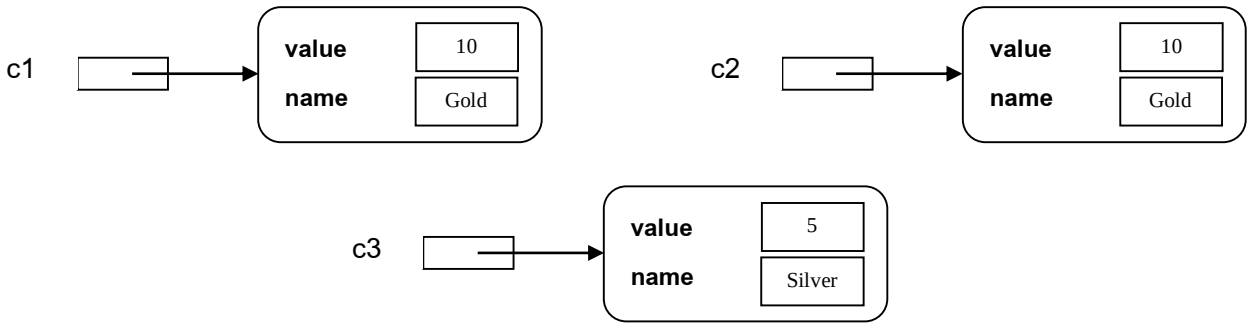
คำอธิบายโปรแกรม

- คลาสทดสอบนี้มีการสร้างอ็อบเจกต์ของเหรียญ 3 อ็อบเจกต์ ดังแสดงในรูปที่ 8.6 และเรียกใช้เมท็อด equals() เพื่อเปรียบเทียบว่า เหรียญแรกเหมือนเหรียญที่สองหรือไม่ และเหรียญที่สองเหมือนเหรียญที่สามหรือไม่

ผลลัพธ์ที่ได้จากโปรแกรมคือ

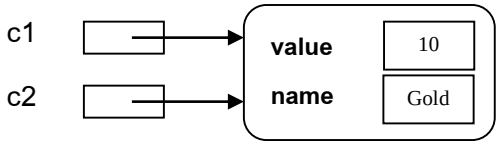
Gold equals Gold

Unequal




รูปที่ 8.6 อ็อบเจกต์ทั้งสามที่สร้างจากคลาส Coin

จากรูปที่ 8.6 อ็อบเจกต์ c1 และ c2 มีค่าตัวแปรประจำอ็อบเจกต์เท่ากัน ในการเปรียบเทียบว่าค่าข้อมูลที่เก็บภายในอ็อบเจกต์ 2 อ็อบเจกต์เท่ากันหรือไม่ จะใช้คำสั่ง if (c1.equals(c2)) … จะไม่ใช้คำสั่ง if (c1 == c2) … เนื่องจากการใช้คำสั่ง if และเครื่องหมาย == เป็นการเปรียบเทียบค่าตำแหน่งที่อยู่ในหน่วยความจำที่เก็บในตัวแปรอ้างอิงถึงอ็อบเจกต์ หากเทียบด้วยเครื่องหมาย == และให้ผลเป็นจริง แสดงว่าตัวแปรอ้างอิงทั้งสองอ้างอิงหรือชี้ไปยังก้อนอ็อบเจกต์ก้อนเดียวกัน ภาพในหน่วยความจำเพื่อแทนเหตุการณ์นี้จะเป็นดังรูปที่ 8.7




รูปที่ 8.7 รูปแสดงตัวแปรอ้างอิง 2 ตัวที่อ้างอิงไปยังก้อนอ็อบเจกต์เดียวกัน

## 8.5 Polymorphism

คำว่า polymorphism เมื่อถูกใช้ในการเขียนโปรแกรมเชิงวัตถุ หมายถึง ความสามารถในการใช้ตัวแปรอ้างอิงหนึ่งตัวเพื่ออ้างอิงถึงอ็อบเจกต์ของคลาสได้หลายชนิดและสามารถเรียกใช้เมท็อดของอ็อบเจกต์เพื่อทำงานโดยใช้คำสั่งเดียวกัน แต่ในการประมวลผล เครื่องจะพิจารณาเองว่าในขณะนั้นตัวแปรอ้างอิงนั้นอ้างอิงไปยัง อ็อบเจกต์ของคลาสใด ก็จะสั่งให้เมท็อดของอ็อบเจกต์นั้นประมวลผล ไม่ได้พิจารณาเลือกเมท็อดในการทำงานตามชนิดของตัวแปรอ้างอิง

ในการเขียนโปรแกรมให้ทำงานแบบ polymorphism ได้นั้นอ็อบเจกต์ที่เกี่ยวข้องในการทำงานจะต้องเป็นอ็อบเจกต์ที่อยู่ในตระกูลเดียวกัน นั่นคือมีความสัมพันธ์แบบ IS-A relationship มีความเป็นคลาสแม่คลาสลูกกันอยู่ ตัวอย่างเช่น

- มีคลาสแม่คือ Animal ซึ่งมีเมท็อด move() สำหรับการเคลื่อนที่
- มีคลาสลูกคือ Fish, Frog และ Bird ซึ่งทุกคลาสต่างก็ override move() เพื่อให้มีการเคลื่อนที่เป็นไปตามลักษณะของตน เช่น ปลาใช้การว่ายน้ำ กบใช้การกระโดด และนกใช้การบิน เป็นต้น

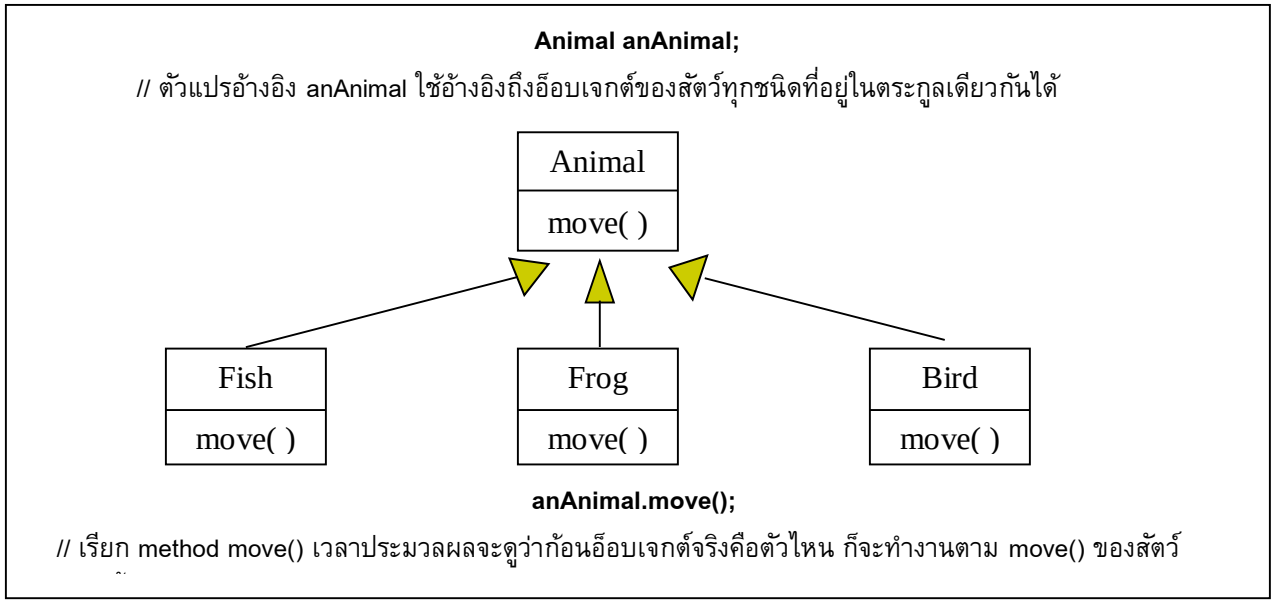
ดังรูปที่ 8.8 เมื่อประกาศตัวแปรอ้างอิงชื่อ anAnimal ให้มีชนิดเป็น Animal ซึ่งเป็นตัวแปรชนิดคลาสแม่ เราสามารถใช้ตัวแปรนี้เพื่ออ้างอิงถึงอ็อบเจกต์ของสัตว์อื่นภายในตระกูลเดียวกันได้ เช่น คำสั่ง

```
anAnimal = new Bird();
```

ทำให้สร้างอ็อบเจกต์จากคลาส Bird ขึ้นมาโดยมีตัวแปรอ้างอิงเป็น anAnimal ซึ่งเป็นตัวแปรอ้างอิงที่มีชนิดเป็นคลาสแม่ หากมีการใช้คำสั่ง

**anAnimal.move();** **// คำสั่งนี้เป็นคำสั่งที่ทำงานแบบ polymorphism**

เมท็อด move() ที่จะถูกประมวลผลคือ move() ของ Bird คือเครื่องจะพิจารณาจากก้อนอ็อบเจกต์จริงที่ทำงานอยู่ ไม่ได้พิจารณาเลือก move() ของ Animal ซึ่งเป็นชนิดของตัวแปรอ้างอิงให้มาประมวลผล




รูปที่ 8.8 ความสัมพันธ์แบบ IS-A relationship ของตระกูล Animal

ตัวอย่างต่อไป เป็นตัวอย่างเพื่อแสดงการทำงานแบบ polymorphism โดยในตัวอย่างนี้ประกอบด้วยคลาส BankAccount, SavingsAccount, CheckingAccount และ AccountTest โดยนำคลาสเดิมจากในเนื้อหาเรื่องก่อนมาปรับปรุง คือในคลาส BankAccount มีการใช้คำสั่ง this(); เพื่อเรียกตัวสร้างอื่นในคลาสเดียวกัน และเพิ่มเมท็อด transfer() ให้โอนเงินจากบัญชีของผู้เรียกใช้เมท็อดไปยังบัญชีอื่น นั่นคือถอนจากบัญชีตัวเองแล้วนำไปฝากเข้าบัญชีอื่น ส่วนคลาสอื่น ๆ เป็นเหมือนเดิมตามเนื้อหาเรื่องการ override toString()

ตัวอย่าง ไฟล์โปรแกรม BankAccount.java

```
1  public class BankAccount  {
2     private double balance;
3     public BankAccount()   {
4        this(0);            // call other constructor
5     }
6     public BankAccount(double initialBalance)   {
7        balance = initialBalance;
8     }
9     public void deposit(double amount)   {
10       balance = balance + amount;
11    }
12    public void withdraw(double amount)   {
```

```
13        balance = balance - amount;
14     }
15     public double getBalance()  {
16        return balance;
17     }
18     public String toString()   {
19           return getClass().getName() + "[balance = " + balance + "]";
20     }
21     /* Transfers money from the bank account to another account */
22     public void transfer(BankAccount other, double amount)   {
23        withdraw(amount);
24        other.deposit(amount);     // polymorphism
25     }
26  }
```

คำอธิบายโปรแกรม

- คลาส BankAccount นี้มีการใช้คำสั่ง this(0); ในบรรทัดที่ 4 เพื่อเรียกใช้ตัวสร้างอีกตัวหนึ่งของคลาสให้ทำงาน โดยในการใช้ this() เพื่อเรียกตัวสร้างอื่นทำงานนั้น เครื่องจะพิจารณาว่าจะเลือกตัวสร้างตัวใดให้ทำงานโดยดูจาก argument ที่ระบุ ในที่นี้ เรียกด้วยคำสั่ง this(0); เครื่องก็จะเรียกตัวสร้างตัวที่มีตัวแปรพารามิเตอร์หนึ่งตัว ซึ่งมีชนิดเป็นตัวเลขให้ทำงาน
- บรรทัดที่ 21-25 คือการกำหนดเมท็อดชื่อ transfer โดยรับพารามิเตอร์ 2 ตัวคือ อ็อบเจกต์ของบัญชีเงินฝาก และจำนวนเงินที่ต้องการโอน ภายในเมท็อดมีคำสั่งเพื่อถอนเงินตามจำนวนที่ระบุจากบัญชีเงินฝากของตน แล้วนำเงินที่ได้นี้นำฝากเข้ายังบัญชีอื่นที่ส่งเข้ามา โดยคำสั่ง

  other.deposit(amount);

  เป็นคำสั่งที่ทำงานแบบ polymorphism นั่นคือ การที่เครื่องจะเรียกเมท็อด deposit() ให้ทำงานนั้น จะพิจารณาจากก้อนอ็อบเจกต์จริงว่าขณะนั้นก้อนอ็อบเจกต์ดังกล่าวคืออ็อบเจกต์ของคลาสใดก็จะประมวลผลเพื่อฝากเงินตามคำสั่งของคลาสนั้น ไม่ได้พิจารณาเพื่อทำการฝากเงินตามชนิดของตัวแปรอ้างอิงคือ BankAccount

ตัวอย่าง ไฟล์โปรแกรม SavingsAccount.java

```
1  public class SavingsAccount extends BankAccount {
2        private double interestRate;
3        public SavingsAccount(double rate)  {
4              interestRate = rate;
5        }
6        public void addInterest()  {
7              double interest = getBalance() * interestRate / 100;
8              deposit(interest);
9        }
10       public String toString()  {
11             return super.toString() + "\n[interestRate = " +
12                           interestRate + "]";
13       }
14 }
```

ตัวอย่าง ไฟล์โปรแกรม CheckingAccount.java

```
1  public class CheckingAccount extends BankAccount  {
```

```
2     private int transactionCount;
3     private static final int FREE_TRANSACTIONS = 3;
4     private static final double TRANSACTION_FEE = 2.0;
5     public CheckingAccount(int initialBalance)   {
6        // construct superclass
7        super(initialBalance);
8        transactionCount = 0;
9     }
10    public void deposit(double amount)    {
11       transactionCount++;
12       super.deposit(amount);
13    }
14    public void withdraw(double amount)    {
15       transactionCount++;
16       super.withdraw(amount); // now subtract amount from balance
17    }
18    /*Deducts the accumulated fees and resets the transaction count.*/
19    public void deductFees()   {
20       if (transactionCount > FREE_TRANSACTIONS)    {
21          double fees = TRANSACTION_FEE *
22             (transactionCount - FREE_TRANSACTIONS);
23          super.withdraw(fees);
24       }
25       transactionCount = 0;
26    }
27
28    public String toString()  {
29       return super.toString() + "\n[transactionCount = " +
30                         transactionCount + "]";
31   }
32 }
```

ตัวอย่าง ไฟล์โปรแกรม AccountTest.java

```
1  public class AccountTest  {
2     public static void main(String[] args)   {
3        SavingsAccount mySavings = new SavingsAccount(5);
4        CheckingAccount BoomChecking = new CheckingAccount(10000);
5        mySavings.deposit(10000);
6        BoomChecking.withdraw(1500);
7        BoomChecking.deposit(1500);
8        BoomChecking.withdraw(1500);
9        BoomChecking.withdraw(400);
10       mySavings.transfer(BoomChecking, 2000);
11       System.out.println(mySavings);
12       System.out.println();
13       System.out.println(BoomChecking);
14       System.out.println();
15       // simulate end of month
16       mySavings.addInterest();
17       BoomChecking.deductFees();
18       System.out.println("My savings balance = $" +
                                mySavings.getBalance());
19       System.out.println("Boom's checking balance = $" +
                                BoomChecking.getBalance());
20    }
21 }
```

คำอธิบายโปรแกรม

- คลาสทดสอบนี้ มีการเพิ่มคำสั่ง mySavings.transfer(BoomChecking, 2000); เพื่อให้มีการโอนเงินจากบัญชีออมทรัพย์ mySavings ไปยังบัญชีแบบใช้เช็ค BoomChecking เป็นจำนวนเงิน 2,000
- เมื่อมีการใช้คำสั่งนี้ การประมวลผลจะทำงานตามคำสั่งของเมท็อด transfer ในคลาส BankAccount ซึ่งตัวแปรพารามิเตอร์ BankAccount other จะรับค่าของตัวแปรอ้างอิง BoomChecking นั่นคือรับค่าที่อยู่ในหน่วยความจำของอ็อบเจกต์ BoomChecking ซึ่งทำให้ตัวแปร other ของเมท็อด transfer ชี้ไปยังก้อนอ็อบเจกต์เดียวกันกับที่ตัวแปร BoomChecking อ้างอิงอยู่และตัวแปรพารามิเตอร์ double amount จะรับค่า 2,000
- ภายในเมท็อด transfer() คำสั่งแรกคือ `withdraw(amount);` ซึ่งคือ `this.withdraw(amount);` เป็นการถอนเงินจากบัญชีของตน และบรรทัดต่อไป `other.deposit(amount);` คือคำสั่งเพื่อฝากเงินเข้าในบัญชีที่ตัวแปรอ้างอิง other ชี้อยู่ ซึ่งในที่นี้ก้อนอ็อบเจกต์ที่ other ชี้อยู่คือ ก้อนอ็อบเจกต์จากคลาส CheckingAccount นั่นคือคำสั่งภายในเมท็อด deposit() ของคลาส CheckingAccount ที่นอกจากจะถอนเงินแล้ว ยังมีการเพิ่มจำนวนตัวนับจำนวนรายการที่ทำ จะถูกประมวลผล ไม่ใช่เมท็อด deposit() ของ BankAccount

ผลลัพธ์ที่ได้จากโปรแกรมคือ

```
SavingsAccount[balance = 8000.0]
[interestRate = 5.0]

CheckingAccount[balance = 10100.0]
[transactionCount = 5]

My savings balance = $8400.0
Boom's checking balance = $10096.0
```

กล่าวโดยสรุป การเขียนโปรแกรมให้ทำงานด้วยเทคนิค polymorphism นั้น

- คลาสต่าง ๆ ที่เกี่ยวข้องจะต้องเป็นคลาสที่อยู่ในตระกูลเดียวกัน (same inheritance hierarchy)
- ทุกคลาสในตระกูลนี้ที่ต้องการเรียกใช้เมท็อดแบบ polymorphism จะต้องมีเมท็อดเดียวกันนี้ในทุกคลาส นั่นคือคลาสลูกจะต้อง override เมท็อดดังกล่าวที่มีในคลาสแม่
- การเขียนคำสั่งเรียกใช้เมท็อด จะเป็นการเรียกใช้ผ่านตัวแปรอ้างอิงระดับคลาสแม่ ส่วนการประมวลผลนั้นเครื่องจะพิจารณาเรียกใช้เมท็อดตามก้อนอ็อบเจกต์จริงที่กำลังประมวลผล

ดังตัวอย่างของเมท็อด transfer() ของคลาส BankAccount ซึ่งมีการใช้คำสั่ง other.deposit() ซึ่งเป็นคำสั่งแบบ polymorphism จะเห็นว่าทั้งคลาส BankAccount และ CheckingAccount เป็นคลาสในตระกูลเดียวกัน และทั้งสองคลาสต่างก็มีเมท็อด deposit() ของตนซึ่งคลาส CheckingAccount ทำการ override เมท็อดนี้ให้มีรายละเอียดในการทำงานที่แตกต่างจากคลาส BankAccount การประมวลผลของคำสั่ง other.deposit() นั้นสามารถทำงานได้แตกต่างกันหลายแบบแล้วแต่ว่าเป็น deposit() ของอ็อบเจกต์จากคลาสใด นั่นคือ หนึ่งคำสั่งแต่ทำงานได้หลายแบบ ซึ่งคือ polymorphism นั่นเอง

## 8.6 เมท็อดและคลาสแบบ final (Final method and class)

หากกำหนดเมท็อดให้เป็น fiinal แล้ว จะไม่สามารถ override เมท็อดนั้น โดยเมท็อดที่กำหนดสิทธิ์ในการเข้าถึงเป็น private หรือเมท็อดแบบ static ซึ่งเป็นเมท็อดประจำคลาส จะถือว่าเป็นเมท็อดแบบ final แล้ว

สำหรับคลาส หากเรากำหนดให้เป็น final แล้ว คลาสนั้นจะไม่สามารถเป็นคลาสแม่ได้อีกแล้ว และเมท็อดภายใน final class จะจัดเป็น final method ด้วย ตัวอย่างเช่น คลาส String ของจาวา

## 8.7 คลาสนามธรรมและเมท็อดนามธรรม (Abstract class and method)

คลาสนามธรรม (abstract class) คือ คลาสที่ยังไม่สมบูรณ์ในตัวเอง ทำให้ไม่สามารถสร้างอ็อบเจกต์จากคลาสนี้ได้ โดยปกติแล้ว คลาสนามธรรมจะทำหน้าที่เป็นคลาสแม่ โดยในคลาสนามธรรมนี้จะมีเมท็อดนามธรรม (abstract method) อย่างน้อย 1 เมท็อด ซึ่งเมท็อดนามธรรมคือ เมท็อดที่มีเพียงบรรทัดหัวของเมท็อด แต่ไม่มีรายละเอียดของคำสั่งต่าง ๆ ของเมท็อด คลาสลูกจะต้อง override เมท็อดนามธรรม โดยการเพิ่มรายละเอียดของคำสั่งต่าง ๆ ในเมท็อด คลาสลูกนั้นจึงจะสมบูรณ์ เรียกว่า concrete class และสามารถนำไปสร้างอ็อบเจกต์ได้ แต่หากคลาสลูกไม่ได้ override เมท็อดนามธรรม คลาสลูกนั้นก็จะต้องเป็นคลาสนามธรรมด้วยเช่นกัน

จุดประสงค์ของการสร้างคลาสนามธรรม คือการเตรียมคลาสแม่ที่เหมาะสมเพื่อให้คลาสอื่นมาสืบทอดจากคลาสนามธรรมนี้และร่วมใช้เมท็อดต่าง ๆ ที่ออกแบบไว้ในคลาสนามธรรมร่วมกัน โดยในคลาสนามธรรมจะกำหนดตัวแปรและเมท็อดที่จำเป็นไว้ให้ และสำหรับเมท็อดที่กำหนดเป็นเมท็อดนามธรรมนั้นจะยังไม่มีรายละเอียดของการทำงานไว้ให้ เพียงแต่ร่างโครงแบบให้เห็นว่า จะมีการกระทำหรือพฤติกรรมเหล่านี้ คลาสลูกที่สืบทอดจากคลาสนามธรรมนี้ จะเป็นผู้กำหนดรายละเอียดภายในเมท็อดนามธรรมได้ตามความต้องการ ตัวอย่างคลาสนามธรรม เช่น คลาส Shape ซึ่งผู้ออกแบบคลาสทราบว่าจะสามารถคำนวณหาพื้นที่และปริมาตรของรูปทรงได้ และรูปทรงต่าง ๆ ต้องมีชื่อของรูปทรง จึงออกแบบคลาสเป็นคลาสนามธรรมชื่อ Shape ซึ่งภายในคลาสยังไม่สมบูรณ์ในตัวเอง เนื่องจากยังไม่ทราบว่าเป็นรูปทรงชนิดใด จึงไม่สามารถสร้างอ็อบเจกต์จากคลาสนามธรรม Shape จะต้องสร้างคลาสลูกอื่นเพื่อสืบทอดจากคลาส Shape นี้ เช่น สร้างคลาสลูกชื่อ Circle, Cylinder คลาสลูกที่สร้างใหม่ก็จะใช้การออกแบบที่ได้ออกแบบไว้ในคลาสแม่ คือสามารถคำนวณหาพื้นที่และปริมาตร รวมทั้งกำหนดชื่อของรูปทรงได้เองตามต้องการ และเมื่อคลาสลูกกำหนดรายละเอียดต่าง ๆ ไว้สมบูรณ์แล้ว ก็จะสามารถนำไปสร้างเป็นอ็อบเจกต์เพื่อทำงานได้ต่อไป

การสร้างคลาสนามธรรม

- ระบุที่หัวคลาสด้วยคำว่า abstract
- ภายในคลาส มีอย่างน้อย 1 เมท็อดที่เป็นเมท็อดนามธรรม
- ภายในคลาส สามารถมีตัวแปรและเมท็อดอื่น ๆ ตามปกติ
- ตัวสร้าง และ static methods ไม่สามารถประกาศเป็น abstract

การสร้างเมท็อดนามธรรม

- ระบุที่หัวเมท็อดด้วยคำว่า abstract
- เมื่อเขียนบรรทัดหัวเมท็อดแล้ว ให้ปิดท้ายด้วยเครื่องหมาย ; และจบแค่นั้น ไม่มีการใส่คำสั่งใด ๆ ของเมท็อด ปล่อยให้คลาสลูกนำไป override และเขียนคำสั่งภายในเมท็อดเอง

ตัวอย่าง ไฟล์โปรแกรม AbstractA.java

```
1  public abstract class AbstractA {
2      private int a = 0;
3      private int b = 1;
4      public abstract int getValue();
5      public int getA() {
6          return a;
7      }
8      public int getB() {
9          return b;
10     }
11 }
```

คำอธิบายโปรแกรม

- คลาส AbstractA เป็นคลาสนามธรรมที่มีตัวแปร 2 ตัวคือ a และ b มีเมท็อดนามธรรมชื่อ getValue() ซึ่งยังไม่สมบูรณ์ และมีเมท็อด getA() และ getB() เพื่อเรียกค่าของตัวแปร a และ b

ตัวอย่าง ไฟล์โปรแกรม AbstractB.java

```
1  public abstract class AbstractB extends AbstractA {
2      private int c;
3      public int getC() {
4          return c;
5      }
6        //still have not implementation of getValue()
7  }
```

คำอธิบายโปรแกรม

- คลาส AbstractB เป็นคลาสที่ขยายต่อจากคลาส AbstractA มีตัวแปรเพิ่มอีก 1 ตัวคือ c และมีเมท็อด getC() เพื่อเรียกค่าของตัวแปร c แต่ยังไม่ได้ override getValue() ดังนั้นคลาส AbstractB จึงยังต้องเป็นคลาสนามธรรมต่อไป

ตัวอย่าง ไฟล์โปรแกรม ClassC.java

```
1  public class ClassC extends AbstractB{
2      public int getValue() {
3          return getA() + getB() + getC();
4      }
5  }
```

คำอธิบายโปรแกรม

- คลาส ClassC เป็นคลาสที่ขยายต่อจากคลาส AbstractB โดยมีการ override getValue() โดยให้มีคำสั่งหาผลรวมของตัวแปร a, b และ c และส่งค่าผลรวมคืนออกจากเมท็อด

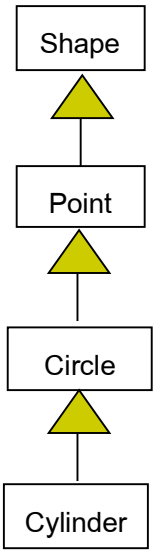
ตัวอย่าง ไฟล์โปรแกรม Main.java

```
1  public class Main {
2      public static void main(String[] args) {
3          ClassC o = new ClassC();
4          System.out.println("a = " + o.getA());
5          System.out.println("b = " + o.getB());
6          System.out.println("c = " + o.getC());
7          System.out.println("value = " + o.getValue());
8      }
9  }
```

คำอธิบายโปรแกรม

- คลาส Main เป็นคลาสทดสอบ โดยมีการสร้างอ็อบเจกต์จากคลาส ClassC แล้วเรียกใช้เมท็อด getA(), getB() และ getC() ที่ได้รับถ่ายทอดมา และเรียกใช้เมท็อด getValue() ที่ ClassC override มา

ตัวอย่างต่อไป จะประกอบด้วยคลาสต่าง ๆ ตาม inheritance hierarchy ดังรูปที่ 8.9




รูปที่ 8.9 ความสัมพันธ์แบบ IS-A relationship ของตระกูล Shape

ตัวอย่าง ไฟล์โปรแกรม Shape.java

```
1   // Shape abstract-superclass declaration.
2   public abstract class Shape {
3         public double getArea() {
4               return 0.0;
5         }
6         public double getVolume() {
7               return 0.0;
8         }
9         // abstract method, overridden by subclasses
10        public abstract String getName();
11  }
```

คำอธิบายโปรแกรม

- คลาส Shape เป็นคลาสนามธรรมเนื่องจากมีเมท็อดนามธรรม ชื่อ getName() ในบรรทัดที่ 10 ซึ่งจะเห็นว่า มีเพียงหัวเมท็อด ซึ่งมีคำว่า abstract และจบท้ายด้วย ; ไม่มีส่วนของคำสั่งในเมท็อด เนื่องจากคลาส Shape นี้เป็น abstract ยังไม่สามารถระบุได้ชัดเจนว่ารูปทรงจะเป็นชนิดใด จึงยังไม่สามารถบอกชื่อชนิดของรูปทรงได้ ผู้เขียนโปรแกรมจึงกำหนดไว้ก่อนว่ารูปทรงต่าง ๆ ควรจะมีชื่อ แต่ในชั้นสูงสุดนี้ยังไม่สามารถระบุได้ว่าชื่ออะไร เป็นหน้าที่ของคลาสลูกที่ขยายต่อจากคลาส Shape ในการ override เมท็อดและระบุชื่อของรูปทรง
- มีเมท็อด getArea() และ getVolume() เพื่อคืนค่า 0 ออกจากเมท็อด หากคลาสลูกเป็นคลาสของสิ่งที่สามารถคำนวณหาพื้นที่หรือปริมาตรได้ ก็จะ override เมท็อดเพื่อคำนวณค่าตามชนิดของรูปทรง

ตัวอย่าง ไฟล์โปรแกรม Point.java

```
1  // Point class declaration inherits from Shape.
2  public class Point extends Shape  {
3        private int x;  // x part of coordinate pair
4        private int y;  // y part of coordinate pair
5        public Point( int xValue, int yValue )  {
6              x = xValue;
7              y = yValue;
8        }
9        public void setX( int xValue ) {
10             x = xValue;
11       }
12       public int getX() {
13             return x;
14       }
15       public void setY( int yValue ) {
16             y = yValue;
17       }
18       public int getY() {
19             return y;
20       }
21       // override abstract method getName
22       public String getName() {
23             return "Point";
24       }
25       // override toString
26       public String toString() {
27             return "[" + getX() + ", " + getY() + "]";
28       }
29 }
```

คำอธิบายโปรแกรม

- คลาส Point ขยายต่อจาก Shape มี instance variable คือ x และ y แทนคู่ลำดับของจุด คลาสนี้ override เมท็อดนามธรรม getName() ในบรรทัดที่ 22-24 จึงทำให้เป็นคลาสที่สมบูรณ์สามารถสร้างอ็อบเจกต์ได้ ที่บรรทัดแรกที่เป็นหัวของคลาสนี้จึงไม่มีคำว่า abstract แล้ว
- บรรทัดที่ 5-8 เป็นตัวสร้างของคลาส เพื่อกำหนดค่าตัวแปรประจำอ็อบเจกต์ตามที่ระบุตอนสร้างอ็อบเจกต์
- บรรทัดที่ 9-20 เป็น accessor และ mutator method คือ getX(), getY(), setX() และ setY()เพื่อเรียกค่าและเปลี่ยนค่าตัวแปร x และ y เนื่องจากสิทธิ์การเข้าถึงตัวแปรเป็น private จึงควรมีเมท็อดเพื่อให้สามารถเข้าถึงค่าเหล่านี้ได้
- บรรทัดที่ 22-24 เป็นการ override เมท็อดนามธรรม getName() โดยในการ override นั้น เราจะเขียนหัวเมท็อดเหมือนกับที่เคยประกาศในคลาสแม่ แต่จะไม่ใส่คำว่า abstract แล้ว เพราะเราได้เพิ่มคำสั่งทำงานภายในเมท็อดจนได้เมท็อดที่สมบูรณ์แล้ว สามารถเรียกใช้งานได้ตามต้องการ ในที่นี้ คำสั่งที่กำหนดคือการส่งคืนค่าคำว่า Point ให้ผู้เรียกใช้เมท็อด
- บรรทัดที่ 26-28 เป็นการ override toString() เพื่อให้คืนค่าของ instance variable

ตัวอย่าง ไฟล์โปรแกรม Circle.java

```
1  // Circle class inherits from Point.
2  public class Circle extends Point  {
3        private double radius;  // Circle's radius
4        public Circle( int x, int y, double radiusValue ) {
5              super( x, y );  // call Point constructor
6              setRadius( radiusValue );
7        }
8        public void setRadius( double radiusValue ) {
9              radius = ( radiusValue < 0.0 ? 0.0 : radiusValue );
10       }
11       public double getRadius()     {
12             return radius;
13       }
14       public double getDiameter() {
15             return 2 * getRadius();
16       }
17       public double getCircumference() {
18             return Math.PI * getDiameter();
19       }
20       // override method getArea to return Circle area
21       public double getArea()  {
22             return Math.PI * getRadius() * getRadius();
23       }
24       // override abstract method getName to return "Circle"
25       public String getName()  {
26             return "Circle";
27       }
28       // override toString to return String representation of Circle
29       public String toString() {
30             return "Center = " + super.toString()
                     + "; Radius = " + getRadius();
31       }
32 }
```

คำอธิบายโปรแกรม

- คลาส Circle ขยายต่อจาก Point โดยตัวแปร x และ y ของ Point ที่เป็นส่วนประกอบหนึ่งของ Circle นั้นจะแสดงถึงจุดศูนย์กลางของวงกลม และมีตัวแปรเพิ่มอีกหนึ่งตัวคือ radius เพื่อแทนรัศมี คลาสนี้ override เมท็อดนามธรรม getName() ในบรรทัดที่ 25-27 จึงทำให้เป็นคลาสที่สมบูรณ์สามารถสร้างอ็อบเจกต์ได้ ที่บรรทัดแรกที่เป็นหัวของคลาสนี้จึงไม่มีคำว่า abstract แล้ว
- บรรทัดที่ 4-7 เป็นตัวสร้างซึ่งจะกำหนดค่าตัวแปรต่าง ๆ ตามที่ระบุตอนสร้างอ็อบเจกต์ โดยบรรทัดแรกภายในตัวสร้างจะเป็นการเรียกตัวสร้างของคลาสแม่เพื่อกำหนดค่าตัวแปร x และ y และบรรทัดต่อมาเป็นการกำหนดค่าให้ตัวแปร radius โดยเรียกใช้เมท็อด setRadius() คำสั่งนี้ หากไม่ต้องการเขียนโดยเรียกใช้เมท็อด setRadius() จะเขียนเป็น

  radius = radiusValue;

  ดังที่เคยทำในโปรแกรมอื่นก่อนหน้านี้ก็ได้
- บรรทัดที่ 8-13 เป็น accessor และ mutator method เพื่อจัดการกับตัวแปร radius
- บรรทัดที่ 14-16 เป็นเมท็อดเพื่อคำนวณหาค่าเส้นผ่านศูนย์กลาง
- บรรทัดที่ 17-19 เป็นเมท็อดเพื่อคำนวณหาค่าเส้นรอบวง

- บรรทัดที่ 21-23 เป็นการ override เมท็อด getArea() เพื่อคำนวณหาค่าพื้นที่
- บรรทัดที่ 25-27 เป็นการ override เมท็อดนามธรรม getName() เพื่อคืนค่าคำว่า Circle ให้ผู้เรียกใช้เมท็อด
- บรรทัดที่ 29-31 เป็นการ override toString() เพื่อให้คืนค่าของ instance variable โดยมีการใช้คำสั่ง super.toString() เพื่อเรียกใช้เมท็อด toString() ของคลาสแม่เพื่อเรียกค่าของ x และ y

ตัวอย่าง ไฟล์โปรแกรม Cylinder.java

```
1  // Cylinder class inherits from Circle.
2  public class Cylinder extends Circle  {
3        private double height;  // Cylinder's height
4        public Cylinder(int x,int y,double radius,double heightValue) {
5              super( x, y, radius );  // call Circle constructor
6              setHeight( heightValue );
7        }
8        public void setHeight( double heightValue )   {
9              height = ( heightValue < 0.0 ? 0.0 : heightValue );
10       }
11       public double getHeight() {
12             return height;
13       }
14       // override method getArea to return Cylinder area
15       public double getArea() {
16             return 2 * super.getArea() +
                     getCircumference() * getHeight();
17       }
18       // override method getVolume to return Cylinder volume
19       public double getVolume()     {
20             return super.getArea() * getHeight();
21       }
22       // override abstract method getName to return "Cylinder"
23       public String getName() {
24             return "Cylinder";
25       }
26       // override toString to return String representation of Cylinder
27       public String toString()      {
28             return super.toString() + "; Height = " + getHeight();
29       }
30 }
```

คำอธิบายโปรแกรม

- คลาส Cylinder ขยายต่อจาก Circle โดยตัวแปร x และ y ของ Point ที่เป็นส่วนประกอบหนึ่งของ Cylinder นั้นจะแสดงถึงจุดศูนย์กลางของปากทรงกระบอก ตัวแปร radius ของ Circle จะเป็นส่วนประกอบหนึ่งของ Cylinder เพื่อแทนรัศมีของทรงกระบอก และมีการประกาศตัวแปรใหม่เพิ่มคือ height เพื่อแทนความสูงของทรงกระบอก คลาสนี้ override เมท็อดนามธรรม getName() ในบรรทัดที่ 23-25 จึงทำให้เป็นคลาสที่สมบูรณ์ สามารถสร้างอ็อบเจกต์ได้ ที่บรรทัดแรกที่เป็นหัวของคลาสนี้จึงไม่มีคำว่า abstract แล้ว
- บรรทัดที่ 4-7 เป็นตัวสร้าง ซึ่งจะกำหนดค่าตัวแปรต่าง ๆ ตามที่ระบุตอนสร้างอ็อบเจกต์ โดยบรรทัดแรกภายในตัวสร้างจะเป็นการเรียกตัวสร้างของคลาสแม่คือ Circle เพื่อกำหนดค่าตัวแปร x, y และ radius และบรรทัดต่อมาเป็นการกำหนดค่าให้ตัวแปร height
- บรรทัดที่ 8-13 เป็น accessor และ mutator method เพื่อจัดการกับตัวแปร height

- บรรทัดที่ 15-17 เป็นการ override เมท็อด getArea() เพื่อคำนวณหาค่าพื้นที่ โดยจะเป็นการนำพื้นที่ของตัวทรงกระบอก (เส้นรอบวงคูณกับความสูง) รวมกับปากของทรงกระบอกทั้งสองข้าง (2 คูณกับพื้นที่ของปากทรงกระบอก) ซึ่งจะเห็นว่า มีการเรียกใช้เมท็อด getArea() และ getCircumference() สังเกตว่า การเรียก getCircumference() นั้น สามารถเรียกใช้ได้ เนื่องจากมันเป็นเมท็อดที่ถ่ายทอดมาจากคลาสแม่ คลาสลูกเรียกใช้ได้ แต่สำหรับ getArea() นั้น ในคลาสลูกก็มีการกำหนดเมท็อด getArea() ขึ้นใช้เองด้วย ซึ่งเป็นการ override เมท็อดของคลาสแม่ ดังนั้น หากต้องการเรียกใช้ getArea() ตัวที่เป็นของคลาสแม่ จะต้องระบุ super กำกับไว้ให้ชัดเจน ไม่เช่นนั้นจะเป็นการเรียก getArea() ของตัวเองแบบ recursive ตัวอย่างกรณีนี้ คล้ายกับที่เคยอธิบายในตัวอย่างของตระกูล BankAccount ที่มีการเรียกใช้ super.deposit() และ super.withdraw()
- บรรทัดที่ 19-21 เป็นการ override เมท็อด getVolume() เพื่อคำนวณหาค่าปริมาตร ซึ่งก็มีการใช้คำสั่ง super.getArea() เช่นกัน เพื่อเรียกใช้เมท็อดของคลาสแม่
- บรรทัดที่ 23-25 เป็นการ override เมท็อดนามธรรม getName() เพื่อคืนค่าคำว่า Cylinder ให้ผู้เรียกใช้เมท็อด
- บรรทัดที่ 27-29 เป็นการ override toString() เพื่อให้คืนค่าของ instance variable โดยมีการใช้คำสั่ง super.toString() เพื่อเรียกใช้เมท็อด toString() ของคลาสแม่เพื่อเรียกค่าของ x, y และ radius

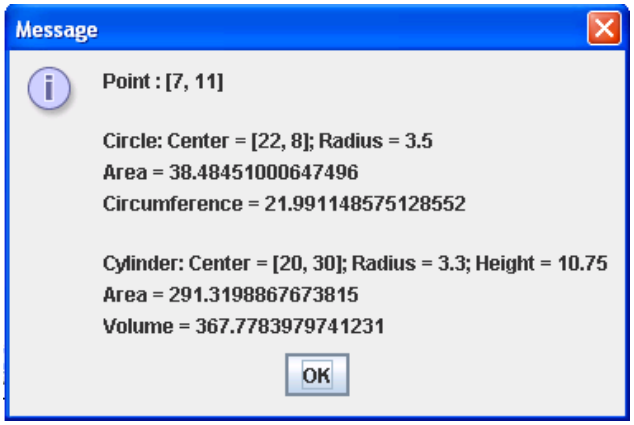
ตัวอย่าง ไฟล์โปรแกรม AbstractTest1.java

```
1  //Test class for shape, point, circle, cylinder hierarchy.
2  import javax.swing.JOptionPane;
3  public class AbstractTest1 {
4      public static void main( String args[] ) {
5        // create Point, Circle and Cylinder objects
6        Point point = new Point( 7, 11 );
7        Circle circle = new Circle( 22, 8, 3.5 );
8        Cylinder cylinder = new Cylinder( 20, 30, 3.3, 10.75 );
9        String output;
10       // obtain name and string representation of each object
11       output = point.getName() + " : " + point + "\n\n";
12       output += circle.getName() + ": " + circle + "\n" +
               "Area = " + circle.getArea() + "\n" +
               "Circumference = " + circle.getCircumference() + "\n\n";
13       output += cylinder.getName() + ": " + cylinder + "\n" +
               "Area = " + cylinder.getArea() + "\n" +
               "Volume = " + cylinder.getVolume();
14       JOptionPane.showMessageDialog( null, output );
15     }
16 }
```

คำอธิบายโปรแกรม

- คลาสนี้สร้างอ็อบเจกต์จากคลาส Point, Circle และ Cylinder อย่างละ 1 อ็อบเจกต์ จากนั้นทดลองเรียกใช้เมท็อดต่าง ๆ ผ่านอ็อบเจกต์แต่ละตัว เพื่อแสดงเป็นผลลัพธ์

ผลลัพธ์ที่ได้จากโปรแกรมคือ




ตัวอย่าง ไฟล์โปรแกรม AbstractTest2.java

```
1  //AbstractTest2 class (using polymorphism)
2  import javax.swing.JOptionPane;
3  public class AbstractTest2  {
4      public static void main( String args[] )    {
5        // create Point, Circle and Cylinder objects
6        Point point = new Point( 7, 11 );
7        Circle circle = new Circle( 22, 8, 3.5 );
8        Cylinder cylinder = new Cylinder( 20, 30, 3.3, 10.75 );
9        // obtain name and string representation of each object
10       String output = point.getName() + ": " + point + "\n" +
               circle.getName() + ": " + circle + "\n" +
               cylinder.getName() + ": " + cylinder + "\n";
11       output += showDetail(point);
12       output += showDetail(circle);
13       output += showDetail(cylinder);
14       JOptionPane.showMessageDialog( null, output );
15     }
16     public static String showDetail(Shape aShape)    {
17           String output = "\n\n" + aShape.getName() + ": " +
                 "\nArea = " + aShape.getArea() +
                 "\nVolume = " +aShape.getVolume();
18         return output;
19     }
20 }
```

คำอธิบายโปรแกรม

- คลาส AbstractTest2 สร้างขึ้นเพื่อทดสอบคลาสต่าง ๆ โดยเรียกใช้เมท็อดด้วยเทคนิค polymorphism
- คลาสนี้สร้างอ็อบเจกต์จากคลาส Point, Circle และ Cylinder อย่างละ 1 อ็อบเจกต์ จากนั้นทดลองเรียกใช้เมท็อด getName() และ toString() ผ่านอ็อบเจกต์แต่ละตัว เพื่อแสดงเป็นผลลัพธ์ และมีการเรียกใช้เมท็อด showDetail() ซึ่งเป็นเมท็อดประจำคลาสหรือ static method ในคลาสทดสอบนี้ เพื่อนำข้อมูลต่าง ๆ ของแต่ละอ็อบเจกต์แสดงเป็นผลลัพธ์
- บรรทัดที่ 16-19 คือ static method showDetail ซึ่งมีตัวแปรพารามิเตอร์หนึ่งตัว ชนิด Shape เมื่อมีการเรียกใช้เมท็อดนี้ ด้วยคำสั่งในบรรทัดที่ 11-13 argument ที่ส่งเข้ามาก็คือตัวแปรอ้างอิงอ็อบเจกต์แต่ละ

ประเภท ซึ่งเมื่อเข้ามาทำงานภายในเมท็อดแล้ว ตัวแปรพารามิเตอร์ aShape ซึ่งมีชนิดเป็นคลาสแม่ของทุกคลาส จะร่วมชี้ไปยังก้อนอ็อบเจกต์นั้น การใช้คำสั่งเพื่อเรียกใช้เมท็อดผ่านตัวแปรระดับคลาสแม่ คือ aShape.getName(), aShape.getArea() และ aShape.getVolume() จัดเป็นการใช้เทคนิค polymorphism นั่นคือ ออกคำสั่งหนึ่งคำสั่ง แต่สามารถประมวลผลได้หลายแบบแล้วแต่ว่าก้อนอ็อบเจกต์จริงในขณะนั้นคืออะไร เช่นถ้าส่งอ็อบเจกต์แบบ Point เข้ามา ก็จะเรียกใช้เมท็อด getName(), getArea() และ getVolume() ของ Point ให้ประมวลผล

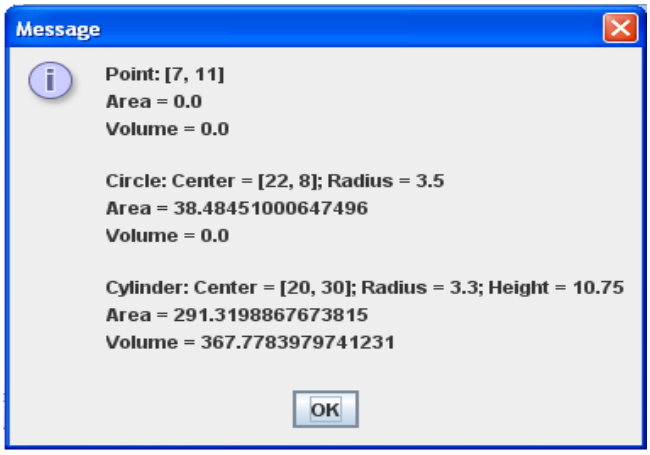
ตัวอย่าง ไฟล์โปรแกรม AbstractTest3.java

```
1  //AbstractTest3 class (using polymorphism, array)
2  import javax.swing.JOptionPane;
3  public class AbstractTest3  {
4      public static void main( String args[] )    {
5        String output = "";
6        // create Point, Circle and Cylinder objects
7        Shape shapeArray[] = new Shape[3];
8        shapeArray[0] = new Point( 7, 11 );
9        shapeArray[1] = new Circle( 22, 8, 3.5 );
10       shapeArray[2] = new Cylinder( 20, 30, 3.3, 10.75 );
11       // obtain data of each object
12       for (int i = 0; i < shapeArray.length; i++) {
13           output += shapeArray[i].getName() + ": " + shapeArray[i] +
                       "\nArea = " + shapeArray[i].getArea() +
                       "\nVolume = " + shapeArray[i].getVolume() + "\n\n";
14       }
15       JOptionPane.showMessageDialog( null, output );
16     }
17 }
```

คำอธิบายโปรแกรม

- คลาส AbstractTest3 ใช้อาร์เรย์ชนิด Shape ซึ่งเป็นตัวแปรระดับคลาสแม่เพื่อเก็บอ็อบเจกต์ที่สร้างจากคลาสต่าง ๆ และเรียกใช้เมท็อดด้วยเทคนิค polymorphism
- คลาสนี้สร้างอ็อบเจกต์จากคลาส Point, Circle และ Cylinder อย่างละ 1 อ็อบเจกต์ เก็บใส่อาร์เรย์ จากนั้นวนลูปเพื่อเรียกใช้เมท็อด getName(), toString(), getArea() และ getVolume() ผ่านตัวแปรอ้างอิง shapeArray[i] เพื่อแสดงเป็นผลลัพธ์ โดยในการเรียกใช้เมท็อดนี้เป็นการทำงานแบบ polymorphism

ผลลัพธ์ที่ได้จากโปรแกรมคือ




## 8.8 อินเทอร์เฟซ (Interface)

อินเทอร์เฟซเป็นคลาสชนิดหนึ่งที่ใช้เมื่อ

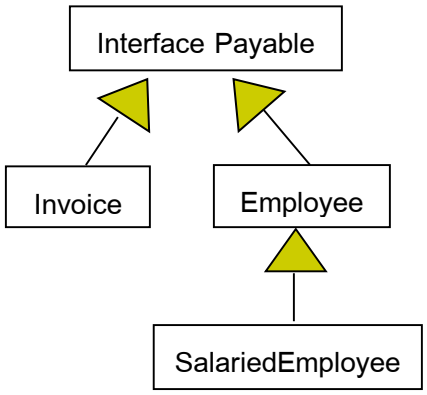
- คลาสอื่น ๆ ที่ไม่มีความสัมพันธ์กันต้องการแบ่งกันใช้ตัวแปรแบบค่าคงที่หรือเมท็อดร่วมกัน นั่นคือเราจะสร้างอินเทอร์เฟซเพื่อกำหนดตัวแปรหรือเมท็อดไว้ แล้วสร้างคลาสลูกอื่น ๆ ขยายต่อจากอินเทอร์เฟซนี้ ซึ่งจะทำให้คลาสลูกอื่น ๆ นั้นสามารถใช้ตัวแปรหรือเมท็อดที่กำหนดในอินเทอร์เฟซร่วมกันได้
- ต้องการทำ multiple inheritance นั่นคือ คลาสลูกมีคลาสแม่มากกว่าหนึ่งคลาส ในภาษาจาวาไม่สามารถสร้างคลาสลูกให้รับทอดจากคลาสแม่ได้มากกว่าหนึ่งคลาส จาวาจะให้สร้างอินเทอร์เฟซ เช่น กำหนดให้มี 1 อินเทอร์เฟซ และ 1 คลาสแม่ แล้วจึงขยายต่อเป็นคลาสลูก

การสร้างอินเทอร์เฟซ

- ใช้คำว่า interface แทน class ในบรรทัดแรกสุด
- ภายในอินเทอร์เฟซ ประกอบด้วย ตัวแปรประจำคลาสแบบค่าคงที่ (public static final fiield) และ public abstract method เท่านั้น โดยในการเขียนคำสั่งประกาศตัวแปรแบบค่าค่งที่สามารถละ public static final ได้ และสำหรับการเขียนหัวเมท็อด สามารถละ public abstract ได้
- เมื่อคอมไพล์อินเทอร์เฟซแล้วจะได้ไฟล์ที่มีนามสกุล .class เช่นเดียวกับการคอมไพล์คลาสปกติ

การสร้างคลาสลูกรับทอดจากอินเทอร์เฟซ

- คลาสลูกต้อง implements อินเทอร์เฟซ ต่างจากกรณีคลาสขยายต่อจากคลาสซึ่งเราจะใช้คำว่า extends ในบรรทัดแรกสุดของคลาสลูก หากมีการ implement อินเทอร์เฟซมากกว่า 1 ตัว ให้ใช้เครื่องหมาย , คั่นระหว่างชื่อของอินเทอร์เฟซ
- คลาสลูกต้อง implement เมท็อดนามธรรม เพื่อกำหนดส่วนของคำสั่งภายในเมท็อด หากคลาสลูกไม่ได้ implement เมท็อดนามธรรมของอินเทอร์เฟซ คลาสลูกนั้นจะต้องเป็นคลาสนามธรรม นั่นคือที่บรรทัดแรกของคลาสลูกต้องระบุคำว่า abstract ไว้ด้วย




รูปที่ 8.10 ความสัมพันธ์แบบ IS-A relationship ของอินเทอร์เฟซ Payable

ในการเรียกความสัมพันธ์ระหว่างอินเทอร์เฟซและคลาสนั้น ก็ยังเป็นเหมือนความสัมพันธ์ระหว่างคลาสทั่วไป นั่นคือ มีความสัมพันธ์แบบ IS-A relationship รูปที่ 8.10 จะใช้สำหรับตัวอย่างการสร้างอินเทอร์เฟซและคลาสลูกต่อไป โดยเหตุผลที่สร้างอินเทอร์เฟซขึ้นมา ก็คือ ไม่ว่าจะเป็นคลาสของ Invoice หรือ Employee ซึ่งเป็นคลาสที่ไม่มีความสัมพันธ์กันเลย ต่างก็ต้องจัดการกับเรื่องของการคำนวณเงินที่ต้องจ่ายเหมือนกัน จึงตั้งเมท็อดgetPaymentAmount() เพื่อคิดเงินที่ต้องจ่ายขึ้นมาเป็นเมท็อดร่วมเก็บไว้ในอินเทอร์เฟซ Payable และให้คลาส Invoice และ Employee ซึ่งเป็นคลาสลูกรับทอดจากอินเทอร์เฟซและ implement เมท็อดนามธรรมนี้ให้ตรงกับการใช้งานของตน นั่นคือของใบ Invoice จะเป็นการคำนวณยอดเงินที่ต้องชำระในการสั่งสินค้า ของ Employee เป็นการคำนวณเงินเดือนหรือค่าตอบแทนที่พนักงานได้รับ

ตัวอย่าง ไฟล์โปรแกรม Payable.java

```
1  // Payable interface declaration.
2  public interface Payable {
3        // calculate payment; no implementation
4        double getPaymentAmount();
5  }
```

คำอธิบายโปรแกรม

- อินเทอร์เฟซ Payable เขียนประกาศให้เป็นอินเทอร์เฟซด้วยการระบุคำว่า interface กำกับไว้หน้าชื่ออินเทอร์เฟซ
- ภายในอินเทอร์เฟซนี้มีเมท็อดนามธรรม ชื่อ getPaymentAmount() โดยสังเกตว่าสามารถที่จะไม่เขียน public abstract ได้

ตัวอย่าง ไฟล์โปรแกรม Invoice.java

```
1   public class Invoice implements Payable  {
2      private String partNumber;
3      private String partDescription;
4      private int quantity;
5      private double pricePerItem;
6      public Invoice(String part, String description, int count, double price){
7         partNumber = part;
8         partDescription = description;
9         setQuantity( count );
10        setPricePerItem( price );
11     }
12     public void setPartNumber( String part )   {
```

```
13          partNumber = part;
14       }
15       public String getPartNumber()    {
16          return partNumber;
17       }
18       public void setPartDescription( String description )    {
19          partDescription = description;
20       }
21       public String getPartDescription()    {
22          return partDescription;
23       }
24       public void setQuantity( int count )    {
25          quantity = ( count < 0 ) ? 0 : count;
26       }
27       public int getQuantity()    {
28          return quantity;
29        }
30       public void setPricePerItem( double price )    {
31          pricePerItem = ( price < 0.0 ) ? 0.0 : price;
32       }
33       public double getPricePerItem()    {
34          return pricePerItem;
35       }
36       public String toString()    {
37          return "invoice: \npart number: " + getPartNumber() + "(" +
                  getPartDescription() + ")\nquantity: " +
                  getQuantity() + "\nprice per item: " + getPricePerItem();
38       }
39       /* method required to carry out contract with interface Payable */
40       public double getPaymentAmount() {
41          return getQuantity() * getPricePerItem(); // calculate total cost
42       }
43    }
```

คำอธิบายโปรแกรม

- คลาส Invoice รับทอดจากอินเทอร์เฟซ Payable ที่หัวคลาสจะเขียนให้มัน **implements** Payable
- ภายในคลาสมีการกำหนดตัวแปรและเมท็อดที่จำเป็นสำหรับการทำงาน นอกจากนี้ ยังมีการ override toString() เพื่อแสดงค่าของข้อมูลต่าง ๆ และ implement เมท็อดนามธรรม getPaymentAmount() โดยหัวเมท็อดเป็นเหมือนในอินเทอร์เฟซแต่ตัดคำว่า abstract ทิ้งไป และกำหนดคำสั่งเพื่อคำนวณจำนวนเงินที่ต้องจ่าย โดยคิดจาก จำนวนสินค้า คูณกับราคาต่อหน่วย

ตัวอย่าง ไฟล์โปรแกรม Employee.java

```
1   public abstract class Employee implements Payable {
2      private String firstName;
3      private String lastName;
4      private String empID;
5      public Employee( String first, String last, String ID )    {
6         firstName = first;
7         lastName = last;
8         empID = ID;
9      }
10     public void setFirstName( String first ) {
11        firstName = first;
12     }
13     public String getFirstName()    {
```

```
14          return firstName;
15       }
16       public void setLastName( String last )   {
17          lastName = last;
18       }
19       public String getLastName()   {
20          return lastName;
21       }
22       public void setEmpID( String ID )   {
23          empID = ID;
24       }
25       public String getEmpID()    {
26          return empID;
27       }
28       public String toString()    {
29          return getClass().getName()+ ":" +getFirstName()+ " " +
                       getLastName() + "\nID :" + getEmpID();
30       }
31
32       /* Note: We do not implement Payable method getPaymentAmount here.
33          So, this class must be declared abstract. */
34    }
```

คำอธิบายโปรแกรม

- คลาส Employee รับทอดจากอินเทอร์เฟซ Payable ที่หัวคลาสจะเขียนให้มัน **implements** Payable
- ภายในคลาสมีการกำหนดตัวแปรและเมท็อดที่จำเป็นสำหรับการทำงาน มีการ override toString() เพื่อแสดงค่าของข้อมูลต่าง ๆ แต่ยังไม่ implement เมท็อดนามธรรม getPaymentAmount() ทำให้คลาส Employee นี้ยังคงเป็นคลาสนามธรรมต่อไป ที่หัวคลาสจึงต้องเขียนคำว่า abstract กำกับไว้ด้วย

ตัวอย่าง ไฟล์โปรแกรม SalariedEmployee.java

```
1   public class SalariedEmployee extends Employee  {
2      private double weeklySalary;
3      public SalariedEmployee (String first, String last, String ID,
                              double salary ) {
4         super( first, last, ID );
5         setWeeklySalary ( salary);
6      }
7      public void setWeeklySalary(double salary)  {
8         weeklySalary = salary < 0.0 ? 0.0 : salary;
9      }
10     public double getWeeklySalary()  {
11        return weeklySalary;
12     }
13     /* calculate earnings; implement interface Payable method that was
14        abstract in superclass Employee */
15     public double getPaymentAmount()   {
16        return getWeeklySalary();
17     }
18     public String toString()  {
19        return super.toString() + "\nweekly salary:" + getWeeklySalary();
20     }
21  }
```

คำอธิบายโปรแกรม

- คลาส SalariedEmployee รับทอดจากคลาสแม่ Employee ที่หัวคลาสจะเขียนให้มัน **extends** Employee
- ภายในคลาสมีการกำหนดตัวแปรและเมท็อดที่จำเป็นสำหรับการทำงานเพิ่มเติม นอกจากนี้ ยังมีการ override toString() เพื่อแสดงค่าของข้อมูลต่าง ๆ และ implement เมท็อดนามธรรม getPaymentAmount() โดยหัวเมท็อดเป็นเหมือนในอินเทอร์เฟซแต่ตัดคำว่า abstract ทิ้งไป และกำหนดคำสั่งเพื่อคำนวณจำนวนเงินที่ต้องจ่าย ซึ่งในที่นี้คือเงินเดือนที่ต้องจ่ายให้พนักงาน

ตัวอย่าง ไฟล์โปรแกรม PayableInterfaceTest.java

```
1  public class PayableInterfaceTest  {
2     public static void main(String args[])  {
3        Payable payableObjects[] = new Payable[4];
4        payableObjects[0] = new Invoice("01234", "seat", 2, 375.00);
5        payableObjects[1] = new Invoice("56789", "tire", 4, 79.95);
6        payableObjects[2] = new SalariedEmployee ("John", "Smith",
                             "111- 11-1111", 800.00);
7        payableObjects[3] = new SalariedEmployee ("Lisa", "Barnes",
                               "888-88-8888", 1200.00);
8        System.out.println("Invoices and Employees processed
                            polymorphically:\n");
9  //generically process each element in array payableObjects
10       for (int i = 0; i < payableObjects.length; i++)  {
11           System.out.println(payableObjects[i].toString() + "\n" +
              "payment due : " + payableObjects[i].getPaymentAmount() + "\n");
12       }
13    }
14 }
```

คำอธิบายโปรแกรม

- คลาส PayableInterfaceTest นี้กำหนดอาร์เรย์เพื่อเก็บอ็อบเจกต์ที่สร้างจากคลาส Invoice และ SalariedEmployee จากนั้นวนลูปเพื่อแสดงค่าข้อมูลของทุกอ็อบเจกต์ในอาร์เรย์ด้วยการใช้คำสั่งเรียกเมท็อดแบบ polymorphism

ผลลัพธ์ที่ได้จากโปรแกรมคือ

Invoices and Employees processed polymorphically:

invoice:

part number: 01234(seat)

quantity: 2

price per item: 375.0

payment due: 750.0

invoice:

part number: 56789(tire)

quantity: 4

price per item: 79.95

payment due: 319.8

payableinterface.SalariedEmployee:John Smith
ID : 111-11-1111
weekly salary:800.0
payment due: 800.0

payableinterface.SalariedEmployee:Lisa Barnes
ID : 888-88-8888
weekly salary:1200.0
payment due: 1200.0

## 8.9 ความสัมพันธ์กับวิชาอื่นที่เกี่ยวข้อง

ในรายวิชา 2301479 การวิเคราะห์และออกแบบเชิงวัตถุ (Object-Oriented Analysis and Design) จะศึกษาถึงการวิเคราะห์และออกแบบระบบเชิงวัตถุ นิสิตจำเป็นต้องมีความรู้พื้นฐานในเรื่องของคลาสและอ็อบเจกต์ และการสืบทอดคุณสมบัติมาก่อน นอกจากนี้ในการพัฒนาโปรแกรมต่าง ๆ ในรายวิชาโครงสร้างข้อมูล โครงงานวิทยาศาสตร์ นิสิตมักจะต้องใช้กลุ่มคลาสมาตรฐานของภาษาจาวา หรือคลาสที่หน่วยงานหรือองค์กรสร้างไว้แล้ว ซึ่งอาจเป็นการเรียกใช้คลาสเหล่านั้นโดยตรงหรือต้องสร้างคลาสลูกจากคลาสเหล่านั้นเพื่อให้สามารถทำงานได้ตรงตามวัตถุประสงค์ของโปรแกรมที่พัฒนา

## แบบฝึกหัดท้ายบท

1. จงสร้าง superclass Employee และ subclass Salesman และ Secretary
   - Employee มีข้อมูลคือ ชื่อ (String) ปีที่เริ่มงาน (int) เงินเดือน (double) และมีเมท็อดคือ getName() ซึ่งคืนค่าชื่อและสกุลออกมา getStartYear() ซึ่งคืนค่าปีที่เริ่มงานออกมา และ getSalary() ซึ่งคืนค่าเงินเดือนออกมา
   - Salesman มีข้อมูลเพิ่มเติมคือ ยอดขาย (double) อัตราคอมมิชชัน (double) และมีเมท็อดคือ getSalary() ซึ่งคืนค่าเงินเดือนรวมกับค่าคอมมิชชันที่ได้
   - Secretary มีข้อมูลเพิ่มเติมคือ ความเร็วในการพิมพ์ดีด (int) หน่วยเป็นคำ/นาที และมีเมท็อดคือ getTyping() ซึ่งคืนค่าความเร็วในการพิมพ์ดีดออกมา

   ให้นิสิตสร้างคลาสเหล่านี้โดยกำหนดตัวสร้างให้เหมาะสม และสร้าง Test class เพื่อทดสอบคลาสที่สร้างขึ้น โดยสร้างอ็อบเจกต์จากคลาสดังกล่าว คลาสละ 1 อ็อบเจกต์ และลองเรียกใช้เมท็อดต่าง ๆ เพื่อแสดงข้อมูลทั้งหมดของแต่ละอ็อบเจกต์

2. จากโจทย์ในข้อ 1 หากกำหนดคำสั่งใน test class ดังนี้

```
Employee e = new Employee("Sasipa Pant", 2000, 25000);
Salesman s = new Salesman("Somying Meejai", 2005, 12500, 150000,
                    0.05);
Secretary c = new Secretary("Somjai Deejing", 2008, 20000, 60);
Employee ec = c;
System.out.println("call getName() from ec = " + ec.getName());
System.out.println("call getTyping() from ec = " + ec.getTyping());
Salesman se = e;
System.out.println("call getName() from se = " + se.getName());
System.out.println("call getSalary() from se = " + se.getSalary());
```

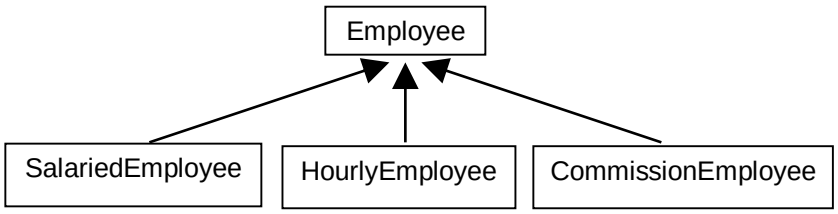
   คำสั่งมีข้อผิดพลาดหรือไม่ ถ้ามี จงบอกว่าผิดที่ใด เพราะเหตุใด และจะแก้ให้ถูกต้องได้อย่างไร แต่หากไม่ผิดพลาด จงแสดงผลลัพธ์

3. จากโจทย์ในข้อ 1 หากกำหนดคำสั่งใน test class ดังนี้

```
Salesman s = new Salesman("Somying Meejai", 2005, 12500, 150000,
                    0.05);
Employee es = s;
System.out.println("call getName() from es = " + es.getName());
System.out.println("call getSalary() from es = " + es.getSalary());
Salesman se = (Salesman) es;
System.out.println("call getName() from se = " + se.getName());
System.out.println("call getSalary() from se = " + se.getSalary());
```

   คำสั่งมีข้อผิดพลาดหรือไม่ ถ้ามี จงบอกว่าผิดที่ใด เพราะเหตุใด และจะแก้ให้ถูกต้องได้อย่างไร แต่หากไม่ผิดพลาด จงแสดงผลลัพธ์

4. จงสร้างคลาสต่าง ๆ ตาม inheritance hierarchy ที่กำหนด โดยให้มี method ต่าง ๆ ตามความเหมาะสม กำหนดให้ Employee เป็น abstract superclass ที่มีเมท็อดนามธรรม ชื่อ earnings และให้ทุกคลาส override method toString()




ดูตารางข้อมูลประกอบดังนี้

| | **Earnings** | **toString()** |
|---|---|---|
| Employee | Abstract | firstName, lastName, empID |
| SalariedEmployee | Return weeklySalary | firstName, lastName, empID, weeklySalary |
| HourlyEmployee | If hours <= 40<br>Return wage*hours<br>Else<br>Return 40*wage+(hours-40)*wage*1.5 | firstName, lastName, empID, wage, hours |
| CommissionEmployee | Return commRate*grossSales | firstName, lastName, empID, grossSales, commRate |

และให้นิสิตสร้าง test class โดยกำหนดอาร์เรย์ลิสต์เพื่อเก็บข้อมูลชนิด Employee แล้วสร้างอ็อบเจกต์ของ SalariedEmployee, HourlyEmployee, CommissionEmployee อย่างละหนึ่งอ็อบเจกต์เก็บในอาร์เรย์ลิสต์ จากนั้นเขียนโปรแกรมด้วยเทคนิค polymorphism เพื่อเรียกใช้เมท็อด toString() และ earnings() เพื่อแสดงข้อมูลของแต่ละอ็อบเจกต์เป็นผลลัพธ์ (ให้นิสิตลองเขียนโปรแกรมโดยใช้อาร์เรย์เพื่อเก็บอ็อบเจกต์แทนการใช้อาร์เรย์ลิสต์ด้วย)

5. จงสร้างคลาสต่าง ๆ ดังนี้
   - interface Shape ซึ่งมี 2 เมท็อด คือ findArea() และ findCircum()
   - Class Circle ซึ่งมี instance variable คือ radius (รัศมี) และ implement findArea() และ findCircum()
   - Class Rectangle ซึ่งมี instance variable คือ width (ความกว้าง), length (ความยาว) และ implement findArea() และ findCircum()
   - Class test ซึ่งใช้อาร์เรย์เก็บข้อมูลชนิด Shapes เพื่อเก็บอ็อบเจกต์ที่สร้างจากคลาส Circle และ Rectangle แล้วเรียกใช้เมท็อดทั้งสองด้วยเทคนิคของ polymorphism

6. จากโจทย์ในข้อ 1 จงสร้าง interface ชื่อ Comparable ให้มีเมท็อดชื่อ compareTo ซึ่งรับพารามิเตอร์เป็น Object o และปรับปรุงคลาส Salesman และ Secretary ให้ implements interface Comparable

- ใน Salesman ให้ implement method compareTo ให้มีการเปรียบเทียบค่า commRate ของอ็อบเจกต์ กับอ็อบเจกต์อื่นที่ส่งผ่านมาทางพารามิเตอร์ว่าเท่ากันหรือไม่ ถ้าเท่าให้ return 0 ถ้าน้อยกว่าให้ return -1 ถ้ามากกว่าให้ return 1
- ใน Secretary ให้ implement method compareTo ให้มีการเปรียบเทียบค่า typing ของอ็อบเจกต์ กับ อ็อบเจกต์อื่นที่ส่งผ่านมาทางพารามิเตอร์ว่าเท่ากันหรือไม่ ถ้าเท่าให้ return 0 ถ้าน้อยกว่าให้ return -1 ถ้ามากกว่าให้ return 1

กำหนด test class ดังนี้

```
public class Main {
   public static void main(String[] args) {
       Salesman s1 = new Salesman("Somying Meejai", 2005, 12500,
                                  150000, 0.05);
       Salesman s2 = new Salesman("Somsak Pakdee", 2000, 14500,
                                  350000, 0.08);
       Secretary c1 = new Secretary("Somjai Deejing", 2008,
                                  20000, 60);
       Secretary c2 = new Secretary("Sompon Deejai", 2003,
                                  25000, 60);
       if (s1.compareTo(s2) == 0)
         System.out.println("Somying and Somsak has same commRate");
       else if (s1.compareTo(s2) < 0)
         System.out.println("Somying has less commRate than Somsak");
       else
         System.out.println("Somying has greater commRate than
                     Somsak");
       if (c1.compareTo(c2) == 0)
         System.out.println("Somjai and Sompon has same typing
                     rate");
       else if (c1.compareTo(c2) < 0)
         System.out.println("Somjai has less typing rate than
                     Sompon");
       else
         System.out.println("Somjai has greater typing rate
                     than Sompon");
   }
}
```