บทที่ ๔ คลาส เมท็อด และ อ็อบเจกต์

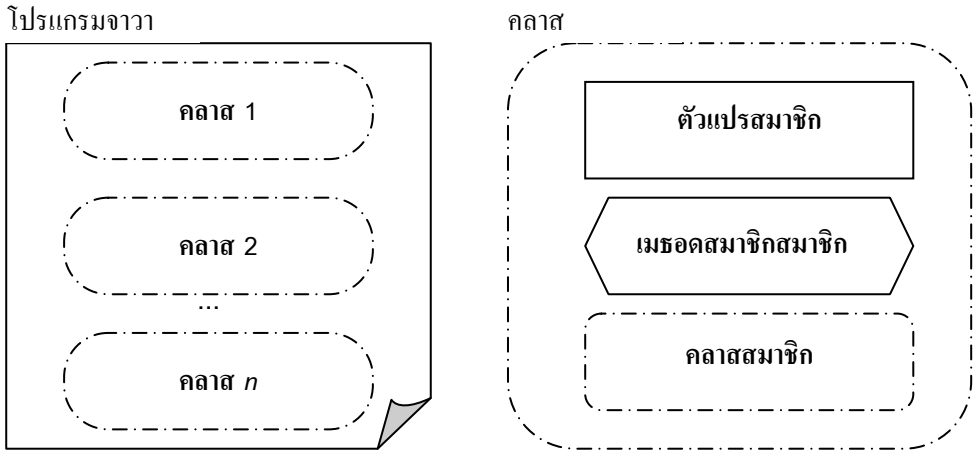
ในบทนี้จะกล่าวถึงวิธีสร้างอ็อบเจกต์ โดยตอนแรกจะกล่าวถึงการสร้างอ็อบเจกต์จากคลาสที่มีให้ใช้อยู่แล้วเพื่อเรียนรู้วิธีการสร้างอ็อบเจกต์อย่างรวดเร็ว ต่อจากนั้นจึงจะกล่าวถึงวิธีเขียนคลาสใหม่ขึ้นใช้เอง ซึ่งคลาสใหม่ที่เราเขียนขึ้นนี้ จะใช้เป็นแม่แบบในการสร้างอ็อบเจกต์ให้มีสมบัติและพฤติกรรมตามที่เราต้องการ ส่วนการที่อ็อบเจกต์จะมีพฤติกรรมใดนั้น ขึ้นอยู่กับ*เมท็อด(method)*ต่าง ๆ ของคลาส หรือกล่าวอีกนัยหนึ่งก็คือขึ้นอยู่กับข้อความสั่งต่าง ๆ ที่เราเขียนขึ้นมาและบรรจุไว้ในคลาสเมท็อดนั่นเอง ในบทนี้จะกล่าวถึงการเขียนคลาสขั้นพื้นฐานเพื่อให้ผู้เรียนสามารถนำคลาสที่เขียนขึ้นไปสร้างอ็อบเจกต์ให้ทำงานอย่างง่าย อันเป็นพื้นฐานในการเรียนบทต่อ ๆ ไป ส่วนรายละเอียดที่มากขึ้นและความรู้ขั้นสูงเกี่ยวกับคลาสและอ็อบเจกต์จะกล่าวเพิ่มเติมในบทที่ ๕ และ ๘

โครงสร้างของคลาส

คลาสถือว่าเป็นแม่แบบที่ใช้ในการสร้างอ็อบเจกต์ คลาสหนึ่ง ๆ สามารถนำไปใช้สร้างอ็อบเจกต์ได้หลายตัว อ็อบเจกต์แต่ละตัวที่สร้างมาจากคลาสเดียวกันจะมีพฤติกรรมตามแม่แบบคือคลาสของมัน อ็อบเจกต์ที่สร้างมาจากคลาสเดียวกันอาจมีสมบัติบางอย่างที่แตกต่างกันไปได้บ้าง บางอ็อบเจกต์อาจมีลักษณะเฉพาะบางอย่างเป็นของตนเอง แต่สมบัติโดยรวมของอ็อบเจกต์ส่วนใหญ่แล้วใกล้เคียงกัน

ในการเขียนโปรแกรมจาวานั้น เราจะเขียนคลาสขึ้นมาเพื่ออธิบายอ็อบเจกต์ที่มีลักษณะและพฤติกรรมแบบเดียวกัน ถ้าอ็อบเจกต์ที่ต้องการมีสมบัติและพฤติกรรมที่แตกต่างกันกันออกไปค่อนข้างมาก เรานิยมเขียนแยกไว้เป็นคลาสอื่น คนละคลาสกัน

โปรแกรมต้นฉบับภาษาจาวาประกอบด้วยคลาสจำนวนหนึ่งเขียนรวมกันไว้ในแฟ้มโปรแกรมต้นฉบับเดียวกัน ในแฟ้มโปรแกรมต้นฉบับหนึ่ง ๆ อาจบรรจุคลาสเพียงคลาสเดียวหรือหลายคลาสก็ได้ ในแต่ละคลาส



รูปที่ ๔-๑ แสดงโครงสร้างของโปรแกรมจาวา และคลาส

อาจประกอบด้วยสมาชิกเพียงประเภทใดประเภทหนึ่งหรือหลายประเภทจากสมาชิกประเภทต่าง ๆ ๓ ประเภท ต่อไปนี้คือตัวแปรสมาชิก(member variable) เมท็อดสมาชิก(member method) และ คลาสสมาชิก(member class) โดยที่สมาชิกแต่ละประเภทสามารถมีอยู่ในคลาสได้เพียงหนึ่งเดียวหรือมากกว่าหนึ่ง หรืออาจไม่มีเลยก็ได้ โครงสร้างของโปรแกรมและคลาสแสดงได้ดังรูปที่ ๔-๑

ตัวแปรสมาชิก เป็นตัวแปรที่กำหนดไว้ในคลาสสำหรับเก็บสมบัติ (property) ของอ็อบเจกต์ ตัวแปรสมาชิกยังแบ่งออกเป็น ๒ ชนิดคือ *ตัวแปรอินสแทนซ์ (instance variable)* และ*ตัวแปรคลาส (class variable)* ตัวแปรอินสแทนซ์ถือว่าเป็นตัวแปรของแต่ละอ็อบเจกต์จะเกิดขึ้นพร้อมกับการสร้างอ็อบเจกต์ใหม่ จะมีอยู่ในทุกอ็อบเจกต์ที่สร้างขึ้นมาและอยู่ไปตลอดจนกว่าอ็อบเจกต์นั้นจะไม่ใช้งานและถูกทำลายไป ดังนั้นถ้าสร้างอ็อบเจกต์หลายตัวขึ้นมาจากคลาสเดียวกัน จะมีตัวแปรอินสแทนซ์อยู่ในทุกอ็อบเจกต์โดยที่ตัวแปรในอ็อบเจกต์เหล่านี้จะเป็นอิสระต่อกัน อ็อบเจกต์ใครอ็อบเจกต์มัน ส่วนตัวแปรคลาสไม่ได้เก็บไว้ในอ็อบเจกต์แต่จะเก็บอยู่ในคลาส ตัวแปรคลาสนี้จะมีตัวตนเกิดขึ้นพร้อมกับการโหลดคลาสเข้าสู่หน่วยความจำ ตัวแปรชนิดนี้แต่ละชื่อที่ประกาศไว้จะมีเพียงตัวเดียวอยู่ในพื้นที่หน่วยความจำของคลาสเท่านั้น ต่างจากตัวแปรอินสแทนซ์ที่เกิดขึ้นหลายชุดตามจำนวนอ็อบเจกต์ที่สร้างขึ้น ตัวแปรคลาสจะอยู่ไปตลอดตราบใดที่คลาสของมันยังทำงานอยู่ในหน่วยความจำ ตัวแปรคลาสจะมีเพียงชุดเดียวอยู่คู่กับคลาส อ็อบเจกต์ต่าง ๆ ของคลาสเดียวกันจะใช้ตัวแปรคลาสที่มีอยู่เพียงที่เดียวนี้ร่วมกัน เนื่องจากว่าตัวแปรคลาสมีตัวตนเกิดขึ้นเมื่อมีการนำ (load) คลาสเข้าสู่หน่วยความจำ จึงสามารถเข้าถึงตัวแปรชนิดนี้ได้โดยที่ไม่ต้องสร้างอ็อบเจกต์ขึ้นมาก็ได้

หมายเหตุ

*ตัวแปรสมาชิก*เป็นคำที่ใช้กันอยู่เป็นการทั่วไปในภาษาโปรแกรมเชิงวัตถุ แต่ในเอกสารของบริษัทซันไมโครซิสเต็มส์ใช้คำ *เขตข้อมูล (field)* แทนคำว่าตัวแปรสมาชิก เพื่อแยกออกให้ชัดจากตัวแปรทั่วไป

- เมท็อดสมาชิก เป็นที่รวมของชุดคำสั่งที่กำหนดไว้ในคลาสสำหรับแสดงพฤติกรรมหรือความสามารถบางอย่าง เมท็อดสมาชิกแบ่งออกได้ ๒ ชนิดทำนองเดียวกับตัวแปรสมาชิกคือ *อินสแทนซ์เมท็อด (instance method)* และ*คลาสเมท็อด(class method)* อินสแทนซ์เมท็อดถือว่าเกาะอยู่กับอ็อบเจกต์ต้องสร้างอ็อบเจกต์ขึ้นมาก่อนจึงจะเรียกใช้เมท็อดชนิดนี้ได้ เวลาเรียกใช้ต้องบอกด้วยว่าเรียกใช้อินสแทนซ์เมท็อดใด ส่วนคลาสเมท็อดถือว่าเกาะอยู่กับคลาสของมัน สามารถเรียกใช้ได้โดยไม่ต้องสร้างอ็อบเจกต์

- คลาสสมาชิก เป็นคลาสซ้อนที่เขียนไว้ภายในคลาสอื่นที่ห่อหุ้มมัน บริษัทซันไมโครซิสเต็มได้ปรับปรุงข้อกำหนดของภาษาจาวาให้มีคลาสสมาชิกได้ตั้งแต่รุ่นที่ 1.1 เป็นต้นมา ในบทนี้จะยังไม่อธิบายประโยชน์และวิธีการเขียนคลาสสมาชิกแต่จะกล่าวถึงโดยละเอียดในบทที่ ๘

ในการสร้างคลาสนั้น สามารถเขียนสมาชิกทั้ง ๓ ประเภทนี้อย่างละกี่ตัวก็ได้ ประเภทใดมาก่อนหรือหลังก็ได้ไม่มีกฎเกณฑ์แต่อย่างใด แต่ควรวางในตำแหน่งที่ดูเข้าใจง่าย สมาชิกทั้ง ๒ ประเภทแรกจะกล่าวถึงในบทนี้ ส่วนคลาสสมาชิกจะกล่าวถึงในบทอื่น

การสร้างอ็อบเจกต์

ในการเขียนโปรแกรมจาวานั้นเรามักกำหนดชุดของคลาสต่าง ๆ ขึ้นมา คลาสเหล่านี้จะถูกใช้เป็นแม่แบบในการสร้างอ็อบเจกต์ ในการเขียนโปรแกรมจาวาเราใช้คลาสที่มีการเขียนกำหนดเอาไว้ล่วงหน้าแล้ว ในตอนแรกของบทนี้ เราจะเรียนรู้วิธีสร้างอ็อบเจกต์ใหม่(instantiation)โดยใช้คลาสจากคลังคลาส(class library) ที่มีให้ใช้อยู่แล้วเป็นแม่แบบ ต่อจากนั้นจึงจะเรียนรู้วิธีสร้างคลาสขึ้นใช้เอง

นิพจน์สร้างอ็อบเจกต์ใหม่

ในโปรแกรมจาวานั้น อ็อบเจกต์ต่าง ๆ ไม่ได้เกิดขึ้นเองโดยอัตโนมัติ นักเขียนโปรแกรมเป็นผู้สั่งว่าจะสร้างอ็อบเจกต์เมื่อใด ในการสร้างอ็อบเจกต์ใหม่แต่ละครั้ง เราใช้นิพจน์ซึ่งประกอบด้วยคำสำคัญ `new` ควบคู่กับชื่อคลาสที่เราต้องการให้เป็นแม่แบบในการสร้างอ็อบเจกต์ ตามด้วยวงเล็บเปิดและวงเล็บปิด ดังตัวอย่าง

```
new  Random()
```

นิพจน์นี้แสดงการสร้างอ็อบเจกต์ของคลาส `Random` วงเล็บที่ใส่ไว้หลังชื่อคลาสมีความสำคัญเป็นอย่างยิ่ง ไม่ใส่ไม่ได้ ภายในวงเล็บสามารถเว้นว่างไว้ได้ หรือบางครั้งภายในวงเล็บอาจบรรจุ*อาร์กิวเมนต์(argument)*หรือ*พารามิเตอร์(parameter)* ซึ่งใช้เป็นค่าเริ่มต้นให้กับตัวแปรหรือสมบัติเริ่มต้นของอ็อบเจกต์ที่เราต้องการสร้าง อาร์กิวเมนต์เป็นนิพจน์ เมื่อหาค่าผลลัพธ์ได้แล้ว ค่านั้นจะถูกส่งไปใช้ประกอบการสร้างอ็อบเจกต์ ตัวอย่างนี้ไม่มีอาร์กิวเมนต์แต่อย่างใด

หมายเหตุ

ตำราหลายเล่มเรียกคำสำคัญ new ว่าเป็นตัวดำเนินการ แต่ในเอกสารข้อกำหนดคุณลักษณะภาษาจาวาของบริษัทซันไมโครซิสเต็มส์ไม่จัด new ว่าเป็นตัวดำเนินการ แต่จัดว่าเป็นส่วนประกอบของนิพจน์สำหรับสร้างอ็อบเจกต์

ตัวอย่างต่อไปนี้เป็นตัวอย่างการสร้างอ็อบเจกต์ใหม่ โดยที่มีการรับอาร์กิวเมนต์ด้วย

```
new Random(123456);
new Point(0,0);
```

ตัวอย่างนี้แสดงการสร้างอ็อบเจกต์ของคลาส `Random` และ `Point` โดยใส่อาร์กิวเมนต์ไว้ในวงเล็บด้วย ตัวอย่างนี้เราใส่อาร์กิวเมนต์ให้คลาส `Point` ไว้ในวงเล็บจำนวน ๒ ตัว ถ้ามีอาร์กิวเมนต์มากกว่า ๑ ตัว จะต้องคั่นอาร์กิวเมนต์แต่ละตัวด้วยจุลภาค (,) จำนวนและชนิด(type)ของอาร์กิวเมนต์ที่สามารถใส่ไว้ในวงเล็บได้นั้นถูกกำหนดโดยคลาสของมันโดยใช้*เมท็อด(method)*พิเศษที่เรียกว่า*ตัวสร้าง* (constructor) ซึ่งจะกล่าวถึงภายหลังในบทนี้ ถ้าเราเขียนโปรแกรมสร้างอ็อบเจกต์ใหม่ของคลาสใดคลาสหนึ่งด้วยจำนวนและชนิดของอาร์กิวเมนต์ที่ต่างไปจากที่กำหนดไว้ในคลาสนั้น ๆ คอมไพเลอร์จะไม่ยอมปล่อยให้ผ่าน มันจะแสดงข้อความแจ้งถึงข้อผิดพลาดที่เกิดขึ้น

การดำเนินงานของนิพจน์สร้างอ็อบเจกต์ใหม่

เมื่อมีการสร้างอ็อบเจกต์ใหม่ด้วยนิพจน์สำหรับสร้างอ็อบเจกต์นั้น จะมีการดำเนินการหลายอย่างเกิดขึ้น เริ่มตั้งแต่มีการจัดสรรที่ว่างในหน่วยความจำเพื่อใช้เป็นที่อยู่ของอ็อบเจกต์ใหม่ที่จะสร้างขึ้น ถ้าที่ว่างในหน่วยความจำไปเพียงพอสำหรับสร้างอ็อบเจกต์จะเกิดข้อผิดพลาดที่เรียกว่า `OutOfMemoryError` โปรแกรมจะยุติการทำงานแบบไม่ปรกติ แต่ถ้าสามารถหาที่ว่างในหน่วยความจำได้เพียงพอ จะมีการสร้างอ็อบเจกต์ลงในที่ว่างที่จัดสรรมาได้ ตัวแปรต่าง ๆ ของออปเจ็กท์ที่มีการกำหนดไว้ในคลาสจะได้รับการบรรจุลงในที่ว่างนี้และถูกกำหนดค่าเริ่มต้นตามค่าโดยปริยายของตัวแปรแต่ละชนิด ตัวอย่างเช่น ถ้าเป็นตัวแปรชนิดตัวเลขจะได้รับการกำหนดค่าเริ่มต้นเป็นศูนย์ ต่อจากนั้นเมท็อดพิเศษที่เรียกว่า*ตัวสร้าง*ที่กำหนดไว้ในคลาสจะถูกเรียกให้ทำงาน

ตัวสร้าง (*constructor*) เป็นเมท็อดพิเศษมีชื่อเหมือนชื่อคลาสใช้สำหรับสร้างและกำหนดการเริ่มต้นให้กับอ็อบเจกต์ใหม่ ตัวสร้างจะกำหนดค่าเริ่มต้นให้ตัวแปรอินสแทนซ์ใหม่ที่สร้างขึ้นมา หรืออาจสร้างอ็อบเจกต์ใหม่อื่น ๆ ที่อ็อบเจกต์นั้นต้องการ และกระทำการอื่นใดที่อ็อบเจกต์นั้นจำเป็นต้องทำเพื่อเริ่มต้นตัวมันเอง

คลาสหนึ่ง ๆ อาจมีตัวสร้างได้หลายตัวแต่ต้องมีจำนวนและชนิดของอาร์กิวเมนต์ที่แตกต่างกัน ในนิพจน์สร้างอ็อบเจกต์ใหม่เราสามารถระบุอาร์กิวเมนต์ที่แตกต่างกันนี้ในรายการอาร์กิวเมนต์ ข้อแตกต่างของชนิดหรือจำนวนอาร์กิวเมนต์นี่เองจะเป็นตัวกำหนดว่าตัวสร้างตัวใดที่ถูกเรียกให้ทำงาน ตัวอย่างเช่นคลาส `Random` ที่กล่าวถึงมาแล้วมีตัวสร้างหลายตัว ตัวสร้างตัวหนึ่งไม่ต้องการอาร์กิวเมนต์ แต่ตัวสร้างอีกตัวต้องการอาร์กิวเมนต์จำนวน ๑ ตัว ดังนั้น ในการสร้างอ็อบเจกต์ของคลาส `Random` ด้วยนิพจน์ข้างต้นจึงใช้ได้ทั้ง ๒ กรณี ในกรณีที่เราสร้างคลาสของเราขึ้นใช้เอง เราสามารถกำหนดให้มีตัวสร้างกี่ตัวก็ได้เท่าที่จำเป็น

การอ้างอิงถึงอ็อบเจกต์

ในการสร้างอ็อบเจกต์ใหม่นั้น จะมีการหาที่ว่างในหน่วยความจำเพื่อใช้เป็นที่อยู่ของอ็อบเจกต์ใหม่ที่กำลังสร้างขึ้น ซึ่งที่อยู่ของมันเราไม่รู้ล่วงหน้าว่าจะได้ที่ใด ในการใช้โปรแกรมแต่ละครั้งไม่จำเป็นว่าอ็อบเจกต์ที่สร้างขึ้นจะได้อยู่ตำแหน่งเดิมเสมอไป และอาจมีการสร้างอ็อบเจกต์ของคลาสเดียวกันหลายตัวก็ได้ ดังนั้น ในการส่งสาร(message)หรือติดต่อกับอ็อบเจกต์ เราจำเนต้องรู้ว่าเราจะส่งสารให้อ็อบเจกต์ตัวใด ซึ่งเราจำเป็นต้องรู้ที่อยู่ของอ็อบเจกต์นั้น เพื่อที่จะได้ติดต่อได้ถูกตัว ในทางปฏิบัติเราไม่จำเป็นต้องรู้ที่อยู่ของอ็อบเจกต์โดยตรง จาวามีวิธีอำนวยความสะดวกในเรื่องนี้ เพื่อความสะดวกในการติดต่อกับอ็อบเจกต์เราควรสร้างตัวแปรขึ้นมาเพื่อเก็บที่อยู่ของอ็อบเจกต์ที่เราต้องการติดต่อด้วย ตัวแปรซึ่งเก็บข้อมูลเป็นที่อยู่ของข้อมูลหรืออ็อบเจกต์อื่นเรียกว่า*ตัวแปรอ้างอิง* *(reference variable)* ถ้าจะเรียกให้เจาะจงยิ่งขึ้น ตัวแปรสำหรับเก็บที่อยู่หรือใช้อ้างอิงถึงอ็อบเจกต์เรียกว่า*ตัวแปรอินสแทนซ์(object variable)* ในการอ้างถึงอ็อบเจกต์เราจะใช้ตัวแปรอ้างอิงหรือนิพจน์ที่ให้ผลลัพธ์ของค่าข้อมูลเป็นชนิดอ้างอิงในการอ้างถึงอ็อบเจกต์ที่เราต้องการติดต่อด้วย

ตัวอย่างต่อไปนี้แสดงการประกาศตัวแปรอ้างอิง

```
Random randNum;
```

การประกาศตัวแปรอ้างอิงมีรูปแบบทำนองเดียวกับการประกาศตัวแปรพื้นฐานทั่วไป ต่างกันแต่เพียงชนิดของข้อมูลเท่านั้น แทนที่จะเป็นชนิดข้อมูลพื้นฐาน ชนิดของข้อมูลจะเป็นคลาส คลาสจัดว่าเป็นชนิดประเภทหนึ่งที่มีความซับซ้อนมากว่าชนิดพื้นฐานทั่วไป ตัวอย่างนี้ Random เป็นชื่อคลาสซึ่งถือว่าเป็นชนิด ส่วน randNum เป็นตัวแปร ใช้ชื่อตามกฎการตั้งชื่อตัวระบุเช่นเดียวกับตัวแปรทั่วไป ตัวแปร randNum ถือว่าเป็นตัวแปรอ้างอิงเพราะว่ามันไม่ได้เก็บอ็อบเจกต์ไว้ในตัวของมัน แต่เก็บที่อยู่ของอ็อบเจกต์ที่มีชนิดเป็นคลาส Random เพื่อใช้ในการเข้าถึงอ็อบเจกต์โดยอ้อม ตัวแปร randNum เป็นตัวแปรที่ใช้ในการอ้างถึงอ็อบเจกต์ของคลาส Random ในบางครั้ง เพื่อความสะดวกและกระชับ เราจะเรียกชื่อตัวแปรอ้างอิงเสมือนว่าเป็นชื่ออ็อบเจกต์

เราไม่อาจกำหนดค่าอื่นใดให้กับตัวแปร(อ้างอิง)อ็อบเจกต์นอกจากที่อยู่ของอ็อบเจกต์ที่สร้างมาจากคลาสที่ตัวแปรอินสแทนซ์อ้างถึง เราอาจสร้างตัวแปรอินสแทนซ์ขึ้นมาก่อนที่จะสร้างอ็อบเจกต์ก็ได้ ในกรณีเช่นนี้ตัวแปรอินสแทนซ์จะยังไม่เก็บที่อยู่ของอ็อบเจกต์ใดจนกว่าเราจะนำค่าที่อยู่ไปกำหนดให้มัน ในกรณีที่เราต้องการให้ตัวแปรอินสแทนซ์ไม่อ้างอิงไปที่อ็อบเจกต์ใดเลย เราอาจกำหนดค่าของตัวแปรอ้างอิงให้เป็น null ซึ่งเป็นค่าคงที่ที่ใช้แทนความหมายว่าไม่ได้อ้างอิงไปยังอ็อบเจกต์ใด เช่น

```
randNum = null;
```

หลังจากทำข้อความสั่งนี้แล้วตัวแปรอินสแทนซ์ randNum จะมีค่าเป็น null ซึ่งหมายความว่าไม่ได้อ้างอิงไปที่ใด หากก่อนหน้านี้ randNum เก็บค่าที่อยู่ของอ็อบเจกต์ใดอยู่ randNum จะไม่อ้างถึงอ็อบเจกต์นั้นอีกต่อไป

โดยทั่วไปแล้ว เรามักจะใช้นิพจน์สร้างอ็อบเจกต์ใหม่ควบคู่กับตัวแปรอ้างอิงในข้อความสั่งกำหนดค่าดังตัวอย่าง

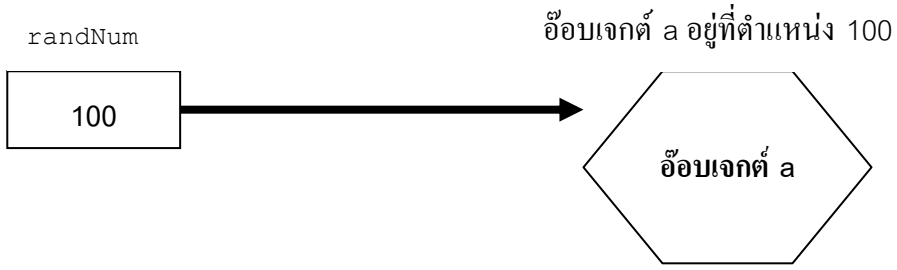
```
randNum = new Random();
```

ตัวอย่างนี้หมายความว่า เมื่อสร้างอ็อบเจกต์ของคลาส Random เสร็จเรียบร้อยแล้วให้เอาที่อยู่ของอ็อบเจกต์นั้นมาใส่ไว้ในตัวแปรอ้างอิง randNum

เพื่อความกระชับของโปรแกรมเรามักประกาศตัวแปรอ้างอิงพร้อม ๆ ไปกับการสร้างอ็อบเจกต์ ดังตัวอย่าง

```
Random randNum = new Random();
```

ตัวอย่างนี้มีการประกาศตัวแปรอ้างอิงชื่อว่า randNum มีชนิดเป็นคลาส Random พร้อมทั้งกำหนดค่าเริ่มต้นให้มีค่าเป็นที่อยู่ของอ็อบเจกต์ของคลาส Random ที่สร้างขึ้นใหม่รูปที่ ๔-๑ แสดงความสัมพันธ์ระหว่างตัวแปรอ้างอิงกับอ็อบเจกต์ สมมติว่าอ็อบเจกต์ a เป็นอ็อบเจกต์ที่สร้างมาจากคลาส Random ตั้งอยู่ในหน่วยความจำที่ตำแหน่ง 100 ตัวแปรอ้างอิง randNum จะเก็บค่าเลขที่อยู่ (address) ของอ็อบเจกต์ a ดังนั้นตัวแปร randNum จะมีค่าเป็น 100



รูปที่ ๔-๒ แสดงความสัมพันธ์ระหว่างตัวแปรอ้างอิงกับอ๊อบเจกต์

โดยปรกติแล้วนักเขียนโปรแกรมไม่จำเป็นต้องจดจำหรือรู้ค่าเลขที่อยู่ของอ็อบเจกต์ว่ามีค่าเป็นเท่าใด ระบบจะใช้วิธี "จดและจำ" ไว้ในตัวแปรอ้างอิง เวลาจะเรียกใช้อ็อบเจกต์ใด เราสามารถใช้อ็อบเจกต์นั้นได้โดยอาศัยตัวแปรอ้างอิงเป็นตัวเชื่อมหรือทางผ่าน จัดว่าเป็นการเข้าถึงอ็อบเจกต์ทางอ้อม

ตัวอย่างที่ ๔-๑ เป็นตัวอย่างโปรแกรมอย่างง่ายที่สมบูรณ์ทำงานได้จริง โปรแกรมนี้จะพิมพ์ค่าตัวเลขแบบสุ่มโดยอาศัยคลาส `Random` ซึ่งเป็นคลาสที่ให้มาพร้อมกับชุด JDK

ตัวอย่างโปรแกรมที่ ๔-๑ แสดงการสร้างอ็อบเจกต์อย่างง่าย

```
1.  import java.util.Random;
2.  class Ex4_1 {
3.    public static void main(String args[]) {
4.      Random randNum;
5.      randNum = new Random();
6.      System.out.println("Random double value : " +
                           randNum.nextDouble());
7.      System.out.println("Random integer value : " +
                           randNum.nextInt());
8.    }
9.  }
```

ในบรรทัดที่ ๑ มีคำสั่ง `import` สำหรับบอกคอมไพเลอร์ว่าเราต้องการใช้คลาสอะไรจากแพคเกจ (package)ใด แพคเกจเป็นที่รวมของคลาสต่าง ๆ ที่เตรียมไว้ใช้งานด้านใดด้านหนึ่ง ตัวอย่างนี้เราต้องการใช้คลาส `Random` จากแพคเกจ `java.util` เราจำเป็นต้องนำเข้า (import) คลาสนี้มาจากภายนอกเนื่องจากเราไม่ได้เขียนคลาส `Random` ขึ้นมาเองในโปรแกรมของเรา `java.util` เป็นชื่อแพคเกจหนึ่งในบรรดาแพคเกจต่าง ๆ ที่บริษัทซันไมโครซิสเต็มพัฒนาขึ้นและแจกจ่ายไปพร้อมกับชุด JDK

ในบรรทัดที่ ๔ มีการประกาศตัวแปรชื่อว่า `randNum` ให้เป็นตัวแปรอ้างอิงมีชนิดเป็นคลาส `Random` หมายความว่าเราจะใช้ตัวแปรนี้สำหรับอ้างอิงถึงอ็อบเจกต์ที่เป็นตัวอย่างหนึ่ง(instance)ของคลาส `Random` ในบรรทัดนี้เป็นเพียงประกาศตัวแปร จะได้ตัวแปรอ้างอิงหนึ่งตัวเท่านั้น ยังไม่มีการสร้างอ็อบเจกต์ ค่าของตัวแปร `randNum` ยังไม่ได้อ้างอิงไปยังอ็อบเจกต์ใด

บรรทัดที่ ๕ เป็นข้อความสั่งสำหรับกำหนดค่าให้กับตัวแปร `randNum` โดยนำค่าที่หาได้จากทางขวาของเครื่องหมายเท่ากับไปใส่ในตัวแปร `randNum` ทางด้านขวาของเครื่องหมายเท่ากับเป็นนิพจน์ที่

ประกอบด้วยตัวดำเนินการ new ซึ่งใช้สำหรับสร้างอ็อบเจกต์ อ็อบเจกต์ที่ได้นี้สร้างขึ้นมาโดยใช้คลาส Random เป็นแม่แบบ ระบบจะจัดหาที่ว่างในหน่วยความจำเพื่อเก็บข้อมูลในสำหรับอ็อบเจกต์ที่สร้างขึ้นใหม่นี้ ค่าเลขที่อยู่ของอ็อบเจกต์จะถูกนำไปใส่ไว้ในตัวแปร randNum เมื่อโปรแกรมทำงานมาถึงบรรทัดนี้จะได้อ็อบเจกต์ใหม่ ๑ ตัว

ในตอนท้ายของบรรทัดที่ ๖ มีการส่งสาร(message)ให้อ็อบเจกต์ การส่งสารให้อ็อบเจกต์ในภาษาจาวาคือการเรียกใช้เมท็อด(method) ใดเมท็อดหนึ่งของอ็อบเจกต์ การเรียกอินสแทนซ์เมท็อดใดจากอีกอ็อบเจกต์หนึ่งนั้นจำเป็นต้องอาศัยตัวแปรอ้างอิงเพื่อบอกให้รู้ว่าเราจะส่งสารไปให้อ็อบเจกต์ใด การเรียกเมท็อดโดยผ่านตัวแปรอ้างอิงเขียนได้ดังนี้

```
randNum.nextDouble()
```

nextDouble เป็นชื่อเมท็อดซึ่งมีการกำหนดไว้ในคลาส Random เราสามารถเรียก (call หรือ invoke) เมท็อดนี้ได้โดยการบอกชื่อเต็มของเมท็อด เนื่องจากอาจมีการสร้างอ็อบเจกต์ของคลาสเดียวกันขึ้นมาหลายตัว การเรียกเมท็อดโดยระบุแต่เพียงชื่อเมท็อดจะไม่เพียงพอเพราะเกิดความกำกวมไม่รู้ว่าจะเรียกอินสแทนซ์เมท็อดใดกันแน่ ดังนั้น การเรียกอ็อบเจกต์จากภายนอกอ็อบเจกต์ของตัวมันเองจึงจำเป็นต้องระบุชื่อเต็มของเมท็อดนั้น ชื่อเต็มที่สมบูรณ์ประกอบด้วยค่าอ้างอิงไปยังที่อยู่ของอ็อบเจกต์และชื่อเมท็อดคั่นด้วยจุด (.) ซึ่งตัวอย่างในที่นี้ชื่อเต็มคือ randNum.nextDouble ความหมายของนิพจน์นี้คือ การเรียกเมท็อด nextDouble() ของอ็อบเจกต์ที่อ้างถึงโดยตัวแปรอ้างอิง randNum เพื่อความสะดวกและกระชับเราจะเรียกสั้น ๆ ว่า เป็นการเรียกเมท็อด nextDouble() ของอ็อบเจกต์ randNum

การเรียกเมท็อดจัดว่าเป็นนิพจน์ โปรดสังเกตุว่านิพจน์เรียกเมท็อดจะต้องมีวงเล็บหลังชื่อเมท็อดที่ถูกเรียกเสมอ

เมื่อมีการเรียกเมท็อด โปรแกรมจะกระโดดข้ามไปดำเนินการในเมท็อดที่ถูกเรียก เมื่อทำงานในเมท็อดที่ถูกเรียกเสร็จเรียบร้อยแล้วจะย้อนกลับมาทำงานต่อจากตำแหน่งที่ถูกเรียก บางเมท็อดเมื่อทำงานเสร็จแล้วอาจส่งค่าข้อมูลบางอย่างกลับมาด้วย เมท็อด nextDouble เป็นเมท็อดที่ใช้หาค่าตัวเลขแบบสุ่ม เมื่อทำงานเสร็จแล้วมันจะส่งค่าตัวเลขที่สุ่มได้เป็นชนิด double กลับมาด้วย เราสามารถนำค่าที่ส่งกลับมานี้ไปใช้งานต่อไปได้ ซึ่งในบรรทัดที่ ๘ มีการนำค่าที่เมท็อดส่งกลับมาไปต่อกับสายอักขระ (string) "Random double value : " ด้วยตัวดำเนินการต่อเชื่อม (+) แล้วส่งให้เมท็อด System.out.println ทำการพิมพ์ออกทางจอภาพ

ในบรรทัดที่ ๗ มีการเรียกใช้เมท็อด nextInt() ของอ็อบเจกต์ randNum เมท็อดนี้จะคืนค่ากลับมาเป็นค่าตัวเลขสุ่มชนิด int ต่อจากนั้นนำไปพิมพ์ออกทางจอภาพทำนองเดียวกับที่ทำในบรรทัดที่ ๖

ถ้าเราคอมไพล์และดำเนินการ (run) โปรแกรมนี้จะได้ผลลัพธ์ดังตัวอย่าง:

```
Random double value : 0.6892960397283345
Random integer value : 1421115631
```

การดำเนินการโปรแกรมนี้แต่ละครั้งจะได้ผลลัพธ์ที่แตกต่างกันไป เนื่องจากโปรแกรมนี้สร้างตัวเลขแบบสุ่มซึ่งค่าของมันจะไม่แน่นอน บางครั้งเราต้องการค่าตัวเลขแบบสุ่มเพื่อนำไปใช้ในการทดสอบโปรแกรมโดยที่เราไม่ต้องกำหนดค่าขึ้นเอง โปรแกรมเกมส์โดยทั่วไปก็มักมีการสร้างตัวเลขแบบสุ่มขึ้นมาใช้งานเช่นเดียวกัน

ตัวอย่างโปรแกรมที่ ๔-๒ เป็นตัวอย่างโปรแกรมอย่างง่ายแสดงการสร้างอ็อบเจกต์ของคลาส `Date` ซึ่งเป็นคลาสหนึ่งในแพคเกจ `java.util` ที่ให้มาพร้อมกับชุด JDK

ตัวอย่างโปรแกรม ๔-๒ แสดงการสร้างอ็อบเจกต์จากคลาส `Date`

```
1:   import java.util.Date;
2:   class Ex4_1 {
3:       public static void main(String args[]) {
4:           Date today;
5:           today = new Date();
6:           System.out.println("Right now is " + today.toString());
7:       }
8:   }
```

ในบรรทัดที่ ๑ มีคำสั่ง `import` สำหรับบอกคอมไพเลอร์ว่าเราต้องการใช้คลาส `Date` ที่อยู่ในแพคเกจ `java.util` ในบรรทัดที่ ๔ มีการประกาศตัวแปรชื่อว่า `today` ให้เป็นตัวแปรอ้างอิงมีชนิดเป็นคลาส `Date` หมายความว่าเราจะใช้ตัวแปรนี้สำหรับอ้างอิงถึงอ็อบเจกต์ที่เป็นตัวอย่าง(instance)ของคลาส `Date` บรรทัดที่ ๕ เป็นข้อความสั่งสำหรับกำหนดค่าให้กับตัวแปร `today` โดยนำค่าที่หาได้จากทางขวาของเครื่องหมายเท่ากับไปใส่ในตัวแปร `today` ทางด้านขวาของเครื่องหมายเท่ากับเป็นนิพจน์ที่ประกอบด้วยตัวดำเนินการ `new` ซึ่งใช้สำหรับสร้างอ็อบเจกต์ของคลาส `Date` เมื่อโปรแกรมทำงานมาถึงบรรทัดนี้จะได้อ็อบเจกต์ใหม่ ๑ ตัว ค่าเลขที่อยู่ของอ็อบเจกต์นี้จะถูกนำไปใส่ไว้ในตัวแปร `today` คลาส `Date` มี*ตัวสร้าง*หลายตัวซึ่งแต่ละตัวจะรับอาร์กิวเมนต์ต่างกัน จากตัวอย่างนี้ไม่มีการส่งอาร์กิวเมนต์ไปให้*ตัวสร้าง*ของคลาส `Date` ดังนั้นตัวสร้างตัวที่ไม่ต้องการอาร์กิวเมนต์ใด ๆ จึงถูกเรียกให้ทำงาน ซึ่ง*ตัวสร้าง*ตัวนี้มีหน้าที่ในการกำหนดข้อมูลภายในอ็อบเจกต์ให้เป็นวันและเวลาปัจจุบัน ณ. ขณะที่กำลังสร้างอ็อบเจกต์ ส่วนบรรทัดที่ 6 มีการพิมพ์ข้อความว่า `"Right now is "` แล้วตามด้วยข้อความที่ได้จากการเรียกใช้เมท็อด `toString()` ของอ็อบเจกต์ที่อ้างถึงโดยตัวแปร `today` เมท็อด `toString()` นี้เป็นเมท็อดที่กำหนดไว้ในคลาส `Date` ใช้สำหรับแปลงข้อมูลภายในออปเจ็กท์ให้เป็นสายอักขระ (string) เราสามารถเรียกใช้อินสแทนซ์เมท็อดใด ๆ ผ่านตัวแปรอ้างอิงได้โดยใช้จุด (`.`) เชื่อมระหว่างชื่อตัวแปรอ้างอิงกับชื่อเมท็อด ในตัวอย่างนี้ `today.toString()` หมายถึงการเรียกเมท็อด `toString()` ของอ็อบเจกต์ที่อ้างอิงโดยตัวแปร `today` หรือจะพูดให้ง่ายและกระชับว่าเป็นการเรียกเมท็อด `toString()` ของอ็อบเจกต์ `today` หลังจากเมท็อด `toString()` ทำงานตามหน้าที่ของมันเสร็จแล้วมันจะส่งค่ากลับคืนมาให้เป็นข้อมูลชนิดสายอักขระ ผลลัพธ์ที่ได้นี้จะนำไปต่อกับสายอักขระ `"Right now is "` เนื่องจากมีตัวดำเนินการบวก (+) เป็นตัวต่อเชื่อมสายอักขระทั้งสองเข้าด้วยกัน โปรแกรมนี้จึงพิมพ์ผลลัพธ์ดังตัวอย่าง:

```
Right now is Sun Nov 01 22:09:09 GMT+07:00 1998
```

วันที่และเวลาที่พิมพ์ออกมาจากการทำงานในแต่ละครั้งของโปรแกรมนี้จะไม่เหมือนกัน เนื่องจากโปรแกรมจะนำวันที่และเวลาปัจจุบันของเครื่องขณะที่กำลังสร้างอ็อบเจกต์ไปใช้เป็นค่าเริ่มต้นของอ็อบเจกต์ ถ้าเรียกโปรแกรมนี้ให้ทำงานหลายครั้งจะได้วันและเวลาที่แตกต่างกันไป

ตัวอย่างโปรแกรมที่ ๔-๓ เป็นตัวอย่างการสร้างอ็อบเจกต์จากคลาส `Point` ซึ่งเป็นคลาสที่อยู่ในแพคเกจ `java.awt` ตัวย่อ `awt` ย่อมาจาก Abstract Window Toolkit เป็นแพคเกจที่รวบรวมคลาสที่เกี่ยวข้องกับการเขียนโปรแกรมที่ทำงานแบบวินโดว์ไว้เป็นจำนวนมาก ในที่นี้จะหยิบยกคลาส `Point` ซึ่งเป็นคลาสเล็ก ๆเข้าใจง่ายมาเป็นตัวอย่าง คลาสนี้เป็นคลาสที่ใช้กำหนดตำแหน่งพิกัด ๒ มิติ โดยมีการเก็บข้อมูลค่าตำแหน่งในแนวแกน `x` และ แกน `y` และมีเมท็อดที่เกี่ยวข้องอยู่จำนวนหนึ่ง

ตัวอย่างโปรแกรม ๔-๓ แสดงการสร้างอ็อบเจกต์จากคลาส `Point`

```
1. import java.awt.Point;
2. class Ex4_2 {
3.     public static void main(String args[]) {
4.         Point pt;
5.         pt = new Point(100, 50);
6.         System.out.println( pt.toString() );
7.         pt.translate(50, 60);
8.         System.out.println( pt.toString() );
9.         pt.move(200, 150);
10.        System.out.println( pt.toString() );
11.    }
12. }
```

บรรทัดที่ ๔ เป็นการประกาศตัวแปรชื่อว่า `pt` เป็นตัวแปรอ้างอิงมีชนิดเป็นคลาส `Point` ในบรรทัดที่ ๕ มีการสร้างอ็อบเจกต์ของคลาส `Point` โดยส่ง 100 และ 50 เป็นอาร์กิวเมนต์ไปให้*ตัวสร้าง ตัวสร้าง*จะรับ 100 และ 50 ไปกำหนดเป็นค่าเริ่มต้นให้ตัวแปร `x` และ ตัวแปร `y` ของอ็อบเจกต์ที่สร้างขึ้นใหม่ตามลำดับ เลขที่อยู่ของอ็อบเจกต์ที่ได้จะนำไปใส่ในตัวแปร `pt`

ในบรรทัดที่ ๖ มีการเรียกเมท็อด `toString()` ของอ็อบเจกต์ `pt` เพื่อแปลงข้อมูลภายในอ็อบเจกต์ให้เป็นสายอักขระเพื่อความสะดวกในการนำไปแสดงผล เมท็อด `toString()` ในที่นี้เป็นเมท็อดคนละตัวกับเมท็อด `toString()` ในคลาส `Date` ตามตัวอย่างที่ ๔-๒ ถึงแม้จะมีชื่อและหน้าที่เหมือนกัน แต่การทำงานจะต่างกัน เราอาจตั้งชื่อเมท็อดซ้ำกันได้แต่มักให้มีความหมายไปในแนวทางเดียวกัน โดยที่การทำงานภายในไม่เหมือนกัน ถ้าสังเกตุ คลาสต่าง ๆ จะมีเมท็อด `toString()` อยู่ด้วยเสมอ สำหรับวัตถุประสงค์ของเมท็อดนี้จะกล่าวถึงโดยละเอียดอีกครั้งในบทที่ ๘

บรรทัดที่ ๗ มีการเรียกใช้เมท็อด `translate()` ของอ็อบเจกต์ `pt` เมท็อดนี้จะเลื่อนตำแหน่งของ `x` และ `y` ออกไปตามค่าของอาร์กิวเมนต์ที่ส่งไปให้ตามลำดับ หลังจากทำเมท็อดนี้แล้วค่าของ `x` และ `y` จะเปลี่ยนไป บรรทัดที่ ๘ เป็นการพิมพ์ค่าภายในอ็อบเจกต์ออกมาดูอีกครั้งเพื่อดูว่าค่าของมันเปลี่ยนไปอย่างไร

บรรทัดที่ ๙ เรียกเมท็อด `move` ของอ็อบเจกต์ `pt` เพื่อกำหนดค่า `x` และ `y` ให้เป็นค่าใหม่ตามอาร์กิวเมนต์ที่ส่งไปให้ หลังจากนั้นในบรรทัดที่ ๑๐ พิมพ์ค่าของอ็อบเจกต์ออกมาดูอีกครั้ง

ถ้าเราคอมไพล์และดำเนินการ (run) โปรแกรมนี้จะได้ผลลัพธ์แสดงออกมาดังนี้

```
java.awt.Point[x=100,y=50]
java.awt.Point[x=150,y=110]
java.awt.Point[x=200,y=150]
```

ตัวอย่างที่ ๔-๔ เป็นตัวอย่างการสร้างอ็อบเจกต์ของคลาส `Point` อีกเช่นเดียวกัน แต่คราวนี้แสดงให้เห็นการเข้าถึงข้อมูลในอ็อบเจกต์

ตัวอย่างโปรแกรมที่ ๔-๔ แสดงการเข้าถึงข้อมูลในอ็อบเจกต์

```
1.  import java.awt.Point;
2.  class Ex4_3 {
3.    public static void main(String args[]) {
4.      Point pt;
5.      pt = new Point(100, 50);
6.      System.out.println( "initial x=" + pt.x + " y=" + pt.y );
7.      pt.translate(50, -30);
8.      System.out.println( "after translation x=" + pt.x +
                            " y=" + pt.y );
9.      pt.x = 200;
10.     pt.y = 150;
11.     System.out.println( "after change value x=" + pt.x +
                            " y=" + pt.y);
12.   }
13. }
```

โปรแกรมนี้แก้ไขมาจากตัวอย่างที่ ๔-๓ ในบรรทัดที่ ๖ แทนที่จะใช้เมท็อด `toString()` ให้แปลงข้อมูลภายในอ็อบเจกต์ให้เป็นสายอักขระนั้น เราใช้วิธีเข้าไปดึงข้อมูลในอ็อบเจกต์มาใช้โดยตรงเสียเอง เราสามารถเข้าถึงตัวแปรในอ็อบเจกต์ผ่านตัวแปรอ้างอิงได้ทำนองเดียวกับการเรียกใช้อินสแทนซ์เมท็อด คลาส `Point` มีการกำหนดตัวแปร `x` และ `y` ไว้เป็นตัวแปรสมาชิกของคลาส เราเข้าใช้ตัวแปรสมาชิกเหล่านี้ได้จากภายนอกคลาสโดยอาศัยตัวดำเนินการจุดร่วมกับตัวแปรอ้างอิงที่อ้างถึงอ็อบเจกต์ที่เราต้องการเข้าถึง ในตัวอย่างนี้ `pt.x` หมายถึงการเข้าถึงตัวแปร `x` ของอ็อบเจกต์ที่อ้างอิงโดยตัวแปร `pt` หรือจะพูดให้สั้นได้ว่า เข้าถึงตัวแปร `x` ของอ็อบเจกต์ `pt` การเข้าถึงเป็นไปได้ทั้ง ๒ ทางคือจะหยิบค่าของตัวแปรมาใช้ หรือจะนำค่าใหม่ไปใส่ในตัวแปรก็ได้ บรรทัดที่ ๖ ๘ และ ๑๑ เป็นการหยิบเอาค่าของตัวแปรสมาชิก `x` และ `y` ออกมาพิมพ์ ส่วนบรรทัดที่ ๙ และ ๑๐ เป็นการนำค่าใหม่ไปใส่ในตัวแปรสมาชิก `x` และ `y` ของอ็อบเจกต์ `pt` ตามลำดับ

หลังจากคอมไพล์และดำเนินการโปรแกรมนี้แล้วจะได้ผลลัพธ์ที่แสดงออกมาดังนี้

```
initial x=100 y=50
after translation x=150 y=20
after change value x=200 y=150
```

แพคเกจ

แพคเกจ (package) ในภาษาจาวาหมายถึงกลุ่มของคลาสที่จัดรวมเข้าด้วยกันไว้ในสารบบ (directory) เดียวกัน การจัดแบ่งกลุ่มของคลาสต่าง ๆ ออกเป็นแพคเกจช่วยให้การจัดระเบียบของงานสะดวกขึ้น และช่วยแบ่งแยกระหว่างคลาสที่เราเขียนขึ้นเองกับที่ผู้อื่นเขียน ภาษาจาวาจะใช้ชื่อสารบบเป็นชื่อแพคเกจ ซึ่งชื่อแพค

แกจต้องเป็นไปตามกฎเกณฑ์การตั้งชื่อตัวระบุ ดังนั้นจึงเป็นการบังคับว่าชื่อสารบบที่ตั้งต้องเป็นไปตามกฎการตั้งชื่อตัวระบุด้วย ในกรณีที่มีการมีการแบ่งสารบบเป็นสารบบย่อย (subdirectory) หลายชั้นซ้อนลึกลงไปนั้น จะนำชื่อสารบบมาเป็นชื่อแพคแกจโดยใช้จุด(.) เป็นตัวเชื่อมระหว่างชื่อสารบบย่อย เราเรียกชื่อแพคเกจที่ประกอบด้วยชื่อสารบบไล่ตามลำดับชั้นแบบนี้ว่าชื่อเต็ม (fully qualified name)

คลังคลาสจาวามาตรฐาน(standard Java library) เป็นที่รวมของคลาสต่าง ๆ จัดแบ่งหมวดหมู่เป็นหลายแพคเกจ เช่น `java.lang` `java.util` และ `java.net` เป็นต้น แพคเกจมาตรฐานเป็นตัวอย่างของการจัดแบ่งคลาสเป็นลำดับชั้น ทำนองเดียวกับการที่เราจัดสารบบย่อยในดิสก์ เราสามารถจัดระเบียบแพคเกจโดยการแบ่งแพคเกจเป็นหลายระดับซ้อนกันเป็นชั้น ๆ แพคเกจมาตรฐานของจาวาทั้งหมดจัดเป็นโครงสร้างลำดับชั้นโดยราก(root) หรือจุดเริ่มต้นของแพคเกจมีชื่อว่า `java`

เหตุผลของการจัดแพคเกจซ้อนเป็นชั้น ๆ ก็เพื่อให้มั่นใจว่าชื่อเต็มของแพคเกจจะไม่ซ้ำกัน สมมุติว่านายสมชายสร้างแพคเกจของเขาขึ้นใช้เองและให้ชื่อว่า `util` ถ้าเขาใส่แพคเกจนี้ไว้ภายใต้โครงสร้างลำดับชั้นของแพคเกจเช่น `somchai.util` จะทำให้แพคเกจของเขาไม่ปะปนและสับสนกับแพคเกจ `util` ของผู้อื่น เราสามารถจัดแบ่งระดับของแพคเกจให้ซ้อนกันได้หลายชั้น เพื่อประกันว่าจะไม่เกิดความซ้ำซ้อนของชื่อแพคเกจ บริษัทซันไมโครซิสเต็มส์แนะนำให้ใช้ชื่อโดเมนของบริษัทหรือองค์การมาเป็นส่วนประกอบของชื่อแพคเกจด้วย โดยกลับตำแหน่งของส่วนประกอบของโดเมนย้อนหลังแล้วนำไปกำกับไว้หน้าชื่อแพคเกจ ตัวอย่างเช่น นายสมชายทำงานอยู่ที่บริษัทรวยดีซึ่งมีชื่อโดเมนว่า `goodrich.co.th` เขาควรตั้งชื่อแพคเกจ ของเขาว่า

```
th.co.goodrich.somchai.util
```

ซึ่งจะทำให้มั่นใจได้ว่าชื่อแพคเกจที่ตั้งขึ้นมานั้นจะไม่ซ้ำกับชื่อแพคเกจของผู้อื่น เพราะว่าชื่อผู้ใช้ และชื่อโดเมนมีการควบคุมไม่ให้ซ้ำกัยด้วยการจดทะเบียนนั่นเอง

การใช้แพคเกจ

แพคเกจเป็นที่รวมของคลาสต่าง ๆ เพื่ออำนวยความสะดวกในการนำไปใช้งาน ภาษาจาวาจะมีการจัดหมวดหมู่ของคลาสโดยแยกแยะคลาสไปไว้ในสารระบบ ต่าง ๆ ตามประเภทหรือความสัมพันธ์ของคลาส คลาสต่าง ๆ ที่นำมารวมไว้ในสารบบเดียวกันถือว่าเป็นคลาสที่อยู่ในแพคเกจเดียวกัน เราอ้างถึงคลาสต่าง ๆ ในแพคเกจได้ ๒ วิธี วิธีแรกคือใส่ชื่อเต็มของคลาส ตัวอย่างเช่น ถ้าสร้างอ็อบเจกต์จากคลาส **`Random`** ของแพคเกจ `java.util`

```
java.util.Random randNum = new java.util.Random();
```

การใช้วิธีนี้ดูเทอะทะเยิ่นเย้อไปหน่อยจึงไม่ค่อยนิยม อีกวิธีที่ง่ายและนิยมใช้มากกว่าคือการนำคำสำคัญ `import` ซึ่งจะทำให้เราอ้างถึงคลาสต่าง ๆ โดยไม่ต้องใช้ชื่อเต็มของแพคเกจก็ได้ เราสามารถนำเข้า(import) เฉพาะคลาสที่เจาะจงเพียงบางคลาสหรือจะนำเข้าทุกคลาสในแพคเกจก็ได้ เราใส่ข้อความสั่ง `import` ไว้ในโปรแกรมต้นฉบับก่อนตำแหน่งที่จะมีการใช้คลาสที่ต้องการ ตัวอย่างเช่น

```
import java.util.Random;
```

เป็นการเจาะจงว่าเราต้องการใช้คลาส Random ของแพคเกจ java.util ถ้าเราใช้คลาสเป็นจำนวนมากจากแพคเกจนี้ คงไม่สะดวกนักที่จะเขียนข้อความสั่ง import หลายบรรทัด เราสามารถเขียนแบบลัดโดยใช้เครื่องหมายดอกจันทร์ช่วยได้ดังนี้

```
import java.util.*;
```

เครื่องหมายดอกจันทร์หมายถึงคลาสอะไรก็ได้ที่อยู่ในแพคเกจ เราสามารถใช้ * ได้กับทีละแพคเกจเท่านั้น เราจะใช้ java.* เพื่อบอกว่าต้องการนำเข้าทุกแพคเกจของ java ไม่ได้

โดยปรกติแล้ว การนำเข้าทุกคลาสในแพคเกจเป็นเรื่องง่าย ไม่มีผลในทางลบแต่อย่างใดไม่ว่าจะเป็นเวลาที่ใช้ในการคอมไพล์หรือขนาดของโปรแกรม ดังนั้นจึงไม่มีเหตุผลใดที่จะไม่ใช้ แต่อย่างไรก็ตาม ถ้ามีคลาสชื่อเดียวกันซ้ำกันอยู่ในหลายแพคเกจ เราไม่สามารถนำเข้าพร้อมกันได้ ต้องเลี่ยงไปใช้ชื่อเต็มของคลาสใดคลาสหนึ่งแทนการนำเข้า เช่น สมมุติว่าแพคเกจ somchai.util ซึ่งชื่อซ้ำกับชื่อคลาส Date ที่อยู่ในแพคเกจ java.util เราจะนำเข้าทั้งสองแพคเกจนี้พร้อมกันในโปรแกรมเดียวกันไม่ได้ ต้องเลือกว่าจะนำเข้าแพคเกจใดเพียงแพคเกจเดียวเท่านั้น สมมติว่าเราเลือกที่จะนำเข้าคลาส Date จากแพคเกจ java.util ถ้าในโปรแกรมเราอ้างถึงคลาส Date โดยไม่ใช้ชื่อเต็ม จะหมายถึงคลาส Date ของแพคเกจ java.util ถ้าเราต้องการใช้คลาส Date ในแพคเกจ somchai.util เราต้องใช้ชื่อเต็มว่า somchai.util.Date

วิธีค้นหาตำแหน่งที่อยู่ของแพคเกจ

ทุกแฟ้มของแพคเกจจะต้องใส่ไว้ในสารบบย่อย (subdirectory) ซึ่งเข้าคู่กับชื่อเต็มของแพคเกจ ตัวอย่างเช่นทุกแฟ้มของแพคเกจ somchai.util จะต้องเก็บไว้ในสารบบย่อย somchai\util (ถ้าใช้ระบบปฏิบัติการ UNIX ต้องเก็บใน somchai/util)

สารบบย่อยไม่จำเป็นต้องแตกกิ่งออกมาจากสารบบราก (root directory) โดยตรง จะแยกสาขาออกมาจากสารบบใดก็ได้ ทั้งนี้คอมไพเลอร์และอินเทอร์พรีเตอร์ของจาวาจะรู้ที่อยู่ของสารบบได้จากตัวแปรแวดล้อม CLASSPATH สมมุติว่า ในเครื่องคอมพิวเตอร์ที่ใช้งานมีการระบุ CLASSPATH ไว้ดังต่อไปนี้

```
CLASSPATH=c:\j2sdk1.4.2\lib;c:\userlib;.
```

เครื่องหมายจุดตัวเดียว (.) หมายถึงสารบบปัจจุบัน (current directory)

และสมมุติว่าในโปรแกรมมีข้อความสั่งต่อไปนี้

```
import java.util.*;
import somchai.util.*;
```

ถ้าต้องการใช้คลาส Time คอมไพเลอร์จะมองหาแฟ้มชื่อ Time.class ในสารบบต่าง ๆ ที่กำหนดใน CLASSPATH ตามลำดับประกอบกับชื่อแพคเกจดังต่อไปนี้

```
c:\j2sdk1.4.2\lib\Time.class
c:\j2sdk1.4.2\lib\java\util\Time.class
c:\j2sdk1.4.2\lib\somchai\util\Time.class
```

```
c:\userlib\Time.class
c:\userlib\java\util\Time.class
c:\userlib\somchai\util\Time.class
.\Time.class
.\java\util\Time.class
.\somchai\util\Time.class
```

ถ้าพบแฟ้มที่ต้องการ คอมไพเลอร์จะตรวจสอบต่อไปว่า ชื่อแพคเกจเข้าคู่กับชื่อพาธ (path) หรือไม่ และภายในแฟ้มมีคลาสสาธารณะชื่อว่า `Time` อยู่ภายในหรือไม่

โดยที่จริงแล้ว ถ้าเข้าไปดูในสารบบ(directory) ที่เก็บคลังคลาสหรือแพคเกจของจาวาที่ติดตั้งไว้ในฮาร์ดดิสก์ เช่นถ้าเข้าไปในโฟลเดอร์ c:\program files\java\ jdk1.7.0\jre\lib จะไม่พบสารบบย่อย `java.util` แต่อาจพบแฟ้มที่ชื่อว่า rt.jar แฟ้มสกุลนี้เป็นแฟ้มที่ได้จากการบีบอัดแฟ้มอื่นหลายแฟ้มแล้วรวมเข้าด้วยกันเพื่อให้ประหยัดเนื้อที่ในดิสก์ ถ้าเราเข้าไปดูแฟ้ม `rt.jar` ด้วยโปรแกรมสำหรับดูแฟ้ม zip เช่น WinZip หรือ WinRar จะเห็นพาธและแฟ้มคลาสต่าง ๆ อยู่ในนั้น ชื่อสารบบและชื่อแฟ้มอาจแตกต่างกันสำหรับจาวาแต่ละรุ่น

นอกจากนี้คอมไพเลอร์จาวาจะค้นหาแพคเกจ `java.lang` เสมอไม่ว่าเราจะระบุ `import java.lang` หรือไม่ก็ตาม เนื่องจากแพคเกจนี้บรรจุคลาสต่าง ๆ ที่จำเป็นของภาษาจาวาเอาไว้ เรามักใช้คลาสใดคลาสหนึ่งจากแพคเกจนี้เสมอ จาวาจึงอำนวยความสะดวกให้เราไม่ต้องระบุการนำเข้าแพคเกจนี้

การเขียนคลาส

ในตอนต้นของบทนี้เราได้รู้จักการสร้างอ็อบเจกต์จากคลาสที่ผู้อื่นสร้างไว้ให้เราใช้ ในการเขียนโปรแกรมจาวาเรามักนำคลาสต่าง ๆ ที่ผู้อื่นเขียนไว้เป็นแพคแกจมาใช้ วัตถุประสงค์หลักของแนวคิดการสร้างโปรแกรมเชิงวัตถุก็คือการใช้ซ้ำ(reuse) ใช้สิ่งที่มีอยู่แล้ว หรือสิ่งที่ผู้อื่นสร้างไว้แล้วให้เกิดประโยชน์มากที่สุด ดังนั้นจึงไม่ใช่เรื่องแปลกที่ผู้เขียนโปรแกรมจาวาจะใช้คลาสที่ผู้อื่นเขียนไว้ให้ใช้มากกว่าเขียนคลาสของตนเองขึ้นมาใหม่ เหตุผลสำคัญเหตุผลหนึ่งที่ทำให้ภาษาจาวาได้รับความนิยมอย่างรวดเร็วก็คือการที่มีคลาสสำเร็จรูปให้ใช้เป็นจำนวนมากทั้งจากผู้คิดค้นภาษาจาวาคือซันไมโครซิสเต็ม เองและบริษัทอื่นที่สร้างคลาสต่าง ๆ ขึ้นมาแจกฟรีหรือขาย ถึงแม้ว่าจะมีคลาสเป็นจำนวนมากให้เราใช้ก็ตาม บ่อยครั้งที่เราจำเป็นหรือควรเขียนคลาสของเราขึ้นใช้เอง ในส่วนนี้จะกล่าวถึงการเขียนคลาสใหม่เพื่อใช้งานในโปรแกรมของเรา

ในภาษาจาวา เราใช้คลาสในการกำหนดส่วนของรหัสคำสั่ง โปรแกรมจาวาแต่ละโปรแกรมจะต้องมีการกำหนดคลาสอย่างน้อย ๑ คลาสเสมอ

โครงของคลาสมีรูปแบบดังนี้

ตัวดัดแปรคลาส$_{\text{ละได้}}$ **class** *ชื่อคลาส*
{
ส่วนประกาศตัวแปร$_{\text{ละได้}}$
ส่วนประกาศเมท็อด$_{\text{ละได้}}$
ส่วนประกาศคลาส$_{\text{ละได้}}$
}

คำที่มีตัวห้อย $_{\text{ละได้}}$กำกับอยู่ด้วยหมายถึงตัวเลือก จะเขียนใส่ไว้ในโปรแกรมของเราหรือไม่ก็ได้แล้วแต่ความจำเป็น

ตัวดัดแปรคลาส *(class modifier)* เป็นคำสำคัญ(keyword) ที่ใส่ไว้หน้าคำสำคัญ `class` เพื่อบ่งบอกสมบัติบางอย่างของคลาส ตัวดัดแปรคลาสแบ่งเป็นตัวดัดแปรการเข้าถึง (access modifier) และตัวดัดแปรอื่น ๆ เราอาจไม่ระบุตัวดัดแปรใด ๆ เลยก็ได้ หรืออาจใส่ตัวดัดแปรเพียงตัวใดตัวหนึ่งหรือหลายตัวร่วมกันก็ได้ แต่ต้องดูความหมายของตัวดัดแปรให้เหมาะสมด้วยว่าใช้ร่วมกันได้หรือไม่ ในช่วงเริ่มต้นนี้เราจะยังไม่ใช้ตัวดัดแปรใด เว้นแต่ในตอนท้ายบทจะกล่าวถึงตัวดัดแปรการเข้าถึง public ส่วนตัวดัดแปรนอกเหนือจากนี้จะกล่าวถึงในบทอื่นต่อไป

`class` เป็นคำสำคัญเพื่อบ่งบอกว่าเรากำลังกำหนดคลาส

ชื่อคลาส เป็นตัวระบุ (identifier) ใช้ตั้งชื่อคลาส เราต้องตั้งชื่อคลาสตามหลักเกณฑ์การตั้งชื่อตัวระบุ

ส่วนประกาศตัวแปร ใช้สำหรับประกาศตัวแปรสมาชิกของคลาส คลาสหนึ่ง ๆ อาจมีตัวแปรสมาชิกได้หลายตัวหรือจะไม่มีตัวแปรสมาชิกเลยก็ได้ กรณีที่มีตัวแปรสมาชิกหลายตัวในคลาสเดียวกัน จะประกาศตัวแปรให้มีชื่อซ้ำกันไม่ได้ ตัวแปรสมาชิกต้องประกาศไว้ในขอบเขต ({...}) ของคลาส และอยู่นอกขอบเขตของเมท็อด ตัวแปรสมาชิกจะได้รับการกำหนดค่าเริ่มต้นโดยอัตโนมัติตามชนิดของตัวแปร ตัวแปรชนิดตัวเลขจะได้รับการกำหนดค่าเริ่มต้นให้เป็นศูนย์ ตัวแปรชนิดบูล(boolean)จะได้รับการกำหนดค่าเริ่มต้นให้เป็น `false` ส่วนตัวแปรอ้างอิงจะได้รับการกำหนดค่าเริ่มต้นให้เป็น `null`

ส่วนประกาศเมท็อด หมายถึงเมท็อดต่าง ๆ ที่เราเขียนขึ้นมาเป็นเมท็อดสมาชิกของคลาส ในเมท็อดจะมีข้อความสั่งต่าง ๆ ที่เราต้องการให้เมท็อดทำงานตามหน้าที่ของมัน

ส่วนประกาศคลาส หมายถึงคลาสชั้นในที่เขียนขึ้นมาให้เป็นสมาชิกของคลาสชั้นนอก

ภายในวงเล็บปีกา { } เป็นองค์(body)ของคลาส ซึ่งจะประกอบด้วยสมาชิกของคลาสอันได้แก่*ตัวแปรสมาชิก (member variable)* ***เมท็อดสมาชิก*** *(member method)* และคลาสสมาชิก คลาสหนึ่ง ๆ จะมีเฉพาะตัวแปรสมาชิก หรือ เมท็อดสมาชิกอย่างใดอย่างหนึ่ง หรือจะมีสมาชิกครบทั้ง ๓ อย่างก็ได้ แต่โดยส่วน

ใหญ่แล้วในคลาสหนึ่ง ๆ มักจะมีประกอบด้วยตัวแปรสมาชิกและเมท็อดสมาชิกเป็นหลัก ส่วนคลาสสมาชิกนั้นมีความจำเป็นในการใช้น้อยกว่าจึงยังไม่อธิบายในบทนี้ แต่จะกล่าวถึงในหัวข้อคลาสชั้นในในบทที่ ๘

หมายเหตุ

เรื่องของตัวดัดแปร

ตัวดัดแปรเป็นคำสำคัญใช้ระบุเพื่อให้เปลี่ยนสมบัติบางอย่างไปจากค่าโดยปริยาย คลาส เมท็อด และตัวแปรสมาชิกต่างก็มีตัวดัดแปรเฉพาะตัว แต่ใช้คำสำคัญบางคำเป็นคำเดียวกัน ซึ่งให้ความหมายใกล้เคียงกัน

ตัวดัดแปรแบ่งเป็นตัวดัดแปรการเข้าถึง (access modifier) และตัวดัดแปรอื่น ๆ ตัวดัดแปรการเข้าถึงใช้ระบุสมบัติในการเข้าถึงคลาส เมท็อด และตัวแปรว่า "ใจกว้าง" แค่ไหน ถ้าไม่ได้ระบุตัวดัดแปรการเข้าถึงจะหมายถึงใจกว้างพอประมาณ ถือว่า "เป็นมิตร(friendly)" กับพวกเดียวกัน ยอมให้พวกเดียวกันเข้าถึงได้ ส่วนพวกอื่นไม่ยอมให้เข้าถึง ถ้าต้องการเปลี่ยนระดับความใจกว้าง จะต้องระบุตัวดัดแปรการเข้าถึงไว้ด้วย ตัวดัดแปรการเข้าถึงมีให้ใช้ ๓ ตัวด้วยกัน ตัวแรกคือ `public` หมายถึงใจกว้างดั่งมหาสมุทร ยกให้เป็นสมบัติสาธารณะใคร ๆ ก็ใช้ได้ ไม่หวงแต่อย่างใด ตัวที่สองคือ `protected` ทำใจให้แคบลง ให้ใช้เฉพาะลูกหลานที่สืบเชื้อสายลงไปเท่านั้น ส่วนตัวสุดท้ายคือ `private` ใจแคบที่สุด ใช้เป็นการส่วนตัวเท่านั้น แม้แต่ลูกหลานก็ไม่ยอมให้ใช้ ที่เป็นเช่นนี้ใช่ว่านิสัยไม่ดี แต่บางทีจำเป็นต้องทำ

ตัวอย่างโปรแกรมที่ ๔-๕ เป็นตัวอย่างโปรแกรมแสดงการเขียนคลาสอย่างง่าย โปรแกรมนี้มี ๒ คลาส คลาสแรกชื่อ Ex4_5 สำหรับทดสอบคลาส `Time` ซึ่งเขียนไว้ที่บรรทัด ๑๖ ถึง ๑๙ สำหรับเก็บข้อมูลเวลาภายในคลาสนี้มีตัวแปรสมาชิก ๒ ตัวคือ `hour` และ `minute` สำหรับเก็บค่าชั่วโมงและนาทีตามลำดับ

บรรทัดที่ ๓ มีการประกาศตัวแปรอ้างอิงชื่อว่า `departureTime` มีชนิดเป็นคลาส `Time` พร้อมทั้งกำหนดค่าเริ่มต้นให้เป็นเลขที่อยู่ของอ๊อบเจกต์ชนิด `Time` ที่สร้างขึ้นใหม่ ตัวดำเนินการ `new` ใช้สำหรับสร้างอ๊อบเจกต์ใหม่หนึ่งตัว ในกรณีนี้สร้างอ๊อปเจ็กท์ชนิดคลาส `Time` ซึ่งมีตัวแปรสมาชิก ๒ ตัว ในขณะที่โปรแกรมกำลังทำงานเมื่อมีการสร้างอ๊อบเจกต์ใหม่จะมีการหาเนื้อที่ว่างในหน่วยความจำสำหรับเก็บตัวแปรสมาชิกของคลาส พร้อมทั้งกำหนดค่าเริ่มต้นให้กับตัวแปรสมาชิก กรณีนี้ตัวแปรสมาชิกเป็นชนิดตัวเลขจะถูกกำหนดค่าเริ่มต้นให้เป็นศูนย์โดยอัตโนมัติ

บรรทัดที่ ๔ และ ๕ เป็นการกำหนดค่าให้กับตัวแปรสมาชิกของอ๊อบเจกต์ `departureTime` ส่วนบรรทัดที่ ๖ เป็นการพิมพ์ค่าของตัวแปรสมาชิกออกมาดู

บรรทัดที่ ๙ เป็นการประกาศตัวแปรอ้างอิงอีกตัวให้ชื่อว่า `arivalTime` พร้อมทั้งกำหนดค่าเริ่มต้นให้เป็นเลขที่อยู่ของอ๊อบเจกต์ชนิด `Time` ตัวใหม่อีกตัว โปรแกรมนี้จึงมีอ๊อบเจกต์ ๒ ตัวอิสระจากกันอยู่กันคนละที่ ต่างก็มีตัวแปรสมาชิกเป็นของตนเองอ๊อบเจกต์ใครอ๊อบเจกต์มัน บรรทัดที่ ๑๐ กำหนดค่าให้ตัวแปร `hour` ของอ๊อบเจกต์ `arivalTime` โปรดสังเกตุว่าตัวอย่างนี้ไม่มีการกำหนดค่าให้ตัวแปร `minute` ของอ๊อบเจกต์ `arivalTime` ส่วนบรรทัดที่ ๑๑ จะพิมพ์ค่าของตัวแปรสมาชิก `hour` และ `minute` ของอ๊อบเจกต์ `arivalTime` ออกมา

ตัวอย่างที่ ๔-๕ แสดงการสร้างคลาสอย่างง่าย

```
1.  class Ex4_5 {
2.      public static void main(String args[]) {
3.          Time departureTime = new Time();
4.          departureTime.hour = 8;
5.          departureTime.minute = 30;
6.          System.out.println("Departure Time is " +
7.              departureTime.hour + ":" + departureTime.minute);
8.
9.          Time arivalTime = new Time();
10.         arivalTime.hour = 19;
11.         System.out.println("Arival Time is " +
12.             arivalTime.hour + ":" + arivalTime.minute);
13.     }
14. }
15.
16. class Time {
17.     byte hour;
18.     byte minute;
19. }
```

ผลลัพธ์จากการดำเนินการของโปรแกรมนี้เป็นดังนี้

```
Departure Time is 8:30
Arival Time is 19:0
```

จะเห็นว่าค่า `minute` ของอ็อบเจกต์ `arivalTime` เป็นศูนย์ทั้ง ๆ ที่ไม่มีส่วนใดของโปรแกรมนี้กำหนดค่าให้ตัวแปรนี้เป็นศูนย์เลย เหตุที่เป็นเช่นนี้ ก็เพราะว่า ขณะที่สร้างอ็อบเจกต์นี้ขึ้นมานั้นจะมีการกำหนดค่าตัวแปรสมาชิกให้เป็นศูนย์โดยอัตโนมัตินั่นเอง

ค่าโดยปริยายของตัวแปรสมาชิก

เมื่อมีการจัดสรรเนื้อที่ในหน่วยความจำสำหรับเก็บตัวแปรสมาชิก จะมีการกำหนดค่าเริ่มต้นให้ตัวแปรสมาชิกที่สร้างขึ้นใหม่นั้นโดยอัตโนมัติ ค่าที่กำหนดให้นั้นเรียกว่า*ค่าโดยปริยาย (default value)* ซึ่งขึ้นอยู่กับชนิดของตัวแปร ตัวแปรชนิดตัวเลขจะได้รับการกำหนดค่าเริ่มต้นให้เป็นศูนย์ ตัวแปรชนิดบูล (`boolean`) จะได้รับการกำหนดค่าเริ่มต้นให้เป็น `false` ส่วนตัวแปรอ้างอิงจะได้รับการกำหนดค่าเริ่มต้นให้เป็น `null` ซึ่งมีความหมายว่าไม่ได้อ้างอิงไปที่ใดเลย

ในตัวอย่างที่ ๔-๕ คลาส `Time` มีแต่ตัวแปรสมาชิก ไม่มีเมท็อดสมาชิกใด ๆเลย เราอาจใช้คลาสแบบนี้สำหรับสร้างอ็อบเจกต์ต่าง ๆ เพื่อเก็บข้อมูล และอนุญาตให้มีการเข้าถึงข้อมูลเหล่านั้นจากคลาสอื่นได้ ตัวอย่างนี้มีการเข้าถึงข้อมูลของอ็อบเจกต์ชนิด `Time` จากคลาส `Ex4_5` การเข้าถึงตัวแปรสมาชิกจากภายนอกคลาสของมันจำเป็นต้องระบุตัวแปรอ้างอิงเสมอ อ็อบเจกต์ที่สร้างจากคลาส `Time` ในตัวอย่างนี้ถือว่าเป็น *อ็อบเจกต์ที่ทำงานด้วยตัวเองไม่ได้ (passive object)* โดยทั่วไปแล้วคลาสที่เราสร้างขึ้นมักจะมีเมท็อดสมาชิกรวมอยู่ด้วยเพื่อให้อ็อบเจกต์ที่สร้างขึ้น*เป็นอ็อบเจกต์ที่สามารถทำงานได้ด้วยตัวเอง (active object)*

การกำหนดเมท็อด

เราสามารถกำหนดให้มีเมท็อดอยู่ในคลาสของเราเพื่อทำหน้าที่ต่าง ๆ ตามที่เราต้องการ รูปแบบของการกำหนดเมท็อดมีรูปแบบเป็นดังนี้

```
ตัวดัดแปรเมท็อด_ละได้  ชนิดของค่าส่งกลับ  ชื่อเมท็อด (รายการพารามิเตอร์_ละได้)
{
    ส่วนประกาศตัวแปรและข้อความสั่งต่าง ๆ
}
```

ชื่อเมท็อด เป็นตัวระบุ(identifier)ที่ตั้งขึ้นมาเพื่อใช้อ้างถึงในการเรียกใช้เมท็อด ชื่อที่ตั้งต้องเป็นไปตามกฎเกณฑ์การตั้งชื่อตัวระบุ

ชนิดของค่าส่งกลับ*(return type)*เป็นชนิดของข้อมูลผลลัพธ์ที่จะส่งกลับไปให้ผู้เรียกเมท็อด เราสามารถระบุชนิดของข้อมูลที่จะส่งกลับโดยระบุชื่อชนิด (type) ของข้อมูล หรือ ชื่อคลาสทำนองเดียวกับการประกาศตัวแปร ค่าที่ส่งคืนจากเมท็อดมีได้เพียงค่าเดียวเท่านั้น หรือจะไม่ส่งค่าใด ๆกลับเลยก็ได้ ถ้าไม่มีความจำเป็นที่จะส่งข้อมูลกลับให้ใช้คำสำคัญ `void` แทนที่*ชนิดของค่าส่งกลับ*

รายการพารามิเตอร์*(parameter list)*เป็นรายการของตัวแปรที่ประกาศขึ้นในเมท็อดใช้สำหรับรับค่าข้อมูลต่าง ๆ ที่ผู้เรียกส่งมาให้ในขณะที่เรียกเมท็อด เมท็อดอาจมีหรือไม่มีพารามิเตอร์ก็ได้ ถ้ามีพารามิเตอร์หลายตัว จะคั่นพารามิเตอร์แต่ละตัวด้วยจุลภาค (comma) ถ้าเมท็อดไม่มีการรับค่าพารามิเตอร์ใด ๆ เลย ยังจำเป็นต้องมีวงเล็บอยู่เสมอแต่จะเว้นในวงเล็บให้ว่างไว้ พารามิเตอร์เป็นตัวแปรที่ประกาศไว้ในวงเล็บหลังชื่อเมท็อด การประกาศพารามิเตอร์จะต้องมีชนิดข้อมูลนำหน้าชื่อตัวแปรที่ทำหน้าที่เป็นพารามิเตอร์เสมอ ถ้ามีพารามิเตอร์หลายตัวชื่อเหล่านี้จะซ้ำกันไม่ได้ พารามิเตอร์ถือว่าเป็นตัวแปรเฉพาะบริเวณ ใช้ได้เฉพาะภายในเมท็อดที่กำหนดมันขึ้นมาเท่านั้น

ตัวดัดแปรเมท็อด*(method modifier)* เป็นคำสำคัญ(keyword)ที่ใช้กำหนดสมบัติของเมท็อด ตัวดัดแปรเป็นตัวเลือกเราสามารถละไม่เขียนก็ได้ เราอาจเขียนตัวดัดแปรมากกว่าหนึ่งตัวไว้หน้าชนิดของค่าส่งกลับก็ได้ เท่าที่ผ่านมาเรารู้จักตัวดัดแปรมาบ้างแล้วคือคำสำคัญ `public` ซึ่งเป็นตัวดัดแปรให้เมท็อดมีสมบัติเป็นสาธารณะสามารถเรียกใช้จากคลาสอื่นได้ และคำสำคัญ `static` ที่ใช้สำหรับกำหนดสมบัติของเมท็อดให้เป็น*เมท็อดสถิต(static method)* หรือเรียกอีกอย่างว่า*คลาสเมท็อด(class method)* ซึ่งไม่ขึ้นกับอ็อบเจกต์ใด อันจะทำให้สามารถเรียกใช้เมท็อดได้โดยไม่ต้องสร้างอ็อบเจกต์ขึ้นมา

การประกาศตัวแปรและข้อความสั่ง บรรจุอยู่ในวงเล็บปีกกาประกอบขึ้นเป็น*องค์ของเมท็อด (method body)* หรือเรียกอีกอย่างว่า*บล็อก* ซึ่งทำหน้าที่เป็น*ข้อความสั่งประกอบ(compound statement)* ที่รวมเอาข้อความประกาศตัวแปรและข้อความสั่งสำหรับทำงานต่าง ๆ ไว้ภายใน ข้อความประกาศตัวแปรเป็นข้อความที่ใช้สำหรับประกาศชื่อและชนิดของตัวแปรที่ใช้ภายในเมท็อด ตัวแปรต่าง ๆที่ใช้ในเมท็อดต้องมีการประกาศ

มาก่อนล่วงหน้าจึงจะสามารถอ้างอิงใช้งานได้ ภายในบล็อกของเมท็อดอาจมีบล็อกย่อยซ้อนอยู่ภายในได้อีกหลายบล็อกถ้าต้องการ แต่*ไม่สามารถสร้างเมท็อดให้ซ้อนอยู่ภายในเมท็อดใด ๆ ได้เลย*

เมท็อดใดที่ถูกเรียกให้ทำงาน มันจะทำงานไปจนจบบล็อกของเมท็อดนั้นแล้วจึงออกจากบล็อกย้อนกลับไปทำงานในลำดับถัดไปจากตำแหน่งที่เรียกเมท็อดนั้น ในกรณีที่ไม่ต้องการให้เมท็อดทำงานไปจนจบบล็อก เราสามารถสั่งให้เมท็อดออกจากบล็อกโดยใช้ข้อความสั่ง `return` กรณีที่เป็นเมท็อดที่มีการส่งค่าข้อมูลกลับคืนด้วยนั้น เราจะใช้ข้อความสั่ง `return` รูปแบบดังนี้

```
return นิพจน์
```

ในการส่งข้อมูลกลับไปยังผู้เรียก เมื่อโปรแกรมทำงานมาถึงข้อความสั่ง `return` *นิพจน์*หลังคำสำคัญ `return` จะถูกหาค่าแล้วส่งค่าที่ได้นั้นกลับไปพร้อมกับการออกไปจากเมท็อดนี้ในทันทีเพื่อไปทำงานในลำดับถัดไปจากตำแหน่งที่เรียกเมท็อดมา ค่าของ*นิพจน์*ที่หาได้จะต้องเป็นชนิดเดียวกันกับ*ชนิดของค่าที่ส่งกลับ*ตามที่ได้กำหนดไว้หน้าชื่อเมท็อด

ตัวอย่างที่ ๔-๖ เป็นโปรแกรมที่ดัดแปลงมาจากตัวอย่างที่ ๔-๕ โดยที่แทนที่จะให้เข้าถึงตัวแปรสมาชิกของคลาส `Time` จากคลาสอื่นโดยตรง คราวนี้เราจะสร้างเมท็อดขึ้นมาเพื่อใช้ในการเข้าถึงตัวแปรสมาชิก ในคลาส `Time` มีการกำหนดเมท็อด `setHour` ขึ้นมาที่บรรทัด ๑๕ เมท็อดนี้เขียนขึ้นมาเพื่อรับค่าชั่วโมงไปใส่ในตัวแปรสมาชิก `hour` ดังนั้นจึงตั้งตัวพารามิเตอร์ขึ้นมา ๑ ตัวเพื่อรับค่าชั่วโมงที่ผู้เรียกส่งมาให้ พารามิเตอร์ตัวนี้ให้ชื่อว่า `h` มีชนิดเป็น `int` เมท็อดนี้ไม่ได้ส่งค่าใด ๆกลับไปยังผู้เรียกจึงใส่คำสำคัญ `void` ไว้หน้าชื่อเมท็อด ภายในเมท็อดนี้มีข้อความสั่ง ๑ ข้อความคือ `hour = (byte)h;` เพื่อนำค่าชั่วโมงที่รับมาอยู่พารามิเตอร์ `h` ไปใส่ไว้ในตัวแปรสมาชิก `hour` แต่เนื่องจากมีขนาดที่แตกต่างกัน ตัวแปรสมาชิก `hour` มีขนาดเล็กกว่าพารามิเตอร์ `h` จึงจำเป็นต้องหล่อหลอม (cast) ค่าของตัวแปร `h` ซึ่งเป็นชนิด `int` ให้เป็นชนิด `byte` เสียก่อนจึงจะนำค่าที่ได้ไปใส่ในตัวแปรสมาชิก `hour` ความจริงแล้วเราสามารถกำหนดให้ตัวพารามิเตอร์ `h` เป็นชนิด `byte` เสียก็ได้ แต่เวลาเรียกเมท็อดนี้เราต้องส่งพารามิเตอร์ให้เป็นชนิด `byte` ด้วย ซึ่งหากเราส่งค่าคงที่เป็นตัวเลขไปให้เมท็อดนี้แล้วละก็ เราต้องหล่อหลอมค่าตัวเลขคงที่นั้นให้เป็นชนิด `byte` อยู่ดี เช่นถ้าจะเรียกเมท็อด `setHour` โดยการส่งพารามิเตอร์เป็นค่าคงที่ `8` ตามตัวอย่างในบรรทัดที่ ๔ เราต้องเขียนการเรียกเมท็อดดังนี้

```
departureTime.setHour( (byte) 8 );
```

ซึ่งหากมีการเรียกเมท็อดทำนองนี้หลายครั้งจะทำให้ยุ่งยากและเสียเวลาในการเขียนโปรแกรมให้มีการหล่อหลอมอยู่เสมอ ดังนั้นเราจึงให้เมท็อด `setHour` รับพารามิเตอร์เป็นชนิด `int` เพื่อความสะดวก และทำการหล่อหลอมพารามิเตอร์นี้เพียงครั้งเดียวก่อนนำข้อมูลไปใส่ในตัวแปรสมาชิก `hour`

สำหรับเมท็อด `setMinute` ในบรรทัดที่ ๒๐ มีการทำงานทำนองเดียวกันกับเมท็อด `setHour` ส่วนเมท็อด `getHour` ในบรรทัดที่ ๒๑ เขียนขึ้นมาเพื่อส่งค่าของตัวแปรสมาชิก `hour` ไปให้ผู้เรียกเมท็อดนี้ เมท็

อดนี้ไม่รับพารามิเตอร์ใดเลย ในวงเล็บหลังชื่อเมท็อด จึงว่างเปล่า ถึงแม้ว่าจะไม่รับพารามิเตอร์ใดเลยก็ตาม จะต้องมีวงเล็บนี้เสมอ

เนื่องจากเราต้องการให้เมท็อดนี้ส่งค่ากลับไปด้วยเมื่อทำงานเสร็จ ดังนั้นเราจึงต้องกำหนดชนิดของตัวแปรที่ต้องการส่งกลับไปด้วยซึ่งในที่นี้คือ `byte` ส่วนภายในเมท็อดไม่ได้ทำอะไรมากเพียงแค่ส่งค่าของตัวแปรสมาชิก `hour` กลับออกมาด้วยข้อความสั่ง `return hour;` เท่านั้น ในขณะที่เมท็อดนี้ทำงาน เมื่อทำงานมาถึงข้อความสั่ง `return` มันจะยุติการทำงานภายในเมท็อดพร้อมทั้งส่งค่าของตัวแปรสมาชิก `hour` กลับออกมาด้วย สำหรับเมท็อด `getMinute` ในบรรทัดที่ ๒๒ คล้ายคลึงกับเมท็อด `getHour`

ตัวอย่างที่ ๔-๖ นี้มีการกำหนดเมท็อดสำหรับรับค่าเป็นพารามิเตอร์แล้วนำไปใส่ในตัวแปรสมาชิกซึ่งทำให้สถานะของอ๊อบเจกต์เปลี่ยนไป เมท็อดที่ทำหน้าที่แบบนี้เรียกว่า*ตัวเปลี่ยน (mutator)* หรือ *ตัวตั้งค่า (setter)* เมท็อดชนิดนี้นิยมตั้งชื่อขึ้นต้นด้วยคำว่า `set` ส่วนเมท็อดที่ใช้สำหรับส่งค่าข้อมูลภายในอ๊อบเจกต์ออกมาให้ผู้เรียกเราเรียกว่า *ตัวเข้าถึง (accessor)* หรือ *ตัวสอบถามค่า (getter)* เมท็อดประเภทนี้นิยมตั้งชื่อขึ้นต้นด้วยคำว่า `get`

ข้อควรปฏิบัติ

ข้อนิยมในการตั้งชื่อคลาสและเมท็อด

- ชื่อคลาสนิยมขึ้นต้นด้วยอักษรโรมันตัวใหญ่ ควรหลีกเลี่ยงการใช้ชื่อซ้ำกับชื่อคลาสในแพคเกจต่าง ๆ ของ JDK เพื่อป้องกันความสับสน
- ชื่อเมท็อดนิยมขึ้นต้นด้วยอักษรโรมันตัวเล็ก ถ้าเป็นเมท็อดที่ทำการเปลี่ยนค่าของตัวแปรสมาชิกนิยมตั้งนำหน้าด้วยคำว่า `set` ถ้าเป็นเมท็อดที่คืนค่าของตัวแปรสมาชิก นิยมขึ้นต้นด้วย `get`

ในคลาส Ex4_6 มีการใช้คลาส `Time` แต่คราวนี้ไม่มีการเข้าถึงตัวแปรสมาชิกของคลาส `Time` โดยตรงเหมือนดังในตัวอย่างที่ ๔-๕ แต่จะใช้*ตัวเปลี่ยน*และ*ตัวเข้าถึง*แทน บรรทัดที่ 4 และ 5 *ใช้ตัวเปลี่ยน* `setHour` และ `setMinute` ในการกำหนดชั่วโมงและนาทีให้อ๊อบเจกต์ `departureTime` ส่วนบรรทัดที่ ๗ มีการใช้*ตัวเข้าถึง*สำหรับส่งค่าตัวแปรสมาชิก `hour` และ `minute` ของอ๊อบเจกต์ `departureTime` ออกมาแสดงทางจอภาพ

สำหรับผลลัพธ์จากการดำเนินการของโปรแกรมนี้ได้ผลลัพธ์เหมือนกับผลลัพธ์ที่ได้จากตัวอย่างที่ ๔-๕

ตัวอย่างที่ ๔-๖ แสดงคลาสซึ่งมีตัวเปลี่ยนและตัวเข้าถึง

```
1.  class Ex4_6 {
2.      public static void main(String args[]) {
3.          Time departureTime = new Time();
4.          departureTime.setHour(8);
5.          departureTime.setMinute(30);
6.          System.out.println("Departure Time is " +
7.                              departureTime.getHour() + ":" +
                                departureTime.getMinute());
8.
9.          Time arivalTime = new Time();
```

```
10.          arivalTime.setHour(19);
11.          System.out.println("Arival Time is " +
12.                              arivalTime.getHour() + ":" +
                                 arivalTime.getMinute() );
13.      }
14.  }
15.
16.  class Time {
17.      byte hour;
18.      byte minute;
19.      void setHour(int h) { hour = (byte)h; }
20.      void setMinute(int m) {minute = (byte)m; }
21.      byte getHour() { return hour; }
22.      byte getMinute() { return minute; }
23.  }
```

เมท็อด*ตัวเปลี่ยน*และเมท็อด*ตัวเข้าถึง*มีทั้งข้อดีและข้อเสีย ข้อเสียที่เห็นได้ชัดก็คือ ต้องเขียนเป็นเมท็อดแทนที่จะให้คลาสอื่นเข้าถึงตัวแปรโดยตรง ทำให้เสียเวลาเขียนโปรแกรมมากขึ้น ทำให้โปรแกรมใหญ่โตมากขึ้น และประสิทธิภาพในการทำงานลดลงเพราะต้องมีขั้นตอนในการเรียกเมท็อดเพิ่มขึ้น เมื่อมีข้อเสียออกอย่างนี้ แล้วจะมีเมท็อด*ตัวเปลี่ยน*และเมท็อด*ตัวเข้าถึง*ไปทำไม ข้อดีของการใช้เมท็อดเหล่านี้ย่อมมีแน่ และดีพอที่จะลบล้างข้อเสียได้ด้วย

ข้อดีของการใช้เมท็อด*ตัวเปลี่ยน*และเมท็อด*ตัวเข้าถึง*คือ ทำให้ผู้ใช้คลาสไม่จำเป็นต้องสนใจในรายละเอียดว่าข้อมูลในคลาสเก็บอย่างไร ตัวแปรชื่ออะไร เป็นตัวแปรชนิดไหน ให้ผู้ใช้คลาสรู้แต่เพียงว่าต้องส่งพารามิเตอร์อะไรไปให้*ตัวเปลี่ยน*บ้าง และรู้ว่าจะรับข้อมูลอะไรชนิดใดกลับมาจาก*ตัวเข้าถึง* ผู้ใช้รู้แค่นี้ก็เพียงพอ เป็นการซ่อนข้อมูลที่แท้จริงไว้ภายในคลาสตามหลักการซ่อนข้อมูล(information hiding) ผู้ใช้คลาสไม่จำเป็นต้องรู้ว่าข้อมูลในคลาสเก็บอย่างไร จะเก็บอย่างไรนั้นเป็นเรื่องภายในที่ผู้ใช้ไม่จำเป็นต้องทราบ การทำอย่างนี้นอกจากทำให้ผู้ใช้ไม่ต้องศึกษาโครงสร้างข้อมูลภายในคลาสแล้ว ยังมีประโยชน์ที่สำคัญคือ หากมีความจำเป็นต้องปรับเปลี่ยนวิธีการจัดเก็บข้อมูลภายในคลาส ผู้เขียนคลาสมักไม่จำเป็นต้องเปลี่ยนวิธีเรียก*ตัวเปลี่ยน*และ*ตัวเข้าถึง* ซึ่งจะไม่กระทบต่อผู้ใช้คลาส ถึงแม้จะมีการเปลี่ยนแปลงวิธีการจัดเก็บข้อมูลภายในคลาสก็ตาม ผู้ใช้คลาสยังคงใช้คลาสแบบเดิม ถ้าเราปล่อยให้ผู้ใช้คลาสเข้าใช้ตัวแปรสมาชิกของคลาสโดยตรง ผู้ใช้คลาสจำเป็นต้องรู้ว่ามีการเปลี่ยนแปลงการจัดเก็บข้อมูลในคลาสไปอย่างไรบ้างและต้องแก้โปรแกรมให้สอดคล้องกัน การรักษาหลักการนี้ไว้เป็นเรื่องสำคัญ มิฉะนั้นแล้วผู้ใช้คลาสโดยเฉพาะผู้อื่นซึ่งอาจมีการใช้คลาสที่มีการปรับแก้ภายในนี้เป็นจำนวนมากต้องแก้ไขโปรแกรมตามไปด้วย สู้แก้ไขเป็นการภายในคลาสเพียงอย่างเดียวจะดีกว่า ดังนั้นถ้าไม่จำเป็นจริง ๆ แล้ว เราจะไม่เปลี่ยนวิธีการเรียก*ตัวเปลี่ยน*และ*ตัวเข้าถึง* แต่จะเปลี่ยนเฉพาะโครงสร้างข้อมูลและคำสั่งที่อยู่ภายใน*ตัวเปลี่ยน*และ*ตัวเข้าถึง*เพื่อให้สอดคล้องกับการจัดเก็บข้อมูลที่เปลี่ยนไปเท่านั้น หากการเปลี่ยนแปลงการจัดเก็บข้อมูลจำเป็นต้องกระทบ*ตัวเปลี่ยน*และ*ตัวเข้าถึง*อย่างหลีกเลี่ยงไม่ได้ เราอาจเขียน*ตัวเปลี่ยน*และ*ตัวเข้าถึง*ตัวใหม่เพิ่มขึ้นแต่ยังคงเก็บรักษา*ตัวเปลี่ยน*และ*ตัวเข้าถึง*ตัวเดิมไว้ถ้าทำได้ ทั้งนี้เพื่อรักษาความเข้ากันได้ (compatibility) เอาไว้นั่นเอง

การสร้าง*ตัวเปลี่ยน*และ*ตัวเข้าถึง*ยังช่วยลดความยุ่งยากในการใช้คลาสในบางกรณีได้ด้วย เช่น ถ้าให้ผู้ใช้คลาสทำการแก้ไขข้อมูลตัวแปรสมาชิกเองหลายตัว ผู้ใช้คลาสอาจหลงลืมแก้ไขไม่ครบถ้วน ถ้าทำเป็นเมท็อด*ตัวเปลี่ยน*โดยรับพารามิเตอร์หลายตัวในคราวเดียวกันจะช่วยป้องกันการหลงลืมแก้ไขข้อมูลไม่ครบได้ หรือในบางกรณีเราอาจไม่จำเป็นต้องเก็บข้อมูลบางอย่างไว้ในตัวแปรสมาชิกก็ได้ ถ้าผู้ใช้คลาสต้องการข้อมูลนั้น ให้เรียกใช้*ตัวเข้าถึง* *ตัวเข้าถึง*จึงจะคำนวณส่งไปให้ วิธีนี้ช่วยประหยัดเนื้อที่ในการเก็บข้อมูลของอ๊อบเจกต์โดยเฉพาะอย่างยิ่งถ้ามีการสร้างอ๊อบเจกต์หลายตัวจะยิ่งประหยัดได้มาก

การที่นักเขียนโปรแกรมจะเขียน*ตัวเปลี่ยน*และ*ตัวเข้าถึง*หรือไม่นั้น ขึ้นอยู่กับดุลยพินิจของนักเขียนโปรแกรมเองที่ต้องชั่งน้ำหนักว่าเหตุผลใดสำคัญกว่ากัน ในบางสถานการณ์เหตุผลเรื่องประสิทธิภาพของโปรแกรมมีความสำคัญเพียงพอที่จะทำให้เราเลือกใช้วิธีเข้าถึงข้อมูลโดยตรงโดยไม่ใช้*ตัวเปลี่ยน*และ*ตัวเข้าถึง*ก็ได้

ตัวอย่างที่ ๔-๖ ถึงแม้ว่าคลาส `Time` จะมีเมท็อดให้เรียกใช้งานก็ตามแต่เป็นเมท็อด*ตัวเปลี่ยน*และเมท็อด*ตัวเข้าถึง*เท่านั้น ยังไม่มีเมท็อดที่ทำงานให้เป็นชิ้นเป็นอันที่แท้จริง คราวนี้เรามาสร้างเมท็อดให้คลาสนี้ทำงานได้มากขึ้น สมมุติว่าเราต้องการให้อ๊อบเจกต์ของคลาสนี้มีความสามารถในการคำนวณบวกเพิ่มเวลาได้ด้วยตัวของมันเอง เราจะเขียนเมท็อดสำหรับบวกเพิ่มชั่วโมงชื่อว่า `addHour` เมท็อดสำหรับบวกเพิ่มนาทีชื่อว่า `addMinute` และเมท็อดสำหรับแสดงเวลาชื่อว่า `print` ตัวอย่างที่ ๔-๗ เป็นตัวอย่างที่เราเพิ่มเมท็อดทั้งสามนี้ให้กับคลาส `Time` นอกจากนี้ยังมีการปรับปรุงเมท็อด*ตัวเปลี่ยน* `setHour` และ `setMinute` ให้สามารถตรวจสอบข้อมูลด้วยว่าพารามิเตอร์ที่ผู้เรียกส่งมาให้นั้นถูกต้อง อยู่ในช่วงตัวเลขที่รับได้หรือไม่

ตัวอย่างที่ ๔-๗ คลาส `Time` ที่ปรับปรุงให้มีความสามารถมากขึ้น

```
1.  class Ex4_7 {
2.      public static void main(String args[]) {
3.          Time arivalTime = new Time();
4.          arivalTime.setHour(19);
5.          arivalTime.setMinute(30); arivalTime.print();
6.          arivalTime.addHour(3);    arivalTime.print();
7.          arivalTime.addMinute(45); arivalTime.print();
8.          arivalTime.addMinute(45); arivalTime.print();
9.          arivalTime.addMinute(1);  arivalTime.print();
10.     }
11. }
12.
13. class Time {
14.     byte hour;
15.     byte minute;
16.     boolean setHour(int h) {
17.         if ( h >=0 && h <= 24 ) {
18.             hour = (byte)h;
19.             return true;
20.         } else
21.             return false;
22.     }
23.     boolean setMinute(int m) {
24.         if ( m >=0 && m < 60 ) {
```

```
25.             minute = (byte)m;
26.             return true;
27.         } else
28.             return false;
29.     }
30.     byte getHour() { return hour; }
31.     byte getMinute() { return minute; }
32.     void addHour(int dh) {
33.         int h;
34.         h = hour + dh;
35.         if ( h >= 24 ) {
36.             hour = (byte)(h % 24);
37.             return;
38.         }
39.         hour = (byte)h;
40.     }
41.     void addMinute(int dm) {
42.         int m;
43.         m = minute + dm;
44.         if ( m >= 60 ) {
45.             minute = (byte)(m % 60);
46.             addHour( m / 60 );
47.         } else
48.             minute = (byte)m;
49.     }
50.     void print() {
51.         if ( hour == 0 && minute == 0)
52.             System.out.println("24:0");
53.         else
54.             System.out.println(hour + ":" + minute);
55.     }
56. }
```

เมท็อด `setHour` และ `setMinute` ได้รับการปรับปรุงให้ตรวจสอบความถูกต้องของพารามิเตอร์ที่รับมาก่อนที่จะนำค่าไปใส่ในตัวแปรสมาชิก `hour` และ `minute` เมท็อดเหล่านี้จะทำการตรวจสอบข้อมูลที่รับมาเป็นพารามิเตอร์ว่ามีค่าอยู่ในช่วงตัวเลขที่ถูกต้องหรือไม่ ถ้าถูกต้องจะทำงานต่อไปจนจบแล้วส่งค่าชนิด `boolean` เป็น `true` กลับออกไปถ้าค่าของพารามิเตอร์ไม่ถูกต้องจะไม่ทำอะไรนอกจากส่งค่า `false` กลับไปเป็นสัญลักษณ์บอกผู้เรียกให้ทราบว่าทำงานไม่สำเร็จเพราะพารามิเตอร์ไม่ถูกต้อง บรรทัดที่ ๑๗ ใช้ข้อความสั่ง `if` ตรวจสอบว่าพารามิเตอร์ `h` มีค่าอยู่ระหว่าง 0 ถึง 24 ใช่หรือไม่ ถ้าใช่ จะนำค่าของ `h` ไปใส่ในตัวแปรสมาชิก `hour` แต่เนื่องจากพารามิเตอร์ `h` เป็นชนิด `int` ซึ่งใหญ่กว่า `hour` ที่เป็นชนิด `byte` ดังนั้นจึงจำเป็นต้องหล่อหลอมค่าของ `h` ให้เป็นชนิด `byte` เสียก่อนจึงจะนำไปใส่ตัวแปร `hour` เป็นอันเสร็จภาระหน้าที่ไม่ต้องทำอะไรต่อไปแล้วจึง ย้อนกลับไปยังผู้เรียกโดยข้อความสั่ง `return` พร้อมทั้งส่งค่า `true` กลับออกไปด้วยเพื่อบอกให้ผู้เรียกรู้ว่าเมท็อดนี้ทำงานสำเร็จ

การที่เมท็อดจะคืนค่าใดกลับไปนั้นขึ้นอยู่กับผู้เขียนโปรแกรมว่าจะกำหนดให้เป็นค่าใด ในกรณีนี้ผู้เขียนโปรแกรมต้องการบอกให้ผู้เรียกใช้เมท็อดทราบว่าเมท็อดทำงานสำเร็จหรือไม่เพียงเท่านั้น จึงสามารถคืนค่าเป็นชนิด `boolean` ได้อย่างเพียงพอ โดยทั่วไปแล้วเรานิยมคืนค่าเป็น `true` หมายถึงทำหน้าที่ได้สำเร็จถูกต้อง การคืนค่าเป็น `false` หมายถึงการทำงานไม่สำเร็จ ซึ่งในที่นี้หมายถึงพารามิเตอร์ที่รับมามีค่าอยู่นอกช่วง

ตัวเลขที่เหมาะสม ดังนั้นถ้าพารามิเตอร์ที่รับมามีค่านอกช่วง 0 - 24 จะไม่มีการนำพารามิเตอร์นี้ไปใส่ในตัวแปร hour แต่จะเลิกทำงานในเมท็อดพร้อมทั้งคืนค่าเป็น false เพื่อบอกให้ผู้เรียกทราบว่าค่าพารามิเตอร์ที่ส่งมานั้นไม่ถูกต้อง ส่วนเมท็อด setMinute มีหลักการตรวจสอบและทำงานคล้ายกับเมท็อด setHour ต่างกันแต่เพียงค่าตัวเลขที่ตรวจสอบเท่านั้น สำหรับเมท็อดนี้จะตรวจสอบว่าค่านาทีที่ส่งมาเป็นพารามิเตอร์นั้นมีค่าอยู่ระหว่าง 0 ถึง 59 หรือไม่

เมท็อด addHour ทำหน้าที่ในการบวกเพิ่มค่าของชั่วโมง โดยนำค่าจากพารามิเตอร์ dh ไปบวกเพิ่มในตัวแปรสมาชิก hour เพื่อความสะดวกในการทำงานภายในเมท็อดจึงกำหนดตัวแปรใหม่ขึ้นมา ๑ ตัว ให้ชื่อว่า h เป็นชนิด int สำหรับใช้ในการเก็บพักข้อมูลชั่วโมงเป็นการชั่วคราว ตัวแปร h ถือว่าเป็นตัวแปรเฉพาะที่ (local variable) ใช้ได้เฉพาะภายในเมท็อด addHour นี้เท่านั้น บรรทัดที่ ๓๔ นำค่าพารามิเตอร์ dh บวกเข้ากับตัวแปรสมาชิก hour แล้วนำผลลัพธ์ที่ได้จากการบวกไปเก็บไว้ที่ตัวแปร h บรรทัดที่ ๓๕ ตรวจสอบว่าค่าชั่วโมงที่บวกกันแล้วนั้นมีค่ามากกว่าหรือเท่ากับ 24 หรือไม่ ถ้ามากว่าหรือเท่ากับ 24 จริง จะทำกลุ่มข้อความสั่งในวงเล็บปีกกาซึ่งถือเสมือนหนึ่งว่าเป็นเพียง ๑ ข้อความสั่งของ if กลุ่มข้อความสั่งในที่นี้หมายถึงบรรทัดที่ ๓๖ และ ๓๗ บรรทัดที่ ๓๙ ทำการหล่อหลอมค่าของ h จาก int ให้เป็น byte แล้วนำไปเก็บในตัวแปรสมาชิก hour

เมท็อด addMinute ในบรรทัดที่ ๔๑ ถึง ๔๙ ทำหน้าที่ในการเพิ่มค่าของตัวแปรสมาชิก minute ขึ้นตามค่าของพารามิเตอร์ หลักการทำงานคล้ายกับเมท็อด addHour แต่มีเพิ่มตรงที่ถ้าค่าของ m ซึ่งเก็บนาทีมีค่ามากกว่าหรือเท่ากับ 60 จะต้องทดให้กับหลักชั่วโมงและลดค่าของนาที

ผลลัพธ์จากการทำงานของตัวอย่างที่ ๔-๗ เป็นดังนี้

```
19:30
22:30
23:15
24:0
0:1
```

การปกป้องข้อมูล

การอนุญาตให้คลาสอื่นเข้ามาแก้ไขข้อมูลในอ็อบเจกต์ได้โดยตรงเป็นสิ่งที่ไม่ดีนัก เพราะในบางครั้งอ็อบเจกต์ควรจะรับทราบว่าข้อมูลภายในอ็อบเจกต์กำลังจะเปลี่ยน เพื่อจะได้ดำเนินการใด ๆ ให้เหมาะสม การเปลี่ยนแปลงข้อมูลใด ๆ ของอ็อบเจกต์จึงควรกระทำโดยเมท็อด*ตัวเปลี่ยน* ถึงแม้ว่าจะมีเมท็อด*ตัวเปลี่ยน*ให้ผู้ใช้คลาสใช้แล้วก็ตาม แต่ถ้ายังปล่อยผู้ใช้คลาสสามารถเปลี่ยนแปลงแก้ไขข้อมูลในอ็อบเจกต์ได้เองโดยตรงแล้วละก็ นับว่าเป็นอันตรายเป็นอย่างยิ่ง หลักการหนึ่งที่สำคัญของแนวคิดการสร้างโปรแกรมเชิงวัตถุคือ***การห่อหุ้ม*** *(encapsulation)* ซึ่งเป็นการป้องกันข้อมูลในอ็อบเจกต์ไม่ให้โปรแกรมจากภายนอกเข้ามาแก้ไข หลักการนี้เปรียบเสมือนว่าข้อมูลเป็นแกนกลางอยู่ภายในและมีเมท็อดเป็นเปลือกห่อหุ้มอยู่ด้านนอกคอยป้องกันไม่ให้ผู้อื่นเข้าใช้ข้อมูลโดยตรง ข้อมูลจะเข้าถึงได้โดยเมท็อดของมันเองเท่านั้น

จาวามีวิธีการป้องกันข้อมูลในอ็อบเจกต์ เราสามารถปกป้องข้อมูลจากการเข้าถึงจากภายนอกอ็อบเจกต์ได้โดยใช้*ตัวดัดแปรการเข้าถึง(access modifier)* ช่วย ตัวดัดแปรการเข้าถึงเป็นคำสำคัญที่ใช้ระบุพร้อมกับการประกาศ คลาส เมท็อด หรือตัวแปร เพื่อบอกคอมไพเลอร์ถึงสมบัติของการเข้าถึงอย่างที่เราต้องการ

ถ้าเราต้องการป้องกันไม่ให้คลาสอื่นเข้าถึงตัวแปรสมาชิกของคลาสโดยตรง เราควรระบุ*ตัวดัดแปรตัวแปร (variable modifier)* ไว้หน้าชนิดของตัวแปรสมาชิก ตัวดัดแปรที่ควรใช้คือ `private` จากตัวอย่างที่ ๔-๕ และ ๔-๖ ถ้าเราต้องการป้องกันตัวแปรสมาชิก `hour` และ `minute` เราควรประกาศตัวแปรทั้งสองนี้ดังนี้

```
private byte hour;
private byte minute;
```

หากมีการปกป้องข้อมูลในตัวแปรสมาชิกเช่นนี้แล้ว ข้อความสั่งที่เขียนไว้ในคลาสอื่นที่ต้องการเข้าถึงข้อมูลชุดนี้จะเข้าถึงโดยตรงไม่ได้ ถ้ายังมีการเข้าถึง(ใช้)ตัวแปรเหล่านี้โดยตรงอยู่ คอมไพเลอร์จะให้ข้อความแสดงข้อผิดพลาดออกมา ถือว่าคอมไพล์ไม่ผ่าน ไม่สามารถนำโปรแกรมไปใช้ได้

อินสแทนซ์เมท็อดและคลาสเมท็อด

เมท็อดที่แสดงในตัวอย่างที่ผ่านมาในบทนี้ล้วนแล้วแต่เป็นอินสแทนซ์เมท็อดทั้งสิ้น อินสแทนซ์เมท็อดจำเป็นต้องเรียกโดยอาศัยตัวแปรอ้างอิงที่อ้างอิงไปยังอ็อบเจกต์ที่ต้องการเรียก ในบางกรณีเราต้องการให้เมท็อดทำงานอะไรบางอย่างโดยที่ไม่เกี่ยวกับอ็อบเจกต์ของคลาสตัวมันเองเลย จึงไม่น่าที่จะต้องสร้างอ็อบเจกต์ขึ้นมาก่อนเพียงเพื่อจะเรียกใช้เมท็อดนั้น จาวาอนุญาตให้ทำอย่างนี้ได้โดยต้องสร้างเป็นคลาสเมท็อด เราสามารถประกาศคลาสเมท็อดได้โดยเพิ่มคำสำคัญ `static` ไว้หน้าชนิดของค่าส่งกลับ คำสำคัญ `static` เป็นหนึ่งในตัวดัดแปรเมท็อด ใช้สำหรับบ่งบอกสมบัติของเมท็อดว่าเป็นเมท็อดที่อยู่กับคลาสตายตัว ในการประกาศเมท็อดถ้าไม่มีคำสำคัญ `static` ปรากฏอยู่หน้าชื่อเมท็อดถือว่าเมท็อดนั้นเป็นอินสแทนซ์เมท็อด ตัวอย่างที่ ๔-๘ มีเมท็อด `starLine` เป็นคลาสเมท็อด ตัวอย่างที่ ๔-๙ มีคลาสเมท็อดชื่อว่า `maxOfThree` จะเห็นว่ามีการเรียกใช้เมท็อดทั้งสองนี้โดยไม่มีการสร้างอ็อบเจกต์แต่อย่างใด

การเรียกเมท็อด

เมท็อดจะทำงานก็ต่อเมื่อมีการสั่งให้มันทำ การสั่งให้เมท็อดทำงานเรียกว่า*การเรียกเมท็อด* (*method call* หรือ *method invocation*) เมท็อดอาจถูกเรียกใช้มาจากข้อความสั่งที่อยู่ในตัวเมท็อดของมันเอง หรือเรียกจากเมท็อดอื่นในคลาสเดียวกัน หรือจะถูกเรียกมาจากคลาสอื่นก็ได้ การเรียกเมท็อดจะระบุชื่อเมท็อดและอาจให้ข้อมูลต่าง ๆ แก่เมท็อดที่จะถูกเรียกต้องการใช้ในการทำงานตามหน้าที่ของมัน หากมีการเรียกเมท็อดที่อยู่ต่างอ็อบเจกต์กัน คือมีการเรียกเมท็อดของอีกอ็อบเจกต์หนึ่งจากอ็อบเจกต์หนึ่ง จำเป็นระบุตัวแปรอ้างอิงหรือที่อยู่ของอ็อบเจกต์ปลายทางที่เราต้องการเรียกด้วย ซึ่งการเรียกเมท็อดผ่านทางตัวแปรอ้างอิงโดยมีจุด (.) เป็นตัวคั่นระหว่างชื่อตัวแปรอ้างอิงกับชื่อเมท็อด นั้นได้กล่าวถึงไปแล้ว กรณีที่เมท็อดที่ต้องการเรียกเป็นเมท็อด

ประเภทคลาสเมท็อด เราสามารถใช้ชื่อคลาสแทนตัวแปรอ้างอิงได้ ถ้าเราสร้างคลาสของเราขึ้นใช้เอง ภายในคลาสที่เราเขียนขึ้นอาจมีการเรียกคลาสเมท็อดอื่นบ้าง และอาจมีการเรียกเมท็อดที่เราสร้างขึ้นใช้ภายในคลาสของเราเองบ้าง การเรียกเมท็อดภายในคลาสเดียวกันไม่จำเป็นต้องอาศัยตัวแปรอ้างอิงในการระบุที่อยู่หรืออ๊อบเจกต์ปลายทางแต่อย่างใด

การเรียกเมท็อดจะทำให้ลำดับการทำงานของโปรแกรมไม่เป็นไปตามลำดับที่ต่อเนื่อง เมื่อมีการเรียกเมท็อด ลำดับการทำงานของโปรแกรมจะเปลี่ยนไป จะมีการกระโดดจากจุดที่มีการเรียกเมท็อดไปทำงานตามข้อความสั่งที่อยู่ในเมท็อดที่ถูกเรียก เมื่อเมท็อดที่ถูกเรียกใช้ทำหน้าที่ของมันเสร็จสิ้นแล้ว โปรแกรมจะย้อนกลับไปทำงานต่อที่ตำแหน่งหลังจุดที่เรียกเมท็อดนั้นมา ถ้าเมท็อดที่ถูกเรียกนั้นมีการส่งค่าข้อมูลกลับไปด้วย (โดยใช้ข้อความสั่ง `return` และ*ชนิดของค่าที่ส่งกลับ* จะต้องไม่เป็น `void`) จุดที่มีการเรียกเมท็อดนั้นสามารถนำค่าที่เมท็อดส่งออกไปมาใช้ได้ ในกรณีนี้การเรียกเมท็อดจะทำตัวเสมือนตัวแปรซึ่งเราสามารถเอาค่าของมันไปใช้ได้

โดยทั่วไปแล้วนักเขียนโปรแกรมไม่จำเป็นต้องรู้ว่าเมท็อดที่เรียกใช้นั้นทำงานอย่างไร รู้แต่เพียงว่าทำอะไรได้ จะเรียกใช้อย่างไร ต้องการพารามิเตอร์อะไรบ้างหรือไม่ และจะส่งค่าอะไรกลับคืนมาก็เพียงพอแล้ว เปรียบเสมือนหัวหน้าสั่งให้ลูกน้องทำงาน เมื่อลูกน้องทำงานเสร็จก็จะเอาผลงานมาส่ง หัวหน้าไม่จำเป็นต้องรู้ว่าลูกน้องทำงานตามที่ได้รับมอบหมายได้อย่างไร ซึ่งลูกน้องอาจจะไปสั่งให้คนอื่นช่วยทำงานบางอย่างให้ด้วยก็ได้ เมท็อดเปรียบเสมือนลูกน้องที่รับข้อมูลมาจากหัวหน้าแล้วนำไปปฏิบัติตามภาระกิจที่ได้รับมอบหมาย ซึ่งเมท็อดที่ถูกเรียกใช้นี้อาจมีการเรียกใช้เมท็อดอื่นให้ช่วยทำงานอีกทอดก็ได้

ตัวอย่างที่ ๔-๘ เป็นตัวอย่างเมท็อดที่มีการทำงานภายในเมท็อด แต่ไม่มีการส่งค่าใด ๆ คืนกลับไปยังผู้เรียก

```
1:  class Ex4_8 {
2:      public static void main(String args[]) {
3:          E4_8.starLine(50);
4:          System.out.println("Would you like to drink Java?");
5:          starLine(60);
6:      }
7:      static void starLine(int n) {
8:          while (n-- > 0)
9:              System.out.print('*');
10:         System.out.println("");
11:     }
12: }
```

บรรทัดที่ ๗ ถึง ๑๑ เป็นเมท็อดที่เขียนขึ้นโดยให้ชื่อว่า `starLine` เมท็อดนี้ต้องการพารามิเตอร์หนึ่งตัว คือ `n` เป็นตัวแปรชนิดจำนวนเต็ม คำสำคัญ `void` ที่เขียนไว้หน้าชื่อเมท็อดบอกให้รู้ว่าเมท็อดนี้จะไม่ส่งค่าใด ๆ กลับคืนไปให้ผู้เรียก เมท็อดนี้จะพิมพ์ดอกจันทร์ตามจำนวน `n` ที่ผู้เรียกส่งมาโดยใช้ข้อความสั่ง `while` สำหรับควบคุมการทำซ้ำ ให้ทำบรรทัดที่ ๙ ซ้ำจำนวน `n` เที่ยว (ข้อความสั่ง while จะกล่าวถึงโดยละเอียดในบทที่ ๙) เมื่อวนทำซ้ำครบตามจำนวนแล้ว จะออกจากการทำซ้ำไปทำข้อความสั่งหลังชุดของ `while` คือบรรทัดที่ ๑๐ เมื่อทำงานเสร็จแล้วจะกลับคืนไปสู่ผู้เรียกโดยไม่ส่งค่าอะไรกลับคืนไปเลย การเรียกเมท็อดนี้จุดแรกอยู่ที่

บรรทัดที่ ๓ นิพจน์ `Ex4_8.starLine(50)` เป็นการเรียกเมท็อด `starLine` พร้อมทั้งส่งค่าจำนวนเต็ม 50 ไปให้เมท็อดนี้ด้วย จุดที่สองที่มีการเรียกเมท็อดนี้คือนิพจน์ `starLine(60)` ในบรรทัดที่ ๕ โปรดสังเกตุว่าการเรียกเมท็อดนบรรทัดที่ ๓ มีชื่อคลาส Ex4_8 กำกับไว้หน้าชื่อเมท็อดด้วย ส่วนบรรทัดที่ ๕ ไม่มีเนื่องจากเป็นการเรียกเมท็อดจากภายในคลาสเดียวกัน จึงไม่ต้องใส่ชื่อคลาสก็ได้ แต่ถ้าต้องการเรียกเมท็อดนี้จากคลาสอื่นจำเป็นต้องระบุชื่อคลาสด้วยเสมอ ตัวอย่างนี้แสดงให้เห็นว่าเราจะระบุชื่อคลาสไว้หน้าชื่อเมท็อดหรือไม่ก็ได้ถ้าเป็นการเรียกเมท็อดจากภายในคลาสของมันเอง

ความจริงแล้ว บรรทัดที่ ๒ ก็เป็นการกำหนดเมท็อดเช่นเดียวกัน เมท็อดนี้มีชื่อว่า `main` เราคุ้นเคยกับการเขียนเมท็อดทำนองนี้มาแล้วโดยที่อาจไม่รู้ว่ามันคือเมท็อด เมท็อด `main` เป็นเมท็อดที่มีความหมายพิเศษ เมท็อดนี้จะถูกเรียกใช้งานเป็นอันดับแรกในโปรแกรมของเรา ผู้เรียกใช้เมท็อดนี้ก็คือจาวาอินเทอร์พรีเตอร์นั่นเอง

เมท็อด `main` ต้องการพารามิเตอร์ชนิด `String` ซึ่งจะอธิบายในบทที่ ๙ ยังไม่ต้องใส่ใจในตอนนี้ ขอให้ใช้ตามตัวอย่างไปก่อน เมท็อด `main` สิ้นสุดที่บรรทัดที่ ๖ ส่วนร่างของเมท็อดนี้มี ๓ บรรทัดเริ่มตั้งแต่บรรทัดที่ ๓ ถึงบรรทัดที่ ๕ โปรแกรมนี้เริ่มต้นทำงานตั้งแต่บรรทัดที่ ๓ เป็นต้นไป บรรทัดที่ ๓ มีนิพจน์ `starLine(50)` ซึ่งเป็นการเรียกเมท็อด `starLine` โดยส่งค่าจำนวนเต็ม 50 ไปให้เมท็อดนี้ด้วย ณ จุดนี้ ที่ซึ่งมีการเรียกเมท็อด ลำดับการทำงานของโปรแกรมจะเปลี่ยนไป โปรแกรมจะกระโดดข้ามไปทำงานในบรรทัดที่ ๘ ซึ่งอยู่ในเมท็อด `starLine` ขณะที่เริ่มเข้ามาทำงานในเมท็อด `starLine` นี้นั้น ตัวแปร `n` จะมีค่าเป็น 50 ตามที่ผู้เรียกส่งมาให้ ดังนั้นเมท็อด `starLine` จะพิมพ์ดอกจันทร์จำนวน 50 ตัว แล้วจึงทำข้อความสั่งในบรรทัดที่ ๑๐ ซึ่งเป็นการพิมพ์บรรทัดใหม่ ต่อจากนั้นจะไปที่บรรทัดที่ ๑๑ พบว่าจบบล็อกของเมท็อด `starLine` เป็นอันเสร็จสิ้นการทำงานในเมท็อดนี้ โปรแกรมจะย้อนกลับไปทำงานต่อจากบรรทัดที่ ๓ ซึ่งในบรรทัดที่ ๔ จะมีการพิมพ์คำว่า `"Would you like to drink Java?"` ออกมา ต่อจากนั้นจึงไปทำงานในบรรทัดที่ ๕ ซึ่งเป็นการเรียกเมท็อด `starLine` อีกครั้งหนึ่งแต่คราวนี้ ส่งค่า 60 เป็นพารามิเตอร์ โปรแกรมจะเปลี่ยนลำดับการทำงาน กระโดดไปทำงานในเมท็อด `starLine` อีกครั้ง เมื่อทำงานเสร็จก็จะย้อนกลับมาทำงานต่อไปในบรรทัดที่ ๖ แต่ปรากฏว่าบรรทัดที่ ๖ ไม่มีข้อความสั่งใดเลย มีแต่วงเล็บปีกกาปิด นั่นหมายถึงว่าการทำงานในเมท็อด `main` ก็สิ้นสุดลงด้วย โปรแกรมจะไม่ไปทำงานในบรรทัดที่ ๗ ซึ่งอยู่ถัดมา แต่จะย้อนกลับไปยังผู้ที่เรียกเมท็อดนี้มาซึ่งก็คือกลับไปยังจาวาอินเทอร์พรีเตอร์นั่นเอง เป็นการจบสิ้นการทำงานของโปรแกรมนี้

ตัวอย่างที่ ๔-๙ เป็นตัวอย่างการเขียนเมท็อดที่มีการส่งค่าคืนกลับไปยังผู้เรียกด้วย ตัวอย่างนี้จะเขียนเมท็อดชื่อว่า `MaxOfThree` ขึ้นมาก่อน แล้วจึงตามด้วยเมท็อด `main` ถึงแม้ว่าจะเขียนเมท็อด `MaxOfThree` ขึ้นมาเป็นอันดับแรกก็ตาม แต่โปรแกรมนี้ไม่ได้เริ่มต้นทำงานที่เมท็อดนี้ ยังคงเริ่มต้นทำงานที่เมท็อด `main` อยู่ดี ถึงแม้ว่าจะเขียนไว้อันดับหลังก็ตาม เพราะว่าจาวาอินเทอร์พรีเตอร์จะเริ่มต้นโดยการเรียกใช้เมท็อด `main` เสมอ

ตัวอย่างที่ ๔-๙ เมท็อดที่มีการส่งค่าคืนกลับไปยังผู้เรียก

```
1:   class Ex4_9 {
2:        static int maxOfThree(int x, int y, int z)
3:        {
4:             int max = x;
5:
6:             if (y > max)
7:                  max = y;
8:             if (z > max)
9:                  max = z;
10:            return max;
11:       }
12:       public static void main(String args[]) {
13:            int a=10, b=20, c=30, d=40;
14:            int max;
15:            max = maxOfThree( a, b, c );
16:            System.out.println("The largest value is " + max );
17:
18:            max = maxOfThree( a, b, c) + d;
19:            System.out.println("Result of the expression is " + max);
20:       }
21:  }
```

ตัวอย่างที่ ๔-๙ นี้ เมท็อด `maxOfThree` ต้องการพารามิเตอร์จำนวน ๓ ตัว แต่ละตัวเป็นชนิดจำนวนเต็ม เมท็อดนี้มีการส่งค่ากลับไปให้ผู้เรียกเป็นข้อมูลชนิดจำนวนเต็มดูได้จากคำสำคัญ `int` ที่อยู่หน้าชื่อเมท็อด เมื่อมีการระบุว่าเมท็อดนี้จะส่งค่ากลับไปยังผู้เรียก ภายในเมท็อดจะต้องมีข้อความสั่ง `return` แล้วตามด้วยนิพจน์ที่ให้ผลลัพธ์เป็นข้อมูลชนิดเดียวกับที่ระบุไว้หน้าชื่อเมท็อด ในบรรทัดที่ ๑๐ มีข้อความสั่ง `return max` ไว้สำหรับบังคับให้โปรแกรมย้อนกลับไปยังผู้เรียกพร้อมทั้งส่งค่าของ `max` ไปให้ผู้เรียกด้วย

โปรแกรมนี้เริ่มต้นทำงานที่เมท็อด `main` โดยเริ่มต้นที่บรรทัดที่ ๑๓ ซึ่งมีการประกาศตัวแปร `a b c` และ `d` พร้อมทั้งกำหนดค่าเริ่มต้นให้แก่ตัวแปรเหล่านี้เป็น 10 20 30 และ 40 ตามลำดับ การเรียกเมท็อด `maxOfThree` เริ่มแรกในนิพจน์ `max = maxOfThree( a, b ,c )` ในบรรทัดที่ ๑๕ การเรียกเมท็อดในครั้งนี้มีการส่งค่าของตัวแปร `a b` และ `c` ซึ่งมีค่าเป็น 10 20 และ 30 ตามลำดับไปให้เมท็อด `maxOfThree` จากจุดที่มีการเรียกเมท็อดนี้ โปรแกรมจะกระโดดไปทำงานยังเมท็อด `maxOfThree` เมื่อทำงานเสร็จแล้วจะย้อนกลับมายังจุดนี้อีกครั้งพร้อมทั้งค่าผลลัพธ์เป็นจำนวนเต็มซึ่งมีค่าสูงที่สุดจากทั้ง ๓ ตัวที่ส่งเป็นพารามิเตอร์ไปให้ ซึ่งในที่นี้ก็คือ 30 ค่า 30 นี้จะเป็นผลของการเรียก `maxOfThree(a, b, c)` ซึ่งเป็นนิพจน์ทางขวาของตัวดำเนินการกำหนดค่า (=) ดังนั้นจึงนำค่า 30 ไปใส่ไว้ในตัวแปรทางซ้ายของตัวดำเนินการกำหนดค่าคือตัวแปร `max` เป็นผลให้หลังจากทำงานในบรรทัดที่ ๑๕ ผ่านไปแล้ว `max` จะมีค่าเป็น 30

การเรียกเมท็อด `maxOfThree` เขียนไว้อีกที่ในบรรทัดที่ ๑๘ ประกอบด้วยการเรียกเมท็อด `maxOfThree` โดยส่งค่าของ `a b` และ `c` เป็นพารามิเตอร์ ซึ่งจะส่งค่า 10 20 และ -30 ไปให้เมท็อด `MaxOfThree` เมท็อด `maxOfThree` ก็จะรับค่าทั้ง ๓ นี้มาหาค่าสูงสุดซึ่งได้แก่ 30 ดังนั้นจึงส่งค่า 30 กลับไป

ให้ผู้เรียกในบรรทัดที่ 18 ส่งผลให้ `maxOfThree(a, b, c)` ในบรรทัดที่ ๑๘ หลังจากทำเมท็อด `maxOfThree` แล้วมีค่าเป็น 30 ดังนั้นหลังจากทำงานในเมท็อด maxOfThree เรียบร้อย แล้วในบรรทัดที่ ๑๘ จึงมีความหมายเหมือนกับ `max = 30 + d;` ดังนั้นเมื่อสิ้นสุดการทำงานในบรรทัดนี้แล้ว `max` จะมีค่าเป็น 70

การส่งผ่านข้อมูลไปให้เมท็อด

ในการเรียกเมท็อดที่มีการรับค่าพารามิเตอร์ด้วยนั้น ก่อนที่โปรแกรมจะกระโดดไปทำงานยังเมท็อดที่กำลังจะถูกเรียก จะมีการหาค่าผลลัพธ์ของนิพจน์ที่ระบุอยู่ในรายการพารามิเตอร์ที่จะส่งไปให้เมท็อด นิพจน์ที่ว่านี้อาจประกอบด้วยค่าคงที่ ตัวแปร หรือการเรียกเมท็อดอย่างใดอย่างหนึ่งหรือหลายอย่างรวมกันโดยมีตัวดำเนินการเป็นตัวเชื่อม หากมีพารามิเตอร์ที่จะส่งไปให้เมท็อดหลายตัว ต้องคั่นนิพจน์ของแต่ละพารามิเตอร์ด้วยจุลภาค (,) ผลลัพธ์ของนิพจน์เหล่านี้จะถูกประเมินค่าตามลำดับจากซ้ายไปขวา แล้วส่งค่าที่ได้นี้ไปเก็บในตัวแปรพารามิเตอร์ของเมท็อด หลังจากนั้นโปรแกรมจึงจะกระโดดไปทำงานในเมท็อดที่ถูกเรียก กรณีที่ระบุตัวแปรไว้ในวงเล็บของการเรียกเมท็อด เช่น `MaxOfThree(a, b, c)` จะมีการทำสำเนาค่าของตัวแปร (ในกรณีตัวอย่างนี้คือ `a` `b` และ `c`) ไปให้เมท็อด ซึ่งหากเมท็อดจะทำการเปลี่ยนแปลงค่าของพารามิเตอร์ไปอย่างไรก็ตามจะไม่มีผลกระทบต่อค่าของตัวแปรต้นตอ `a` `b` และ `c` ที่ส่งไปให้เมท็อดแต่อย่างใด ส่งผ่านพารามิเตอร์ไปให้เมท็อดวิธีนี้เรียกว่า*ส่งผ่านโดยค่า* (pass by value) คือส่งเฉพาะค่าของตัวแปรไปเท่านั้น เมท็อดไม่ได้เข้ามาใช้ตัวแปรโดยตรง แต่ใช้สำเนาของมัน

ตัวแปรอัตโนมัติ

ตัวแปรและพารามิเตอร์ซึ่งประกาศไว้ในเมท็อดจัดว่าเป็นตัวแปรเฉพาะที่(local variable) ตัวแปรและพารามิเตอร์เหล่านี้จะเกิดขึ้นโดยอัตโนมัติ (เรียกว่าตัวแปรอัตโนมัติ - automatic variable) ไม่ได้เกิดขึ้นพร้อมกับการสร้างอ็อบเจกต์ แต่จะมีตัวตนเกิดขึ้นโดยอัตโนมัติเมื่อโปรแกรมเข้าสู่การทำงานในเมท็อดหรือบล็อกที่ประกาศตัวแปรเหล่านั้น ตัวแปรเหล่านี้จะมี*อายุ* (*duration* หรือ *lifetime*) อยู่ระยะหนึ่งเฉพาะช่วงเวลาที่โปรแกรมทำงานอยู่ในเมท็อดที่สร้างมันขึ้นมาเท่านั้น เมื่อโปรแกรมเริ่มเข้ามาทำงานในบล็อกที่มีการประกาศตัวแปรอัตโนมัติ ตัวแปรเหล่านี้จึงจะถูกสร้างขึ้นมาในหน่วยความจำและคงอยู่ไปตลอดเวลาตราบใดที่ยังทำงานคำสั่งต่าง ๆอยู่ภายในบล็อกที่ประกาศตัวแปรนั้นขึ้นมา เมื่อทำงานในบล็อกเสร็จสิ้นแล้ว โปรแกรมจะออกไปทำงานที่อื่นนอกบล็อก ตัวแปรอัตโนมัติต่าง ๆที่สร้างขึ้นมาในบล็อกนี้จะถูกทำลายไปด้วย เป็นอันสิ้นสุดอายุขัยของตัวแปรเหล่านั้นไปโดยอัตโนมัติ การใช้ตัวแปรแบบนี้ทำให้ประหยัดหน่วยความจำเพราะว่าพื้นที่ในหน่วยความจำจะถูกจับจองสำหรับเก็บข้อมูลก็ต่อเมื่อโปรแกรมเข้าไปทำงานในบล็อกที่ประกาศมันขึ้นมา และเมื่อโปรแกรมออกจากบล็อกที่ประกาศมันขึ้นมาแล้วหน่วยความจำที่เก็บตัวแปรประเภทนี้จะถูกคืนให้กับระบบ

ตัวแปรอัตโนมัติถูกสร้างขึ้นมาเมื่อโปรแกรมเข้าทำงานในบล็อกที่มีการประกาศตัวแปรนั้น ตัวแปรที่สร้างขึ้นนี้มิได้ถูกกำหนดค่าเริ่มต้นให้โดยอัตโนมัติเหมือนอย่างตัวแปรสมาชิก เป็นหน้าที่ของผู้เขียนโปรแกรม

เองที่จะต้องกำหนดค่าเริ่มต้นให้ก่อนที่จะนำตัวแปรไปใช้ หากมิได้กำหนดค่าให้กับตัวแปรก่อนนำไปใช้ คอมไพเลอร์จะไม่ยอมให้ผ่าน คอมไพเลอร์จะฟ้องว่า "ตัวแปรอาจยังไม่ได้รับการกำหนดค่าเริ่มต้น" (`"Variable ... may not have been initialized"`) ซึ่งเป็นกลไกหนึ่งของจาวาที่ป้องกันการหลงลืมการกำหนดค่าให้ตัวแปรก่อนใช้งาน หากไม่ป้องกันเช่นนี้ อาจเกิดข้อผิดพลาดในโปรแกรมโดยที่ผู้เขียนโปรแกรมไม่รู้ตัวและหาที่ผิดได้ยาก

ขอบเขตของตัวแปร

ขอบเขต(scope)ของตัวแปรใด ๆ หมายถึงส่วนใดส่วนหนึ่งของโปรแกรมที่สามารถอ้างอิงถึงตัวแปรนั้นได้ เช่นเมื่อมีการประกาศตัวแปรเฉพาะที่ขึ้นมาในบล็อกหนึ่งแล้ว ตัวแปรนี้สามารถอ้างถึงได้เฉพาะภายในบล็อกเดียวกันนี้หรือบล็อกที่ซ้อนลงชั้นในเท่านั้น

เราสามารถประกาศตัวแปรขึ้นมาใช้เป็นการเฉพาะภายในบล็อกต่าง ๆ ได้ ถ้ามีบล็อกซ้อนกันอยู่หลายชั้น บล็อกชั้นในสามารถอ้างถึงตัวแปรของบล็อกชั้นนอกได้ แต่ข้อความสั่งที่อยู่ในบล็อกชั้นนอกไม่สามารถอ้างถึงตัวแปรของบล็อกชั้นในที่ลึกลงไปกว่าได้ ในแต่ละบล็อกไม่สามารถตั้งชื่อตัวแปรซ้ำซ้อนกันได้ แต่ในบล็อกชั้นในสามารถตั้งตัวแปรใหม่โดยใช้ชื่อซ้ำกับตัวแปรที่อยู่ในบล็อกชั้นนอกได้โดยถือว่าเป็นคนละตัวกัน กรณีที่ตั้งชื่อตัวแปรเดียวกันแต่อยู่กันคนละบล็อกนั้น ขณะที่โปรแกรมกำลังทำคำสั่งของบล็อกนอกอยู่นั้น จะรู้จักเฉพาะตัวแปรที่ประกาศไว้ในบล็อกชั้นนอก เมื่อโปรแกรมทำงานเข้าสู่บล็อกชั้นในที่มีชื่อตัวแปรซ้ำกับตัวแปรที่อยู่ในบล็อกชั้นนอกนั้น ถ้าอ้างถึงตัวแปรที่ใช้ชื่อซ้ำกันนี้จะหมายถึงตัวแปรที่อยู่ในบล็อกชั้นใน และเมื่อโปรแกรมทำงานออกนอกบล็อกชั้นในก็จะกลับมารู้จักตัวแปรชื่อเดียวกันที่อยู่บล็อกชั้นนอกอีกครั้งและจะไม่รู้จักตัวแปรที่อยู่ในบล็อกชั้นในอีกต่อไป

การซ้ำชื่อเมท็อด

ภาษาจาวายอมให้มีการตั้งชื่อคลาสเมท็อดเดียวซ้ำกันได้ ตราบใดที่เมท็อดเหล่านี้มีชุดพารามิเตอร์ที่แตกต่างกัน(แตกต่างกันโดยจำนวนพารามิเตอร์ หรือ ชนิดของพารามิเตอร์ หรือลำดับของพารามิเตอร์) เมท็อดหลายเมท็อดที่ใช้ชื่อเดียวกันนี้เรียกว่า*การซ้ำชื่อเมท็อด*(method overloading) เมื่อมีการเรียกเมท็อดที่มีการซ้ำชื่อ คอมไพเลอร์จาวาจะเลือกเมท็อดที่เหมาะสมได้ถูกตัวโดยดูจากจำนวน ชนิด และลำดับของพารามิเตอร์ที่ใช้เรียก เมท็อดที่ใช้ชื่อซ้ำกันเหล่านี้อาจมีชนิดของข้อมูลที่จะส่งกลับ(return type)แตกต่างกันก็ได้ ความแตกต่างของข้อมูลส่งกลับจะไม่มีผลต่อการเลือกเมท็อดแต่อย่างใด พารามิเตอร์เท่านั้นที่ใช้ในการเปรียบเทียบเพื่อเลือกว่าจะเรียกเมท็อดใด โดยทั่วไปแล้วเรามักสร้างเมท็อดที่ใช้ชื่อเดียวกันหลาย ๆตัวที่ทำหน้าที่คล้ายกัน แต่จะแตกต่างกันที่ชนิดของข้อมูลที่ต้องการกระทำ

ตัวอย่างที่ ๔-๑๐ เป็นตัวอย่างการซ้ำชื่อเมท็อด ตัวอย่างนี้มีการสร้างเมท็อดชื่อ `square` ขึ้นมา ๒ ตัว ตัวแรกมีพารามิเตอร์ ๑ ตัวเป็นชนิด `int` ตัวที่ ๒ มีพารามิเตอร์ ๑ ตัวเป็นชนิด `double` ในบรรทัดที่ ๖ มีการเรียกเมท็อด `square` พร้อมกับส่งค่าของตัวแปร `y` ซึ่งเป็นจำนวนเต็มไปให้ คอมไพเลอร์จะรู้ได้ว่าเราต้องการ

เรียกเมท็อด square ตัวที่ต้องการพารามิเตอร์เป็น int ซึ่งเมท็อดตัวนี้เขียนไว้เริ่มตั้งแต่บรรทัดที่ ๑๐ ส่วนบรรทัดที่ ๗ มีการเรียกเมท็อด square อีกครั้งหนึ่ง แต่คราวนี้ส่งพารามิเตอร์เป็น z ซึ่งมีค่าเป็นชนิด double ดังนั้นคราวนี้คอมไพเลอร์จะเรียกเมท็อด square อีกตัวหนึ่งซึ่งเริ่มบรรทัดที่ ๑๔

ตัวอย่างที่ ๔-๑๐ การซ้ำชื่อเมท็อด

```
1:   class MethodOverload {
2:       public static void main (String args[]) {
3:           int y=5;
4:           double z = 25.0;
5:
6:           System.out.println("The square of " + y + " is "
                                + square(y));
7:           System.out.println("The square of " + z + " is "
                                + square(z));
8:       }
9:
10:      static int square(int x) {
11:          return x * x;
12:      }
13:
14:      static double square(double x) {
15:          return x * x;
16:      }
17:  }
```

ผลลัพธ์จากการทำงานของโปรแกรมเป็นดังนี้

```
The square of 5 is 25
The square of 25 is 625.0
```

ตัวสร้าง

เมื่อมีการสร้างอ็อบเจกต์ใหม่ขึ้นมา เรามักต้องการทำอะไรบ้างอย่างในช่วงเริ่มต้นของการสร้างอ็อบเจกต์ เช่นทำการกำหนดค่าเริ่มต้นให้กับข้อมูลในอ็อบเจกต์ การทำงานอะไรบ้างอย่างเพื่อเตรียมความพร้อมก่อนที่อ็อบเจกต์จะถูกเรียกใช้ในครั้งต่อ ๆไป การที่จะให้อ็อบเจกต์ใด ๆ ทำงานบางอย่างในช่วงเริ่มต้นชีวิตของมัน เราสามารถทำได้โดยเขียนงานที่ต้องการให้ทำใส่ไว้ในเมท็อดใดเมท็อดหนึ่งแล้วหาโอกาสเรียกเมท็อดนั้น จาวามีวิธีการอำนวยความสะดวกในการเรียกเมท็อดที่มีวัตถุประสงค์นี้ โดยการกำหนดให้มีเมท็อดพิเศษซึ่งจะถูกเรียกโดยอัตโนมัติเมื่อมีการสร้างอ็อบเจกต์ใหม่ เมท็อดพิเศษที่ว่านี้เรียกว่า*ตัวสร้าง(constructor)*

ตัวสร้างเป็นเมท็อดที่มีชื่อเดียวกับชื่อคลาส มีสมบัติพิเศษที่แตกต่างจากเมท็อดทั่วไปคือ เมท็อดนี้จะถูกเรียกใช้เองโดยอัตโนมัติเมื่อมีการสร้างอ็อบเจกต์ใหม่ นักเขียนโปรแกรมไม่สามารถเรียกเมท็อดตัวสร้างได้เองโดยตรง คอมไพเลอร์จะแทรกข้อความสั่งสำหรับเรียกตัวสร้างให้เองทุกครั้งที่มีการใช้ตัวดำเนินการ new ข้อแตกต่างจากเมท็อดทั่วไปที่สำคัญอีกอย่างคือ ห้ามระบุชนิดของข้อมูลที่จะส่งคืนจากตัวสร้างในตอนประกาศเมท็อดตัวสร้าง ห้ามแม้กระทั่งคำสำคัญ void ก็ใส่ไม่ได้ ถึงแม้จะไม่คืนค่าใด ๆ เลยจากตัวสร้างก็ตาม ทั้งนี้

เป็นเพราะว่า โดยข้อเท็จจริงแล้ว ตัวสร้างจะคืนค่ากลับมาเป็นที่อยู่ของอ็อบเจกต์ที่สร้างขึ้นมาใหม่เสมอ ไม่เปิดโอกาสให้ผู้เขียนโปรแกรมส่งอย่างอื่นกลับคืนออกมา เพื่ออำนวยความสะดวก ป้องกันข้อผิดพลาดและความสับสน คอมไพเลอร์จึงจัดการเรื่องนี้เสียเอง โดยหลังจากทำงานในตัวสร้างตามที่ผู้เขียนโปรแกรมกำหนดไว้เสร็จแล้ว คอมไพเลอร์จะแทรกข้อความสั่ง `return` ให้ส่งค่าที่อยู่ของอ็อบเจกต์กลับคืนออกมาด้วย สำหรับชนิดของค่าส่งกลับจะได้เป็นที่อยู่หรือค่าอ้างอิงของอ็อบเจกต์ของคลาสมันเอง ซึ่งไม่จำเป็นต้องระบุไว้ตอนประกาศตัวสร้างเช่นเดียวกัน

รูปแบบของการเขียนตัวสร้างเป็นดังนี้

ชื่อตัวสร้าง (*รายการพารามิเตอร์*$_{\text{ละได้}}$)
{
    การประกาศตัวแปรและข้อความสั่งต่าง ๆ
}

ชื่อตัวสร้าง คือชื่อเมท็อดต้องใช้ชื่อเดียวกับชื่อคลาสเสมอ

รายการพารามิเตอร์ คือตัวแปรต่าง ๆ ที่ตั้งขึ้นมาเพื่อรับพารามิเตอร์ที่ผู้เรียกส่งมาให้ (คอมไพเลอร์เรียกให้แทนผู้เขียนโปรแกรม) *รายการพารามิเตอร์*เป็นเพียงตัวเลือกเท่านั้นจะมีหรือไม่มีก็ได้แล้วแต่ความจำเป็นว่าต้องการให้ผู้เรียกส่งข้อมูลมาให้ตัวสร้างหรือไม่

ตัวสร้างที่ไม่รับพารามิเตอร์ใดเลยเรียกว่า *ตัวสร้างโดยปริยาย (default constructor)* นักเขียนโปรแกรมที่เขียนคลาสขึ้นมาจะไม่เขียนตัวสร้างเลยก็ได้ถ้าไม่ต้องการทำงานอะไรเป็นพิเศษในตอนสร้างอ็อบเจกต์ ถึงแม้ผู้เขียนโปรแกรมจะไม่เขียนตัวสร้างก็ตาม คอมไพเลอร์จะเตรียมตัวสร้างโดยปริยายให้เอง เพื่อทำงานบางอย่างที่จำเป็น

ผู้เขียนโปรแกรมอาจเขียนตัวสร้างของแต่ละคลาสขึ้นมาหลายตัวก็ได้ แต่ต้องมีจำนวนหรือชนิดของพารามิเตอร์ที่แตกต่างกันตามหลักการซ้ำชื่อเมท็อด การที่คลาสหนึ่ง ๆมีตัวสร้างได้หลายตัวเรียกว่า*การซ้ำชื่อตัวสร้าง (constructor overloading)*

ตัวอย่างที่ ๔-๑๑ เป็นโปรแกรมที่ดัดแปลงมาจากตัวอย่างที่ ๔-๗ โดยเพิ่มตัวสร้างของคลาส `Time` เข้าไป ๒ ตัว ให้ตัวสร้างตัวแรกรับพารามิเตอร์ ๑ ตัวเป็นค่าตัวเลขชั่วโมงที่ต้องการกำหนดให้กับอ็อบเจกต์ ส่วนตัวสร้างตัวที่สองรับพารามิเตอร์ ๒ ตัว คือชั่วโมงและนาที ตัวสร้าง ๒ ตัวนี้ทำหน้าที่กำหนดค่าเริ่มต้นให้กับตัวแปรสมาชิก `hour` และ `minute` ถ้าต้องการกำหนดค่าเริ่มต้นให้ `hour` เพียงอย่างเดียวโดยต้องการให้ `minute` เป็นศูนย์ การกำหนดค่าเริ่มต้นให้ตัวแปรสมาชิก ตัวสร้างสามารถเปลี่ยนค่าได้เอง แต่ในตัวอย่างนี้มีการเรียกใช้เมท็อด `setHour` และ `setMinute` ให้ทำแทนให้เนื่องจากเมท็อดทั้งสองมีการตรวจสอบความถูกต้องของพารามิเตอร์ด้วย ตัวสร้างจึงไม่ต้องตรวจสอบเอง นักเขียนโปรแกรมสามารถกำหนดค่าเริ่มต้นให้ตัวแปร `hour` และ `minute` ไว้ในวงเล็บหลังชื่อคลาสหลังตัวดำเนินการ `new` ได้เลย ซึ่งถ้าจะว่าไปแล้วมันมีรูปแบบเหมือนกับการเรียกเมท็อดนั่นเอง นักเขียนโปรแกรมสามารถสั่งให้เรียกตัวสร้างตัวแรกได้โดยใส่

พารามิเตอร์เพียงตัวเดียวดังตัวอย่างที่บรรทัด ๒ ถ้าต้องการกำหนดค่าเริ่มต้นทั้งชั่วโมงและนาที สามารถสร้างอ็อบเจกต์ได้ตามตัวอย่างในบรรทัดที่ ๕

ตัวอย่างที่ ๔-11 แสดงคลาสที่มีการซ้ำชื่อตัวสร้าง

```
1.  class Ex4_11 {
2.      public static void main(String args[]) {
3.          Time time1 = new Time(4);
4.          time1.print();
5.          Time time2 = new Time(8, 30);
6.          time2.print();
7.          // Time time3 = new Time(); // default constructor no longer
    valid
8.      }
9.  }
10.
11. class Time {
12.     byte hour;
13.     byte minute;
14.     Time(int h) {
15.         setHour(h);
16.         minute = 0;
17.     };
18.     Time(int h, int m) {
19.         setHour(h);
20.         setMinute(m);
21.     }
22.     boolean setHour(int h) {
23.         if ( h >=0 && h <= 24 ) {
24.             hour = (byte)h;
25.             return true;
26.         } else
27.             return false;
28.     }
29.     boolean setMinute(int m) {
30.         if ( m >=0 && m < 60 ) {
31.             minute = (byte)m;
32.             return true;
33.         } else
34.             return false;
35.     }
36.     byte getHour() { return hour; }
37.     byte getMinute() { return minute; }
38.     void addHour(int dh) {
39.         int h;
40.         h = hour + dh;
41.         if ( h >= 24 ) {
42.             hour = (byte)(h % 24);
43.             return;
44.         }
45.         hour = (byte)h;
46.     }
47.     void addMinute(int dm) {
48.         int m;
49.         m = minute + dm;
50.         if ( m >= 60 ) {
```

```
51.             minute = (byte)(m % 60);
52.             addHour( m / 60 );
53.         } else
54.             minute = (byte)m;
55.     }
56.     void print() {
57.         if ( hour == 0 && minute == 0)
58.             System.out.println("24:0");
59.         else
60.             System.out.println(hour + ":" + minute);
61.     }
62. }
```

โปรดจำไว้ว่า ตัวสร้างถูกเรียกโดยตัวดำเนินการ `new` เท่านั้น เราไม่สามารถเรียกตัวสร้างโดยตรงได้เอง ดังนั้นเราจึงไม่สามารถเรียกใช้ตัวสร้างกับอ็อบเจกต์เก่าที่เกิดขึ้นแล้วเพื่อกำหนดค่าใหม่ให้กับตัวแปรสมาชิกอีกครั้งหนึ่งได้ ตัวอย่างเช่น ถ้าเราจะเปลี่ยนเวลาของอ็อบเจกต์ `time1` ให้เป็น 10 นาฬิกา 30 นาที จะเรียกตัวสร้างให้เปลี่ยนเวลาให้ด้วยวิธีการต่อไปนี้ไม่ได้

```
time1.Time(10, 30)
```

ถ้าเขียนโปรแกรมลักษณะข้างต้นนี้ จะเกิดข้อผิดพลาดขณะคอมไพล์ หากเรายังต้องการเปลี่ยนแปลงชั่วโมงและนาทีของอ็อบเจกต์อยู่ ควรใช้เมท็อด*ตัวเปลี่ยน* `setHour` และ `setMinute` ช่วย

เมื่อเราเขียนตัวสร้างตัวใดตัวหนึ่งของคลาสใด ๆ ขึ้นมา คอมไพเลอร์จะไม่*สร้างตัวสร้างโดยปริยาย*ของคลาสนั้นให้อีกต่อไป ดังนั้น การสร้างอ็อบเจกต์ตามตัวอย่างในบรรทัดที่ ๗ จึงทำไม่ได้อีกต่อไป ทั้ง ๆ ที่ก่อนที่เราจะเขียนตัวสร้างยังทำได้อยู่ หากเรายังคงต้องการสร้างอ็อบเจกต์โดยไม่ส่งพารามิเตอร์ เราจำเป็นต้องเขียนตัวสร้างที่ไม่รับพารามิเตอร์ขึ้นเอง เช่นตัวอย่างต่อไปนี้

```
Time( ) {
    hour = 0;
    minute = 0;
}
```

หากไม่อยากเขียนตัวสร้างที่ไม่ต้องการพารามิเตอร์ลักษณะนี้ขึ้นมา เราอาจเลี่ยงไปสร้างอ็อบเจกต์ด้วยข้อความสั่งดังต่อไปนี้

```
Time time3 = new Time(0, 0);
```

หรือ

```
Time time3 = new Time(0);
```

ทดแทนได้

ตัวแปรอินสแทนซ์และตัวแปรคลาส

ตัวแปรสมาชิก `hour` และ `minute` ของคลาส `Time` ในตัวอย่างที่ ๔-๕ ถึง ๔-๗ เป็นตัวแปรอินสแทนซ์ทั้งสิ้น ตัวแปรอินสแทนซ์ถือว่าอยู่กับอ็อบเจกต์ จะเกิดขึ้นก็ต่อเมื่อมีการสร้างอ็อบเจกต์ขึ้นมา อ็อบเจกต์

แต่ละตัวจะมีตัวแปรอินสแทนซ์เป็นของตนเองอิสระจากอ็อบเจกต์อื่น สมมุติว่ามีการสร้างอ็อบเจกต์ของคลาส Time ๑๐ ครั้ง จะได้ตัวแปรสมาชิก hour และ minute ๑๐ ชุดตามจำนวนอ็อบเจกต์ที่สร้างขึ้น เมื่อมีตัวแปรชื่อเดียวกันอยู่ในหลายอ็อบเจกต์เช่นนี้ จึงเลี่ยงไม่พ้นที่จะต้องระบุว่าเราจะใช้ตัวแปรอินสแทนซ์ใด ดังนั้นการเข้าถึงตัวแปรเหล่านี้จึงต้องใช้ควบคู่กับตัวแปรอ้างอิง เว้นเสียแต่ว่าอ้างอิงจากภายในอ็อบเจกต์(คลาส) ตัวมันเอง ไม่ต้องใช้ตัวแปรอ้างอิงใด ๆ

ตัวแปรอินสแทนซ์เกิดมาคู่กับอ็อบเจกต์ เมื่อไม่มีการใช้อ็อบเจกต์อีกต่อไป ระบบจะทำลายอ็อบเจกต์ให้เอง ตัวแปรอินสแทนซ์ก็จะสิ้นสุดลงด้วย ไม่สามารถใช้ตัวแปรนั้นได้อีกต่อไป

ตัวแปรคลาสต่างจากตัวแปรอินสแทนซ์คือไม่เก็บอยู่ที่อ็อบเจกต์แต่เก็บไว้ที่คลาสและมีอยู่ชุดเดียวเท่านั้น ไม่ว่าจะมีการสร้างอ็อบเจกต์ใหม่เพิ่มขึ้นอีกกี่ตัวก็ตาม ตัวแปรคลาสยังคงมีอยู่เพียงชุดเดียวเหมือนเดิม ตัวแปรคลาสถือว่าเป็นสมบัติร่วมของทุกอ็อบเจกต์ ทุกอ็อบเจกต์ของคลาสเดียวกันสามารถใช้ตัวแปรคลาสได้เท่าเทียมกัน ตัวแปรคลาสมีอยู่ที่เดียว ดังนั้นหากอ็อบเจกต์ใดอ็อบเจกต์หนึ่งทำการเปลี่ยนแปลงค่าของตัวแปรคลาสตัวใดไป เมื่ออ็อบเจกต์อื่นมาใช้ตัวแปรนั้นจะรับรู้ค่าใหม่ที่เปลี่ยนไปด้วย ต่างจากตัวแปรอินสแทนซ์ซึ่งอิสระเป็นของอ็อบเจกต์ใครอ็อบเจกต์มัน ถ้ามีการเปลี่ยนค่าตัวแปรอินสแทนซ์ในอ็อบเจกต์ใดไม่มีผลกระทบต่ออ็อบเจกต์อื่น

เราสามารถใช้ตัวแปรคลาสได้โดยที่ไม่จำเป็นต้องสร้างอ็อบเจกต์หรืออ้างอิงผ่านตัวแปรอ้างอิงก็ได้ กรณีที่เข้าถึงตัวแปรคลาสมาจากคลาสอื่นเพียงแต่ระบุชื่อคลาสแทนตัวแปรอ้างอิงเท่านั้น

ด้วยสมบัติที่กล่าวไปนี้ เราจึงควรเลือกใช้ชนิดของตัวแปรสมาชิกให้เหมาะสม ถ้ามีข้อมูลหลายชุดที่ต้องการจัดเก็บ ควรใช้ตัวแปรอินสแทนซ์ ถ้ามีข้อมูลเพียงชุดเดียว หรือเป็นข้อมูลที่ซ้ำซ้อนเหมือน ๆ กันควรเก็บเป็นตัวแปรคลาส

เราสามารถประกาศตัวแปรสมาชิกให้เป็นตัวแปรคลาสได้โดยเขียนคำสำคัญ static ไว้หน้าชนิดของตัวแปรสมาชิก ตัวอย่างเช่นคลาส Abc ต่อไปนี้

```
class Abc {
    int a;            // a is a instance variable
    static int b;     // b is a class variable
}
```

ตัวแปรสมาชิก a เป็นตัวแปรอินสแทนซ์ ส่วนตัวแปรสมาชิก b เป็นตัวแปรคลาส เราสามารถเข้าถึงตัวแปรคลาสได้โดยใช้ชื่อคลาสนำหน้าชื่อตัวแปรสมาชิก เช่น Abc.b

ตัวอย่างที่ ๔-๑๒ เป็นตัวอย่างแสดงการใช้ตัวแปรคลาสควบคู่กับตัวแปรของเมท็อด ตัวอย่างนี้แสดงให้เห็นคลาส Dice ซึ่งมีตัวแปรสมาชิก ๒ ตัว ตัวแรกชื่อว่า count เป็นตัวแปรคลาส ตัวที่ ๒ ชื่อว่า face เป็นตัวแปรอินสแทนซ์ ตัวแปร count ใช้นับจำนวนอ็อบเจกต์ของคลาส Dice ที่สร้างขึ้นมาแล้ว ตัวแปรนี้จะถูกเพิ่มค่าทีละ 1 ทุกครั้งที่ตัวสร้างของคลาส Dice ทำงาน ส่วนตัวแปร face ใช้เก็บเลขหน้าที่ออกของ

ลูกเต๋าแต่ละลูกซึ่งมีค่าที่เป็นไปได้ระหว่าง 1 ถึง 6 เนื่องจากอาจมีการสร้างอ็อบเจกต์ลูกเต๋าหลายลูก แต่ละลูกจึงควรเก็บข้อมูลหน้าที่ออกเป็นของตนเอง ดังนั้นจึงใช้เป็นตัวแปรอินสแทนซ์ ส่วนตัวแปร `count` ใช้นับจำนวนลูกเต๋าทั้งหมดที่สร้างขึ้น ควรมีตัวแปรนี้เพียงตัวเดียว จึงกำหนดให้เป็นตัวแปรคลาส

เมท็อด `toss` ทำหน้าที่กำหนดค่าให้ตัวแปร `face` โดยใช้คลาส `Random` ช่วยในการสร้างเลขแบบสุ่ม เนื่องจากลูกเต๋ามี ๖ หน้า จึงนำเลขสุ่มที่ได้มาหาเศษเหลือจากการหารด้วยตัวดำเนินการ `%` ค่าที่ได้เป็นเลขอยู่ระหว่าง 0 ถึง 5 แต่อาจมีค่าเป็นลบได้ จึงใช้ข้อความสั่ง `if` ตรวจสอบว่าเป็นลบ (น้อยกว่าศูนย์) หรือไม่ ถ้าน้อยกว่าทำให้เป็นบวกโดยใช้ตัวดำเนินการลบ (-) ต่อจากนั้นบวกด้วย 1 เพื่อทำให้ค่าเปลี่ยนเป็นอยู่ระหว่าง 1 ถึง 6 ตามที่ต้องการ

ในตัวอย่างนี้มีการสร้างอ็อบเจกต์หลายครั้ง ในแต่ละครั้งที่มีการสร้างอ็อบเจกต์ ตัวสร้างจะถูกเรียกให้ทำงาน โดยจะมีการเพิ่มค่าของตัวแปร `count` และเรียกเมท็อด `toss` เพื่อหาค่าตัวเลขสุ่มไปใส่ในตัวแปร `face` ของอ็อบเจกต์ที่สร้างใหม่

ตัวอย่างที่ ๔-๑๒ แสดงการใช้ตัวแปรคลาส

```
1.  import java.util.Random;
2.  class Ex4_12 {
3.      public static void main(String args[]) {
4.          System.out.println("Initial number of dices = " +
5.                             Dice.getNumberOfDices() );
6.
7.          Dice d1 = new Dice();
8.          System.out.println("Total number of dices = " +
9.                             Dice.getNumberOfDices() );
10.         System.out.println("Current face of dice 1 is " + d1.face);
11.
12.         Dice d2 = new Dice();
13.         System.out.println("Total number of dices = " +
14.                            Dice.getNumberOfDices() );
15.         System.out.println("Current face of dice 2 is " + d2.face);
16.
17.         Dice d3 = new Dice();
18.         System.out.println("Total number of dices = " +
19.                            Dice.getNumberOfDices() );
20.         System.out.println("Current face of dice 3 is " + d3.face);
21.
22.         System.out.println("Total number of dices = " +
23.                            d1.getNumberOfDices() );
24.         d3.toss();
25.         System.out.println("Current face of dice 3 is " + d3.face);
26.     }
27. }
28.
29. class Dice {
30.     static int count; // number of dices created
31.     byte face;
32.     Dice() {
33.         count++;
34.         toss();
```

```
35.      }
36.      void toss() {
37.          Random randNum = new Random();
38.          int x = randNum.nextInt() % 6;
39.          if (x < 0)
40.              x = -x;
41.          face = (byte) ++x;
42.      }
43.      static int getNumberOfDices() { return count; }
44.  }
```

ผลลัพธ์จากการทำงานของโปรแกรมตัวอย่างที่ ๔-๑๒ แสดงในกรอบสี่เหลี่ยมต่อไปนี้ จะเห็นว่า `Total number of dices` มีค่าเพิ่มขึ้นทุกครั้งที่มีการสร้างอ็อบเจกต์ของลูกเต๋าตัวใหม่ ส่วน `Current face of dice` จะมีค่าไม่แน่นอนในการดำเนินการของโปรแกรมแต่ละครั้ง เนื่องจากคำนวณมาจากเลขสุ่ม

```
Initial number of dices = 0
Total number of dices = 1
Current face of dice 1 is 2
Total number of dices = 2
Current face of dice 2 is 2
Total number of dices = 3
Current face of dice 3 is 3
Total number of dices = 3
Current face of dice 3 is 4
```

ค่าอ้างอิง *this*

ชื่อตัวแปรสมาชิกในคลาสหนึ่ง ๆ จะซ้ำกันไม่ได้ แต่ในเมท็อดต่าง ๆ ของคลาสเดียวกันนี้สามารถมีตัวแปรเฉพาะที่ที่มีชื่อซ้ำกับตัวแปรสมาชิกได้ หากใช้ชื่อเดียวกันแล้วจะเกิดความสับสนในการใช้ตัวแปรหรือไม่ คำตอบก็คือภาษาจาวามีหลักเกณฑ์ในการแยกแยะว่าจะใช้ตัวแปรใดในกรณีที่อ้างถึงชื่อเดียวกันอยู่แล้ว แต่สำหรับผู้เขียนโปรแกรมแล้ว ถ้าไม่เข้าใจหลักเกณฑ์หรือไม่ทันระวังอาจเกิดความสับสน เข้าใจผิดและโปรแกรมอาจทำงานผิดพลาดได้

แต่ละอ็อบเจกต์สามารถเข้าถึงตัวแปรสมาชิกของมันโดยอ้างชื่อตัวแปร สำหรับตัวแปรสมาชิกจะมีชื่อเต็มของมันคือ `this.`*`member`* เมื่อ `this` คือค่าอ้างอิงที่ชี้ไปยังอ็อบเจกต์ปัจจุบัน (current object)ที่บรรจุตัวแปรสมาชิกนั้นอยู่ ส่วน *`member`* หมายถึงชื่อสมาชิกที่ต้องการเข้าถึง ถ้าไม่มีความกำกวมเรื่องชื่อแล้วไม่จำเป็นต้องใช้ `this` ก็ได้ โดยทั่วไปแล้วเราใช้ `this` ในกรณีที่ตัวแปรเฉพาะที่หรือพารามิเตอร์ของเมท็อดมีชื่อซ้ำกับชื่อตัวแปรสมาชิกซึ่งกำหนดไว้ในคลาส เมื่อเป็นเช่นนี้จำเป็นต้องอ้างถึงตัวแปรสมาชิกด้วยชื่อเต็มโดยการใส่คำนำหน้า `this` ลงไปด้วยเพื่อบ่งบอกว่าต้องการใช้ตัวแปรสมาชิกของคลาสไม่ใช่ตัวแปรเฉพาะที่

ตัวอย่างที่ ๔-๑๔ ตัวสร้างในบรรทัดที่ ๑๑ มีพารามิเตอร์ชื่อ `hour` ซึ่งซ้ำชื่อกับตัวแปรสมาชิก ถ้าเราใช้ชื่อ `hour` โดยไม่มีคำนำหน้าชื่อกำกับไว้ด้วย จะหมายถึงตัวแปรที่ประกาศไว้ใกล้ตัวที่สุดคือตัวแปรเฉพาะที่ `hour` ไม่ใช่ตัวแปร `hour` ซึ่งเป็นสมาชิกของอ็อบเจกต์ ดังนั้นเพื่อไม่ให้กำกวมจึงต้องระบุ `this` ไว้หน้าตัวแปร `hour` ที่หมายถึงตัวแปรสมาชิก

ในบางครั้งเราต้องการเข้าถึงอ็อบเจกต์ปัจจุบันในฐานะของอ็อบเจกต์โดยรวม ไม่ใช่ตัวแปรสมาชิกตัวใดตัวหนึ่งเป็นการเฉพาะ จาวามีวิธีการอย่างย่อให้ใช้อย่างสะดวกโดยใช้ค่าอ้างอิง `this` ช่วย ถ้าใช้ในเมท็อด คำสำคัญ `this` อ้างถึงอ็อบเจกต์ที่เมท็อดนั้นกำลังดำเนินการอยู่

ตัวอย่างเช่น คลาสของจาวาจำนวนมากมีเมท็อดที่ชื่อ `toString()` สำหรับแปลงข้อมูลในอ็อบเจกต์ให้อยู่ในรูปสายอักขระ สมมุติว่าคลาสของเรามีเมท็อดนี้อยู่ด้วย เราสามารถพิมพ์ข้อมูลในอ็อบเจกต์อย่างง่าย ๆ ลักษณะต่อไปนี้ได้

```
System.out.println( this );
```

การทำเช่นนี้ใช้ประโยชน์ได้มากโดยเฉพาะอย่างยิ่งในการค้นหาจุดบกพร่อง(debug) ของโปรแกรม เพราะผลลัพธ์จากการเรียกเมท็อด `toString()` คือสายอักขระที่ได้มาจากการแปลงข้อมูลต่าง ๆ ที่อยู่ในอ็อบเจกต์ ซึ่งจะนำไปแสดงผลได้โดยสะดวก ทำให้เรารู้ว่าในขณะที่เรียกเมท็อด `toString()` นั้น ในอ็อบเจกต์มีค่าข้อมูลเป็นอย่างไรบ้าง

คำสำคัญ `this` มีอีกความหมายคือ ถ้า*ข้อความสั่งแรกของตัวสร้าง*อยู่ในรูป `this(...)` หมายถึงให้ตัวสร้างเรียกตัวสร้างตัวอื่นอีกตัวของคลาสเดียวกัน ดังตัวอย่างในบรรทัดที่ ๑๘ ของตัวอย่างที่ ๔-๑๓ `this` ในที่นี้เป็นการเรียกตัวสร้างที่รับพารามิเตอร์ ๒ ตัว ซึ่งอยู่ที่บรรทัด ๑๗ แทนที่เราจะเขียนตัวสร้างเองทั้งหมดเราอาจเรียกใช้ตัวสร้างตัวอื่นก็ได้ ตัวอย่างนี้ต้องการอำนวยความสะดวกให้นักเขียนโปรแกรมที่ใช้คลาสนี้ ถ้าต้องการให้นาทีของอ็อบเจกต์ใหม่เป็นศูนย์แต่ไม่อยากให้นักเขียนกำหนดนาทีให้เป็นศูนย์ให้ยุ่งยาก จึงเขียนตัวสร้างอีกตัวให้รับพารามิเตอร์เฉพาะชั่วโมง ส่วนข้อมูลนาทีตัวสร้างกำหนดให้เป็นศูนย์เอง แต่แทนที่จะเขียนตัวสร้างอีกตัวที่ทำหน้าที่ครบถ้วนเบ็ดเสร็จในตัวเอง เราไปเรียกใช้ตัวสร้างอีกตัวซึ่งเขียนไว้แล้ว วิธีนี้ทำให้โปรแกรมกระชับ ลดข้อยุ่งยากหากมีการแก้ไขโปรแกรมในภายหลัง ถ้าดูในตัวอย่างนี้อาจไม่เห็นความจำเป็นที่จะต้องเรียกใช้ตัวสร้างอีกตัว เพราะไม่เห็นตัวสร้างทำอะไรที่ซับซ้อน จะเขียนตัวสร้างขึ้นมาใหม่อีกตัวไม่ใช่เรื่องหนักหนาอะไร แต่โดยข้อเท็จจริงแล้ว โปรแกรมที่ใช้งานจริงมีความซับซ้อนกว่านี้หลายเท่านัก ตัวอย่างนี้พยายามไม่ให้โปรแกรมดูยุ่งยากเพื่อให้ง่ายแก่การทำความเข้าใจ หากเป็นโปรแกรมที่ใช้งานจริงแล้ว การกำหนดข้อมูลให้ตัวแปรสมาชิกควรมีการตรวจสอบความถูกต้องของข้อมูลด้วย เช่นไม่ยอมให้ชั่วโมงมีค่าเกิน 24 เป็นต้น การเรียกใช้ตัวสร้างที่มีอยู่แล้วให้เป็นประโยชน์จึงช่วยในการเขียนโปรแกรมเป็นอย่างมาก ทั้งนี้ในการออกแบบตัวสร้างควรแบ่งหน้าที่ให้ดี จึงจะเรียกใช้กันได้โดยไม่ทำงานซ้ำซ้อน

การใช้ `this(hour, 0)` ดูเหมือนว่าจะเป็นการเรียกเมท็อด `Time(hour, 0)` แต่ความจริงแล้วไม่ใช่เสียทีเดียว ตัวสร้างเป็นอะไรที่ซับซ้อนกว่าเมท็อดทั่วไป ก่อนที่เมท็อดจะถูกเรียก ตัวแปรอินสแทนซ์จะต้องถูกกำหนดค่าเริ่มต้นตามที่เราระบุหรือไม่ก็ถูกกำหนดให้เป็นค่าเริ่มต้นโดยปริยายตามแต่ชนิดของตัวแปร ด้วยเหตุนี้ภาษาจาวาจึงบังคับให้เราเรียกตัวสร้างเป็นอันดับแรกในเมท็อดก่อนที่จะไปทำข้อความสั่งอื่นใด หากคอมไพเลอร์อนุญาตให้เราทำข้อความสั่งอื่นก่อนได้ อาจเกิดข้อผิดพลาดอย่างร้ายแรงได้ เนื่องจากตัวแปรสมาชิกยังไม่ได้รับการกำหนดค่าเริ่มต้น เราอาจนำค่าที่ผิดไปใช้ หรือถ้าเราทำการเปลี่ยนแปลงค่าของตัวแปรสมาชิกไปแล้ว ถ้าเราเรียกตัวสร้างทีหลังได้จริง ตัวสร้างจะทำการกำหนดค่าเริ่มต้นของตัวแปรสมาชิกภาย

หลังจากที่เราเปลี่ยนค่าตัวแปรสมาชิกไปแล้ว ซึ่งอาจเป็นความผิดพลาดร้ายแรงที่เกิดขึ้นโดยเราไม่ทันระวังและค้นหาข้อผิดพลาดนี้ได้ยาก ดังนั้นคอมไพเลอร์จึงห้ามไม่ให้เราเรียกตัวสร้างเองโดยตรง ในบรรทัดที่ ๑๘ ถ้าเปลี่ยน `this(hour, 0)` ให้เป็น `Time(hour, 0)` แล้วคอมไพล์ใหม่ คอมไพเลอร์จะแจ้งว่าไม่พบเมท็อด `Time` เป็นการป้องกันไม่ให้ผู้เขียนโปรแกรมเรียกใช้ตัวสร้างได้เองโดยตรง

ตัวอย่างที่ ๔-๑๓ แสดงการใช้ค่าอ้างอิง this

```
1.  class Ex4_13 {
2.      public static void main(String args[] ) {
3.          Time TG234Time = new Time(18,30);
4.          Time TG456Time = new Time(15);
5.
6.          TG234Time.print();
7.          TG456Time.print();
8.          TG234Time.setFormat( 1);
9.          TG234Time.print();
10.         TG456Time.print();
11.     }
12. }
13.
14. class Time {
15.     static int format;
16.     int  hour, minute;
17.     Time(int hour, int mm) { this.hour = hour; minute = mm; }
18.     Time(int hour) { this(hour, 0); }
19.     void setFormat ( int format) { Time.format =format; }
20.     void print() {
21.         if (format == 0)
22.             System.out.println( hour + ":" + minute);
23.         else {
24.             if ( hour > 12)
25.                 System.out.println (hour-12 + ":" + minute + "PM");
26.             else
27.                 System.out.println( hour + ":" + minute + "AM");
28.         }
29.     }
30. }
```

ตัวอย่างที่ ๔-๑๓ นอกจากจะแสดงการใช้ค่าอ้างอิง `this` แล้ว ยังมีวัตถุประสงค์ให้เป็นตัวอย่างของการใช้ตัวแปรคลาสอีกตัวอย่างหนึ่งด้วย คลาส `Time` มีตัวแปรคลาส ๑ ตัวคือ `format` ใช้เก็บสมบัติโดยรวมของอ็อบเจกต์ ถ้าตัวแปร `format` มีค่าเป็นศูนย์ หมายถึงพิมพ์เวลาโดยใช้เลขชั่วโมงได้ถึง 24 ถ้าไม่ใช่ศูนย์หมายถึงการบอกเวลาแบบ AM PM ค่าของตัวแปร `format` เปลี่ยนครั้งเดียวมีผลต่อทุกอ็อบเจกต์ เพราะทุกอ็อบเจกต์ใช้ตัวแปรนี้ร่วมกัน เริ่มแรกตัวแปร format มีค่าเป็นศูนย์โดยปริยาย แต่ต่อมาถูกเปลี่ยนให้เป็น 1 อันเป็นผลมาจากการเรียกเมท็อด `setFormat` ในบรรทัดที่ ๘ บรรทัดนี้เรียกเมท็อด `setFormat` ของอ็อบเจกต์ `TG234Time` ถ้าตามไปดูการทำงานในเมท็อดแล้วจะเห็นว่ามีการนำค่าของพารามิเตอร์ชื่อ `format` ซึ่งรับค่ามาเป็น 1 ไปใส่ไว้ในตัวแปรคลาสที่ชื่อ `format` ซึ่งเป็นชื่อเดียวกัน เพื่อไม่ให้กำกวมจึงต้องใส่คำนำหน้าชื่อของตัวแปรคลาสด้วยชื่อคลาส เป็น `Time.format` ความจริงแล้วในกรณีนี้เราจะใส่คำนำหน้าชื่อ

ด้วย this เป็น this.format ก็ได้ ซึ่งหมายถึงตัวแปร format ของอ็อบเจกต์นี้ที่เมท็อดกำลังดำเนินการอยู่ แต่เนื่องจาก ตัวแปรสมาชิก format ตัวนี้เป็นตัวแปรคลาส ซึ่งมีเพียงตัวเดียวเท่านั้นไม่ได้อยู่ในอ็อบเจกต์ใด การเขียนเป็น Time.format จึงเป็นการสร้างความชัดเจน ว่า format ตัวนี้เป็นตัวแปรคลาส

ถึงแม้ว่าตัวแปรคลาส format จะถูกสั่งให้เปลี่ยนค่าโดยอ็อบเจกต์ TG234Time ก็ตาม แต่ผลของมันมีกับทุกอ็อบเจกต์ เพราะทุกอ็อบเจกต์ใช้ตัวแปรสมาชิก format ร่วมกัน ดังนั้น หลังจากเปลี่ยนค่าของตัวแปร format แล้ว เราสั่งให้แต่ละอ็อบเจกต์พิมพ์เวลาของมันออกมาอีกครั้ง คราวนี้จะพิมพ์อีกรูปแบบที่ต่างไปจากเดิม เนื่องจากเมท็อด print มี ๒ รูปแบบให้เลือก ผลลัพธ์จากการทำงานของตัวอย่างโปรแกรมนี้เป็นดังนี้

```
18:30
15:0
6:30PM
3:0PM
```

เมท็อดคลาสและอ็อบเจกต์

เมท็อดที่ประกาศโดยมีคำสำคัญ static กำกับอยู่ด้วยถือว่าเป็นเมท็อดคลาส(class method) เมท็อดชนิดนี้ถือว่าเป็นเมท็อดของคลาสไม่ขึ้นไม่เกี่ยวข้องกับอ็อบเจกต์ใด จะถูกเรียกโดยไม่ต้องอ้างถึงอ็อบเจกต์ การอ้างถึงตัวแปรอินสแทนซ์ (instance variable) จากภายในเมท็อดประเภทนี้ไม่ว่าจะมีค่าอ้างอิง this กำกับอยู่ด้วยหรือไม่ก็ตาม ถือว่าไม่ถูกต้อง จะเกิดข้อผิดพลาดขณะคอมไพล์ นั่นก็คือเมท็อดคลาสไม่มีสิทธ์ใช้ตัวแปรอินสแทนซ์ มันสามารถใช้ได้เฉพาะตัวแปรคลาสและตัวแปรเฉพาะบริเวณ (local variable) เท่านั้น

ส่วนเมท็อดที่ประกาศโดยไม่มีคำสำคัญ static กำกับอยู่ด้วยถือว่าเป็นเมท็อดของอ็อบเจกต์ดังที่กล่าวไปก่อนหน้านี้แล้ว เมท็อดประเภทนี้จะถูกเรียกควบคู่กับตัวแปรอ้างอิงที่อ้างถึงอ็อบเจกต์ที่เกี่ยวข้องเสมอ ซึ่งค่าอ้างอิงที่ใช้ตอนเรียกเมท็อดนี้จะถูกนำไปใช้เป็นค่าอ้างอิง this ในเมท็อดที่ถูกเรียกให้กำลังทำงาน

การสร้างแพคเกจ

ถ้าเราต้องการจัดระเบียบของแพคเกจ ตอนที่เราเขียนแพคเกจควรใส่ชื่อของแพคเกจไว้ในบรรทัดแรก ๆ ของแฟ้มโปรแกรมต้นฉบับ ก่อนที่จะถึงการกำหนดคลาสใด ๆ ในแพคเกจ การตั้งชื่อแพคเกจเราจะใช้คำสำคัญ package ช่วยดังตัวอย่างต่อไปนี้

```
package somchai.util;
```

การเขียนข้อความสั่งนี้หมายความว่า ทุกคลาสที่เขียนไว้ในแฟ้มโปรแกรมต้นฉบับเดียวกันนี้ถูกจัดให้อยู่ในแพคเกจ somchai.util ข้อความสั่ง package จะต้องเป็นข้อความสั่งที่มาก่อนข้อความสั่งอื่นใดในแฟ้ม สิ่งที่จะมาก่อนข้อความสั่งนี้ได้มีเพียงหมายเหตุ (comment) เพียงอย่างเดียวเท่านั้น

เมื่อเราสร้างแพคเกจ เราต้องนำแฟ้มที่ได้จากการคอมไพล์ (แฟ้มที่มีสกุลเป็น class) ไปใส่ไว้ในสารบบย่อยให้ถูกด้วย ตัวอย่างเช่นถ้าเราคอมไพล์แฟ้มต้นฉบับที่ระบุชื่อแพคเกจไว้ว่า

```
package somchai.util;
```

เราต้องเอาผลลัพธ์แฟ้มคลาสที่ได้ไปไว้ที่สารบบย่อย somchai\util เอง คอมไพเลอร์ไม่ใส่ให้เรา

ถ้าในแฟ้มโปรแกรมต้นฉบับไม่มีการประกาศชื่อแพคเกจ จาวาจะจัดให้คลาสต่าง ๆที่เขียนไว้ในแฟ้มนี้อยู่ในแพคเกจที่เรียกว่าแพคเกจโดยปริยาย (default package)

ขอบเขตของแพคเกจ

เราได้รู้จักตัวดัดแปรการเข้าถึงของตัวแปรสมาชิกมาบ้างแล้วในเรื่องการปกป้องข้อมูล ถ้าเราประกาศตัวแปรสมาชิกให้เป็น private ตัวแปรนี้จะเข้าถึงได้เฉพาะคลาสของมันเองเท่านั้นคลาสอื่นไม่สามารถใช้ตัวแปรนี้ได้ คลาสก็มีตัวดัดแปรการเข้าถึงทำนองเดียวกับตัวแปรสมาชิก แต่ใช้ด้วยวัตถุประสงค์ที่ต่างกัน กรณีของตัวแปรสมาชิกเรามักปกป้อง แต่กรณีของคลาสเรามักเปิดเผย เพราะว่าเราสร้างคลาสขึ้นมาไม่ใช่เพื่อให้คลาสเดียวกันใช้กันเอง เราสร้างคลาสขึ้นมาเพื่อให้คลาสอื่นใช้เป็นการทั่วไปเสียมากกว่า ดังนั้นจาวาจึงไม่ยอมให้ใช้ตัวดัดแปรการเข้าถึงคลาสเป็น private ตัวอย่างที่ผ่านมาในบทนี้ไม่ได้ระบุตัวดัดแปรการเข้าถึงคลาสเลย ถ้าไม่มีการระบุตัวดัดแปรการเข้าถึงในตอนกำหนดคลาส คลาสนั้นจะถูกใช้ได้เฉพาะในแพคเกจของมันเท่านั้น นั่นหมายความว่าคลาสเมท็อดอื่น ๆ ที่อยู่ในแพคเกจเดียวกันสามารถใช้มันได้ แต่คลาสเมท็อดในแพคเกจอื่นไม่สามารถใช้มันได้ ถ้าเราต้องการให้คลาสที่เขียนขึ้นสามารถใช้จากคลาสใด ๆ ก็ได้ ไม่จำเป็นว่าจะต้องเป็นคลาสในแพคเกจเดียวกัน เราต้องประกาศคลาสนั้นให้เป็นคลาสสาธารณะ โดยระบุตัวดัดแปรการเข้าถึงคลาสเป็น public ในตอนกำหนดคลาส

ถ้าเราสร้างคลาสให้เป็นสาธารณะเพื่อให้คลาสอื่นรู้จักและใช้ได้แล้ว มีประเด็นที่เราต้องพิจารณาต่อไปก็คือ การเข้าถึงสมาชิกของคลาส จะยอมให้คลาสอื่นเข้าถึงสมาชิกของคลาสไม่ว่าจะเป็นตัวแปรสมาชิกหรือเมท็อดสมาชิกก็ตามได้อย่างไรบ้าง

กรณีของตัวแปรสมาชิกถ้ากำหนดการเข้าถึงเป็น private คลาสอื่นไม่สามารถเข้าถึงมันได้ ถ้าไม่ระบุตัวดัดแปรการเข้าถึงไว้เลย ตัวแปรสมาชิกนั้นสามารถเข้าถึงได้จากทุกคลาสของแพคเกจเดียวกัน แพคเกจอื่นไม่สามารถเข้าถึงได้ ถึงแม้คลาสของมันจะเป็นสาธารณะก็ตาม ถ้าต้องการให้ตัวแปรสมาชิกสามารถเข้าถึงได้จากทุกคลาสรวมทั้งคลาสต่างแพคเกจด้วย ต้องระบุตัวดัดแปรการเข้าถึงตัวแปรเป็น public

ส่วนเมท็อดสมาชิก ถ้าไม่ระบุตัวดัดแปร หมายถึงเมท็อดนั้นสามารถเข้าถึงได้จากคลาสต่าง ๆ ในแพคเกจเดียวกัน ส่วนแพคเกจอื่นเข้าใช้ไม่ได้ ถ้าต้องการสงวนไว้ใช้เฉพาะคลาสตัวเองต้องระบุเป็น private ถ้าต้องการให้คลาสใดก็ใช้ได้ไม่จำเป็นต้องเป็นแพคเกจเดียวกัน ให้ระบุเป็น public

ตัวอย่างที่ ๔-๑๔ เป็นตัวอย่างของการสร้างคลาส Time ของแพคเกจ somchai.util ตัวอย่างนี้พยายามกำหนดตัวดัดแปรให้กับคลาส เมท็อดและตัวแปรสมาชิก

ตัวอย่างที่ ๔-๑๔ แสดงการสร้างคลาส Time ไว้ในแพคเกจ somchai.util

```
1. package somchai.util;
2. public class Time {
```

```
3.      private static int format;
4.      private byte hour;
5.      private byte minute;
6.      Time(int h) {
7.          setHour(h);
8.          minute = 0;
9.      };
10.     Time(int h, int m) {
11.         setHour(h);
12.         setMinute(m);
13.     }
14.     public boolean setHour(int h) {
15.         if ( h >=0 && h <= 24 ) {
16.             hour = (byte)h;
17.             return true;
18.         } else
19.             return false;
20.     }
21.     public boolean setMinute(int m) {
22.         if ( m >=0 && m < 60 ) {
23.             minute = (byte)m;
24.             return true;
25.         } else
26.             return false;
27.     }
28.     public byte getHour() { return hour; }
29.     public byte getMinute() { return minute; }
30.     public void addHour(int dh) {
31.         int h;
32.         h = hour + dh;
33.         if ( h >= 24 ) {
34.             hour = (byte)(h % 24);
35.             return;
36.         }
37.         hour = (byte)h;
38.     }
39.     public void addMinute(int dm) {
40.         int m;
41.         m = minute + dm;
42.         if ( m >= 60 ) {
43.             minute = (byte)(m % 60);
44.             addHour( m / 60 );
45.         } else
46.             minute = (byte)m;
47.     }
48.     public void print() {
49.         if (format == 0)
50.             System.out.println( hour + ":" + minute);
51.         else {
52.             if ( hour > 12)
53.                 System.out.println (hour-12 + ":" + minute + "PM");
54.             else
55.                 System.out.println( hour + ":" + minute + "AM");
56.         }
57.     }
58. }
```

จะเห็นว่ากำหนดคลาส `Time` ให้เป็น `public` เมท็อดทุกตัวกำหนดให้เป็น `public` เพื่อให้คลาสอื่นเรียกใช้ได้ ส่วนตัวแปรสมาชิกกำหนดให้เป็น `private` เพื่อป้องกันไม่ให้คลาสอื่นมาใช้ตัวแปรโดยตรง ต้องใช้ผ่านเมท็อดเท่านั้น

หมายเหตุ แฟ้มโปรแกรมต้นฉบับสามารถบรรจุคลาสสาธารณะได้เพียงคลาสเดียวเท่านั้น และชื่อคลาสสาธารณะนั้นต้องเป็นชื่อเดียวกับชื่อแฟ้มต้นฉบับด้วย

คลังคลาส

เมื่อเรามีคลาสสะสมมากเข้า เรามักอยากจัดระเบียบหรือจัดหมวดหมู่ของคลาสเหล่านั้น วิธีการจัดระบบที่ใช้ในภาษาจาวาคือการนำคลาสที่สัมพันธ์หรือเกี่ยวกับเรื่องใดเรื่องหนึ่งไปไว้รวมกันในสารบบเดียวกันที่เรียกว่าแพคเกจ เราอาจสร้างสารบบขึ้นมาเพื่อใช้เป็นที่เก็บรวบรวมคลาส ชื่อสารบบจะใช้เป็นชื่อแพคเกจ ที่รวมของแพคเกจต่าง ๆ เรียกว่าคลังคลาส (class library)

คำแนะนำในการออกแบบคลาส

๑. พยายามให้ข้อมูลเป็นส่วนตัว

ให้ยึดหลักการห่อหุ้ม ต้องปกป้องข้อมูลให้มากที่สุด ถึงเราจะต้องเขียน*ตัวเปลี่ยน*และ*ตัวเข้าถึง*บ้างในบางครั้ง แต่เราควรรักษาข้อมูลให้เป็นส่วนตัวจะดีกว่า นักเขียนโปรแกรมที่มีประสบการณ์จะพบว่าวิธีการจัดเก็บข้อมูลอาจมีการเปลี่ยนแปลงได้ แต่วิธีการใช้มักไม่ค่อยเปลี่ยน ถ้าเราให้ข้อมูลเป็นส่วนตัว การเปลี่ยนแปลงวิธีการจัดเก็บภายในจะไม่กระทบต่อผู้ใช้คลาส และสามารถค้นหาจุดบกพร่องในโปรแกรมง่ายกว่า

๒. กำหนดค่าเริ่มต้นให้ข้อมูลเสมอ

จาวาไม่ทำการกำหนดค่าเริ่มต้นให้ข้อมูลเฉพาะที่ แต่จะกำหนดค่าเริ่มต้นให้กับตัวแปรสมาชิกของอ็อบเจกต์ อย่าไว้วางใจค่าโดยปริยายที่คอมไพเลอร์ทำให้ เราควรทำการกำหนดค่าเริ่มต้นให้ตัวแปรเองให้เห็นชัดเจนจะดีกว่า ถึงแม้จะกำหนดค่าเดียวกับค่าโดยปริยายก็ตาม

๓. ไม่จำเป็นว่าทุกตัวแปรสมาชิกจะต้องมีตัวเข้าถึงและตัวเปลี่ยน

ข้อมูลบางตัวอาจมีความจำเป็นต้อง `get` หรือ `set` แต่ไม่ได้หมายความว่าตัวแปรทุกตัวจะต้องมี*ตัวเปลี่ยน*และ*ตัวเข้าถึง* บ่อยครั้งที่อ็อบเจกต์มีตัวแปรสมาชิกแต่ไม่ต้องการให้ผู้อื่น `get` หรือ `set`

๔. อย่าให้คลาสใหญ่โตจนเกินไป

ถ้าคลาสมีทีท่าว่าจะซับซ้อนเกินไป ควรแบ่งออกเป็นหลายคลาสที่ง่ายและเล็กลง แบ่งหน้าที่ความรับผิดชอบกันไป แต่ไม่ใช่แบ่งออกเป็นหลาย ๆ คลาส แต่ละคลาสมีขนาดเล็กจนแทบไม่มีอะไรจะทำ ตัวอย่างเช่น คลาส `Customer` ต้องการเก็บที่อยู่ของลูกค้าไว้ด้วย ถ้าคลาสเริ่มใหญ่โตจนซับซ้อน เราอาจแบ่ง

ข้อมูลเกี่ยวกับที่อยู่แยกออกไปเป็นอีกคลาส เช่นเป็นคลาส `Address` แล้วบรรจุคลาส `Address` เป็นข้อมูลสมาชิกของคลาส `Customer` แทน

๕. ตั้งชื่อคลาสและเมท็อดให้ตรงกับหน้าที่และความรับผิดชอบ

ตั้งชื่อให้สื่อความหมายและสะท้อนสิ่งที่มันทำ แนวทางที่ดีคือตั้งชื่อคลาสด้วยคำนาม ส่วนเมท็อดตั้งชื่อด้วยคำกริยา การตั้งชื่อควรเป็นไปตามข้อนิยมซึ่งนักเขียนโปรแกรมส่วนใหญ่ใช้กันคือนิยมตั้งชื่อคลาสขึ้นต้นด้วยอักษรโรมันตัวใหญ่ ส่วนชื่อเมท็อดนิยมขึ้นต้นด้วยอักษรโรมันตัวเล็ก ถ้าเป็นเมท็อด*ตัวเข้าถึง*ควรนำหน้าชื่อเมท็อดด้วย `get` ถ้าเป็นเมท็อด*ตัวเปลี่ยน*ควรนำหน้าชื่อเมท็อดด้วย `set`