จาวา

สำหรับผู้เริ่มต้น

จาวา สำหรับผู้เริ่มต้น
ISBN 974-09479-9-0
จำนวน 347 หน้า
เขียนโดย นรินทร์ โอฬารกิจอนันต์
- ซันจาวา 2 เซอร์ติฟายด์ โปรแกรมเมอร์ (SJCP)

สงวนลิขสิทธิ์ตามพระราชบัญญัติลิขสิทธิ์ พุทธศักราช 2537 เนื้อหาทั้งหมดในหนังสือเล่มนี้เป็นลิขสิทธิ์ของ นาย นรินทร์ โอฬารกิจอนันต์ แต่เพียงผู้เดียว ห้ามมิให้ผู้ใดนำส่วนหนึ่งส่วนใดหรือทั้งหมดของหนังสือเล่มนี้ไปหาประโยชน์ในทางธุรกิจ โดยไม่ได้รับอนุญาตจากผู้เขียนเป็นลายลักษณ์อักษร

สร้างสรรค์โดย
เดคิซูกิ ดอทเนท
URL : http://www.dekisugi.net/java
email : webmaster@dekisugi.net

บรรณานุกรม

-A Programmer's Guide to Java Certification, A Comprehensive Premier by Khalid A. Mughal and Rolf W. Rasmussen, Addison-Wesley, ISBN 0-201-59614-8

-The Java Tutorials : A short course on the basics by Mary Campione, Kathy Walrath and Alison Huml, Addison-Westley, ISBN 0-201-70393-9

-Essential Java 2 Fast : How to develop applications and applets with Java 2 (Essential Series) by John Cowell, Spinger Verlag ,ISBN 1-852-33071-6

โครงการหนังสือจาวาในอนาคต

-เจเอสพี สำหรับเวบโปรแกรมเมอร์
-เอ็นเตอร์ไพรส์ จาวาบีน สำหรับองค์กรธุรกิจ
-สร้างโปรแกรมบนออร์แกนไนเซอร์ด้วย เจทูเอ็มอี

จาวา เป็นเครื่องหมายการค้าของบริษัท ซัน ไมโครซิสเต็มท์ วินโดว์ เป็นเครื่องหมายการค้าของบริษัทไมโครซอฟท์ ยูนิกซ์ เป็นเครื่องหมายการค้าของบริษัท เอ็กซ์โอเพ็น แมคอินทอช เป็นเครื่องหมายการค้าของบริษัทแอปเปิล เนตสเคป เนวิเกเตอร์ เป็นเครื่องหมายการค้าบริษัทเอโอแอล

สารบัญ

บทที่ 1 จาวา และ โปรแกรมเชิงวัตถุ 5
บทที่ 2 จาวาเวอร์ชัวน์แมทชีน และ จาวาคอมไพล์เลอร์ 14
บทที่ 3 โปรแกรม Hello World 24
บทที่ 4 ค่าคงตัว และ ตัวแปรพื้นฐาน 30
บทที่ 5 เครื่องหมาย 44
บทที่ 6 บล๊อคเงื่อนไข 55
บทที่ 7 บล๊อควนลูป 65
บทที่ 8 คลาส และ วัตถุ 73
บทที่ 9 ตัวแปรอ้างอิง 83
บทที่ 10 ตัวแปรคลาส 89
บทที่ 11 อะเรย์ 96
บทที่ 12 แมธธอส 101
บทที่ 13 คอนสตรัคเตอร์ 116
บทที่ 14 ตัวแปรสตริง 122
บทที่ 15 คลาสสำหรับตัวแปรพื้นฐาน 134
บทที่ 16 คลาส Math 138
บทที่ 17 การสืบทอด 141
บทที่ 18 แพจเกจ 163
บทที่ 19 ตัวกำกับตัวแปรคลาส และแมธธอส 177
บทที่ 20 คลาส Object 183
บทที่ 21 อินเตอร์เฟส 190
บทที่ 22 คอลเล็กชั่น 194
บทที่ 23 เอ็กซ์เซฟชั่น 209
บทที่ 24 คลาสฟอร์แมท 220
บทที่ 25 วันที่และเวลา 223
บทที่ 26 เทรด 234
บทที่ 27 การอ่านเขียนไฟล์ 242
บทที่ 28 สวิง 259
บทที่ 29 การจัดวางหน้าจอ 272
บทที่ 30 กราฟฟิค 293
บทที่ 31 คีย์บอร์ดและเมาส์ 299
บทที่ 32 จาวาแอพเพลต 308

“ แด่ทุกคนที่รักบ้านเกิดเมืองนอน”

1

จาวา และ โปรแกรมเชิงวัตถุ

ไชโย ในที่สุดคุณก็ตัดสินใจที่จะเรียนรู้จาวาอย่างจริงๆ จังๆ เสียที

ใครที่เคยใช้อินเตอร์เนตต่างเคยได้ยินคำว่า จาวา มาแล้วจากที่ไหนสักแห่ง แต่ถ้าถามว่า จาวา คืออะไร คุณจะได้ยินคำตอบสารพัดรูปแบบ อาทิ กราฟฟิกบนเบราเซอร์ ภาษาคอมพิวเตอร์ หรือ แฟลดฟอร์ม บางคนสับสนกับคำว่า จาวาสคริปต์ ก็มี ในบทนี้คุณจะได้รู้จักกับคำว่า จาวา ดีขึ้น ไม่ว่าคุณจะเป็นผู้ใช้อินเตอร์เนต นักเขียนโปรแกรมมือใหม่ หรือ นักเขียนโปรแกรมภาษาอื่น ก็ตาม

เราได้ยินคำว่า จาวา จากอินเตอร์เนตมากที่สุด แต่ที่จริงแล้วจาวามีใช้ในเทคโนโลยีรูปแบบอื่นด้วย จะว่าไปแล้ว ต้นกำเนิดของจาวาไม่ได้เริ่มจากอินเตอร์เนต แต่เริ่มจากการพัฒนาภาษาคอมพิวเตอร์ที่ใช้สำหรับสร้างโปรแกรมขนาดจิ๋วบนเครื่องใช้อิเล็กทรอนิกส์ ปัจจุบันจาวามีที่ใช้อยู่ทุกหนทุกแห่ง ตั้งแต่ เครื่องใช้อิเล็กทรอนิกส์ ปาล์มออแกนไนเซอร์ คอมพิวเตอร์ส่วนบุคคล จนถึงเครื่องคอมพิวเตอร์แม่ข่ายระดับองค์กร แต่ จาวา เป็นที่รู้จักในวงกว้างเป็นครั้งแรกตอนที่มันถูกนำเข้าไปใช้ในการสร้างสีสันให้กับโฮมเพจบน

อินเตอร์เนต ทำให้หลายคนรู้จักจาวาในฐานะของกราฟฟิกเคลื่อนไหวบนโฮมเพจ หรือที่เราเรียกว่า จาวาแอพเพลต ทั้งที่ความจริงแล้วจาวามีความหมายกว้างกว่านั้น

บนอินเตอร์เนตมีเทคโนโลยีอีกตัวหนึ่งที่มีชื่อว่า จาวาสคริปต์ ซึ่งเป็นภาษาที่ใช้สำหรับกำกับเบราเซอร์ให้แสดงผลโฮมเพจให้มีลูกเล่นต่างๆ ตามใจชอบ จาวาสคริปต์ พัฒนาโดยบริษัทเนตสเคป โดยใช้โครงสร้างของภาษาจาวาเป็นพื้นฐาน บางคนนิยามสับสนระหว่างจาวา กับจาวาสคริปต์ ที่จริงแล้ว จาวาสคริปต์ ไม่ถือเป็นส่วนหนึ่งของจาวา และไม่เกี่ยวข้องกับจาวาแต่ประการใด

นิยามของจาวาที่จัดว่าเหมาะสมที่สุดมีสองนิยามได้แก่ จาวาคือภาษาคอมพิวเตอร์ และจาวาคือแฟลตฟอร์ม

ภาษาจาวา

ภาษาจาวา พัฒนาขึ้นโดยบริษัท ซัน ไมโครซิสเต็มท์ ชื่อของ จาวา มาจากชื่อชนิดของกาแฟที่ทีมวิศวกรของซันดื่ม ตอนที่ร่วมพัฒนาภาษาจาวาต้นแบบด้วยกัน จาวาเป็นเทคโนโลยีเปิด ที่มี ซัน เป็นผู้กำกับทิศทาง และคอยระวังไม่ให้ใครเอาจาวาไปดัดแปลงประยุกต์ใช้ในทางที่เบี่ยงเบนออกจากจุดประสงค์เดิมของมัน การกำหนดทิศทางโดยซันเป็นไปเพื่อให้เกิดความชัดเจนในแง่ของทิศทางการพัฒนา

ภาษาจาวามีคำสั่งพื้นฐานคล้ายภาษาซีพลัสพลัสเป็นอย่างมาก นักเขียนโปรแกรมที่ใช้ภาษาซีพลัสพลัสสามารถเรียนรู้ภาษาจาวาได้ในเวลาอันรวดเร็ว เหตุผลที่ทีมวิศวกรของซันไม่เลือกใช้ภาษาซีพลัสพลัสในการพัฒนาภาษาสำหรับโปรแกรมขนาดจิ๋วบนเครื่องใช้อิเล็กทรอนิกส์เป็นเพราะ เครื่องใช้อิเล็กทรอนิกส์ มีเนื้อที่สำหรับเก็บโปรแกรมจำกัด พวกเขาจึงสร้างภาษาคอมพิวเตอร์ภาษาใหม่ขึ้นมาให้ชื่อว่า **โอ๊ค** ซึ่งตั้งชื่อตามต้นไม้ใหญ่ที่อยู่ในสวนของบ้านที่ทีมวิศวกรใช้เป็นสถานที่สำหรับทำงาน ภาษาใหม่นี้มีความกระชับมากกว่าเดิม แต่มีคำสั่งพื้นฐานเหมือนภาษาซีพลัสพลัส เนื่องจากต้องการให้นักเขียนโปรแกรมภาษาซีพลัสพลัส ซึ่งมีอยู่มากที่สุดในขณะนั้นสร้างความคุ้นเคยได้อย่างรวดเร็ว ต่อมาพวกเขาเปลี่ยนชื่อภาษาใหม่นี้เป็น จาวา ตามชื่อชนิดของกาแฟ ที่พวกเขาดื่ม

ภาษาจาวาจัดเป็นภาษาคอมพิวเตอร์เชิงวัตถุเช่นเดียวกับภาษาซีพลัสพลัส แต่สิ่งที่ภาษาจาวาต่างกับ ภาษาซีพลัสพลัส เป็นอย่างมาก คือ โปรแกรมภาษาจาวาต้องเขียนเป็นแบบเชิงวัตถุเท่านั้น ในขณะที่ภาษาซีพลัสพลัส สามารถเขียนแบบเชิงวัตถุ หรือเขียนแบบโครงสร้าง ก็ได้ ที่เป็นเช่นนี้เนื่องจากภาษาซีพลัสพลัสมีต้นกำเนิดมาจากภาษาซี ซึ่งเป็นภาษาแบบโครงสร้าง ดังนั้นภาษาซีพลัสพลัสจึงต้องสนับสนุนการเขียนโปรแกรมแบบโครงสร้างด้วยเพื่อให้เข้ากันได้กับภาษาซี อย่างไรก็ตาม ภาษาแบบโครงสร้างเป็นเทคโนโลยีที่ล้าสมัย โปรแกรมประยุกต์ในท้องตลาดปัจจุบันนี้ล้วนแต่เขียนด้วยภาษาเชิงวัตถุทั้งสิ้น จาวาจึงไม่สนับสนุนภาษาโครงสร้าง

จาวาแฟลตฟอร์ม

นิยามที่เหมาะสมอันหนึ่งของจาวาคือ จาวาเป็น แฟลตฟอร์ม คำว่า **แฟลตฟอร์ม** โดยทั่วไปมีความหมายใกล้เคียงกับคำว่า ระบบปฏิบัติการ ตัวอย่างเช่น ระบบปฏิบัติการวินโดว์ บางทีเราก็เรียกว่า แฟลตฟอร์มวินโดว์ ความหมายตรงตัวของคำว่า แฟลตฟอร์ม ในพจนานุกรมหมายถึง สถานี เช่น ชานชลาสถานีรถไฟ สาเหตุที่เราเรียกระบบปฏิบัติการว่าเป็น แฟลตฟอร์ม เป็นเพราะ เวลาเราเขียนโปรแกรมประยุกต์อะไรก็ตามขึ้นมา เวลาเราจะใช้งานมัน เราจะต้องรันมันบนระบบปฏิบัติการ ตัวอย่างเช่น ไมโครซอฟท์เวิร์ด จะใช้งานได้ต้องรันบนระบบปฏิบัติการวินโดว์ ระบบปฏิบัติการวินโดว์จึงเป็นเสมือนสถานีปฏิบัติการสำหรับโปรแกรมไมโครซอฟท์เวิร์ด

โปรแกรมภาษาจาวา ไม่เหมือนโปรแกรมที่เขียนขึ้นด้วยภาษาคอมพิวเตอร์ภาษาอื่น โปรแกรมภาษาจาวาไม่ได้รันบนระบบปฏิบัติการ แต่รันบนแฟลตฟอร์มเสมือน ซึ่งเราเรียกว่า **จาวาแฟลตฟอร์ม** หรือ **จาวาเวอร์ชัวน์แมทชีน** ด้วยเหตุนี้เราจึงกล่าวว่า จาวา เป็นแฟลตฟอร์ม

จาวาเวอร์ชัวน์แมทชีน เป็นสิ่งที่ซ่อนโปรแกรมภาษาจาวาจากระบบปฏิบัติการของเครื่องคอมพิวเตอร์ โปรแกรมที่เขียนขึ้นด้วยภาษาจาวา ไม่ว่าจะนำไปรันบนระบบปฏิบัติการใดมันจะมองไม่เห็นความแตกต่างของระบบปฏิบัติการที่มันรันอยู่ เนื่องจากมันไม่ได้ติดต่อกับระบบปฏิบัติการของเครื่องคอมพิวเตอร์โดยตรง แต่มันจะติดต่อกับจาวาเวอร์ชัวน์แมทชีนแทน และจาวาเวอร์ชัวน์แมทชีนจะติดต่อกับระบบปฏิบัติการอีกที

จาวาเวอร์ชัวน์แมทชีนในทุกๆ ระบบปฏิบัติการมีหน้าตาเหมือนกันหมด ดังนั้นโปรแกรมที่เขียนขึ้นด้วยภาษาจาวาสามารถนำไปรันบนระบบปฏิบัติการใดก็ได้ หรือกล่าวอีกนัยหนึ่งก็คือ จาวาเวอร์ชัวน์แมทชีนก็คือระบบปฏิบัติการสำหรับโปรแกรมภาษาจาวา

รูปที่ 1-1 จาวาแฟลตฟอร์ม

ปกติแล้วโปรแกรมประยุกต์ที่เขียนด้วยภาษาอื่น ถ้าพัฒนาขึ้นมาเพื่อระบบปฏิบัติการใด จำเป็นที่จะต้องรันบนระบบปฏิบัติการนั้น เช่น ไมโครซอฟท์เวิร์ดสำหรับระบบปฏิบัติการวินโดว์จะต้องรันบนระบบปฏิบัติการวินโดว์เท่านั้น ไม่สามารถนำไปใช้งานบนระบบปฏิบัติการอื่นเช่น ลินิกซ์ หรือแมคอินทอชได้ เนื่องจากระบบปฏิบัติการแต่ละอันมีความแตกต่างกันอยู่ นี่เป็นความได้เปรียบของการเขียนโปรแกรมด้วยภาษาจาวา เพราะไม่ว่าจะเขียนขึ้นบนระบบปฏิบัติการใด เมื่อเขียนเสร็จแล้วจะสามารถนำไปรันได้บนระบบปฏิบัติการอื่นทุกระบบที่มีจาวาเวอร์ชัวน์แมทชีน เราเรียกคุณสมบัตินี้ของโปรแกรมภาษาจาวาว่า **`Write Once, Run Anywhere`**

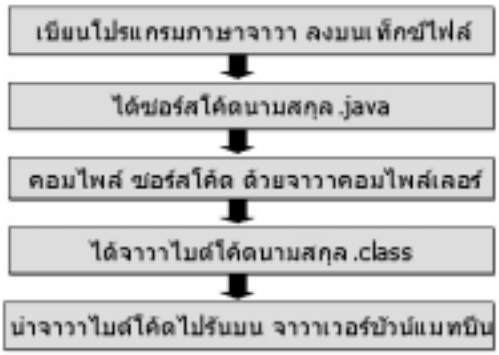
นักเขียนโปรแกรมภาษาจาวา

จาวาเวอร์ชัวน์แมทชีนไม่ได้มีอยู่แต่ในเฉพาะโลกของคอมพิวเตอร์ตั้งโต๊ะเท่านั้น แต่ยังมีอยู่บนทุกๆ สิ่งทุกอย่างตั้งแต่ สมาร์ทการ์ด โทรศัพท์มือถือ ปาล์มออแกนไนเซอร์ พีซีเบราเซอร์ หรือเครื่องคอมพิวเตอร์แม่ข่าย การเรียนรู้ภาษาจาวาจึงเป็นการลงทุนที่คุ้มค่าสำหรับนักเขียนโปรแกรม เพราะรู้ภาษาเดียวแต่สามารถนำไปใช้พัฒนาโปรแกรมบนอะไรก็ได้ นักเขียนโปรแกรมภาษาจาวาคนหนึ่งอาจใช้ภาษาจาวาพัฒนาโปรแกรมบนเครื่องคอมพิวเตอร์แม่ข่ายที่สำนักงานที่ตนทำงานอยู่ ยามว่างก็พัฒนาเวบไซต์ให้บริษัทอื่นเพื่อหา

รายได้พิเศษด้วยภาษาจาวา หรืออาจพัฒนาโปรแกรมเล็กๆ บนปาล์มออแกนไนเซอร์ไว้สำหรับแจกเพื่อนฝูงเป็นงานอดิเรกด้วยภาษาจาวา การที่ภาษาจาวาทำได้ทุกอย่าง ทำให้นักเขียนโปรแกรมที่เลือกเรียนภาษาจาวา ไม่จำเป็นต้องเรียนรู้ภาษาคอมพิวเตอร์ภาษาอื่นอีกเลย เราเรียกแนวคิดนี้ว่า **100% pure java**

ขั้นตอนการเขียนโปรแกรมภาษาจาวานั้นไม่ต่างกับการพัฒนาโปรแกรมในภาษาอื่นมากนัก การเขียนโปรแกรมเริ่มต้นจากการเขียนคำสั่งภาษาจาวาลงบนเท็กซ์ไฟล์ เราเรียกไฟล์เหล่านี้ว่า **ซอร์สโค้ด** ซึ่งซอร์สโค้ดภาษาจาวาจะต้องมีนามสกุลเป็น .java เสมอ

เมื่อเขียนโปรแกรมเสร็จแล้ว เราจะทำการคอมไพล์ซอร์สโค้ด การคอมไพล์ซอร์สโค้ด ก็คือการเปลี่ยนคำสั่งภาษาจาวาเป็นภาษาเฉพาะอย่างหนึ่งซึ่งจาวาเวอร์ชัวน์แมทชีนเข้าใจ ตัวที่ทำหน้าที่ในการคอมไพล์ซอร์สโค้ดเรียกว่า **จาวาคอมไพล์เลอร์** ซึ่งเป็นซอร์ฟแวร์ที่สามารถอ่านคำสั่งในไฟล์ .java ของคุณแล้วแปลเป็นภาษาเฉพาะที่จาวาเวอร์ชัวน์แมทชีนเข้าใจได้ ภาษาเฉพาะที่จาวาเวอร์ชัวน์แมทชีนเข้าใจนี้เราเรียกว่า **จาวาไบต์โค้ด** ซึ่งคอมไพล์เลอร์จะเก็บจาวาไบต์โค้ดที่ได้ไว้ในไฟล์นามสกุล .class ไฟล์นามสกุล .class ที่ได้จากจาวาคอมไพล์เลอร์นี่เองคือตัวโปรแกรมที่แท้จริงของคุณ เมื่อใดที่คุณต้องการรันโปรแกรมที่คุณเขียนขึ้น คุณก็เพียงแต่นำไฟล์ .class ไปรันบนจาวาเวอร์ชัวน์แมทชีน จาวาเวอร์ชัวน์แมทชีนเข้าใจจาวาไบต์โค้ดและจะทำงานตามคำสั่งในโปรแกรมที่คุณเขียนขึ้นโดยการอ่านจากจาวาไบต์โค้ด สรุปขั้นตอนการพัฒนาโปรแกรมได้ดังในรูป



รูปที่ 1-2 ขั้นตอนการพัฒนาโปรแกรมภาษาจาวา

จำไว้ว่า คุณอาจเข้าใจภาษาจาวาแต่จาวาเวอร์ชัวน์แมทชีนไม่เข้าใจ จาวาเวอร์ชัวน์แมทชีนเข้าใจจาวาไบต์โค้ดซึ่งคุณอ่านไม่รู้เรื่อง คอมไพเลอร์ก็คือตัวกลางที่จะแปลภาษาจาวาที่คุณเขียนให้กลายเป็นจาวาไบต์โค้ดนั้นเอง

โปรแกรมเชิงวัตถุ

การเขียนโปรแกรมคอมพิวเตอร์มีสองแบบ คือ การเขียนโปรแกรมแบบโครงสร้าง และการเขียนโปรแกรมเชิงวัตถุ

การเขียนโปรแกรมแบบโครงสร้างเป็นการเขียนโปรแกรมแบบที่มนุษย์คุ้นเคย คือ การเขียนคำสั่งเรียงต่อกันไปเรื่อยๆ ทีละบรรทัด โปรแกรมจะเริ่มทำงานจากคำสั่งแรกสุดเรื่อยไปจนถึงคำสั่งท้ายสุด เป็นอันจบโปรแกรม อาจมีการสร้างเป็นโปรแกรมย่อยๆ ในโปรแกรมใหญ่บ้าง เพื่อลดคำสั่งที่ซ้ำซ้อน แต่หลักการกว้างๆ ยังคงเหมือนเดิม ตัวอย่างของภาษาที่มีวิธีการเขียนโปรแกรมเป็นแบบโครงสร้างได้แก่ ภาษาเบสิก ภาษาโคบอล ภาษาฟอร์แทรน ภาษาปาสคาล และ ภาษาซี

การเขียนโปรแกรมเชิงวัตถุ มีการสร้างวัตถุสมมติขึ้นมาก่อน แล้วเขียนคำสั่งนิยามวัตถุนั้นจนสามารถทำให้วัตถุนั้นทำงานตามที่เราต้องการได้ ซอร์สโค้ดของโปรแกรมเชิงวัตถุแทนที่จะเป็นคำสั่งเขียนเรียงต่อกันไปเรื่อยๆ จะเป็นนิยามของวัตถุเขียนเรียงต่อไปเรื่อยๆ แทน และโปรแกรมจะทำงานได้เองถ้าวัตถุนั้นถูกนิยามขึ้นอย่างเหมาะสม การเขียนโปรแกรมเชิงวัตถุต้องใช้เวลาในการศึกษานานพอสมควร โดยเฉพาะอย่างยิ่งนักเขียนโปรแกรมต้องมีความชำนาญในการสร้างวัตถุสมมติที่ทำงานตามอย่างที่เราต้องการได้ โปรแกรมประยุกต์ที่เราใช้งานจริงในปัจจุบันล้วนแล้วแต่เขียนด้วยโปรแกรมเชิงวัตถุทั้งสิ้น การศึกษาการเขียนโปรแกรมเชิงวัตถุจึงเป็นสิ่งที่นักเขียนโปรแกรมรุ่นใหม่ทุกคนควรจะฝึกฝนไว้ ตัวอย่างของภาษาที่มีการเขียนโปรแกรมแบบเชิงวัตถุคือ ภาษาจาวา และภาษาซีพลัสพลัส (ภาษาซีพลัสพลัสเขียนได้ทั้งแบบโครงสร้างและวัตถุ)

การที่โปรแกรมภาษาจาวาต้องเขียนเป็นแบบเชิงวัตถุเสมอ จัดว่าเป็นทั้งจุดเด่นและจุดด้อยของภาษาจาวา การที่ภาษาจาวาไม่สนับสนุนการเขียนโปรแกรมแบบโครงสร้าง ซึ่งเป็นวิธีการเขียนโปรแกรมที่ล้าสมัย ทำให้ภาษาจาวามีความกะทัดรัดมากกว่าภาษาซีพลัสพลัส แต่

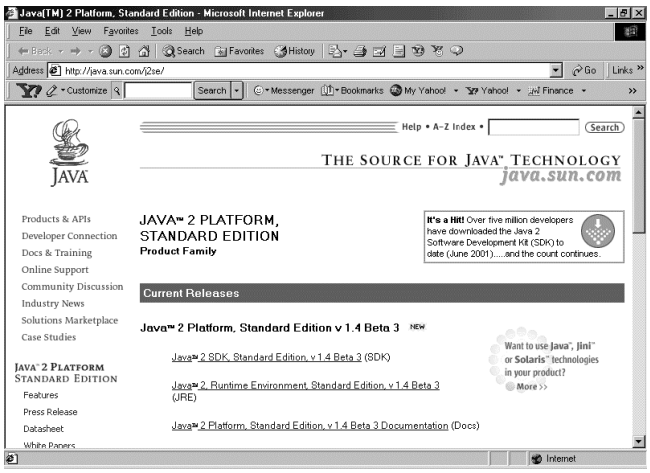
ในเวลาเดียวกันก็ทำให้ต้องใช้เวลาศึกษานาน โดยเฉพาะอย่างยิ่งคนที่ไม่ใช่นักเขียนโปรแกรมมืออาชีพ เพราะการเขียนโปรแกรมเชิงวัตถุ ไม่ใช่เรื่องที่จะเรียนรู้ได้ในเวลาอันรวดเร็ว

หนังสือเล่มนี้มีเนื้อหาที่เหมาะสำหรับคนที่เคยเขียนภาษาโครงสร้างมาบ้าง อาทิ ภาษาซี ภาษาปาสคาล ภาษาเบสิก ภาษาโคบอล หรือภาษาฟอร์แทน ถ้าหากคุณไม่เคยเขียนโปรแกรมคอมพิวเตอร์มาก่อนเลย ขอแนะนำให้อ่านหนังสือหัดเขียนโปรแกรมภาษาใดก็ได้ที่กล่าวมาข้างต้นเสียก่อน ถ้าจะให้ดีที่สุด ขอแนะนำให้อ่านภาษาซี เนื่องจากภาษาซีมีรูปแบบพื้นฐานคล้ายภาษาจาวา แต่คุณไม่จำเป็นต้องมีพื้นฐานการเขียนโปรแกรมเชิงวัตถุ เพราะหนังสือเล่มนี้จะอธิบายการเขียนโปรแกรมเชิงวัตถุตั้งแต่ระดับพื้นฐาน

จุดมุ่งหมายของหนังสือเล่มนี้ คือ ต้องการให้ผู้อ่านมีความเข้าใจการเขียนโปรแกรมเชิงวัตถุดีพอที่จะนำภาษาจาวาไปใช้ในเชิงประยุกต์ได้ ดังนั้นจึงไม่มีการกล่าวถึงทฤษฎีของการเขียนโปรแกรมเชิงวัตถุอย่างละเอียดลึกซึ้ง เพราะจะทำให้มีลักษณะเป็นวิชาการมากเกินไป ในขณะเดียวกันก็จะไม่มีการกล่าวถึงการนำภาษาจาวาไปใช้ในเชิงประยุกต์มากนัก เพราะไม่ต้องการให้เนื้อหาของหนังสือเยิ่นเย้อมากเกินไปจนเสียวัตถุประสงค์หลักของหนังสือ หนังสือเล่มนี้จึงเป็นจุดเริ่มต้นที่ดีสำหรับผู้ที่ต้องการใช้ภาษาจาวาในการทำงานจริงๆ แต่หนังสือเล่มนี้ยังไม่เพียงพอสำหรับการนำภาษาจาวาไปใช้งานจริง ขั้นตอนต่อไปคือ ผู้อ่านต้องศึกษาการนำจาวาไปใช้งานจากหนังสือเล่มอื่น ประโยชน์ของหนังสือเล่มนี้คือ ช่วยให้ผู้อ่านศึกษาหนังสือเหล่านั้นได้ง่ายขึ้น

จาวา 2

ภาษาจาวามีการพัฒนาอย่างต่อเนื่อง โดยที่ ซัน เป็นผู้กำหนดว่าโครงสร้างภาษาจาวา และคำสั่งต่างๆ ต้องมีหน้าตาเป็นอย่างไร และออกเป็นข้อกำหนดออกมา ภาษาจาวาตั้งแต่เวอร์ชัน 1.2 ขึ้นไป มีชื่อเรียกใหม่ว่า จาวา 2 ในหนังสือเล่มนี้เราครอบคลุมเนื้อหาของภาษาจาวาถึงเวอร์ชัน 1.2 แต่เพื่อความกระชับในการสื่อสาร เราจะยังคงใช้คำว่า จาวาเฉยๆ อยู่ คุณสามารถค้นหาข้อมูลใหม่ๆ เกี่ยวกับ จาวา โดยเฉพาะจาวาเวอร์ชันใหม่ล่าสุดได้จาก `http://java.sun.com/j2se` ซึ่งมีการเปลี่ยนแปลงอยู่อย่างสม่ำเสมอ



รูปที่ 1-3 http://java.sun.com/j2se

จาวาเป็นเทคโนโลยีเปิดซึ่งมี ซัน เป็นผู้กำหนดหน้าตาของคำสั่ง ดังนั้น จะมีบริษัทผลิตซอฟท์แวร์อื่นอีกจำนวนมากที่ออกผลิตภัณฑ์ที่ใช้เทคโนโลยีจาวาออกมา ตัวอย่างเช่น ไอบีเอ็ม ออราเคิล แมคโครมีเดีย บีอีเอ รวมทั้งซันเองด้วย ผลิตภัณฑ์จากบริษัทเหล่านี้ถูกพัฒนาขึ้นตามข้อกำหนดของภาษาจาวาอย่างเคร่งครัด ดังนั้น โปรแกรมภาษาจาวา ของคุณจึงสามารถทำงานบนผลิตภัณฑ์ของบริษัทใดเหล่านี้ก็ได้โดยไม่ต้องมีการดัดแปลง ในอนาคตถ้าทุกบริษัทหันมาใช้เทคโนโลยีจาวา สิ่งที่เกิดขึ้นก็คือ ซอฟท์แวร์ จะมีความยุ่งยากในการเชื่อมต่อหรือทำงานประสานกันน้อยลง และในขณะเดียวกันบริษัทเหล่านี้ก็จะหันมาแข่งขันกันในแง่ของการทำให้ผลิตภัณฑ์ของบริษัทตนทำงานได้เร็วกว่า แทนที่จะแข่งกันเป็นผู้ผูกขาดเทคโนโลยีอย่างแต่ก่อน

2

จาวาเวอร์ชัวน์แมทชีน

และ จาวาคอมไพล์เลอร์

โปรแกรมที่เขียนด้วยภาษาจาวารันบน **จาวาเวอร์ชัวน์แมทชีน** ดังนั้นคุณต้องมีจาวาเวอร์ชัวน์แมทชีนบนเครื่องคอมพิวเตอร์ของคุณถ้าคุณต้องการรันโปรแกรมที่เขียนด้วยภาษาจาวา

โปรแกรมที่เขียนด้วยภาษาจาวาก่อนจะนำไปรันได้ต้องผ่านการคอมไพล์ด้วย **จาวาคอมไพล์เลอร์** ก่อน เนื่องจากคุณต้องการเป็นนักเขียนโปรแกรมภาษาจาวา คุณจึงต้องมีทั้งจาวาคอมไพเลอร์ และ จาวาเวอร์ชัวน์แมทชีน เพื่อเขียนและลองรันโปรแกรมภาษาจาวา ในบทนี้คุณจะได้เตรียมสิ่งที่จำเป็นทั้งสองอย่างนี้

เนื่องจากจาวาเป็นเทคโนโลยีเปิด ดังนั้นคุณอาจจะเลือกใช้จาวาเวอร์ชัวน์แมทชีนและจาวาคอมไพล์เลอร์ ของบริษัทใดก็ได้ ในหนังสือเล่มนี้เราจะเลือกใช้จาวาเวอร์ชัวน์แมทชีนและ

จาวาคอมไพล์เลอร์ที่ดาวน์โหลดได้จากเวบไซต์จาวาของซัน เนื่องจากเป็นที่รู้จักดีและที่สำคัญคือฟรี

อีกสิ่งที่คุณต้องมีก็คือคอมพิวเตอร์ส่วนบุคคล ซึ่งจะเป็นระบบปฏิบัติการอะไรก็ได้เพราะจาวารันได้ทุกระบบปฏิบัติการ ในหนังสือเล่มนี้เราเลือกใช้ระบบปฏิบัติการไมโครซอฟท์วินโดว์ (95/98/Me/2000/XP)เพราะเป็นระบบปฏิบัติการที่หาได้ง่ายที่สุด สำหรับผู้ที่ไม่นิยมระบบปฏิบัติการของไมโครซอฟท์อาจจะต้องลองติดตั้งจาวาเวอร์ชัวน์แมทชีนและจาวาคอมไพล์เลอร์สำหรับระบบปฏิบัติการที่คุณใช้อยู่ด้วยตัวเอง หลังจากติดตั้งเสร็จแล้วการใช้งานปกติแทบจะไม่มีความแตกต่างกัน

จาวาเวอรชัวน์แมทชีนบนคอมพิวเตอร์ส่วนบุคคลของบริษัทซัน มีชื่อว่า **JRE** หรือ จาวา รันไทม์ เอ็นไวรอนเมนท์ ส่วนจาวาคอมไพล์เลอร์ของซันมีชื่อว่า **SDK** ทั้งสองสามารถดาวน์โหลดได้จากเวบไซต์ http://java.sun.com/j2se

คุณไม่จำเป็นต้องดาวน์โหลด JRE เพราะ SDK จะรวม JRE มาให้ด้วยในตัว

ติดตั้ง SDK

ให้คุณต่อไปยังอินเตอร์เนตเพื่อเข้าเวบไซต์เจทูเอสอีของซัน http://java.sun.com/j2se เพื่อดาวน์โหลดซอฟท์แวร์ที่มีชื่อว่า Java 2 SDK ซึ่งมีการเปลี่ยนเวอร์ชันใหม่อยู่อย่างสม่ำเสมอ คุณสามารถใช้เวอร์ชันอะไรก็ได้ที่สูงกว่าเวอร์ชัน 1.2 แต่ขอแนะนำให้ใช้เวอร์ชันที่ใหม่ที่สุดที่ไม่ใช่เวอร์ชันทดสอบ

เลือกระบบปฏิบัติการวินโดว์ คุณต้องเลือกระบบปฏิบัติการให้ถูกต้องเวลาดาวน์โหลด เพราะแม้ว่าโปรแกรมภาษาจาวาจะไม่ขึ้นกับระบบปฏิบัติการ แต่ตัวจาวาคอมไพล์เลอร์และจาวาเวอร์ชัวน์แมทชีนขึ้นกับระบบปฏิบัติการ

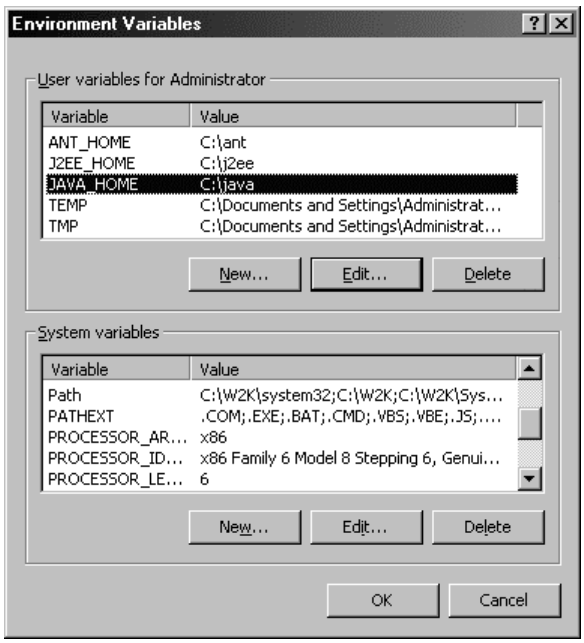
เมื่อดาวน์โหลดเสร็จแล้วก็ให้ทำการติดตั้งได้ด้วยการดับเบิลคลิกที่ไอคอนของไฟล์ที่ดาวน์โหลดมา โปรแกรมจะเข้าสู่การติดตั้ง SDK ซึ่งเป็นวิซซาร์ดเหมือนกันการติดตั้งโปรแกรมบนวินโดว์ทั่วไปซึ่งคุณควรติดตั้งได้ด้วยตนเอง

ในขั้นตอนของการเลือกโฟลเดอร์ที่จะติดตั้งโปรแกรม ขอแนะนำให้คุณติดตั้งลงบนโฟล์เดอร์ชื่อว่า C:\java เพราะในหนังสือเล่มนี้เราจะสมมติว่าคุณติดตั้งมันลงในโฟลเดอร์ C:\java แต่ในความเป็นจริงคุณจะใช้ชื่อที่มีมาให้หรือตั้งชื่ออื่นๆ ก็ได้

รูปที่ 2-1 ติดตั้ง SDK บนระบบปฏิบัติการวินโดว์

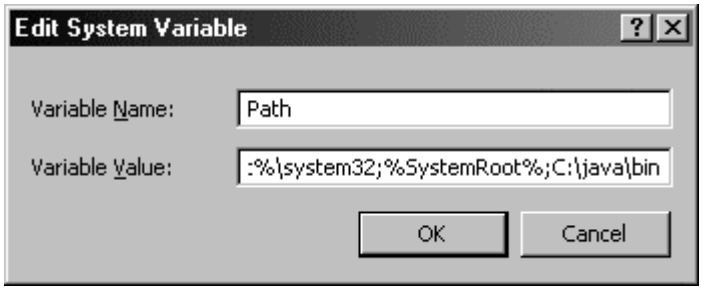
เมื่อติดตั้งเสร็จแล้วคุณจะได้ทุกสิ่งทุกอย่างของ SDK จะอยู่ในโฟล์เดอร์ C:\java ทั้งหมด รวมทั้ง JRE ที่พ่วงมาด้วย

ขั้นตอนการติดตั้งยังไม่เสร็จสมบูรณ์ ก่อนจะใช้งานได้คุณต้องระบุ PATH ให้กับระบบปฏิบัติการของคุณเสียก่อน ในระบบปฏิบัติการวินโดว์ทำได้โดยการคลิกเมาส์ขวาที่ไอคอน My Computer บนเดสก์ท็อป แล้วเลือก Properties จากนั้นเลือกแท็ปที่ให้เซตค่าตัวแปรระบบได้ ซึ่งวินโดว์แต่ละชนิดจะมีหน้าจอในส่วนนี้ไม่เหมือนกัน ตัวอย่างในภาพเป็นการเซตตัวแปรระบบบนวินโดว์ 2000



รูปที่ 2-2 เซตตัวแปรระบบบนวินโดว์ 2000

ตัวแปรระบบที่ชื่อ **Path** จะมีค่าเดิมของมันอยู่ให้คุณคงค่าเดิมไว้แต่เติมต่อท้ายค่าเดิมด้วย ;C:\java\bin เช่นตัวอย่างในภาพเป็นการเซตค่าตัวแปรระบบ Path ใหม่บนวินโดว์ 2000 สังเกต ;C:\java\bin ที่ท้ายสุดของช่อง Variable Value



รูปที่ 2-3 การเติม ;C:\java\bin ต่อท้าย ตัวแปรระบบ Path

เท่านี้การติดตั้ง SDK ของคุณก็เป็นอันเสร็จเรียบร้อย

เคล็ดลับ ตัวแปรระบบ path ใช้ระบุโฟล์เดอร์ที่เป็นที่อยู่ของคำสั่งต่างๆ ในดอส ดังนั้นการใส่ C:\java\bin ไว้เป็นส่วนหนึ่งของตัวแปรระบบ path เป็นการบอกให้ดอสมองหาคำสั่งจากโฟลเดอร์นี้ด้วย ซึ่งเป็นที่อยู่ของจาวาคอมไพล์เลอร์ จาวาเวอร์ชัวน์แมทชีน และโปรแกรมสนับสนุนอื่นๆ ของ SDK

คุณสามารถเซตค่าของตัวแปรระบบ path ด้วยการสั่งผ่านดอสโดยตรงก็ได้ โดยเริ่มจาก

```
C:\> path
PATH=C:\WINNT\system32;C:\WINNT;C:\
```

คำสั่งนี้จะแสดงค่าปัจจุบันของตัวแปร path ออกมาซึ่งประกอบด้วยชื่อโฟล์เดอร์จำนวนมาก คั่นด้วยเครื่องหมาย ; ชื่อโฟล์เดอร์ที่คุณได้อาจแตกต่างกันไปแล้วแต่เครื่อง

เซตค่าของตัวแปร path ใหม่ให้มี C:\java\bin อยู่ด้วย ด้วยการสั่งคำสั่งต่อไปนี้

```
C:\> set path=C:\WINNT\system32;C:\WINNT;C:\;C:\java\bin
```

สิ่งที่อยู่หลังเครื่องหมายเท่ากับและอยู่หน้า ;C:\java\bin คือค่าเก่าทั้งหมดของตัวแปร path ซึ่งได้จากคำสั่งที่แล้ว เสร็จแล้วตรวจสอบค่าใหม่ของตัวแปร path ด้วยคำสั่ง

```
C:\> path
PATH=C:\WINNT\system32;C:\WINNT;C:\;C:\java\bin
```

ดอสจะแสดงค่าใหม่ของตัวแปร path ออกมา ซึ่งคุณควรจะเห็น C:\java\bin เป็นชื่อสุดท้าย

แต่ข้อเสียของการเซตด้วยคำสั่งดอสโดยตรงก็คือ (ณต้องสั่งใหม่ทกครั้งที่

เปิดหน้าจอดอสใหม่ คุณสามารถกำหนดให้ดอสรันคำสั่งนี้เองโดยอัตโนมัติทุกครั้งที่เปิดหน้าจอดอสได้ด้วยการใส่คำสั่งนี้ลงไปในไฟล์ autoexec.bat แต่เนื่องจากไฟล์ autoexec.bat บนวินโดว์แต่ละเวอร์ชันมีวิธีการใช้งานที่แตกต่างกัน จึงไม่ขอลงรายละเอียด

จาวาโปรแกรมแรกของคุณ

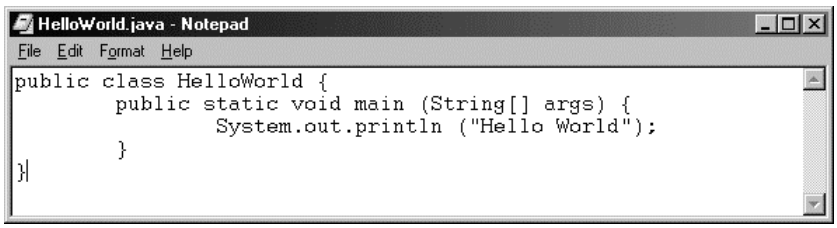
เพื่อทดสอบว่าการติดตั้งเสร็จสมบูรณ์หรือไม่ เราจะทดลองเขียนโปรแกรมภาษาจาวาสั้นๆ โปรแกรมหนึ่ง แล้วทดลองคอมไพล์และรันบนจาวาเวอร์ชัวน์แมทชีน โปรแกรมที่ว่านี้มีชื่อว่า HelloWorld ซึ่งไม่ทำอะไรมากนอกจาก แสดงข้อความว่า Hello World ออกที่หน้าจอดอส

เรียกโปรแกรม Notepad แล้วพิมพ์ข้อความนี้ลงไปโดยไม่ต้องสนใจว่าเนื้อหา (เราจะยังไม่เรียนเรื่องโครงสร้างภาษาจาวาในบทนี้) สำหรับคนที่ใช้ระบบปฏิบัติการอื่นให้เรียกโปรแกรมสำหรับสร้างเท็กซ์ไฟล์บนระบบปฏิบัติการนั้นๆ ขึ้นมา ตัวอย่างเช่น vi บนยูนิกซ์

โปรแกรมที่ 2 – 1 : HelloWorld.java

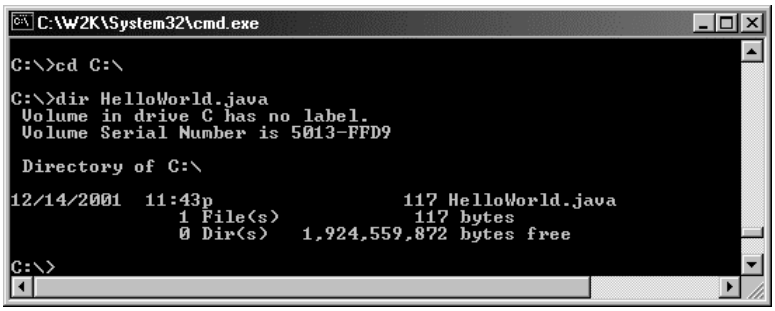
```
public class HelloWorld {
  public static void main ( String []  args ) {
    System.out.println ( “Hello World” ) ;
  }
}
```

พยายามพิมพ์ให้เหมือนที่สุด โดยเฉพาะอย่างยิ่งตัวอักษรพิมพ์เล็กพิมพ์ใหญ่ และการเว้นวรรคตอน เวลาย่อหน้าคุณอาจใช้แท็ปก็ได้ จากนั้นทำการบันทึกไฟล์นี้ลงบนที่ไหนก็ได้ในฮาร์ดดิสก์ ขอแนะนำให้บันทึกลงบน C:\ และ ตั้งชื่อไฟล์ว่า HelloWorld.java ระวังเรื่องตัวพิมพ์เล็กพิมพ์ใหญ่ของชื่อไฟล์



รูปที่ 2-4 เขียนโปรแกรม HelloWorld.java ด้วย Notepad

ไฟล์ไฟล์นี้ก็คือซอร์สโค้ดภาษาจาวาที่เคยกล่าวถึงในบทที่แล้วนั้นเอง สิ่งที่พิมพ์ลงไปในไฟล์ก็คือคำสั่งภาษาจาวา ซึ่งในตอนนี้เรายังไม่สนใจว่าคำสั่งเหล่านี้มีความหมายว่าอย่างไรบ้าง ไฟล์ซอร์สโค้ดภาษาจาวาจะต้องมีนามสกุล `.java` เสมอ เมื่อเราได้โปรแกรมภาษาจาวาที่ชื่อ `HelloWorld.java` แล้ว เราจะลองทำการคอมไพล์ ด้วยการเรียกหน้าจอดอสออกมา แล้วไปที่ `C:\` จากนั้นทดลองตรวจสอบดูว่ามีไฟล์ชื่อ `HelloWorld.java` อยู่หรือไม่ด้วยการพิมพ์คำสั่งดังในภาพ



รูปที่ 2-5 ตรวจสอบไฟล์ HelloWorld.java

เราเรียกคอมไพล์เลอร์ใน SDK ขึ้นมาใช้งานด้วยการใช้คำสั่ง `javac` คำสั่งนี้จะถูกเรียกจากที่ไหนก็ได้เพราะเราได้บอกระบบปฏิบัติการไปแล้วว่าให้คอมไพล์เลอร์อยู่ที่ไหนด้วยการเซตตัวแปร path ของระบบ ที่สำคัญคือที่ที่เรียกคำสั่ง `javac` ต้องมีซอร์สโค้ดของเราอยู่ ซึ่งในกรณีนี้ซอร์สโค้ดอยู่ที่ `C:\` เราจึงเรียกคำสั่ง `javac` จาก `C:\` คำสั่ง `javac` จะตามด้วยชื่อซอร์สโค้ด ดังนี้

```
C:\> javac HelloWorld.java
```

ถ้าดอสไม่ฟ้องความผิดพลาดใดๆ ออกมา แสดงว่าคุณติดตั้ง SDK ได้อย่างถูกต้อง แต่ถ้ามีข้อความเกิดขึ้นแสดงว่าเกิดความผิดพลาดขึ้น ซึ่งสาเหตุอาจเกิดจากการติดตั้ง SDK ที่ยังไม่ถูกต้อง หรืออาจเกิดคำสั่งภาษาจาวาที่คุณพิมพ์ลงไปไม่ถูกต้อง ขอให้ย้อนกับไปตรวจสอบทั้งสองกรณีและแก้ไขจนกว่าดอสจะไม่ฟ้องความผิดพลาดใดๆ ออกมา

ถ้าทุกอย่างเรียบร้อยให้ตรวจสอบดูจะพบว่าเกิดไฟล์ใหม่ขึ้นชื่อ HelloWorld.class ไฟล์นี้คือโปรแกรมภาษาจาวาที่คอมไพล์เสร็จแล้ว และพร้อมที่จะใช้งานได้ โปรแกรมภาษาจาวาที่คอมไพล์แล้วจะมีนามสกุล .class เสมอ และเป็นไบนารีไฟล์คือถ้าคุณลองเปิดไฟล์เหล่านี้ดูด้วย Notepad คุณจะพบว่าเนื้อความข้างในไม่ใช่ภาษามนุษย์ อ่านไม่รู้เรื่อง มันเป็นภาษาที่เราเรียกว่า จาวาไบต์โค้ด ซึ่งจาวาเวอร์ชัวน์แมทชีนเท่านั้นที่เข้าใจ

เคล็ดลับ ถ้าต้องการใช้โปรแกรมช่วยเขียนโปรแกรม เช่น มีการนับบรรทัดของซอร์สโค้ดให้ มีการใช้สีของตัวอักษรในการแยกความแตกต่างระหว่างคำสั่งแต่ละประเภท หรือมีระบบตรวจสอบความผิดพลาดของโปรแกรมแบบทีละบรรทัด คุณอาจหันไปใช้โปรแกรมประเภท IDE ตัวอย่างเช่น Borland JBuilder, WebGain VisualCafe, Oracle JDeveloper, MetroWerks Code Warriors, Forte for Java ฯลฯ แทนการใช้ Notepad

คำสั่ง sourcepath

ถ้าคุณต้องการเรียกคอมไพล์เลอร์จากโฟล์เดอร์อื่นที่ไม่ใช่โฟล์เดอร์ที่มีไฟล์ .java ของคุณอยู่ คุณสามารถบอกตำแหน่งที่เก็บไฟล์ .java ของคุณได้ด้วยคำสั่ง **sourcepath** ตัวอย่างเช่น เรารู้ว่าตอนนี้ไฟล์ HelloWorld.java อยู่ที่ C: ให้เราลองเรียกคำสั่ง javac จากที่อื่น เช่น C:\java ดังนี้

```
C:\> cd java
C:\java> javac HelloWorld.java
```

คุณจะพบว่าไม่สามารถคอมไพล์ได้ เพราะคอมไพล์เลอร์จะหาตัวโปรแกรมไม่เจอ คราวนี้ลองสั่งใหม่ด้วยคำสั่งดังต่อไปนี้

```
C:\java> javac –sourcepath C:\ HelloWorld.java
```

คราวนี้จะพบว่าสามารถคอมไพล์ได้ตามปกติ ถ้าลองตรวจสอบดูจะพบไฟล์ HelloWorld.class ในโฟลเดอร์ C:\java คำสั่ง sourcepath อยู่ต่อท้ายคำสั่ง javac และมีเครื่องหมาย – นำหน้า และตามด้วยชื่อโฟลเดอร์ที่เป็นที่อยู่ของไฟล์ .java ส่วนชื่อไฟล์อยู่ท้ายสุดเสมอ

แม้ว่าเราจะสามารถรันคำสั่ง javac จากที่ไหนก็ได้ เพราะเราสามารถใช้คำสั่ง sourcepath ในการระบุตำแหน่งของไฟล์ .java แต่เพื่อความสะดวกในการอ้างอิงของคุณให้เก็บไฟล์ .java ใดๆ ก็ตามที่คุณเขียนขึ้นไว้ในโฟล์เดอร์ C:\java และคอมไพล์ในโฟลเดอร์นี้ตลอดเนื้อหาในหนังสือเล่มนี้

และขอแนะนำให้คุณลองพิมพ์โปรแกรมตัวอย่างทุกโปรแกรมในหนังสือด้วยตัวเอง บันทึก คอมไพล์ และรัน ดูว่าใช้งานได้จริงหรือไม่ทุกโปรแกรม การพิมพ์โปรแกรมตัวอย่างเองทีละบรรทัดอาจเสียเวลาแต่จะทำให้เกิดความคุ้นเคยและเข้าใจอย่างถ่องแท้

ตอนนี้ให้คุณย้ายไฟล์ HelloWorld.java ของคุณเข้ามาในโฟล์เดอร์ C:\java ดังนี้

```
C:\java> copy C:\HelloWorld.java C:\java
C:\java> del C:\HelloWorld.java
C:\java> del C:\HelloWorld.class
```

ลองรันโปรแกรม HelloWorld

เมื่อเราได้โปรแกรมภาษาจาวาที่คอมไพล์แล้ว ซึ่งก็คือไฟล์ HelloWorld.class เวลาจะรันต้องรันด้วยจาวาเวอร์ชัวน์แมทชีน เราเรียกจาวาเวอร์ชัวน์แมทชีนได้ด้วยคำสั่ง java ตามด้วยชื่อของโปรแกรมภาษาจาวาที่ต้องการจะรัน ดังนี้

```
C:\java> java HelloWorld
```

โปรดสังเกตว่าเวลาเรียกคำสั่ง จาวา ชื่อของโปรแกรมที่ตามมาจะไม่มีนามสกุล .class ต่อท้าย ทั้งๆ ที่โปรแกรมภาษาจาวาคือไฟล์นามสกุล .class ต่างกับกรณีของการเรียกคำสั่ง javac ที่ต้องมีนามสกุล .java ต่อท้ายชื่อไฟล์โปรแกรมเสมอ

ผลลัพธ์ที่ได้จากการรันโปรแกรม HelloWorld.class ก็คือ โปรแกรมจะพิมพ์คำว่า Hello World ออกมาที่หน้าจอดังภาพ

```
C:\java> java HelloWorld
Hello World

C:\java>
```

คำสั่ง java ต้องเรียกในโฟล์เดอร์เดียวกันกับโฟลเดอร์ที่มีไฟล์นามสกุล .class ที่เราจะรันอยู่

ถ้าทุกอย่างเรียบร้อย แสดงว่าการติดตั้ง SDK สมบูรณ์แบบ และคุณก็ได้เขียนและรันโปรแกรมจาวาโปรแกรมแรกในชีวิตของคุณเสร็จเรียบร้อยแล้ว

ถ้าคุณมีเครื่องคอมพิวเตอร์ที่รันระบบปฏิบัติการอื่น คุณสามารถนำไฟล์ HelloWorld.class ของคุณไปทดลองรันบนระบบปฏิบัติการนั้นๆ ได้ ขอให้มีจาวาเวอร์ชัวน์แมทชีนอยู่ โปรแกรมภาษาจาวาสามารถรันได้ทุกระบบปฏิบัติการโดยไม่ต้องมีการแก้ไขอะไรเลย

3

โปรแกรม Hello World

ในบทนี้เรามาวิเคราะห์สิ่งที่อยู่ภายในไฟล์ HelloWorld.java กัน

อย่างที่เคยบอกไปแล้ว จาวา เป็นภาษาเชิงวัตถุ ซอร์สโค้ดของโปรแกรมภาษาเชิงวัตถุประกอบด้วยนิยามของวัตถุหลายๆ วัตถุเขียนเรียงต่อกันไปเรื่อยๆ นิยามของวัตถุในภาษาจาวาเราเรียกว่า **คลาส**

โปรแกรม HelloWorld มีเนื้อหาดังต่อไปนี้

โปรแกรม 3 - 1 : HelloWorld.java

```
public class HelloWorld {
  public static void main (String []  args) {
    System.out.println ("Hello World");
  }
}
```

โปรแกรมนี้มีคลาสแค่หนึ่งคลาส เพราะเป็นโปรแกรมง่ายๆ สังเกตคำสั่ง class ในบรรทัดแรกสุด ชื่อของคลาสคลาสนี้คือคำสั่งที่อยู่ตามมาได้แก่คำว่า HelloWorld นั้นคือคลาสนี้มีชื่อเหมือนกับชื่อโปรแกรมและชื่อไฟล์ด้วย ตัวพิมพ์เล็กพิมพ์ใหญ่มีความสำคัญมากในภาษาจาวา ชื่อ HELLOWORLD ,Helloworld, helloworld และ HelloWorld ถือว่าไม่เหมือนกัน

โปรแกรมที่มีคลาสแค่คลาสเดียว คลาสนั้นต้องมีชื่อเหมือนกับชื่อไฟล์เสมอ และต้องมีคำว่า public นำหน้าคำสั่ง class ตอนนี้ยังไม่ต้องสนใจว่าทำไมต้องมีคำว่า public

ชื่อคลาสต้องเป็นคำๆ เดียว ดังนั้นเราจึงตั้งชื่อคลาสว่า HelloWorld แทนที่จะเป็น Hello World

ถัดไปจากคำว่า HelloWorld คือ เครื่องหมายวงเล็บปีกกาเปิด ถ้าคำว่า public class HelloWorld ซึ่งอยู่หน้าเครื่องหมายวงเล็บปีกกาเปิดเรียกว่า **ส่วนหัวของคลาส** HelloWorld สิ่งที่อยู่ระหว่างวงเล็บปีกกาเปิดอันแรกสุด กับวงเล็บปีกกาเปิดอันสุดท้ายของไฟล์คือ **ส่วนตัวของคลาส** HelloWorld

เนื้อความของคลาส HelloWorld ประกอบด้วยแมธธอสหนึ่งแมธธอส ชื่อว่า main คำว่า **แมธธอส** หมายถึงสิ่งที่บรรจุคำสั่งที่บอกให้โปรแกรมทำอะไรต่อมิอะไร สังเกตคำว่า main ในบรรทัดที่สอง นั้นคือตำแหน่งของชื่อแมธธอส โปรแกรมทุกโปรแกรมในภาษาจาวาต้องมีแมธธอสหนึ่งแมธธอสที่มีชื่อว่า main เสมอ

ยังไม่ต้องสนใจคำสั่ง public static void ที่มาก่อนคำว่า main และคำสั่ง (String [] args) ที่ตามหลังมา ขอให้เข้าใจว่าทั้งหมดคือ **ส่วนหัวของแมธธอส** ขอให้สังเกตวงเล็บก้ามปูเปิดที่อยู่ท้ายบรรทัดที่สอง ข้อความทั้งหมดที่อยู่ระหว่างวงเล็บก้ามปูเปิดนี้กับวงเล็บก้ามปูปิดในบรรทัดรองสุดท้ายคือ **ส่วนตัวของแมธธอส**

จะเห็นได้ว่าทั้งคลาสและแมธธอสมีโครงสร้างคล้ายกันและซ้อนกันอยู่ คือประกอบด้วยส่วนหัวซึ่งมีชื่อของคลาสหรือแมธธอสอยู่ และส่วนเนื้อความซึ่งอยู่ในวงเล็บก้ามปูที่ตามมา แต่เรานิยมวางวงเล็มก้ามปูเปิดไว้ท้ายส่วนหัวของคลาสหรือแมธธอส และวางวงเล็บก้ามปูปิดไว้โดดเดี่ยวในบรรทัดสุดหลังจบเนื้อความโดยย่อหน้าให้ตรงกับส่วนหัว ส่วนเนื้อความจะอยู่ในบรรทัดระหว่างวงเล็บก้ามปูเปิดและวงเล็บก้ามปูปิด โดยย่อหน้าเข้าไปหนึ่งระดับให้ลึกกว่าส่วนหัวทุกบรรทัด ทั้งหมดนี้เป็นเพียงความนิยมในการเขียนโปรแกรมให้ดูง่ายสะอาดตาเท่านั้น การเว้นบรรทัดในภาษาจาวาไม่มีความหมายใดๆ ทั้งสิ้น โปรแกรม HelloWorld อาจเขียนติดๆ กันเป็นแบบนี้ก็ได้

โปรแกรม 3 - 2 : HelloWorld.java

```
public class HelloWorld {public static void main (String[] args)
{System.out.println ("Hello World"); } }
```

แต่ไม่นิยมเพราะอ่านยาก

เคล็ดลับ สำหรับการเว้นบรรทัดไม่มีกฎแน่นอนตายตัวว่าคุณควรเว้นบรรทัดเมื่อใด ทั้งหมดขึ้นอยู่กับวิจารณญาณของคุณเอง การเว้นบรรทัดมีประโยชน์เพียงเพื่อให้โปรแกรมของคุณอ่านง่าย
การย่อหน้าคุณอาจใช้เครื่องหมายแท็บก็ได้ แต่วิธีการที่ดีกว่าคือการเว้นวรรคธรรมดาด้วยการเคาะ Space สองครั้ง เพราะเครื่องหมายแท็บอาจมีปัญหาเวลาใช้งานข้ามระบบปฏิบัติการ

ระหว่างคำทุกคำในโปรแกรมภาษาจาวาคุณจะเว้นวรรคด้วยการเคาะ Space กี่ครั้งก็ได้ตั้งแต่หนึ่งครั้งเป็นต้นไป ยกเว้นเมื่อระหว่างคำมีเครื่องหมายคั่นกลางอยู่ เช่น . ( ) { } [ ] ; เราอาจไม่ต้องเว้นวรรคเลยก็ได้ เพราะจาวาจะถือว่าคำที่อยู่หน้าเครื่องหมายเป็นคนละคำกับคำที่อยู่หลังเครื่องหมายโดยอัตโนมัติ เช่น

```
   public static void main() (String args[]) {
public static void main ( ) ( String args [ ] ) {
   public static void main( ){String args[ ]){
    public static void main()(String args[]){
```

ทุกบรรทัดข้างต้นถือว่าไม่มีความแตกต่าง

แมธธอสคือส่วนที่บอกให้โปรแกรมทำอะไรต่อมิอะไร ดังนั้นถ้าเราต้องการให้โปรแกรมของเราทำอะไร เราก็เพียงแต่นำคำสั่งเหล่านั้นมาใส่ไว้ในเนื้อความของแมธธอส ดังนั้นคุณก็คงเดาออกว่าคำสั่ง

```
System.out.println ("Hello World");
```

คือคำสั่งให้โปรแกรมพิมพ์คำว่า Hello World ออกหน้าจอนั้นเอง

และถ้าคุณใส่คำสั่งนี้เข้าไปในเนื้อความสองคำสั่งติดต่อกัน คุณคงเดาได้ว่าจะเกิดอะไรขึ้น

โปรแกรม 3 - 3 : HelloWorld.java

```
public class HelloWorld {
  public static void main (String []  args) {
```

```
        System.out.println (“Hello World”);
        System.out.println (“Hello World”);
    }
}
```

ลองสร้างโปรแกรมนี้ด้วย Notepad ดู ลองเรียก Notepad จากดอส ดังนี้

```
C:\java> notepad HelloWorld.java
```

คำสั่งนี้เรียก Notepad ออกมาเพื่อสร้างเท็กซ์ไฟล์ชื่อ HelloWorld.java คุณควรพบเนื้อหาของโปรแกรม HelloWorld อันเก่าค้างอยู่ในไฟล์ แก้ไขเพิ่มเติมด้วยการต่อเติมให้เหมือนโปรแกรม 3-3 แล้วบันทึกลงดิสก์ จากนั้นลองคอมไพล์โปรแกรมดู ดังนี้

```
C:\java> javac HelloWorld.java
```

รันโปรแกรมดูโดยการเรียกจาวาเวอร์ชัวน์แมทชีนจะได้ผลดังนี้

```
C:\java> java HelloWorld
Hello World
Hello World
```

ถ้าต้องการให้โปรแกรมเขียนคำว่า Hello World สิบครั้ง ก็ใส่สิบคำสั่ง ประเด็นอยู่ว่าคำสั่งที่เราต้องการให้โปรแกรมทำงานจะอยู่ในส่วนตัวของแมธธอส main() ระหว่างวงเล็บก้ามปู โดยเขียนเรียงต่อไปเรื่อยๆ คำสั่งในภาษาจาวาทุกคำสั่งจะจบด้วยเครื่องหมาย ; เพื่อเป็นการบอกว่าสิ้นสุดคำสั่งแล้ว จาวาคอมไพล์เลอร์จะใช้เครื่องหมายนี้ในการแบ่งแยกคำสั่งที่มาก่อนกับคำสั่งที่อยู่ถัดไป แทนที่จะดูจากการเว้นบรรทัด แต่อย่างไรก็ดีเรานิยมเว้นบรรทัดระหว่างคำสั่งด้วย เพื่อให้โปรแกรมของเราอ่านง่าย

ลองดูคำสั่งที่ให้เขียนออกหน้าจออีกครั้ง สังเกตให้ดีตรงคำว่า Hello World

```
System.out.println (“Hello World”);
```

คุณคงเดาออกว่า ถ้าเราต้องการให้โปรแกรมเขียนข้อความอะไร ก็ให้ใส่ข้อความนั้นลงไปแทนที่คำว่า Hello World ในตำแหน่งเดียวกัน ซึ่งเป็นการเดาที่ถูกต้อง ขอให้จำไว้ว่าเราสามารถสั่งให้โปรแกรมของเราเขียนคำว่าอะไรก็ได้ออกหน้าจอด้วยการใช้คำสั่งข้างต้น

โปรแกรม Hello World เป็นโปรแกรมง่ายๆ สั้นๆ แต่เมื่อคุณต้องเขียนโปรแกรมที่ใหญ่และซับซ้อน โปรแกรมของคุณอาจมีเป็นสิบๆ คลาส ร้อยๆ แมธธอส พันๆ บรรทัด ซึ่งยากต่อการอ่าน โดยเฉพาะอย่างยิ่งเมื่อเวลาที่คอมไพล์ไม่ผ่านและต้องการหาที่ผิด ดังนั้น บางทีคุณอาจต้องการเขียนโน้ตอธิบายส่วนต่างๆ ของโปรแกรมเล็กๆ น้อยๆ ตามตำแหน่งต่างๆ ในโปรแกรมภาษาจาวาของคุณ เพราะให้เวลาที่คุณกลับมาดูโปรแกรมของคุณอีกครั้ง คุณจะเข้าใจได้รวดเร็วขึ้น การเขียนข้อความใดๆ นอกเหนือจากคำสั่งภาษาจาวาสามารถทำได้สองวิธี

วิธีแรกคือการใช้เครื่องหมาย /* และ */ คั่นระหว่างข้อความของคุณ เมื่อคอมไพล์เลอร์เจอเครื่องหมายนี้ มันจะข้ามข้อความที่อยู่ภายในเครื่องหมายนี้ทั้งหมดไปเลย ดังนั้นคุณสามารถเขียนอะไรลงไปก็ได้โดยไม่ทำให้โปรแกรมของคุณผิดเพี้ยน ตัวอย่างข้างล่างนี้เป็นการระบุชื่อผู้เขียนและวันที่ที่เขียนโปรแกรม

โปรแกรม 3 - 4 : HelloWorld.java

```
/* HelloWorld
       Written by Narin
       Since December 16, 2001
       All rights reserved
*/

public class HelloWorld{
  public static void main(String[]args){
    System.out.println("Hello World");
  }
}

/* End of program*/
```

สังเกตว่าข้อความที่อยู่ข้างในเครื่องหมาย /* */ จะยาวกี่บรรทัดก็ได้ และจะอยู่ที่ไหนก็ได้ในโปรแกรมของคุณ

อีกวิธีหนึ่งเป็นการเขียนข้อความในกรณีที่เป็นข้อความสั้นๆ จบในบรรทัดเดียว และอยู่ท้ายสุดของบรรทัด คุณสามารถใช้เครื่องหมาย // ในการบอกคอมไพล์เลอร์ให้ข้ามสิ่งที่อยู่ต่อท้ายเครื่องหมายไปเลยจนจบบรรทัด กรณีนี้คุณไม่ต้องปิดข้อความด้วยเครื่องหมายใดๆ เพราะคอมไพล์เลอร์จะดูจากการขึ้นบรรทัดใหม่ของคุณเป็นการบอกว่าข้อความส่วนตัวของ

คุณสิ้นสุดแล้ว ตัวอย่างข้างล่างเป็นการใช้ข้อความส่วนตัวในการบันทึกว่าคำสั่งในบรรทัดนั้นคือคำสั่งที่บอกให้โปรแกรมเขียนคำว่า HelloWorld ออกหน้าจอ

โปรแกรม 3 - 5 : HelloWorld.java

```
public class HelloWorld{
  public static void main(String[]args){
    System.out.println("Hello World"); // Echo Hello World
  }
}
```

ในหนังสือเล่มนี้บรรทัดบางบรรทัดในโปรแกรมอาจจะยาวจนล้นบรรทัด แต่ขอให้เข้าใจว่าเป็นคำสั่งในบรรทัดเดียวกัน ตัวอย่างเช่นในโปรแกรมข้างต้น คำว่า World ตกลงไปอยู่ในบรรทัดใหม่เองเนื่องจากความยาวของหน้ากระดาษไม่พอ เวลาลองรันโปรแกรมนี้จึงไม่ควรเคาะ Enter เพื่อให้คำว่า World ขึ้นบรรทัดใหม่

4

ค่าคงตัว และ ตัวแปรพื้นฐาน

ประโยชน์อย่างแรกสุดที่มนุษย์รู้จักใช้เกี่ยวกับคอมพิวเตอร์ก็คือการใช้คอมพิวเตอร์คำนวณแทนเรา คอมพิวเตอร์มีหน่วยความจำหรือที่เราเรียกกันติดปากว่า แรม เอาไว้เก็บตัวเลขหรือข้อมูล แล้วก็มีส่วนของหน่วยประมวลผลหรือที่เราเรียกกันติดปากว่า ซีพียู ซึ่งสามารถบวกลบคูณหารข้อมูลที่อยู่ในแรมได้ โปรแกรมคอมพิวเตอร์คือคำสั่งที่บอกให้ซีพียูบวกลบคูณหารตามแบบที่เราต้องการ ดังนั้นในบทนี้เราจะเริ่มต้นจากการเรียนรู้วิธีสั่งให้คอมพิวเตอร์เอาตัวเลขหรือข้อมูลที่เราต้องการจะคำนวณไปเก็บไว้ในแรม

คนที่เขียนโปรแกรมภาษาโครงสร้างมีพอสมควร คงจะรู้สึกว่าเนื้อหาในบทนี้เป็นเรื่องง่าย คุณสามารถอ่านผ่านๆ ไปได้อย่างรวดเร็ว แต่ไม่ขอแนะนำให้ข้ามไปเลย เพราะภาษาจาวามีรายละเอียดปลีกย่อยที่ไม่เหมือนภาษาอื่น และในบทนี้เราจะเน้นข้อแตกต่างเหล่านั้นเป็นหลัก

ค่าคงตัว

ค่าคงตัว คือ ข้อมูลที่เราป้อนให้กับคอมพิวเตอร์ เพื่อให้คอมพิวเตอร์นำไปใช้ในการคำนวณตามที่เราต้องการ ค่าคงตัวในภาษาจาวามี 5 ชนิดหลักๆ ได้แก่ ค่าคงตัวจำนวนเต็ม ค่าคง

ตัวทศนิยม ค่าคงตัวตรรก ค่าคงตัวตัวอักษร และ ค่าคงตัวข้อความ แต่ละชนิดมีรูปแบบการเขียนที่เฉพาะเจาะจง

ค่าคงตัวจำนวนเต็ม

ค่าคงตัวชนิดแรกคือจำนวนเต็ม จำนวนเต็มในภาษาจาวามีสองแบบคือ ค่าคงตัวจำนวนเต็มปกติ และค่าคงตัวจำนวนเต็มแบบยาว

รูปแบบการเขียนค่าคงตัวจำนวนเต็มปกติ เราใช้ตัวเลขจำนวนเต็มธรรมดา เช่น
13, `200, -10, 12541`

ค่าคงตัวจำนวนเต็มปกติเป็นได้ทั้งค่าบวกและลบ และมีค่าได้ระหว่าง –214783648 ถึง +2147483647 ถ้าต้องการเขียนเลขจำนวนเต็มที่มีค่ามากกว่านี้ ต้องใช้ค่าคงตัวจำนวนเต็มอีกแบบหนึ่งคือ ค่าคงตัวจำนวนเต็มแบบยาว

รูปแบบการเขียนค่าคงตัวจำนวนเต็มแบบยาว เราใช้ตัวเลขจำนวนเต็มต่อท้ายด้วยอักษร `L` หรือ `l` เช่น
13`L, 200L, -10L, 12541L, 50000000000L`

ค่าคงตัวจำนวนเต็มแบบยาวมีค่าได้ระหว่าง -9223372036854775808 ถึง 9223372036854775807

สำหรับคนที่คุ้นเคยกับเลขฐานสอง เลขฐานแปด และเลขฐานสิบหก เราสามารถเขียนค่าคงตัวจำนวนเต็มเป็นเลขฐานได้ด้วย รูปแบบของการเขียนก็คือใช้สัญลักษณ์ 0 นำหน้าถ้าเป็นเลขฐานแปด และใช้สัญลักษณ์ `0x` นำหน้าถ้าเป็นเลขฐานสิบหก ตัวอย่างเช่น ตัวเลข 27 ในฐานสิบ มีค่าเท่ากับ `33` และ `1B` ในฐานแปดและสิบหกตามลำดับ เราอาจเขียนเป็น `033` หรือ `0x1b` ก็ได้

ค่าคงตัวทศนิยม

ค่าคงตัวทศนิยมในภาษาจาวามีสองแบบคือ ค่าคงตัวทศนิยมปกติ และค่าคงตัวทศนิยมแบบยาว

ตัวเลขทศนิยมจะต้องมีจุดเสมอแม้ว่าทศนิยมตัวนั้นจะมีค่าเท่ากับจำนวนเต็ม เช่น 5 ต้องเขียนว่า 5.0 เพราะจาวาจะใช้จุดในการแยกแยะระหว่างทศนิยมกับจำนวนเต็ม

รูปแบบการเขียนค่าคงตัวทศนิยมปกติ เราใช้ตัวเลขทศนิยมธรรมดาตามด้วยอักษร F หรือ f เช่น

10.2F, 21501.45F, 5.0F, 3152.34F

ค่าคงตัวทศนิยมปกติเป็นได้ตั้งแต่ -1.4×10^{45} และ $+3.4 \times 10^{38}$ ถ้าต้องการเขียนทศนิยมมีค่ามากกว่านี้ ต้องใช้ค่าคงตัวจำนวนเต็มอีกแบบหนึ่งคือ ค่าคงตัวทศนิยมแบบยาว

รูปแบบการเขียนค่าคงตัวทศนิยมแบบยาว เราใช้เลขทศนิยมต่อท้ายด้วยอักษร D หรือ d หรือจะเขียนเลขทศนิยมเฉยๆ ก็ได้ เช่น

10.2D, 21501.45D, 5.0D, 5.0, 3152.34

ค่าคงตัวทศนิยมมีค่าได้ตั้งแต่ -4.9×10^{-324} ถึง 1.7×10^{308}

เราสามารถเขียนตัวเลขทศนิยมให้อยู่ในรูปของเลขยกกำลังได้ด้วย โดยเราใช้สัญลักษณ์ e เช่น 1.7e308 หมายถึง 1.7×10^{308}

ค่าคงตัวตรรก

ค่าคงตัวตรรก คือค่าความจริงทางตรรกศาสตร์ ซึ่งเป็นไปได้แค่สองกรณีเท่านั้น คือ จริง หรือ เท็จ เราใช้คำเฉพาะว่า true และ false ตามลำดับ ตัวอย่างเช่น

true, false

ทั้งสองคำต้องเขียนด้วยอักษรตัวพิมพ์เล็กเท่านั้น

ค่าคงตัวตัวอักษร

ค่าคงตัวตัวอักษร ได้แก่ตัวอักษร ตัวเลข เครื่องหมาย และสัญลักษณ์ต่างๆ เราใช้เครื่องอัญประกาศขาเดียวล้อมตัวอักษรนั้นเวลาเขียน ตัวอย่างเช่น

'a', 'A', 'z', 'Z', '0', '9' , '$'

มีเครื่องหมายบางตัวเวลาเขียนต้องใส่เครื่องหมาย \ไว้ข้างหน้าตัว เครื่องหมายในกลุ่มนี้มีหลายตัวแต่ที่ควรจดจำ ได้แก่

ตารางที่ 4-1 เครื่องหมายพิเศษและวิธีการเขียนให้อยู่ในรูปของค่าคงตัวตัวอักษร

ตัวอักษร	วิธีเขียนโดยใช้เครื่องหมายอัญประกาศ
Backspace	'\b'
Horizontal tab	'\t'
Linefeed	'\n'
Form feed	'\f'
Carriage return	'\r'
Apostrophe-quote	'\''
Quotation mark	'\"'
Backslash	'\\'

ค่าที่เป็นไปได้ของค่าคงตัวตัวอักษรไม่ได้มีแค่อักษรที่เห็นบนแป้นพิมพ์เท่านั้น แต่เป็นอักขระอะไรก็ได้ที่อยู่ในรหัสยูนิโค้ด 16 บิตสากล อักขระเหล่านี้จะมีรหัสประจำตัวอยู่เป็นเลข 16 บิต ตัวอย่างของอักขระบางส่วนของรหัสยูนิโค้ด 16 บิตสากล และรหัสประจำตัวเป็นดังนี้

ตารางที่ 4-2 บางส่วนของอักขระในรหัสยูนิโค้ด 16 บิต พร้อม รหัสประจำตัว

ตัวอักษร	วิธีการเขียนโดยใช้เลขฐานสิบหกประจำรหัสยูนิโค้ด
0	0030
9	0039
a	0061
A	0041
z	007a
Z	005a
ȧ	008c
ß	00a7

อักขระเหล่านี้บางตัวไม่มีอยู่บนแป้นพิมพ์ ดังนั้น จาวาจึงยอมให้เราเขียนค่าคงตัวตัวอักขระในรูปของเลข 16 บิตประจำตัวของมันได้ด้วย โดยต้องเขียนคำว่า `\u` นำหน้าเช่น `'a'` อาจเขียนเป็น `\u0061` ก็ได้

ค่าคงตัวข้อความ

ค่าคงตัวข้อความ มีลักษณะคล้ายค่าคงตัวแบบตัวอักษรแต่มีความยาวได้มากกว่าหนึ่งตัวอักษรและใช้เครื่องหมายอัญประกาศสองขาคร่อมข้อความไว้ ตัวอย่างเช่น

`“Hello World” , “This is the message.”, “a”`

เราสามารถผสมรหัสยูนิโค้ดหรือ escape character ได้เช่นเดียวกับค่าคงตัวอักษร ตัวอย่างเช่น

`“I said \”yes.\””`
หมายถึง ข้อความ `I said “yes.”`

`“Here comes a tab. \t And here comes another one \u0009”`
หมายถึง ข้อความ `Here comes a tab.[tab]And here comes another one [tab]`

ตัวแปร

ตัวแปร คือหน่วยความจำในแรม ซึ่งสามารถใช้เก็บค่าคงตัวได้ ตัวแปรพื้นฐานในภาษาจาวามี 8 ชนิดได้แก่ ตัวแปรจำนวนเต็ม ตัวแปรทศนิยม ตัวแปรตรรก และ ตัวแปรตัวอักษร

ตัวแปรจำนวนเต็ม

ตัวแปรจำนวนเต็มมีสี่ชนิดคือ byte, short, int และ long แต่ละชนิดมีขนาดไม่เท่ากัน ขนาดของตัวแปรนับเป็นไบต์ ซึ่งก็คือขนาดของหน่วยความจำนั้นเอง ตัวแปรจำนวนเต็มขนาดเล็กใช้เก็บค่าได้ไม่ใหญ่เท่าตัวแปรจำนวนเต็มขนาดใหญ่ แต่กินเนื้อที่ในแรมน้อยกว่า

ค่าสูงสุด ต่ำสุดที่ตัวแปรแต่ละชนิดเก็บได้ และขนาดในแรม สรุปได้ดังตาราง

ตารางที่ 4-3 ตัวแปรจำนวนเต็ม

ชื่อชนิดของตัวแปร	ค่าสูงสุด	ค่าต่ำสุด	ขนาดในแรม
byte	+127	-128	1 ไบต์
short	+32767	-32768	2 ไบต์
int	+2147483647	-214783648	4 ไบต์
long	9223372036854775807	-9223372036854775808	8 ไบต์

ตัวแปรทศนิยม

ตัวแปรทศนิยมมีสองชนิดคือ `float` และ `double` ค่าสูงสุด ต่ำสุดที่ตัวแปรแต่ละชนิดเก็บได้ และขนาดในแรม สรุปได้ดังตาราง

ตารางที่ 4 - 4 ตัวแปรทศนิยม

ชื่อชนิดของตัวแปร	ค่าสูงสุด	ค่าต่ำสุด	กินเนื้อที่แรม
float	3.40282×10^{38}	-1.4×10^{-45}	4 ไบต์
double	1.79769×10^{308}	-4.9×10^{-324}	8 ไบต์

ตัวแปรตรรก

ตัวแปรตรรกใช้เก็บค่าความจริงทางตรรกศาสตร์มีชนิดเดียวคือ `boolean`

ตัวแปรตัวอักษร

ตัวแปรตัวอักษรใช้เก็บตัวอักษรมีชนิดเดียวคือ `char` และมีขนาด 2 ไบต์

ชื่อของตัวแปร

เพื่อให้เราสามารถอ้างอิงถึงตัวแปรที่เราบอกให้จาวาเวอร์ชัวน์แมทชีนสร้างขึ้นได้ตลอดโปรแกรมที่เราเขียนขึ้น เราต้องมีการตั้งชื่อให้กับตัวแปร การตั้งชื่อตัวแปรนั้นจะใช้ชื่ออะไรก็ได้ ตัวอักษรที่ประกอบเป็นชื่อจะเป็นตัวอักษรภาษาอังกฤษตัวพิมพ์เล็ก ตัวพิมพ์ใหญ่ ตัวเลข เครื่องหมายสกุลเงินต่างๆ อักษรโรมัน รวมทั้งเครื่องหมาย _

มีข้อแม้คือชื่อของตัวแปรห้ามขึ้นต้นด้วยตัวเลข และตัวพิมพ์ใหญ่กับตัวพิมพ์เล็กถือว่าเป็นคนละตัวอักษร นอกจากนี้ยังห้ามตั้งชื่อด้วยคำสงวน ซึ่งมีดังต่อไปนี้

```
abstract    do          import      public      throws
boolean     double      instanceof  return      transient
break       else        int         short       try
byte        extends     interface   static      void
case        final       long        strictfp    volatile
catch       finally     native      super       while
char        float       new         switch      null
class       for         package     synchronized true
continue    if          private     this        false
```

```
default     implements     protected     throw
```

ตัวอย่างต่อไปนี้เป็นชื่อตัวแปรที่ถูกหลักภาษาจาวา

```
number, Number, sum_$, $_100
```

ต่อไปนี้เป็นตัวอย่างชื่อตัวแปรที่ไม่ถูกหลักภาษาจาวา

```
4Thai, all/done, get-big-fast
```

กฎการตั้งชื่อของตัวแปรนี้นำไปใช้ในการตั้งชื่อคลาสและแมธธอสได้ด้วย ชื่อตัวแปรในภาษาจาวานิยมใช้ชื่อที่ขึ้นต้นด้วยอักษรภาษาอังกฤษตัวพิมพ์เล็ก และถ้าประกอบด้วยคำหลายๆ คำจะใช้ตัวพิมพ์ใหญ่ขึ้นต้นคำทุกคำที่ตามมาเช่น

```
x, anInt, getBigFast
```

ทั้งนี้เป็นเพียงแค่ความนิยมเท่านั้น

เมื่อเราได้ชื่อของตัวแปรที่เราต้องการจะสร้างแล้ว คำสั่งที่ใช้ในการสร้างตัวแปรมีสองขั้นตอน คือคำสั่งในการประกาศตัวแปร และคำสั่งในการกำหนดค่าคงตัวให้กับตัวแปร

การประกาศตัวแปร

เราสามารถบอกให้จาวาเวอร์ชัวน์แมทชีนจองที่ในแรมให้เราเพื่อไว้ใช้เป็นตัวแปรเก็บค่าคงตัวได้ด้วยการประกาศตัวแปร รูปแบบของคำสั่งในการประกาศตัวแปรเป็นดังตัวอย่าง

```
int anInt;
byte aByte;
long l;
short aShort;
char c;
float f;
boolean oasis;
```

นั้นคือเราประกาศตัวแปรได้ด้วยการใช้ชื่อของชนิดของตัวแปร ตามด้วยชื่อของตัวแปรซึ่งเป็นชื่ออะไรก็ได้ที่เราตั้งขึ้นเอง คำสั่งนี้เป็นการบอกให้จาวาเวอร์ชัวน์แมทชีนจองเนื้อที่ไว้ใช้เป็นตัวแปรโดยขนาดของเนื้อที่จะขึ้นอยู่กับชนิดของตัวแปรที่เราประกาศ อย่าลืมเครื่องหมาย ; ที่บอกจุดสิ้นสุดของคำสั่ง

ในกรณีที่ต้องการประกาศตัวแปรทีละหลายๆ ตัว สามารถทำได้ในคำสั่งเดียว แต่ตัวแปรเหล่านั้นต้องเป็นตัวแปรชนิดเดียวกัน เช่น

```
int i,j,k;
```

ตัวอย่างนี้เป็นการประกาศตัวแปร int จำนวนทั้งสิ้นสามตัวชื่อ i j และ k ตามลำดับ เราใช้เครื่องหมายจุลภาคในการคั่นระหว่างชื่อของตัวแปร

การกำหนดค่าตัวแปร

เมื่อเราประกาศตัวแปรแล้ว ขั้นตอนต่อไปก็คือการกำหนดค่าคงตัวให้กับตัวแปร ในขณะที่การประกาศตัวแปรเป็นการบอกให้เวอร์ชัวน์แมทชีนกันที่ในแรมไว้ให้ การกำหนดค่าตัวแปรเป็นการนำค่าคงตัวมาใส่ไว้ในที่ว่างนั้น รูปแบบของคำสั่งในการกำหนดค่าตัวแปรเป็นดังตัวอย่าง

```
anInt = 5;
aByte = 15;
l = 2000L;
aShort = 129;
c = ‘g’;
f = 10.05F;
oasis = true;
```

รูปแบบของคำสั่งกำหนดค่าให้ตัวแปร เริ่มจากชื่อของตัวแปร ตามด้วยเครื่องหมายเท่ากับ และตามด้วยค่าคงตัวที่ต้องการเก็บ ชนิดของตัวแปรและชนิดของค่าคงตัวต้องสัมพันธ์กัน ตัวอย่างเช่น ตัวแปร int ใช้เก็บค่าคงตัวจำนวนเต็ม จะกำหนดให้มีค่าเท่ากับค่าคงตัวตรรกไม่ได้

ถ้าชนิดของตัวแปรและค่าคงตัวที่กำหนดให้สอดคล้องกัน แต่ขนาดของตัวแปรไม่สามารถรับค่าคงตัวได้ ก็ไม่สามารถกำหนดค่าให้ได้ เช่น ตัวแปร int ไม่สามารถรับค่าคงตัวจำนวนเต็ม 5000000000000L ได้เพราะใหญ่เกินพิสัยของมัน

เวลากำหนดตัวแปรจำนวนเต็ม ตัวแปร byte, short และ int จะรับค่าคงตัวจำนวนเต็มปกติ ส่วนตัวแปร long จะรับค่าคงตัวจำนวนเต็มปกติ หรือค่าคงตัวจำนวนเต็มแบบยาวก็ได้ เช่น

```
aByte = 5;
aShort = 50;
```

```
anInt = 500;
l = 50000000000L;
l = 5000;
anInt = 500L; // Error
```

ในกรณีสุดท้าย แม้ตัวแปร int จะรับค่าขนาด 500 ได้ แต่เราเขียน 500 เป็นแบบค่าคงตัวจำนวนเต็มแบบยาว จาวาจะถือว่ามีขนาดใหญ่กว่าตัวแปร int จึงใช้ตัวแปร int เก็บไม่ได้

ตัวแปร short และ byte มีพิสัยแคบกว่าตัวแปร int ห้ามรับค่าเกินพิสัย เช่น

```
aByte = 130; //Error
```

คำสั่งนี้ใช้ไม่ได้เพราะ ตัวแปร byte มีค่าสูงสุดได้แค่ 127 เท่านั้น

ตัวแปร char นอกจากจะกำหนดให้มีค่าเป็นตัวอักษรแล้ว ยังกำหนดให้มีค่าเป็นตัวเลขได้ด้วย เช่น

```
c = 97;
```

แต่ผลที่ได้ก็คือจะได้ c = 'a' เพราะอักขระ a มีรหัสยูนิโค้ดประจำตัวเป็น \u0061 ซึ่งเท่ากับ 97 ในระบบเลขฐานสิบ หรืออีกนัยหนึ่งเราสามารถกำหนดค่าให้ตัวแปร c มีค่าเท่ากับอักขระ a ได้สามแบบดังนี้

```
c = 'a';
c = \u0061;
c = 97;
```

เมื่อทราบวิธีประกาศและกำหนดค่าของตัวแปรแล้ว เรามาลองเขียนโปรแกรมจริงๆ โดยใช้คำสั่งเหล่านี้ดู ลองเรียกดอสขึ้นมา แล้วเรียก Notepad ในโฟล์เดอร์ C:\java เพื่อสร้างเท็กซ์ไฟล์ชื่อ TestVariable.java ดังนี้

```
C:\java> notepad TestVariable.java
```

พิมพ์โปรแกรมต่อไปนี้ลงไปแล้วบันทึกไว้ใน C:\java ระวังไม่ให้มีนามสกุล .txt ต่อท้ายเวลาบันทึก

โปรแกรม 4 - 1 : TestVariable.java

```
public class TestVariable {
  public static void main(String[]args){
    int i; // (1)
    i = 20; // (2)
    System.out.println(i); // (3)
    i = 97; // (4)
    System.out.println(i); // (5)
```

```
        char c; // (6)
        c = i; // (7)
        System.out.println(c); // (8)
    }
}
```

คงยังจำได้ว่าโปรแกรมในภาษาจาวาประกอบด้วยคลาส ในคลาสมีแมธธอส และถ้าเราต้องการสั่งให้โปรแกรมทำอะไรต่อมิอะไรเราก็เพียงแต่ใส่คำสั่งเหล่านั้นลงไปในส่วนตัวของแมธธอส โปรแกรมข้างต้นก็เช่นกัน ประกอบด้วยคลาสหนึ่งคลาสชื่อ TestVariable ภายในมีแมธธอสชื่อ main ส่วนตัวของแมธธอส main มีคำสั่ง 8 คำสั่ง

คำสั่งในบรรทัด (1) คือคำสั่งประกาศตัวแปรชนิด int ให้ชื่อว่า i

คำสั่งในบรรทัด (2) คือคำสั่งกำหนดค่าตัวแปร i ให้เท่ากับ 20

คำสั่งในบรรทัด (3) คือคำสั่งให้แสดงค่าของตัวแปร i ออกนอกจอ ซึ่งก็คือค่าคงตัวที่ i เก็บไว้ ได้แก่ 20

คำสั่งในบรรทัด (4) กำหนดค่าใหม่ให้ i มีค่าเท่ากับ 97

คำสั่งในบรรทัด (5) แสดงค่า i ออกนอกจอใหม่ คราวนี้มีค่าเป็น 97

คำสั่งในบรรทัด (6) เป็นการประกาศตัวแปร char ให้ชื่อว่า c

คำสั่งในบรรทัด (7) กำหนดให้ตัวแปร c มีค่าเท่ากับค่าของตัวแปร i ซึ่งก็คือ 97

คำสั่งในบรรทัด (8) แสดงค่า c ออกนอกจอ คราวนี้จะได้เป็น a ทดลองคอมไพล์โปรแกรมนี้และรันดู ดังนี้

```
C:\java> javac TestVariable.java
C:\java> java TestVariable
20
97
a
```

สิ่งที่ได้จากโปรแกรมนี้ก็คือ ตัวแปรเป็นแค่เนื้อที่ว่างในแรม ซึ่งเราสามารถกำหนดค่าให้ใหม่ได้ตลอดเวลา และเราสามารถกำหนดค่าของตัวแปรให้มีค่าเท่ากับค่าของตัวแปรอื่นได้ด้วย

สังเกตว่าคราวนี้สิ่งที่อยู่ในวงเล็บหลังคำสั่ง `System.out.println` ไม่มีเครื่องหมายคำพูดคร่อมอยู่ เครื่องหมายคำพูดเป็นการบ่งบอกลักษณะของค่าคงตัวแบบข้อความ แต่ครั้งนี้เราต้องการแสดงค่าตัวแปรไม่ใช่ข้อความ ดังนั้นจึงเขียน `i` หรือ `c` เฉยๆ ไม่มีเครื่องหมายคำพูด ถ้าเราใส่เครื่องหมายคำพูดคร่อม `i` หรือ `c` สิ่งที่ได้คือตัวอักษร `i` หรือ `c` ออกนอกจอ ไม่ใช่ค่าของ `i` หรือ `c` ที่เราต้องการ

อันที่จริงแม้การประกาศตัวแปรกับการกำหนดค่าให้ตัวแปรจะเป็นคนละเรื่องกัน แต่จาวายอมให้คุณรวมคำสั่งทั้งสองนี้ไว้เป็นคำสั่งเดียวกันได้ด้วย รูปของคำสั่งจะเป็นดังตัวอย่าง

```
int anInt = 5;
long aLong = 10;
```

คำสั่งเริ่มด้วยชื่อชนิดของตัวแปร ตามด้วยชื่อของตัวแปร ตามด้วยเครื่องหมายเท่ากับ และค่าคงตัวที่ต้องการกำหนดให้ รูปแบบคำสั่งแบบนี้ใช้ได้กับตัวแปรทุกชนิด ต่อไปนี้เราจะใช้คำสั่งแบบย่อแบบนี้เป็นหลัก เพื่อความกะทัดรัด นอกจากนี้เรายังประกาศหรือกำหนดค่าตัวแปรมากกว่าหนึ่งตัวในคำสั่งเดียวด้วย หากตัวแปรเป็นตัวแปรชนิดเดียวกันและค่าคงตัวที่กำหนดให้เหมือนกัน เช่น

```
int i, j;
i = j = 10;
```

ค่าของตัวแปรสามารถเปลี่ยนไปได้เรื่อยๆ ด้วยการกำหนดค่าใหม่ให้ เราสามารถบังคับให้ตัวแปรที่เราสร้างขึ้นถูกกำหนดค่าได้แค่ครั้งเดียวได้ด้วยคำสั่ง `final` เช่น

```
final int x = 5;
```

กรณีนี้ `x` จะมีค่าเท่ากับ 5 ตลอดโปรแกรม จะกำหนดค่าใหม่ให้อีกภายหลังไม่ได้แล้ว

การกำหนดค่าให้กับตัวแปรทศนิยม ทำได้ดังตัวอย่าง

```
double aDouble = 10.0D;
float aFloat = 1.5F;
double x = 1.234e20;
double y = 150;
float z = 30000L
```

ตัวแปรทศนิยมแบบ float รับค่าคงตัวทศนิยมปกติ ส่วนตัวแปรทศนิยมแบบ double รับค่าคงตัวทศนิยมแบบปกติหรือแบบยาวก็ได้ นอกจากนี้ตัวแปรทศนิยมยังรับค่าคงตัวจำนวนเต็มได้ด้วย เช่น ตัวแปร y ในตัวอย่างจะมีค่าเป็น 150.0D จาวาแปลงค่าคงตัวจำนวนเต็มเป็นค่าคงตัวทศนิยมให้เราโดยอัตโนมัติ

ในทางตรงกันข้ามตัวแปรจำนวนเต็มรับค่าทศนิยมไม่ได้ เช่น

```
int i = 50.0D; //Error
```

เพราะจาวาไม่ต้องการถือวิสาสะปัดเศษทศนิยมให้ถ้ามี เพราะอาจทำให้การคำนวณของเราผิดพลาด

สำหรับตัวแปรตรรก เรากำหนดค่าอย่างง่ายๆ ได้ด้วยค่าคงตัวตรรกคือคำว่า true หรือ false ตัวอย่างเช่น

```
boolean x = true;
boolean y = false;
```

การแคส

เวลาคอมไพล์เลอร์ทำงาน มันจะตรวจสอบความเข้ากันได้ระหว่างตัวแปรทางด้านซ้าย กับค่าคงตัวหรือตัวแปรทางด้านขวาของเครื่องหมายเท่ากับ ถ้าเข้ากันไม่ได้มันจะไม่ยอมคอมไพล์โปรแกรมให้ เราสามารถบังคับให้คอมไพล์เลอร์คอมไพล์ได้ด้วย **การแคส** ตัวอย่างเช่น

```
byte x = (byte) 130;
```

ปกติแล้ว x รับค่า 130 ไม่ได้ เพราะค่าสูงสุดที่เป็นไปได้ของตัวแปรแบบ byte คือ 127 เราสามารถบังคับให้คำสั่งนี้ผ่านได้ด้วยการแคส ซึ่งก็คือการใช้คำสั่ง (byte) กำกับเลข 130 คอมไพเลอร์จะไม่สนใจค่าของตัวเลข 130 เพราะเรากำกับว่าตัวเลขตัวนี้ไม่ว่ามีค่าเป็นเท่าไรถือว่าเข้ากันได้กับตัวแปรแบบ byte ดังนั้นคอมไพเลอร์จะปล่อยผ่าน คำสั่งนี้จึงเป็นคำสั่งที่ไม่ผิดหลักภาษาจาวา

แต่อย่างไรก็ตามผลที่ได้คือ x จะมีค่าแค่ 127 เท่านั้น เพราะ 127 เป็นค่าที่ใกล้เคียง 130 ที่สุดที่สามารถกำหนดให้ตัวแปรแบบ byte ได้

หรืออย่างในกรณีของตัวแปร `float` กับค่าคงตัวทศนิยมแบบยาว ถ้าต้องการจับให้เท่ากันได้ก็ให้แคสค่าคงตัวทศนิยมแบบยาวให้เป็น `float` เช่น

```
float f = (float) 10.3D;
```

ในกรณีนี้ค่าของ `f` จะเท่ากับ 10.3F ไม่ต้องมีการปัดเพราะในความเป็นจริง `float` รับค่าทศนิยม 10.3 ได้อยู่แล้ว

บ่อยครั้งที่ทางขวาของเครื่องหมายเท่ากับเป็นตัวแปรไม่ใช่ค่าคงตัว คอมไพเลอร์จะเช็คขนาดของตัวแปรแทนที่จะดูจากค่าในตัวแปร และมักทำให้เกิดปัญหาในการกำหนดค่าขึ้น เช่น

```
int i = 20;
byte b = i; \\Error
```

ที่จริงแล้วไม่น่ามีปัญหาเพราะตัวแปร `byte` รับค่า 20 ได้อยู่แล้ว แต่เพราะเราใช้ตัวแปร `i` แทนที่จะใช้ตัวเลข 20 โดยตรง แบบนี้คอมไพล์เลอร์จะไม่ให้ผ่าน เพราะมันจะคิดว่าเป็นตัวแปร `i` มีค่ามากกว่า 127 ก็ได้ เราแก้ปัญหานี้ได้ด้วยการแคส ดังนี้

```
int i = 20;
byte b = (byte) i;
```

ตัวอย่างอื่นๆ ของการแคสเป็นดังนี้

```
float  f = (float) 100.5D;
int i = (int) f;
byte b = (byte) i;
char c = (char) 3.14F;
short s = (short) 'a';
byte b = 32;
char d = (char) b;
```

จากตัวอย่างข้างต้นจะเห็นว่าเราใช้การแคสในการบังคับให้ตัวแปรทางซ้ายรับค่าตัวแปรทางขวาซึ่งมีขนาดใหญ่กว่า หรือใช้การแคสในกรณีที่มีการใช้ตัวแปร char ในฐานะของตัวเลข จาวาจะหาค่าที่เหมาะสมที่สุดที่เป็นไปได้ให้ เช่น ในกรณีนี้ ตัวแปร i จะมีค่า 100 ส่วนตัวแปร c จะมีค่า \u0003 เป็นต้น

เคล็ดลับ วิธีที่ดีสุดในการป้องกันปัญหาเรื่องความเข้ากันไม่ได้ของตัวแปรและค่าคงตัว คือ การใช้ตัวแปร `int` หรือ `long` เท่านั้นสำหรับการเก็บจำนวนเต็ม และการใช้ตัวแปร `double` เท่านั้นสำหรับการเก็บทศนิยม การใช้ตัวแปร `byte` `short` และ `float` อาจดีในแง่ของการประหยัดเนื้อที่ในแรมแต่ประโยชน์ที่ได้ไม่คุ้ม

เมื่อเปรียบเทียบกับความเสี่ยงที่เกิดจากความผิดพลาดของโปรแกรมเนื่องจากความเข้ากันไม่ได้ของตัวแปรและค่าคงตัว

5

เครื่องหมาย

เครื่องหมายคือสิ่งที่บอกให้จาวาเวอร์ชัวน์แมทชีนนำค่าคงตัวที่อยู่ในตัวแปรไปคำนวณในลักษณะที่เราต้องการ เครื่องหมายในภาษาจาวามีอยู่อย่างมากมาย เครื่องหมายพื้นฐานที่สุดได้แก่ เครื่องหมายคณิตศาสตร์ บวก ลบ คูณ หาร

เครื่องหมายคณิตศาสตร์

เครื่องหมายคณิตศาสตร์สามารถใช้ได้กับ ค่าคงตัวจำนวนเต็ม ค่าคงตัวทศนิยม ตัวแปรจำนวนเต็ม และตัวแปรทศนิยม มีดังนี้

ตารางที่ 5-1 เครื่องหมายคณิตศาสตร์

ชื่อเครื่องหมาย	สัญลักษณ์	ชนิด	ตัวอย่าง
เลขบวก	+	เดี่ยว	+5, +5.0 , +a
เลขลบ	-	เดี่ยว	-2, - 2.0, -a
การบวก	+	คู่	1 + 2, 1.0+ 2.0, a + b
การลบ	-	คู่	2 – 1, 2.0 - 1.0, a - b
การคูณ	*	คู่	4 * 5, 4.0 * 5.0, a * b
การหาร	/	คู่	10 / 2, 10.0/2.0, a / b
มอดูโล	%	คู่	10 % 3, 10.0 % 3.0, a % b

เครื่องหมายคณิตศาสตร์แบ่งออกเป็นสองจำพวกคือ เครื่องหมายเดี่ยว กับเครื่องหมายคู่

เครื่องหมายเดี่ยวมีสองตัวได้แก่ + และ – ซึ่งเขียนอยู่หน้าค่าคงตัวหรือตัวแปร ส่วนเครื่องหมายคู่เป็นการกระทำกันระหว่างเลขสองตัว

เครื่องหมายหารหรือ / จะให้ผลลัพธ์เป็นจำนวนเต็ม ถ้าตัวตั้งเป็นจำนวนเต็มทั้งคู่ โดยจะปัดเศษทิ้งเสมอ แต่ถ้าตัวตั้งเป็นเลขทศนิยมจะคงเศษไว้ในรูปของทศนิยม และถ้าตัวตั้งเป็นจำนวนเต็มกับทศนิยม ผลลัพธ์จะเป็นเลขทศนิยม

```
int i = 5 / 2;
double d = 5.0 / 2.0;
double e = 5 / 2.0;
```

กรณีนี้ i มีค่าเท่ากับ 2 ส่วน d และ e มีค่าเท่ากับ 2.5

เครื่องหมายมอดูโล คือการหาเศษของผลหาร ถ้าตัวตั้งเป็นจำนวนเต็ม ผลลัพธ์ก็คือเศษของผลหาร ถ้าตัวตั้งเป็นเลขทศนิยม ผลลัพธ์ก็คือเศษของผลหารเหมือนกันแต่เก็บไว้ในรูปของเลขทศนิยม

```
int i = 5 % 2;
double d = 5.0 % 2.0;
double e = 5 % 2.0;
```

กรณีนี้ i มีค่าเท่ากับ 1 ส่วน d และ e มีค่า 1.0

ปกติแล้ว *, /, % จะมีลำดับความสำคัญสูงกว่า +,- ดังนั้น ถ้ามีเครื่องหมายมากกว่าหนึ่งเครื่องหมาย จาวาเวอร์ชัวน์แมทชีนจะทำ *, /, % ให้เสร็จก่อน โดยคำนวณเรียงตามลำดับจากซ้ายไปขวา แล้วจึงค่อยทำ +,- โดยเรียงจากซ้ายไปขวาเช่นกัน และเราสามารถใช้เครื่องหมายวงเล็บในการกำหนดให้จาวาเวอร์ชัวน์แมทชีนเรียงลำดับความสำคัญใหม่ตามที่เราต้องการได้

```
int i = 3 + 5 * 4 – 4 / 2;
int j = 3 + ( 5 * 4 ) – ( 4 / 2 );
int k = ( 3 + 5 ) *  (4 – (4 /2) );
```

กรณีนี้ i และ j จะมีค่าเท่ากันคือ 21 ส่วน k จะมีค่า 16

ส่วนในกรณีของเครื่องหมายบวกลบหน้าตัวเลข จะมีลำดับความสำคัญสูงกว่า *,/,%,+,- เสมอ

```
int i = -3 * -5 + +2 / +1;
```

กรณีนี้ i มีค่าเป็น 17

นอกจากนี้เครื่องหมายบวกลบหน้าตัวเลขยังกระทำจากขวาไปซ้าย ไม่ใช่ซ้ายไปขวา ถ้ามีมากกว่าหนึ่งตัว

```
int i = --4;
```

ในกรณีนี้ i = -(-4)) หรือ 4 นั้นเอง

ผลลัพธ์ของการกระทำคณิตศาสตร์จะเป็นชนิดเดียวกับตัวตั้งเสมอ เช่น ค่าคงตัวจำนวนเต็มปกติสองตัวบวกกัน ผลลัพธ์ที่ได้จะเป็นค่าคงตัวจำนวนเต็มปกติด้วย หรือ ตัวแปร int บวกตัวแปร int ผลลัพธ์ก็จะเป็นค่าคงตัวจำนวนเต็มปกติ แต่ถ้าเป็นการกระทำระหว่างตัวตั้งต่างชนิดกัน ผลลัพธ์จะเป็นชนิดเดียวกับตัวตั้งตัวที่ใหญ่กว่าเสมอ โดยเรียงลำดับจากใหญ่ไปเล็กดังนี้ ทศนิยมแบบยาว ทศนิยมปกติ จำนวนเต็มแบบยาว จำนวนเต็มปกติ ตัวอย่างเช่น ตัวแปรแบบ long บวกค่าคงตัวทศนิยมปกติ จะได้ผลลัพธ์เป็นค่าคงตัวทศนิยมปกติ เป็นต้น

ในกรณีของตัวแปร short และ byte ไม่ว่าจะกระทำกับอะไรจะต้องได้ผลลัพธ์อย่างน้อยเป็นจำนวนเต็มแบบปกติเสมอ เช่น

```
byte b = 2;
short s = 3
int i = s * b;
```

ตัวแปร i ต้องประกาศให้เป็น int ถ้าประกาศให้เป็น short จะรับผลลัพธ์ของ s * b ไม่ได้ หรืออย่างเช่น

```
short s = 3;
s = (short) -s;
```

กรณีนี้ต้องมีการแคส –s ให้เป็น short ด้วย เพราะ s ถูกกระทำด้วย – แล้วจะกลายเป็นจำนวนเต็มแบบปกติซึ่งใหญ่กว่าตัวแปร short

หรืออย่างเช่น

```
short h = 40;
h = h + 2; // Error
```

ในกรณีนี้ h + 2 กลายเป็นค่าคงตัวขนาด 4 ไบต์ตามตัวตั้ง 2 พอเวลาแทนค่ากลับเข้าไปที่ h จะเกิดปัญหา เพราะขนาดของทางขวาใหญ่กว่าทางซ้าย และทางขวามีตัวแปรเป็นส่วนประกอบ ดังนั้นต้องมีการแคสด้วย

```
short h = 40;
h = (short) (h + 2);
```

สังเกตว่าการแก้ปัญหาด้วยวิธีการข้างล่างนี้ไม่ได้ผล

```
short h = 40;
h = h + (short) 2; //Error
```

เพราะแม้จะปรับ 2 ให้มีขนาด 2 ไบต์เท่ากับ h แต่เมื่อถูกกระทำด้วยเครื่องหมายผลลัพธ์จะมีขนาดอย่างน้อย 4 ไบต์เสมอ ซึ่งใหญ่กว่าตัวแปรทางซ้ายของเครื่องหมายเท่ากับอยู่ดี ตัวแปร char แม้ไม่ใช่ตัวเลข แต่สามารถใช้เครื่องหมายคณิตศาสตร์ได้ด้วย ค่าของมันจะเท่ากับตัวเลขของรหัสยูนิโค้ดที่มันแทน ลองดูตัวอย่างต่อไปนี้

โปรแกรม 5 - 1 : TestOperator.java

```
public class TestOperator {
  public static void main(String[]args){
    char x = 'a' ;
    char y = 'b' ;
    System.out.println(x+y);
  }
}
```

ในโปรแกรมนี้ เราประกาศตัวแปร char สองตัวชื่อ x และ y และกำหนดให้มีค่าเป็น 'a' และ 'b' ตามลำดับ เมื่อแสดงค่าของ x + y จะได้ผลเป็นตัวเลขจำนวนเต็ม 195 ดังนี้

```
C:\java> java TestOperator
195
```

ที่เป็นเช่นนี้เป็นเพราะ รหัสยูนิโค้ดของ 'a' คือ \u0061 หรือ 97 ในระบบเลขฐานสิบ ส่วนรหัสยูนิโค้ดของ 'b' คือ \u0062 หรือ 98 เมื่อนำมาบวกกัน จาวาจะเปลี่ยนตัวอักษรทั้งสองเป็นจำนวนเต็ม ทำให้ได้ผลลัพธ์เป็น 195

ดังนั้นตัวแปร char สามารถกระทำกับตัวแปรจำนวนเต็มอื่นได้ราวกับว่าตัวมันเป็นจำนวนเต็ม และผลลัพธ์ที่ได้ทางคณิตศาสตร์ของตัวแปร char จะกลายเป็นจำนวนเต็ม เช่น

```
char c = 'c';
int i = c + 4;
```

ค่าคงตัวแบบข้อความสามารถใช้เครื่องหมายบวกได้ด้วย แต่ผลลัพธ์คือการเชื่อมข้อความเข้าด้วยกัน ตัวอย่างเช่น

โปรแกรม 5 - 2 : TestOperator.java

```
public class TestOperator {
        public static void main(String[]args){
          System.out.println("Hello"+"World");
        }
}
```

ผลการรันโปรแกรมข้างต้นเป็นดังนี้

```
C:\java> java TestOperator
HelloWorld
```

เครื่องหมายอินครีเมนท์

เครื่องหมายอินครีเมนท์เป็นเครื่องหมายทางคณิตศาสตร์ประเภทหนึ่ง มีความหมายดังตาราง

ตารางที่ 5-2 เครื่องหมายอินครีเมนท์

สัญลักษณ์	ความหมาย	ตัวอย่าง
++	+ 1	++i , i++
- -	- 1	- -i, i- -

ตัวอย่างเช่น

```
int i = 10;
i++;
```

ท้ายที่สุด i จะมีค่าเป็น 11 หรืออีกนัยหนึ่ง คำสั่ง i++ มีความหมายเหมือนกับคำสั่ง i = i + 1

ทั้ง ++i และ i++ เป็นการบวกหนึ่งให้กับตัวแปร i แต่มีความหมายต่างกันถ้านำไปใช้ในการกำหนดค่าของตัวแปร เช่น

```
int i = 10;
int j = ++i;
int x = 10;
int y = x++;
```

ตัวแปร i และ j ท้ายที่สุดแล้วจะมีค่าเป็น 11 เพราะ i ซึ่งมีค่า 10 ถูกบวกด้วยหนึ่งกลายเป็น 11 แล้วนำค่าใหม่นี้ไปกำหนดให้กับ j ด้วย ส่วนในกรณีของ x กับ y จะต่างกัน x ท้ายที่สุดแล้วมีค่า 11 แต่ y จะมีค่าแค่ 10 เพราะคำสั่ง x++ จะทำการกำหนดค่า y ให้มีค่าเท่ากับ x ก่อน แล้วจึงเพิ่มค่าของ x ด้วยหนึ่งทีหลัง

นอกจากการบวกและการลบด้วยหนึ่งแล้ว จาวายังมีโอเปอร์เรเตอร์พิเศษอีกกลุ่มหนึ่งที่มีลักษณะคล้ายเครื่องหมายอินครีเมนท์ แต่ใช้สำหรับการกำหนดตัวแปรโดยเฉพาะ

ตารางที่ 5-3 เครื่องหมายอินครีเมนท์

รูปแบบคำสั่ง	ความหมาย
x *= a	x = x * a
x /= a	x = x / a
x %= a	x = x % a
x += a	x = x + a
x -= a	x = x - a

โดยที่จะมีการแคสให้โดยอัตโนมัติด้วยถ้าจำเป็น ตัวอย่างเช่น

```
byte b = 2;
b += 10;
```

บรรทัดที่ (2) จะมีความหมายเหมือน b = (byte) ((int) b + 10) ซึ่งได้ผลลัพธ์เป็น 12 ซึ่งแตกต่างกับในกรณีที่เราเขียนคำสั่งแบบตรงๆ

```
byte b = 2;
b = b + 2; //Error
```

แบบนี้คอมไพล์ไม่ผ่านเพราะต้องมีการแคสด้วย ดังนี้

```
byte b = 2;
b = (byte) (b + 2);
```

เครื่องหมายสมการและอสมการ

ตารางที่ 5-3 เครื่องหมายสมการและอสมการ

รูปแบบคำสั่ง	ความหมาย
a < b	a น้อยกว่า b หรือไม่
a <= b	น้อยกว่าหรือเท่ากับ หรือไม่
a > b	มากกว่า หรือไม่
a >= b	มากกว่าหรือเท่ากับ หรือไม่
a == b	เท่ากับ หรือไม่
a != b	ไม่เท่ากับ หรือไม่

ตัวตั้งของเครื่องหมายแบบสมการและอสมการเป็นได้ทั้งตัวเลขและตัวอักษร แต่ผลลัพธ์ที่ได้จะเป็นค่าทางตรรกซึ่งมีค่าเป็น `true` หรือ `false` เท่านั้นและมีค่าเทียบเท่ากับตัวแปรตรรก แม้ว่าตัวตั้งจะไม่ใช่ตัวแปรตรรกก็ตาม

```
int b = 5;
boolean y = b > 4;
```

ในกรณีนี้ `y` มีค่าเท่ากับ `true`

เครื่องหมายแบบสมการและอสมการ ไม่ต้องมีการแคส แม้ว่า `b` และ 4 จะไม่ใช่ตัวแปรแบบตรรก วลี `b > 4` เป็นวลีทางตรรกศาสตร์ ซึ่งมีค่าเป็นจริงหรือเท็จ ซึ่งเข้ากันได้กับตัวแปร `y` ซึ่งเป็นตัวแปรแบบ `boolean` อยู่แล้ว

สังเกตว่าเครื่องหมายเท่ากับใช้เครื่องหมายเท่ากับสองอัน เพราะเครื่องหมายเท่ากับอันเดียวเป็นสัญลักษณ์ของการกำหนดค่าตัวแปร ไม่ใช่เครื่องหมายสมการและอสมการ

เครื่องหมายตรรก

เครื่องหมายตรรกใช้เชื่อม ค่าคงที่ตรรก ตัวแปรตรรก และวลีที่เกิดจากเครื่องหมายแบบสมการและอสมการ ผลลัพธ์ที่ได้เป็นค่าความจริงทางตรรก

ตารางที่ 5-4 เครื่องหมายตรรก

รูปแบบคำสั่ง	ความหมาย
!x	ค่าความจริงที่ตรงข้ามกับ x
x & y	ค่าความจริงของ x AND y
x | y	ค่าความจริงของ x OR y
x ^ y	ค่าความจริงของ x XOR y
x && y	ค่าความจริงของ x AND y

x || y	ค่าความจริงของ x OR y

ค่าความจริงทางตรรกศาสตร์เป็นดังตาราง

ตารางที่ 5-5 ตารางค่าความจริง

x	y	!x	x & y	x | y	x ^ y	x && y	x || y
true	true	false	true	true	false	true	true
true	false	false	false	true	true	false	true
false	true	true	false	true	true	false	true
false	false	true	false	false	false	false	false

ตัวอย่าง

```
boolean b = true;
boolean c = b & ( 4 > 5 );
boolean d = !c;
```

ตัวแปร `c` ในตัวอย่างข้างต้นมีค่าเท่ากับ `false` ส่วนตัวแปร `d` มีค่า `true`

สังเกตว่าเครื่องหมาย `&` และ `&&` มีค่าความจริงเหมือนกันทุกประการ เครื่องหมาย `|` กับ `||` ก็เช่นเดียวกัน ที่จริงแล้วมีความแตกต่างกันเล็กน้อย กล่าวคือเครื่องหมาย `&` และ `|` จะหาผลลัพธ์ทางตรรกข้างซ้ายและข้างขวาของมันจนเสร็จเสมอ ในขณะที่ `&&` และ `||` มีลักษณะ ลัดวงจร กล่าวคือ ถ้าผลลัพธ์สามารถหาได้โดยไม่ต้องคำนวณจนจบมันจะหยุดคำนวณทันที ดังตัวอย่าง

```
int i = 10;
boolean b = false & (i++ > 11) ;
```

ตัวอย่างนี้ในที่สุดจะได้ `b` มีค่า `false` และ `i` มีค่าเป็น `11` ในขณะที่

```
int i = 10
boolean b = false && (i++ > 11);
```

จะได้ `b` มีค่า `false` เช่นกัน แต่ `i` จะมีค่าแค่ 10 เพราะสำหรับเครื่องหมาย `&&` ถ้าข้างใดข้างหนึ่งของเครื่องหมายเป็น `false` เรารู้ทันทีว่าผลลัพธ์ต้องเป็น `false` แน่นอน ในกรณีนี้ด้านซ้ายของเครื่องหมายเป็น `false` ดังนั้นจาวาเวอร์ชัวน์แมทชีนจะทิ้งสิ่งที่อยู่ทางขวาของเครื่องหมายไปเลย ทำให้ไม่มีการบวกหนึ่งให้กับ `i` เกิดขึ้น

ลองดูวลีทางตรรกที่ซับซ้อนขึ้น

```
int d = 10;
char m =  'x';
boolean r = (d >  4) && (5.5 < 3.0) || (m == 'x');
```

ตัวอย่างนี้ตัวแปร r จะมีค่าเป็นจริง เพราะวลี d > 4 มีค่าเป็นจริง วลี 5.5 < 3.0 เป็นเท็จ ส่วนวลี m == 'x' มีค่าเป็นจริง เมื่อนำสามวลีมากระทำกันจะได้เป็น จริง && เท็จ || จริง ซึ่งจาวาเวอร์ชัวน์แมทชีนจะเริ่มกระทำจากซ้ายไปขวา หรือเท่ากับ ((จริง && เท็จ) || จริง) ซึ่งก็คือ (เท็จ || จริง) ซึ่งก็คือ จริง ในที่สุด

ลำดับความสำคัญของเครื่องหมายแบบสมการและอสมการที่มากกว่าเครื่องหมายตรรก ดังนั้นถ้าไม่ใส่วงเล็บ จาวาเวอร์ชัวน์แมทชีนจะทำเครื่องหมายแบบสมการและอสมการจนเสร็จหมดก่อน แล้วจึงค่อยทำเครื่องหมายตรรก

เครื่องหมายต่อไปนี้เป็นเครื่องหมายตรรกแบบที่ใช้สำหรับการกำหนดค่าให้ตัวแปร boolean โดยเฉพาะ

ตารางที่ 5-6 เครื่องหมายตรรก

รูปแบบคำสั่ง	ความหมาย
b &= a	b = b & a
b ^= a	b = b ^ a
b |= a	b = b | a

ตัวอย่างเช่น

```
boolean b = true;
b &= 5.5 <= 3.0
```

ในกรณีนี้ b จะได้ค่าเป็นเท็จ เพราะ b = (b & (5.5 <= 3.0))

เครื่องหมายบิต

เครื่องหมายบิตเป็นเครื่องหมายที่ใช้กับการคำนวณดิจิตอล มีดังนี้

ตารางที่ 5-7 เครื่องหมายบิต

ชื่อเครื่องหมาย	รูปแบบคำสั่ง	ความหมาย
Bitwise NOT	~a	(ดูได้ในตารางค่าความจริง)
Bitwise AND	a & b	(ดูได้ในตารางค่าความจริง)
Bitwise OR	a | b	(ดูได้ในตารางค่าความจริง)
Bitwise XOR	a ^ b	(ดูได้ในตารางค่าความจริง)
Shift left	a << n	เลื่อนทุกบิตใน a ไปทางซ้าย n ตำแหน่ง เติมศูนย์เข้าทางขวา
Shift right with sign bit	a >> n	เลื่อนทุกบิตใน a ไปทางขวา n ตำแหน่ง เติมทางซ้ายด้วยบิตเครื่องหมายของ a

Shift right with zero fill	a >>> n	เลื่อนทุกบิตใน a ไปทางขวา n ตำแหน่ง เติมทางซ้ายด้วย 0

ทางซ้ายและทางขวาของเครื่องหมายต้องเป็น จำนวนเต็ม โดยถ้าเป็นค่าคงตัวแบบ 4 ไบต์, ตัวแปรแบบ `int`, `short` หรือ `byte` ให้มองเป็นเลขฐานสอง 32 บิต ส่วนค่าคงตัวแบบ 8 ไบต์หรือตัวแปรแบบ `long` ให้ มองเป็นเลขฐานสอง 64 บิต

สังเกตว่ามีการใช้เครื่องหมาย `&` และ `|` ซ้ำกับเครื่องหมายตรรก แต่ในกรณีของเครื่องหมายตรรก ทางซ้ายและทางขวาของเครื่องหมายจะเป็นสมการตรรกศาสตร์ ส่วนในกรณีของเครื่องหมายบิตจะเป็นจำนวนเต็ม

ตารางผลลัพธ์ของเครื่องหมายบิตเป็นดังนี้

ตารางที่ 5-8 ตารางผลลัพธ์ของเครื่องหมายบิต

x	y	~x	x & y	x | y	x ^ y
1	1	0	1	1	0
1	0	0	0	1	1
0	1	1	0	1	1
0	0	1	1	0	0

เครื่องหมายต่อไปนี้เป็นเครื่องหมายบิตที่มีไว้สำหรับการกำหนดค่าให้ตัวแปรโดยเฉพาะ

ตารางที่ 5-9 เครื่องหมายบิต

รูปแบบคำสั่ง	ความหมาย
b &= a	b = b & a
b |= a	b = b | a
b ^= a	b = b ^ a
x <<= a	x = x << a
x >>= a	x = x >> a
x >>>= a	x = x >>> a

สำหรับเครื่องหมายแบบบิตนี้ไม่ขอลงรายละเอียด เนื่องจากมีที่ใช้น้อย

6

บล๊อคเงื่อนไข

บล๊อคเงื่อนไขคือการสั่งให้จาวาเวอร์ชัวน์แมทชีนตัดสินใจภายใต้เงื่อนไขที่เรากำหนดขึ้น มีอยู่ด้วยกันสองคำสั่งคือ `if` และ `switch`

คำสั่ง `if`

ลองดูตัวอย่างคำสั่ง `if` ที่ง่ายที่สุดในโปรแกรมตัวอย่างข้างล่างนี้

โปรแกรม 6 - 1 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
    int i = 1;
    int j = 3;
        if (j > 0) i = 4;
          System.out.println("i = " + i);
        }
}
```

ในโปรแกรมนี้ตอนแรกเรากำหนดค่าให้ตัวแปร `i` เท่ากับ 1 และ `j` เท่ากับ 3 ต่อมาเราใช้คำสั่ง `if` ในการพิจารณาเงื่อนไขว่า ถ้า `j` มีค่ามากกว่า `0` ให้เปลี่ยนค่าของ `i` เป็น 4

เนื่องจาก j มีค่าเป็น 3 อยู่ก่อนแล้ว และ 3 มากกว่า 0 ดังนั้นเงื่อนไขนี้เป็นจริง คำสั่ง if จึงทำงาน เป็นผลให้ i เปลี่ยนค่าเป็น 4 เมื่อพิมพ์ผลลัพธ์ออกหน้าจอจะได้ค่าของ i เป็น 4

นั้นคือคำสั่ง if จะตามด้วยวงเล็บ สิ่งที่อยู่ในวงเล็บคือเงื่อนไข ซึ่งอาจอยู่ในรูปของเงื่อนไขทางตรรกซึ่งต้องมีค่าเป็นจริงหรือเท็จเท่านั้น ถ้าจริงจาวาเวอร์ชัวน์แมทชีนจะทำคำสั่งที่ตามหลังวงเล็บมา แต่ถ้าเป็นเท็จคำสั่งที่ตามหลังวงเล็บมาก็จะถูกข้ามไปเลย เช่นในกรณีของตัวอย่างข้างต้น ถ้าเป็นเท็จ i ก็จะยังคงเท่ากับ 1 เมื่อพิมพ์ออกหน้าจอ

ลองพิจารณาคำสั่ง if อีกรูปแบบหนึ่งในโปรแกรมข้างล่าง

โปรแกรม 6 - 2 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
      int i;
      int j = 1;
      if (j > 2)
        i = 4;
      else
        i = 1;
      System.out.println("i = " + i);
  }
}
```

คำสั่ง if ในโปรแกรมนี้ยาวขึ้นอีกนิดหนึ่ง คือมี else ต่อท้าย จาวาเวอร์ชัวน์แมทชีนจะพิจารณาเงื่อนไขในวงเล็บของคำสั่ง if ถ้าจริงจะทำคำสั่งที่อยู่ต่อท้ายวงเล็บ ถ้าเป็นเท็จแทนที่จะกระโดดข้ามไปเลยจะทำคำสั่งที่อยู่ท้ายคำว่า else แทนก่อนจะจากไปทำคำสั่งที่อยู่ถัดไป

ในโปรแกรมนี้ตอนแรกเราไม่ได้กำหนดค่าใดๆ ให้ i เมื่อโปรแกรมทำงานมาถึงคำสั่ง if และพบว่าเงื่อนไขที่อยู่ในวงเล็บเป็นเท็จ มันจะข้ามคำสั่ง i = 4 ไป แล้วทำคำสั่ง i = 1 แทน จากนั้นค่อยพิมพ์ออกหน้าจอซึ่งย่อมได้ i = 1

โปรดสังเกตว่าต้องมี ; อยู่หน้า else ด้วยทั้งๆ ที่ทั้งหมดเป็นคำสั่งเดียวกัน

บางครั้งเราต้องการให้ทำคำสั่งมากกว่าหนึ่งคำสั่ง เราสามารถใช้วงเล็บปีกกาในการครอบกลุ่มของคำสั่งได้ด้วย เราเรียกกลุ่มของคำสั่งที่ถูกครอบด้วยเครื่องหมายปีกกาว่า **บล๊อคปีกกา** ดังตัวอย่างต่อไปนี้

โปรแกรม 6 - 3 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
    int i, k, l ;
    int j = 1;
    if (j > 2) {
        i = 4;
        k = 5;
        l = 2;
   } else {
        i = 1;
        k = 1;
        l = 4;
      }
    System.out.println("i = " + i);
    System.out.println("k = " + k);
    System.out.println("l = " + l);
  }
}
```

ในกรณีนี้ถ้าเงื่อนไขในวงเล็บเป็นจริง จาวาเวอร์ชัวน์แมทชีนจะทำคำสั่งทั้งหมดที่อยู่ในบล๊อคปีกกาที่อยู่ตามหลังวงเล็บเงื่อนไข แต่ถ้าเป็นเท็จจะทำคำสั่งทั้งหมดที่อยู่ในบล๊อคปีกกาที่อยู่หลัง `else`

กรณีที่เราใช้บล๊อคปีกกาเราไม่จำเป็นต้องมี `;` หน้า `else` อีกต่อไป อีกทั้งเมื่อจบบล๊อคปีกกา เราสามารถละเครื่องหมาย `;` ได้ด้วย

ต่อไปนี้เป็นตัวอย่างของการใช้บล๊อคปีกกาผสมกับการใช้คำสั่งเดี่ยวๆ

โปรแกรม 6 - 4 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
    int i, k, l ;
    int j = 1;
    if (j > 2)
        i = 4;
    else {
        i = 1;
        k = 1;
```

```
            l = 4;
        }
        System.out.println("i = " + i);
      }
    }
```

แบบนี้คุณยังคงต้องมีเครื่องหมาย ; หน้า else เพราะ คำสั่งหลังวงเล็บเงื่อนไขเป็นคำสั่งเดี่ยว ลองพิจารณาอีกตัวอย่างหนึ่งซึ่งกลับกัน

โปรแกรม 6 - 5 : TestIf.java

```
    public class TestIf{
      public static void main(String[]args){
    int i, k, l ;
    int j = 1;
    if (j > 2) {
      i = 4;
      k = 5;
      l = 2;
    } else
      i = 1;
      System.out.println("i = " + i);
  }
}
```

กรณีนี้คำสั่งหลังวงเล็บเงื่อนไขเป็นบล๊อคปีกกา ส่วนคำสั่งหลัง else เป็นคำสั่งเดี่ยว ดังนั้นต้องมีเครื่องหมาย ; ตามหลังคำสั่งเดี่ยวด้วย

คำสั่ง if ทั้งที่มี else และไม่มี else สามารถซ้อนกันเป็นหลายๆ ชั้นได้ดังตัวอย่างต่อไปนี้

โปรแกรม 6 - 6 : TestIf.java

```
    public class TestIf{
      public static void main(String[]args){
        int i, j, k;
        i = j = k = 1;
            if (j > 0)
              if (k == 1) i = 1;
            else
              if (k == 2)
                i = 4;
              else
                i = 5;
        System.out.println("i = " + i);
      }
    }
```

การทำงานของโปรแกรมนี้จะเริ่มจากการประกาศและกำหนดค่าของตัวแปร `i, j` และ `k` ให้เท่ากับ 1 จากนั้นพิจารณาเงื่อนไขในวงเล็บคือ `j>0` หรือไม่ ถ้าใช่ จะทดสอบเงื่อนไขว่า `k == 1` หรือไม่ ถ้าใช่คำสั่ง `i = 1` ก็จะทำงาน แต่ถ้าไม่ใช่จะข้ามไปยัง `else` ตัวแรก ซึ่งเริ่มจากการตรวจสอบเงื่อนไข `k == 2` ถ้าจริงก็จะทำคำสั่ง `i = 4` แต่ถ้าไม่ก็จะทำคำสั่ง `i = 5` แทน ก่อนที่จะทำคำสั่งถัดไปคือ การพิมพ์ค่าของ `i` ออกหน้าจอ

ลองดูตัวอย่างอีกตัวอย่างหนึ่ง

โปรแกรม 6 - 7 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
    int i, j, k;
    i = j = k = 1;
   if (j > 0) {
       if (k == 1) i = 1;
       if (k == 2) i = 3;
   } else {
     if (k == 1) i = 5;
     if (k != 2)
       if (k >= 2) i = 7;
     else
       i = 4;
    }
    System.out.println("i = " + i);
  }
}
```

โปรแกรมนี้เป็นการใช้คำสั่ง `if` ที่มีการซ้อนกันหลายชั้นเหมือนกันแต่มีการใช้บล๊อคปีกกาด้วย คำสั่ง `i = 4` ตอนท้ายของโปรแกรมเป็น `else` ของคำสั่ง `if (k >= 2)` ไม่ใช่ `else` ของคำสั่ง `if (k != 2)` ดังนั้นต้องระวังให้ดีเนื่องจากการย่อหน้าแบบนี้ทำให้เกิดการเข้าใจผิด เราควรย่อหน้าแบบนี้แทน

โปรแกรม 6 - 8 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
    int i, j, k;
    i = j = k = 1;
    if (j > 0) {
```

```
            if (k == 1) i = 1;
            if (k == 2) i = 3;
        } else {
            if (k == 1) i = 5;
            if (k != 2)
              if (k >= 2)
                    i = 7;
              else
                    i = 4;
        }
        System.out.println("i = " + i);
    }
}
```

การตัดสินว่า else หนึ่งๆ เป็นของ if ตัวไหน มีหลักง่ายๆ คือ มันจะเป็นของ if ตัวที่อยู่ใกล้ที่สุดเสมอ

เคล็ดลับ วิธีการที่ดีที่สุดในการป้องกันความสับสนของคำสั่ง if คือ การใช้บล๊อคปีกกาเสมอแม้ว่าจะมีคำสั่งแค่คำสั่งเดียวในเงื่อนไข โปรแกรมจะอ่านง่ายขึ้นมาก

สิ่งสำคัญอีกอย่างหนึ่งเกี่ยวกับการใช้บล๊อคปีกกาคือ ถ้ามีการประกาศตัวแปรภายในบล๊อคปีกกาตัวแปรนั้นจะถูกเรียกใช้ได้เฉพาะภายในบล๊อคปีกกาเท่านั้น เช่น

โปรแกรม 6 - 9 : TestIf.java

```
public class TestIf{
  public static void main(String[]args){
      int j = 1;
      if (j == 1) {
        int i = 2;
      }
      System.out.println("i = " + i); // (1) Error
  }
}
```

โปรแกรมนี้คอมไพล์ไม่ผ่านเพราะตัวแปร i ถูกประกาศไว้ภายใต้บล๊อคปีกกาของคำสั่ง if ไม่สามารถถูกกล่าวถึงนอกบล๊อคได้อย่างในบรรทัด (1)

และเนื่องจากตัวแปรที่ประกาศภายในบล๊อคปีกกาอยู่แต่ในบล๊อคปีกกาเท่านั้นเราจึงประกาศตัวแปรใหม่ที่มีชื่อซ้ำกับตัวแปรในบล๊อคปีกกาได้หลังจากบล๊อคปีกการจบไปแล้ว เช่น

โปรแกรม 6 - 10 : TestIf.java

```
          public class TestIf{
            public static void main(String[]args){
              int j = 1;
                //int i = 1; //(1) Error
                if (j == 1) {
                  int i = 2;  // (2)
                }
                int i = 1;   // (3)
            }
}
```

ถ้าคำสั่งในบรรทัด (1) ไม่ถูกกันไว้ด้วยเครื่องหมาย // โปรแกรมจะคอมไพล์ไม่ผ่านเพราะตัวแปร i ที่สร้างขึ้นในบรรทัด (1) จะคงอยู่ตลอดทั้งโปรแกรม เมื่อประกาศตัวแปรชื่อเดิมอีกภายในบล๊อคปีกกาในบรรทัด (2) จะเกิดความซ้ำซ้อนขึ้น ในทางตรงกันข้ามหากเราเอาคำสั่งในบรรทัด (1) ออก คำสั่งในบรรทัด (2) จะไม่มีปัญหาและจะไม่ขัดแย้งกับคำสั่งในบรรทัด (3) ด้วยเพราะตัวแปร i ในบรรทัด (2) จะหายไปทันทีที่จบบล๊อค if จึงประกาศตัวแปร i ตัวใหม่ได้ในบรรทัด (3)

คำสั่ง switch

ถ้าในกรณีที่เราต้องการให้จาวาเวอร์ชัวน์แมทชีนตัดสินใจว่าตัวแปรตัวหนึ่งมีค่าเท่าไร แล้วค่าที่เป็นไปได้มีหลายค่า เราอาจใช้คำสั่ง if ซ้อนกันไปเรื่อยๆ ดังนี้

โปรแกรม 6 - 10 : TestIf.java

```
        public class TestIf{
                public static void main(String[]args){
                        int i;
                        int j = 3;
                        if (j == 1)
                                i = 1 ;
                        else
                                if (j == 2)
                                        i = 2;
                                else
                                        if (j == 3)
                                                i = 3;
                                        else
                                                if (j == 4)
                                                        i = 4;
                                                else
```

```
                                                i = 5;
                System.out.println (“i = “ + i);
        }
}
```

แต่คำสั่งจะยาวและดูยาก เราสามารถใช้คำสั่ง switch ซึ่งออกแบบมาเพื่อการนี้โดยเฉพาะ

โปรแกรม 6 - 11 : TestSwitch.java

```
public class TestSwitch{
  public static void main(String[]args){
    int i;
    int j = 3;
    switch (j) {
        case (1) : i = 1 ; break;
        case (2) : i = 2 ; break;
        case (3) : i = 3 ; break;
        case (4) : i = 4 ; break;
        default : System.out.println(“i = “ + i);
    }
  }
}
```

สิ่งที่อยู่ในวงเล็บหลังคำสั่ง switch จะเป็นตัวแปร int, short, byte หรือ char ก็ได้ โดยจาวาเวอร์ชัวน์แมทชีนจะทำการเทียบค่าของตัวแปรนั้นกับค่าที่อยู่ในวงเล็บหลังคำสั่ง case ถ้าตรงจะ ทำคำสั่งหลังเครื่องหมาย : ถ้าไม่ตรงจะ ข้ามไป เป็นเช่นนี้เรื่อยไปจนมาถึงคำว่า default จะทำคำสั่งที่อยู่หลัง default เสมอไม่ว่าที่ผ่านมาจะทำคำสั่ง case หรือไม่ ซึ่งจริงๆ แล้วบล๊อค default ไม่จำเป็นต้องมีก็ได้ถ้าไม่จำเป็น

คำสั่งที่อยู่หลังเครื่องหมาย : ของคำสั่ง case อาจมีมากกว่าหนึ่งคำสั่งก็ได้ โดยเขียนเรียงกันไปเรื่อยๆ โดยมีเครื่องหมาย ; คั่นระหว่างคำสั่งแต่ละคำสั่ง แต่สุดท้ายต้องจบด้วยคำสั่ง break เสมอ

ลองพิจารณาโปรแกรมนี้

โปรแกรม 6 - 12 **:** TestSwitch.java

```
public class TestSwitch{
  public static void main(String[]args){
    int i;
```

```
        int j = 3;
        switch (j) {
       case (1) : i = 1 ;
       case (2) : i = 2 ;
       case (3) : i = 3 ;
       case (4) : i = 4 ;
         default : System.out.println (“i = “ + i);
        }
      }
    }
```

โปรแกรมนี้ตัดคำสั่ง break ออกไป ผลที่ได้ก็คือจาวาเวอร์ชัวน์แมทชีนจะพิมพ์เลข 4 ออกมาแทนที่จะเป็นเลข 3 เหมือนเดิม เนื่องจากถ้าไม่มีเครื่องหมาย break เมื่อใดที่พบว่า j มีค่าเท่ากับ case ใดก็ตาม case ที่เหลือทั้งหมดที่ตามมาจะทำงานด้วยทันที ในกรณีนี้เมื่อมาถึง case (3) ซึ่งตรงกับค่าของ j ตัวแปร i จะถูกกำหนดค่าให้เป็น 3 แต่โปรแกรมจะยังไม่ออกจากบล๊อค switch แต่โปรแกรมจะทำ case อื่นที่เหลือถัดไปทั้งหมด ทำให้ตัวแปร i ถูกกำหนดค่าให้เป็น 4 ทับค่าเดิมอีกที โปรแกรมจึงพิมพ์ว่า i = 4 ในตอนสุดท้าย

เครื่องหมาย ? :

ก่อนจะจบคำสั่งเงื่อนไข ขอแนะนำให้รู้จักกับเครื่องหมายตัวหนึ่งในภาษาจาวาซึ่งมีลักษณะคล้ายคำสั่งเงื่อนไข คือ เครื่องหมาย ? : ตัวอย่างการใช้งานเป็นดังนี้

```
boolean b = false;
int a = b ? 10 : 20;
```

เครื่องหมาย ? : มีไว้ใช้ในการกำหนดค่าตัวแปรเป็นหลัก ตัวแปร a ในตัวอย่าง เป็นตัวแปร int แต่ถูกจับให้มีค่าเท่ากับ b ? 10 : 20 ค่าของวลี b ? 10 : 20 จะมีค่าเท่ากับ 10 เมื่อ b เป็นจริง และจะมีค่าเท่ากับ 20 เมื่อ b เป็นเท็จ ในตัวอย่าง b มีค่าเป็นเท็จอยู่ก่อนแล้ว ดังนั้น a จึงมีค่าเป็น 20

เครื่องหมาย ? : สามารถซ้อนกันหลายๆ ชั้นได้ด้วย ตัวอย่างเช่น

```
boolean b = false;
boolean c = true;
boolean d = false;
char x = b?c?d?’m’:’n’:’o’:’p’;
```

วลี b?c?d?'m':'n':'o':'p' เทียบได้กับ (b?(c?(d?'m':'n'):'o'):'p') นั้นคือ จาวาเวอร์ชัวน์แมทชีนจะเริ่มพิจารณาจากวงเล็บในสุดก่อน d มีค่าเป็นเท็จ ดังนั้นวงเล็บในสุดมีค่าเทียบเท่า 'n' ในวงเล็บชั้นถัดมา c มีค่าเป็นจริง ดังนั้นจะมีค่าเทียบเท่า 'n' อีก ส่วนสุดท้ายในวงเล็บนอกสุด b มีค่าเป็นเท็จ ดังนั้นจึงมีค่าเทียบเท่า 'p' นั้นคือ x จะมีค่าเท่ากับ 'p'

7

บล๊อควนลูป

สิ่งหนึ่งที่คอมพิวเตอร์ทำได้ดีกว่ามนุษย์คือ การทำงานเก่าซ้ำๆ ซากๆ เป็นร้อยเป็นพันครั้ง คอมพิวเตอร์ไม่เคยเหนื่อย ไม่เคยเบื่อ และ ไม่เคยบ่น

บล๊อควนลูปคือคำสั่งที่ทำให้จาวาเวอร์ชัวน์แมทชีนทำคำสั่งใดคำสั่งหนึ่งหรือบล๊อคปีกกาซ้ำๆ กันหลายๆ ครั้ง คำสั่งวนลูปในภาษาจาวาได้แก่คำสั่ง `while` และคำสั่ง `for`

คำสั่ง `while`

ลองพิจารณาโปรแกรมที่มีคำสั่ง `while` อย่างง่ายอยู่ดังนี้

โปรแกรม 7 - 1 : TestWhile.java

```
public class TestWhile{
  public static void main(String[]args){
    while (true) System.out.println("Hello World");
  }
}
```

โปรแกรมนี้จะพิมพ์คำว่า Hello World ออกหน้าจอไปเรื่อยๆ เป็นร้อยเป็นพันบรรทัด ไม่มีวันหยุด คำสั่ง while ที่อยู่ข้างหน้าคำสั่งเขียนข้อความออกหน้าจอเป็นการบอกให้โปรแกรมทำคำสั่งในบรรทัดนี้ซ้ำแล้วซ้ำเล่า

โปรแกรมจะทำคำสั่งในบรรทัด while ไปเรื่อยๆ ตราบใดที่ค่าความจริงในวงเล็บที่ตามหลังคำสั่ง while มีค่าเป็นจริง ในกรณีนี้สิ่งที่อยู่ในวงเล็บคือค่าคงตัว true ซึ่งมีค่าเป็นจริงเสมอ โปรแกรมจึงพิมพ์ข้อความ Hello World ซ้ำแล้วซ้ำเล่าไม่สิ้นสุด การทำงานซ้ำๆ โดยที่เราไม่ต้องเขียนคำสั่งเดิมหลายๆ ครั้งในโปรแกรมเดียวกันทำให้ชีวิตง่ายขึ้น แต่การทำซ้ำๆ อย่างไม่มีวันหยุดคงไม่ค่อยดีเท่าไรนัก เราสามารถกำหนดให้คำสั่ง while ทำคำสั่งที่ตามมาซ้ำกี่ครั้งก็ได้ตามใจเราด้วยการสร้างเงื่อนไขในวงเล็บที่ตามมาให้มีค่าเปลี่ยนจากจริงเป็นเท็จภายหลังได้ดังตัวอย่างต่อไปนี้

โปรแกรม 7 - 2 : TestWhile.java

```
public class TestWhile{
  public static void main(String[]args){
    int i = 1;
    while (i < 100) i++; //(1)
    System.out.println("i = " + i);
  }
}
```

เมื่อจาวาเวอร์ชัวน์แมทชีนพบคำสั่ง while มันจะตรวจสอบเงื่อนไขในวงเล็บที่ตามมา ซึ่งในกรณีนี้ก็คือ i < 1000 ถ้าเงื่อนไขเป็นจริง มันจะทำคำสั่งที่อยู่หลังวงเล็บเงื่อนไข เมื่อเสร็จแล้วมันจะตรวจเงื่อนไขเดิมซ้ำอีกครั้ง ถ้าเงื่อนไขยังเป็นจริงอยู่ มันจะทำคำสั่งที่อยู่หลังวงเล็บเงื่อนไขอีกเช่นนี้เรื่อยไปจนกว่าจะพบว่าเงื่อนไขมีค่าเป็นเท็จ จึงจะหลุดจากลูป

ในกรณีนี้เมื่อโปรแกรมทำงานมาถึงบรรทัด (1) โปรแกรมจะบวกค่าของ i ด้วยหนึ่ง ไปเรื่อยๆ จนกว่า i จะมีค่าเท่ากับ 100 ซึ่งทำให้เงื่อนไขในวงเล็บที่อยู่หลังคำสั่ง while เป็นเท็จ จึงหลุดออกจากลูป แล้วทำคำสั่งในบรรทัดต่อไปซึ่งจะพิมพ์ค่า i = 100 ออกมาที่หน้าจอ

```
C:\java> java TestWhile
```

```
i=100
```

คำสั่งที่ตามหลัง while มา อาจเป็นบล๊อคปีกกาก็ได้ เช่น

โปรแกรม 7 - 3 : TestWhile.java

```
public class TestWhile{
  public static void main(String[]args){
    int i = 1;
    while (i < 100) {
        i++;
        System.out.println("i = " + i);
    }
    System.out.println("Finished");
  }
}
```

คำสั่งทุกคำสั่งในบล๊อคปีกกาหลังคำสั่ง while จะถูกทำงานซ้ำจนกว่าเงื่อนไขในวงเล็บหลังคำสั่ง while จะเป็นเท็จ ในกรณีนี้โปรแกรมจะบวกหนึ่งให้กับ i และพิมพ์ค่า i ออกมาหน้าจอทุกครั้งหลังการบวก ทำอย่างนี้เรื่อยไปหนึ่งร้อยครั้ง แล้วปิดท้ายด้วยคำว่า Finished ครั้งเดียวตอนจบโปรแกรม

คำสั่ง while ยังมีอีกรูปแบบหนึ่งที่เราเรียกว่าคำสั่ง do-while ดังตัวอย่างต่อไปนี้

โปรแกรม 7 - 4 : TestWhile.java

```
public class TestWhile{
  public static void main(String[]args){
    int i = 1;
    do {
        i++;
        System.out.println("i = " + i);
    } while (i < 100);
        System.out.println("Finished");
  }
}
```

โปรแกรมนี้ให้ผลเหมือนโปรแกรม 7-3 เพียงแต่ข้อแตกต่างอยู่ตรงที่เวลาตรวจสอบเงื่อนไขจาวาเวอร์ชัวน์แมทชีนจะทำคำสั่งที่อยู่ในลูปก่อนแล้วค่อยตรวจสอบเงื่อนไขในวงเล็บที่อยู่หลังคำสั่ง while ดังนั้นคำสั่งที่อยู่ในลูปจะต้องถูกใช้งานอย่างน้อยหนึ่งครั้งเสมอคือครั้งแรก

สุดไม่ว่าเงื่อนไขที่อยู่ในวงเล็บจะเป็นอย่างไร ในขณะที่การใช้คำสั่ง while แบบไม่มี do คำสั่งในลูปอาจไม่เคยถูกใช้งานเลยถ้าเงื่อนไขไม่เป็นจริงตั้งแต่แรก

ถ้ามีการประกาศตัวแปรภายในบล๊อค while เราจะอ้างถึงตัวแปรนั้นได้อีกเฉพาะภายในบล๊อคเท่านั้นเช่นเดียวกับคำสั่ง if ตัวแปรจะหมดสภาพทันทีที่โปรแกรมหลุดจากบล๊อค

คำสั่ง for

คำสั่งเป็นคำสั่งวนลูปอีกประเภทหนึ่งซึ่งมีรูปแบบคำสั่งดังตัวอย่างต่อไปนี้

โปรแกรม 7 - 5 : TestFor.java

```
        public class TestFor{
          public static void main(String[]args){
            for (int i = 1; i <= 100; i++)
        System.out.println("i = " + i);
        System.out.println("Finished");
    }
}
```

ภายในวงเล็บที่ตามหลังคำสั่ง for จะแบ่งเป็นสามส่วนและคั่นด้วยเครื่องหมาย ; ส่วนที่หนึ่งเป็นส่วนที่ใช้ประกาศและกำหนดค่าตัวแปรจำนวนเต็ม ที่เราจะใช้เป็นตัวนับจำนวนครั้งในการวนลูป ส่วนที่สองเป็นเงื่อนไขที่ให้ตรวจสอบ ตราบใดที่เงื่อนไขยังเป็นจริงอยู่ให้โปรแกรมทำการวนลูปต่อไปอีก และส่วนที่สามคือส่วนที่ใช้เปลี่ยนค่าของตัวแปรที่ใช้นับจำนวนครั้ง ทุกครั้งที่ทำครบหนึ่งลูป ดังในตัวอย่างเราประกาศตัวแปร i และกำหนดค่าเริ่มต้นให้เท่ากับ 1 จากนั้นโปรแกรมจะตรวจสอบเงื่อนไขในส่วนที่สอง ถ้าเป็นจริงก็จะทำคำสั่งที่อยู่ในลูป แล้วเพิ่มค่าของ i อีกหนึ่ง ตรวจสอบเงื่อนไขใหม่ ทำไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นเท็จค่อยออกจากลูป ซึ่งในกรณีนี้จะมีการวนลูปทั้งสิ้น 100 ครั้ง และตัวแปร i มีค่าเริ่มจาก 1 จนถึง 100 ในการวนลูปครั้งสุดท้าย หลังจากนั้นเงื่อนไขจะเป็นเท็จแล้ว เพราะ i จะมีค่าเป็น 101 ซึ่งมากกว่า 100

ถ้าต้องการวนลูปโดยให้ตัวแปรนับจำนวนลูปมีค่าลดลงก็ทำได้เช่นกัน ตัวอย่างเช่น

```
for (int i = 100; i >= 1; i--)
```

ในกรณีนี้ตัวแปร `i` จะเริ่มต้นจาก 100 และจะลดลงเรื่อยๆ ทีละหนึ่งต่อการวนลูปหนึ่งครั้ง จนมีค่าเท่ากับ 1 ในการวนลูปครั้งที่ 100 ซึ่งเป็นครั้งสุดท้าย

ตัวแปรนับจำนวนลูปในคำสั่ง `for` จะเกิดขึ้นเมื่อโปรแกรมเริ่มวนลูป และจะหายไปเมื่อสิ้นสุดการวนลูป ดังนั้นเราไม่สามารถใช้ตัวแปรนับจำนวนลูปนอกลูป `for` ได้ ตัวอย่างเช่น

โปรแกรม 7 - 6 : TestFor.java

```
public class TestFor{
  public static void main(String[]args){

          for (int i = 1;i >= 100; i++)
          System.out.println("Hello World");
          System.out.println("i = " + i);   //  Error
        }
      }
```

โปรแกรมคอมไพล์ไม่ผ่านเนื่องจากมันไม่รู้จักตัวแปรชื่อ `i` ที่อยู่นอกลูป `for` และในกรณีที่มีตัวแปรชื่อ `i` อยู่ก่อนแล้วในโปรแกรม เราไม่อาจใช้ชื่อ `i` ได้อีกในการตั้งชื่อตัวแปรนับจำนวนลูป `for` เพราะตัวแปร `i` ที่อยู่นอกลูปจะคงอยู่ตลอดทั้งโปรแกรม

โปรแกรม 7 - 7 : TestFor.java

```
public class TestFor{
  public static void main(String[]args){
    int i = 50;
    for (int i = 1;i >= 100; i++) //  Error
    System.out.println("Hello World");
  }
}
```

การประกาศตัวแปรนับจำนวนลูปในคำสั่ง `for` สามารถทำได้มากกว่าหนึ่งตัวโดยคั่นด้วยเครื่องหมายจุลภาค และจะนำมาใช้หรือไม่ก็ได้ ตัวอย่างเช่น

```
for (int i = 1, j= 2; i < 100; i++)
```

ถ้าทุกส่วนในวงเล็บเป็นเพียงความว่างเปล่า โปรแกรมจะวนลูป `for` โดยไม่มีวันสิ้นสุดเช่น

โปรแกรม 7 - 8 : TestFor.java

```
public class TestFor{
  public static void main(String[]args){
```

```
        for (;;)
      System.out.println("Hello World");
  }
}
```

เคล็ดลับ ถ้าคุณอยากรู้ว่ารหัสยูนิโค้ดสากลแต่ละตัวมีค่าเป็นเท่าไรบ้าง คุณสามารถดูได้จากการรันโปรแกรมข้างล่างนี้

```
public class ShowUnicode{
  public static void main(String[]args){
   char c = 0;
   for (int i =0; i <128 ; i++) {
    c = (char) i;
    System.out.println(i + '=' + c);
   }
  }
}
```

อย่าลืมตั้งชื่อซอร์สโค้ดว่า `ShowUnicode.java`

ที่จริงแล้วรหัสยูนิโค้ดมีเป็นหมื่นๆ ตัวอักขระ แต่ในโปรแกรมนี้เราให้แสดงแค่ 128 อักขระแรกเท่านั้น เพราะรหัสตัวหลังๆ เป็นภาษานานาชาติซึ่งคอมพิวเตอร์ของคุณอาจแสดงผลออกหน้าจอไม่ได้ อย่าลืมแคสตัวแปร `i` ด้วย เพราะ `i` เป็นตัวแปร คอมไพล์เลอร์ไม่รู้ว่าค่าของมันจะเกินค่าที่ `c` รับได้หรือไม่

คำสั่ง break

เราสามารถใช้คำสั่ง `break` ในการบอกให้โปรแกรมออกจากลูปกะทันหันได้ซึ่งใช้ได้ทั้งลูป `while` และ `for` ตัวอย่างเช่น

โปรแกรม 7 - 9 : TestBreak.java

```
public class TestBreak{
  public static void main(String[]args){
   int i = 1;
   while (i <= 10)
   System.out.println(i + " :Hello World");
   if (i == 4) break;
   i++;
  }
}
```

แทนที่โปรแกรมจะพิมพ์คำว่า Hello World ออกหน้าจอ 10 ครั้ง มันจะพิมพ์เพียง 4 ครั้ง เพราะเมื่อวนลูปครั้งที่สี่ เงื่อนไขในวงเล็บหลังคำสั่ง if จะมีค่าเป็นจริง เป็นผลทำให้คำสั่ง break ทำงาน ซึ่งก็คือโปรแกรมจะกระโดดออกจากลูป while กะทันหัน โดยไม่สนใจคำสั่งที่ตามหลังคำสั่ง break ในลูป

คำสั่ง continue

คำสั่ง continue คล้ายกับคำสั่ง break เพียงแต่แทนที่จะหนีออกจากลูปอย่างถาวร มันจะทิ้งคำสั่งในลูปที่ตามหลังมันมาเพียงครั้งเดียว แล้ววนลูปครั้งใหม่โดยเริ่มที่ต้นลูปใหม่ ตราบใดที่เงื่อนไขของลูปยังเป็นจริง ตัวอย่างเช่น

โปรแกรม 7 - 10 : TestContinue.java

```
public class TestContinue {
  public static void main(String[] args){
    for (int i = 1; i <= 10; i++) {
      if (i == 4) continue;
      System.out.println(i +“ : Hello World”);
    }
  }
}
```

โปรแกรมนี้จะพิมพ์คำว่า Hello World แค่เก้าครั้ง โดยจะข้ามครั้งที่สี่ไป เพราะในการวนลูปครั้งที่สี่ เงื่อนไขหลังคำสั่ง if จะเป็นจริง ทำให้โปรแกรมกระโดดข้ามคำสั่งที่ตามมาในลูปในครั้งที่สี่แล้วกลับมาวนลูปครั้งที่ห้าต่อไปโดยเริ่มจากต้นลูป โปรแกรมจึงไม่พิมพ์คำว่า Hello World ในครั้งที่สี่เท่านั้น

8

คลาสและวัตถุ

ที่ผ่านมาเราเขียนโปรแกรมด้วยการสร้างคลาสขึ้นมาหนึ่งคลาสชื่อเดียวกับชื่อซอร์สโค้ด ภายในคลาสมีแมธธอสหนึ่งแมธธอสชื่อ `main` และถ้าเราต้องการให้โปรแกรมทำอะไรเราก็เอาคำสั่งเหล่านั้นใส่เข้าไปในส่วนตัวของแมธธอส `main` โปรแกรมจะเริ่มรันจากคำสั่งแรกไปเรื่อยๆ จนจบคำสั่งสุดท้าย โปรแกรมที่มีความซับซ้อนมากขึ้นจะมีคลาสมากกว่าหนึ่งคลาส และคลาสเหล่านั้นไม่ต้องมีชื่อเหมือนชื่อของไฟล์ซอร์สโค้ด แต่ละคลาสก็อาจมีแมธธอสมากกว่าหนึ่งแมธธอสได้ด้วย

จาวาเป็นภาษาเชิงวัตถุ ทุกสิ่งทุกอย่างภาษาจาวาจึงมีลักษณะเป็นวัตถุ สิ่งที่เราใช้ในการนิยามวัตถุเรียกว่า คลาส ซอร์สโค้ดภาษาจาวา คือ คลาสตั้งแต่หนึ่งคลาสขึ้นไปเขียนรวมๆ กันอยู่ในไฟล์ๆ เดียวนั้นเอง

หลักการของวัตถุและคลาสอาจจะเป็นเรื่องที่เข้าใจยากสักหน่อย เนื่องจากเราเคยชินกับการเขียนโปรแกรมแบบเก่าคือการเขียนโปรแกรมแบบโครงสร้าง ซึ่งโปรแกรมจะเริ่มทำงานจากคำสั่งแรกสุดในโปรแกรม เรียงต่อไปเรื่อยจนจบโปรแกรม ในบทที่ผ่านมาโปรแกรมของเราก็มีลักษณะแบบนั้น แต่ภาษาเชิงวัตถุจริงๆ ไม่ได้มีแค่นั้น โปรแกรมภาษาเชิงวัตถุไม่ได้ทำงานเรียงบรรทัด ต้องใช้เวลาพอสมควรในการทำความคุ้นเคยกับการเขียนโปรแกรมในลักษณะนี้

ในบทนี้เราจะปูพื้นฐานเกี่ยวกับคลาสและวัตถุในภาษาจาวา อันเป็นพื้นฐานที่สำคัญมากสำหรับการเรียนรู้ภาษาจาวาในระดับต่อไป เพราะทุกสิ่งทุกอย่างในภาษาจาวาเป็นวัตถุ

คลาส

คลาส คือ นิยามของวัตถุ ดังที่ได้เกริ่นนำไปแล้วว่าทุกสิ่งทุกอย่างในภาษาจาวาเป็นวัตถุ ดังนั้นการเขียนโปรแกรมภาษาจาวาจริงๆ แล้วก็คือการเขียนคลาสหลายคลาสต่อๆ กันไปเรื่อยๆ นั้นเอง จาวาเวอร์ชัวน์แมทชีนจะอ่านนิยามของวัตถุที่อยู่ในคลาสแล้วสร้างวัตถุขึ้นมาเพื่อทำให้ได้ผลลัพธ์เป็นสิ่งที่ผู้เขียนโปรแกรมต้องการ

การเขียนโปรแกรมเชิงวัตถุโดยทั่วไปจะเริ่มจากการออกแบบวัตถุก่อน ซึ่งส่วนใหญ่แล้วเรามักสร้างวัตถุเลียนแบบวัตถุที่มีอยู่จริงๆ ในโลก ความยากอยู่ที่ว่าทำอย่างไรวัตถุที่เราออกแบบนั้นจะทำให้โปรแกรมของเราทำงานอย่างที่เราต้องการได้ ถ้าสังเกตดูให้ดีจะเห็นว่าโปรแกรมคอมพิวเตอร์ที่เราใช้กันอยู่ทุกวันนี้มีการสร้างวัตถุซึ่งคล้ายกับวัตถุที่มีอยู่จริงๆ ในโลกทั้งสิ้น ตัวอย่างที่เห็นได้ชัดที่สุดก็คือ การที่เราสั่งงานซอร์ฟแวร์ด้วยการใช้เมาส์คลิกปุ่มต่างๆ ที่อยู่บนหน้าจอมอนิเตอร์ ปุ่มเหล่านั้นเป็นเพียงภาพไม่ใช่ปุ่มจริงๆ แต่นักเขียนโปรแกรมใช้มันแทนปุ่มจริงๆ เพราะเป็นวิธีการสื่อสารกับผู้ใช้ที่ง่ายที่สุด ผู้ใช้เห็นปุ่มก็เข้าใจได้ทันทีว่ามีไว้ใช้ บังคับการ

รูปที่ 8-1 ปุ่มในโปรแกรมต่างๆ

ปุ่มในซอร์ฟแวร์คือวัตถุ และคลาสก็คือสิ่งที่นักเขียนโปรแกรมเขียนขึ้นเพื่อสาธยายให้คอมพิวเตอร์เข้าใจว่า ปุ่ม คืออะไร มีหน้าตาอย่างไร ทำอะไรได้บ้าง ในโปรแกรมเชิงวัตถุ

ทุกสิ่งทุกอย่างถูกสร้างขึ้นเลียนแบบวัตถุที่มีอยู่จริงทั้งสิ้น ปุ่มเป็นเพียงตัวอย่างตัวอย่างหนึ่งเท่านั้น

การสร้างวัตถุในโปรแกรมเชิงวัตถุ ทำได้ด้วยการนิยามวัตถุนั้นหรือการเขียนคลาสนั้นเอง รายละเอียดของคลาสคือส่วนที่บอกว่า วัตถุที่เราออกแบบต้องมีคุณสมบัติ หรือพฤติกรรมอะไรบ้าง

เพื่อให้เข้าใจมากขึ้นว่าคลาสมีไว้ทำอะไร ลองนึกถึงคำว่า รถยนต์ รถยนต์ที่เป็นรถยนต์จริงๆ (ขอให้คุณลืมโลกของคอมพิวเตอร์ไปสักพักหนึ่งก่อน) ทำไมเวลาเราสื่อสารกับคนอื่นพอเราพูดคำว่า รถยนต์ ทุกคนก็เข้าใจเป็นอย่างเดียวกันว่า รถยนต์ หมายถึงอะไร การที่ทุกคนเข้าใจเป็นเพราะมีอะไรบางอย่างที่ใช้เป็นเครื่องตัดสินว่าสิ่งไหนที่เรียกว่ารถยนต์ และเครื่องตัดสินเหล่านี้เป็นที่เข้าใจตรงกัน และวัตถุชิ้นไหนไม่มีสิ่งเหล่านี้อยู่ก็ไม่จัดเป็นรถยนต์ ตัวอย่างเช่น ทุกคนถือว่ารถยนต์ต้องมีล้อ ต้องมีเครื่อง และต้องวิ่งได้

รูปที่ 8-2 รถยนต์

รถยนต์เป็นวัตถุ และการสร้างคลาสรถยนต์คือการบอกว่าวัตถุที่จัดว่าเป็นรถยนต์ได้จะต้องมีคุณลักษณะอะไรบ้าง แม้ว่ารถยนต์จะมีรูปแบบหลากหลาย บางคันเล็ก บางคันใหญ่ บางคันมีสีแดง บางคันมีสีขาว บางคันวิ่งเร็ว บางคันวิ่งช้า แต่รถยนต์ทุกคันต้องมีลักษณะร่วมกันบางประการ ที่แน่ๆ ก็คือ มีล้อ มีเครื่อง และ วิ่งได้

ถ้าสังเกตให้ดีจะเห็นว่าสิ่งที่ใช้บอกความเป็นรถยนต์หรือวัตถุใดๆ จะมีอยู่สองลักษณะคือ บอกว่าวัตถุต้องมีอะไร (ในกรณีของรถยนต์ก็คือ ล้อ และเครื่อง) และบอกว่าวัตถุต้องทำอะไรได้ (ในกรณีของรถยนต์ได้แก่ วิ่งได้)

เราเรียกสิ่งที่วัตถุหนึ่งๆ ต้อง “มี” ว่า **คุณสมบัติของวัตถุ** และเรียกสิ่งที่วัตถุหนึ่งๆ ต้อง ”ทำ” ได้ว่า **พฤติกรรมของวัตถุ** ทั้งคุณสมบัติของวัตถุและพฤติกรรมของวัตถุคือสิ่งที่

ประกอบขึ้นเป็นคลาสนั้นเอง คุณสมบัติของวัตถุในภาษาจาวาแทนด้วย **ตัวแปรคลาส** ส่วนพฤติกรรมของวัตถุในภาษาแทนด้วย**แมธธอส**

การนิยามคลาสจึงได้แก่การสร้าง ตัวแปรคลาส และ แมธธอส นั้นเอง บางทีเราอาจกล่าวว่าคลาสประกอบด้วย ตัวแปรคลาส และแมธธอส หรือกล่าวว่า ตัวแปรคลาส และแมธธอส เป็น**สมาชิกของคลาส**

ตัวอย่างในการเขียนคลาสของรถยนต์ในภาษาจาวาอาจมีรูปแบบดังนี้

โปรแกรม 8 - 1 : Vehicle.java

```
class Vehicle {                                          // (1)

        int numberOfWheels;                              // (2)
        boolean hasEngine;                               // (3)

        void run(){                                              // (4)
                System.out.println("I am running");   // (5)
        }                                                        // (6)

}                                                                // (7)
```

โปรแกรม `Vehicle.java` คอมไพล์ได้แต่รันไม่ได้ ตอนนี้ยังไม่ขออธิบายว่าทำไมจึงรันไม่ได้ ขอให้สนใจแต่เรื่องรูปแบบของคำสั่งในการสร้างคลาสก่อน

การสร้างคลาสเริ่มต้นด้วยคำสั่งว่า `class` ตามด้วยชื่อของคลาสซึ่งจะตั้งชื่ออะไรก็ได้ แต่ต้องเป็นไปตามกฎของการตั้งชื่อในภาษาจาวาดังที่ได้เคยอธิบายไปแล้วตอนที่เราตั้งชื่อของตัวแปร

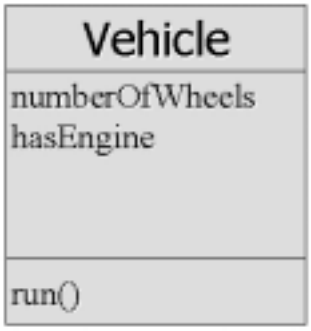
ชื่อของคลาสต้องขึ้นต้นด้วยตัวพิมพ์ใหญ่เสมอ ถ้าประกอบด้วยคำหลายคำต้องเขียนติดกันให้เป็นคำเดียวและนิยมใช้ตัวพิมพ์ใหญ่ขึ้นต้นทุกๆ ตัวอย่างเช่นชื่อต่อไปนี้เป็นชื่อคลาสที่ตั้งถูกหลักภาษาจาวาและถูกต้องตามความนิยม

`Lamp, Calculator, SwimmingPool, PetShopBoys`

สิ่งที่อยู่ตามมาในคลาสจะถูกครอบด้วยวงเล็บปีกกา วงเล็บเปิดอยู่ในบรรทัดเดียวกันกับชื่อของคลาส `(1)` ส่วนวงเล็บปิดอยู่ในบรรทัด (7) เรานิยมวางวงเล็บไว้ในลักษณะนี้เพื่อความ

สะดวกในการมอง แต่ความจริงแล้วการขึ้นบรรทัดใหม่ไม่มีผลต่อการคอมไพล์โปรแกรมแต่อย่างใด

สิ่งที่อยู่ระหว่างบรรทัด (1)และบรรทัด (7) คือรายละเอียดของคลาสซึ่งได้แก่การนิยาม ตัวแปรคลาส และ แมธธอส นั้นเอง เรานิยมย่อหน้าเนื้อหาทั้งหมดในคลาสให้หลบเข้าไปหนึ่งขั้นเพื่อความสะดวกในการมอง คลาสรถยนต์ประกอบด้วยตัวแปรคลาสสองตัว และแมธธอสหนึ่งแมธธอส ดังในภาพ



รูปที่ 8-3 คลาสรถยนต์

บรรทัด (2) คือการประกาศตัวแปรคลาสชื่อ numberOfWheels ซึ่งเป็นตัวแปรแบบ int ตัวแปร numberOfWheels จะมีไว้เก็บจำนวนล้อของรถ เพราะเราแน่ใจว่ารถยนต์มีล้อแน่ๆ แต่รถยนต์แต่ละชนิดอาจมีล้อไม่เท่ากัน

บรรทัด (3) คือการประกาศตัวแปรคลาสตัวที่สองชื่อ hasEngine ซึ่งเป็นตัวแปรแบบ boolean ตัวแปร hasEngine มีไว้บอกว่ารถยนต์มีเครื่องหรือไม่ (ด้วยสามัญสำนึกคุณคงนึกออกว่าตัวแปร hasEngine ควรมีค่าเป็นจริงเสมอ เพราะรถยนต์ต้องมีเครื่อง)

ข้อสังเกตคือตัวแปรคลาสต่างกับตัวแปรที่เราเคยผ่านๆ มาในบทอื่นๆ ตรงที่ ปกติแล้วตัวแปรต้องถูกประกาศ กำหนดค่า และใช้งานภายในแมธธอสเท่านั้น ในขณะที่ตัวแปรคลาสกลับอยู่ภายนอกแมธธอส

บรรทัด (4) (5) และ (6) เป็นการประกาศแมธธอส การประกาศแมธธอสเริ่มต้นด้วย คำว่า void ตามด้วยชื่อของแมธธอส ซึ่งเป็นชื่ออะไรก็ได้ที่เป็นไปตามกฎการตั้งชื่อของภาษา

จาวา ชื่อแมธธอสต้องลงท้ายด้วยเครื่องหมาย () เสมอ เช่นในกรณีนี้แมธธอสนี้มีชื่อว่า run() การตั้งชื่อแมธธอสนิยมขึ้นต้นด้วยอักษรตัวพิมพ์เล็ก และถ้าประกอบด้วยคำหลายคำจะเขียนติดกันหมดและใช้ตัวพิมพ์ใหญ่ขึ้นต้นคำทุกคำที่ตามมา เช่นเดียวกับการตั้งชื่อตัวแปรทุกประการ

สิ่งที่อยู่หลังชื่อของแมธธอสคือบล๊อคปีกกาซึ่งอธิบายพฤติกรรม run() หรือส่วนตัวของแมธธอสนั้นเอง ภายในบล๊อคปีกกาเราใช้คำสั่งในภาษาจาวาเขียนต่อกันไปเรื่อยๆ เพื่ออธิบายพฤติกรรมการวิ่ง ตัวอย่างเช่น ในกรณีนี้ พฤติกรรม run() คืออาการที่รถยนต์วิ่งได้ ซึ่งเราแทนด้วยการแสดงผลที่หน้าจอว่า I am running และเช่นเคยเรานิยมย่อหน้าคำสั่งทั้งหมดในบล๊อคปีกกาให้ลึกกว่าวงเล็บปีกกาเอง เพื่อความสะดวกในการมอง

คลาสคลาสหนึ่งจะประกอบด้วยตัวแปรคลาสกี่ตัวก็ได้ แมธธอสกี่แมธธอสก็ได้ คลาสที่ไม่มีตัวแปรคลาสหรือแมธธอสเลยสักตัวเดียวเรียกว่า คลาสว่าง ซึ่งมีได้ในภาษาจาวา เช่น

```
class A {
}
```

คลาส A เป็นคลาสว่าง ถ้าลองนำไปคอมไพล์ดูจะเห็นว่าคอมไพล์ได้ด้วย แต่คลาสว่างคงไม่ค่อยมีประโยชน์อะไรในทางปฏิบัติ

กลับมามองโปรแกรมภาษาจาวาที่เราเคยเขียนมาก่อนในอดีต โปรแกรมภาษาวาจาที่ผ่านมาล้วนมีโครงสร้างพื้นฐานเป็นดังนี้

โปรแกรม 8 - 2 : HelloWorld.java

```
public class HelloWorld {
        public static void main ( String []  args ) {


        }
}
```

นั้นคือที่ผ่านมาโปรแกรมของเราประกอบด้วยคลาสหนึ่งคลาสที่มีชื่อเหมือนชื่อซอร์สโค้ดเสมอ คลาสคลาสนี้ไม่มีตัวแปรคลาสเป็นสมาชิก และมีแมธธอสแค่หนึ่งแมธธอส ชื่อว่า

`main()` (อย่าเพิ่งสนใจคำว่า `public static` หรือ `String[] args`) การเขียนโปรแกรมในภาษาจาวาแท้ที่จริงก็คือการเขียนนิยามของ `class` นั้นเอง

รูปที่ 8-4 คลาส HelloWorld

โปรแกรมในภาษาจาวาที่สามารถรันได้ต้องมีคลาสอย่างน้อยหนึ่งคลาสที่มีชื่อเหมือนชื่อของซอร์สโค้ด และในคลาสนั้นต้องมีแมธธอสชื่อ `main()` จาวาเวอร์ชัวน์แมทชีนจะมองหาแมธธอสชื่อนี้เสมอทุกครั้งที่รัน เพราะมันจะใช้แมธธอส `main()` เป็นจุดเริ่มต้นของการรัน นี่เป็นเหตุผลที่ทำไมคลาสรถยนต์ของเราจึงคอมไพล์ได้แต่รันไม่ได้ และที่ผ่านมาเราเขียนโปรแกรมให้สั้นที่สุดแต่พอรันได้ด้วยการให้มีแค่คลาสเดียวชื่อเหมือนชื่อโปรแกรม และมีแมธธอสชื่อ `main()` โปรแกรมที่ใหญ่และซับซ้อนกว่านี้อาจมีคลาสเป็นร้อยๆ คลาสในซอร์สโค้ด

วัตถุ

วัตถุ คือ สิ่งที่มีคุณสมบัติและพฤติกรรมตามที่กำหนดไว้ในคลาส วัตถุของคลาสรถยนต์ก็คือตัวรถยนต์ คลาสเป็นแค่คำนิยามไม่มีตัวตนอยู่จริง สิ่งที่มีตัวตนอยู่จริงคือ วัตถุ

คลาสรถยนต์มีได้แค่คลาสเดียว เพราะ ถ้านิยามของคำว่ารถยนต์มีได้หลายนิยามเวลาสนทนาเรื่องรถยนต์คงสับสนวุ่นวายน่าดู แต่รถยนต์ที่เป็นวัตถุมีได้หลายคัน ในกรุงเทพจังหวัดเดียวมีรถยนต์มากกว่าหนึ่งล้านคัน ในโปรแกรมภาษาจาวาโปรแกรมหนึ่ง จะมีคลาส

Vehicle ได้คลาสเดียวแต่มีวัตถุรถยนต์ได้หลายวัตถุ วัตถุในภาษาจาวาเรียกว่า **อินสแตนท์**

การสร้างวัตถุเราเรียกว่า **การสร้างอินสแตนท์** ให้กับคลาส ดังนั้นบางทีเราก็เรียกวัตถุว่า **อินสแตนท์ของคลาส** ด้วย ในหนังสือเล่มนี้เราจะเรียกวัตถุว่า อินสแตนท์ เป็นหลัก ขอให้ระลึกไว้ว่าคือสิ่งเดียวกัน

ตัวอย่างของการสร้างอินสแตนท์ให้กับคลาสรถยนต์เป็นดังต่อไปนี้

โปรแกรม 8 - 3 : BuildACar.java

```
class Vehicle {

  int numberOfWheels;
  boolean hasEngine;
  void run(){
        System.out.println("I am running.");
  }
}

public class BuildACar {
  public static void main (String[] args) {
        new Vehicle();                // (1)
        new Vehicle();                // (2)
        System.out.println("Two cars have been built.");
  }
}
```

โปรแกรม BuildACar.java มีสองคลาส คลาสแรกคือคลาสรถยนต์ คลาสที่สองคือคลาส BuildACar ซึ่งมีชื่อเหมือนชื่อโปรแกรม คลาสนี้เป็นคลาสหลักของโปรแกรม เพราะมีแมธธอส main() อยู่

อันที่จริง ลำดับของคลาสในไฟล์จะเรียงลำดับอย่างไรก็ได้ เราอาจเขียนคลาส BuildACar อยู่ก่อนคลาส Vehicle ก็ได้

สิ่งที่น่าสนใจอย่างหนึ่งก็คือถ้าลองคอมไพล์โปรแกรม BuildACar.java เราจะได้ไฟล์นามสกุล .class ถึงสองไฟล์คือ BuildACar.class และ Vehicle.class นั้นคือในกรณีที่ซอร์สโค้ดของเราประกอบด้วยคลาสมากกว่าหนึ่งคลาส เมื่อนำมาคอมไพล์จะได้ไฟล์ .class หนึ่งไฟล์สำหรับทุกๆ คลาสในซอร์สโค้ด แยกเป็นอิสระจากกัน ไฟล์นามสกุล

.class ทุกไฟล์มีความสำคัญต่อการรันโปรแกรมทั้งสิ้น จาวาเวอร์ชัวน์แมทชีนจะต้องเห็นไฟล์ .class ครบทุกไฟล์มิฉะนั้นโปรแกรมจะรันไม่ได้

โปรแกรมนี้เริ่มต้นที่แมธธอส main() คำสั่งในบรรทัด (1) (2) คือคำสั่งสร้างอินสแตนท์ให้กับคลาส Vehicle หรือก็คือการสร้างรถยนต์ นั้นเอง เราใช้คำสั่ง new ตามด้วยชื่อคลาสโดยมีเครื่องหมาย () ปิดท้ายในการสร้างอินสแตนท์ของคลาสนั้นๆ ในโปรแกรมนี้เราสั่งสองที ดังนั้นเราจะได้รถยนต์สองคัน

เวลารันโปรแกรมนี้บนจาวาเวอร์ชัวน์แมทชีน จาวาเวอร์ชัวน์แมทชีนจะอ่านนิยามของคลาสทุกคลาสที่มีอยู่ในไฟล์ .class แล้วลอกนิยามเหล่านั้นลงบนแรม ซึ่งคลาสหนึ่งๆ จะถูกลอกลงบนแรมเพียงครั้งเดียวเท่านั้นตอนเริ่มโปรแกรม บางทีเราเรียกช่วงเวลาที่จาวาเวอร์ชัวน์แมทชีนกำลังลอกนิยามของคลาสลงบนแรมว่า ช่วงโหลดโปรแกรม นั้นเอง ในกรณีนี้มันจะโหลดเนื้อหาของคลาส Vehicle และ BuildACar เข้าไปในแรม

หลังจากโหลดโปรแกรมเสร็จโปรแกรมจะเริ่มรันจากแมธธอส main() (โปรแกรมภาษาจาวาทุกโปรแกรมจะเริ่มรันจากแมธธอส main() เสมอ) ในกรณีนี้แมธธอส main() เริ่มด้วยคำสั่งสร้างอินสแตนท์ของคลาส Vehicle สองอินสแตนท์ จาวาเวอร์ชัวน์แมทชีนอ่านนิยามที่อยู่ในคลาสว่าอินสแตนท์ของคลาส Vehicle ต้องมีอะไรบ้าง แล้วจะกันเนื้อที่ใหม่ในแรมไว้ให้กับอินสแตนท์แต่ละอินสแตนท์ เป็นที่เก็บตัวแปรคลาส และแมธธอส ของแต่ละอินสแตนท์ คลาสเดียวกันสามารถถูกสร้างอินสแตนท์ได้หลายๆ อินสแตนท์ตามชอบใจ จาวาเวอร์ชัวน์แมทชีนจะกันเนื้อที่ในแรมใหม่ให้กับอินสแตนท์ใหม่ทุกอินสแตนท์ที่สร้างขึ้น ไม่มีการแบ่งใช้เนื้อที่ร่วมกัน ในโปรแกรมนี้เราจะได้อินสแตนท์ของคลาส Vehicle เกิดขึ้นในแรมสองชุด

เคล็ดลับ จำไว้ว่าภายในแรม คลาสมีสำเนาอยู่แค่ชุดเดียวซึ่งสร้างขึ้นตอนโหลดโปรแกรม ส่วนอินสแตนท์จะมีสำเนาอยู่กี่ชุดก็ได้แล้วแต่ว่าเราจะต้องการสร้างวัตถุขึ้นมาจำนวนมากแค่ไหน

อินสแตนท์ที่ถูกสร้างขึ้นจะอยู่ในแรม และตอนนี้เรายังไม่สามารถนำมันมาใช้งานได้เพราะเราไม่รู้จะอ้างถึงแต่ละตัวให้ต่างกันได้อย่างไร การอ้างถึงอินสแตนท์ในแรมเพื่อเอาอินส

แตนท์มาใช้งานเราต้องอาศัยตัวแปรพิเศษชนิดหนึ่งที่เรียกว่า **ตัวแปรอ้างอิง** ซึ่งจะกล่าวถึงในบทต่อไป

9

ตัวแปรอ้างอิง

ตัวแปรอ้างอิง คือ ตัวแปรที่ใช้เก็บเลขบอกตำแหน่งในแรม ตำแหน่งในแรมเหล่านั้นอาจเป็นที่อยู่ของอินสแตนท์ที่เราสร้างขึ้นมา เวลาที่เราต้องการอ้างถึงอินสแตนท์นั้นเราจะใช้การอ้างถึงตัวแปรอ้างอิงที่เก็บเลขบอกตำแหน่งที่อยู่ของอินสแตนท์นั้นแทน หรืออีกนัยหนึ่งตัวแปรอ้างอิงก็คือตัวชี้อินสแตนท์นั้นเอง

ตัวแปรอ้างอิงไม่มีชื่อเรียกเฉพาะของมันเอง ชื่อของมันเปลี่ยนไปเรื่อยตามชื่อคลาสของอินสแตนท์ที่มันชี้ ลองดูตัอย่างการประกาศตัวแปรอ้างอิงที่ใช้ชี้อินสแตนท์รถยนต์ที่เราสร้างขึ้นเมื่อบทที่แล้วดูดังต่อไปนี้

โปรแกรม 9 - 1 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;
        void run(){
                System.out.println("I am running");
        }

}

public class BuildACar {
        public static void main (String[] args) {
                Vehicle myCar,yourCar; // (1)
                myCar = new Vehicle();  // (2)
```

```
            yourCar = new Vehicle(); // (3)
            System.out.println(myCar); // (4)
            System.out.println(yourCar); // (5)
        }
}
```

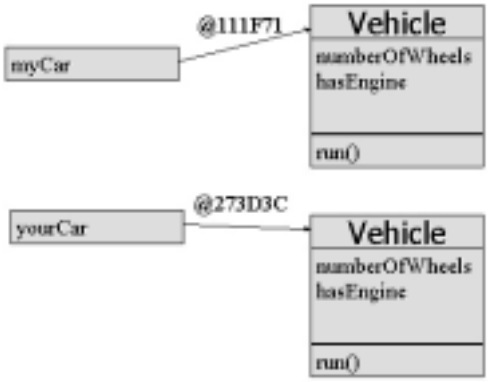
ในบรรทัด (1) เราประกาศตัวแปรอ้างอิงสองตัวชื่อ myCar และ yourCar ชื่อชนิดของตัวแปรอ้างอิงเหมือนชื่อคลาส Vehicle เพราะเราต้องการให้มันใช้ชี้อินสแตนท์ของคลาส Vehicle ได้ ถ้าต้องการตัวแปรอ้างอิงไว้ใช้อินสแตนท์คลาสอื่น ก็ใช้ชื่อคลาสนั้นๆ แทนในการประกาศ

ในบรรทัด (2) (3) เราสร้างอินสแตนท์ของคลาส Vehicle ขึ้นแล้วกำหนดให้ค่าของตัวแปร myCar และ yourCar มีค่าเท่ากับเลขชี้ตำแหน่งของอินสแตนท์ทั้งสองในแรม

ในบรรทัด (4) (5) เราลองแสดงค่าของ myCar และ yourCar ออกนอกจอดู ผลการรันโปรแกรมนี้อาจเป็นดังภาพข้างล่าง

```
C:\java> java BuildACar
Vehicle@1111f71
Vehicle@2373f3d
```

จะเห็นได้ว่า myCar และ yourCar มีค่าเป็นตัวเลขฐานสิบหก เลขฐานสิบหก นี้คือตำแหน่งในแรมที่อินสแตนท์ที่สร้างขึ้นอยู่ ส่วนคำว่า Vehicle@ เป็นเพียงการกำกับว่าอินสแตนท์นี้เป็นอินสแตนท์ของคลาส Vehicle เฉยๆ



รูปที่ 9-1 ตำแหน่งของอินสแตนท์ในแรม

ตำแหน่งในแรมนี้จาวาเวอร์ชัวน์แมทชีนเป็นคนเลือกให้ ซึ่งอาจไม่เท่ากันในการรันแต่ละครั้ง เราไม่มีสิทธิ์เลือกตามใจเรา

คำสั่งประกาศตัวแปรอ้างอิง และคำสั่งสร้างอินสแตนท์สามารถยุบรวมเป็นคำสั่งเดียวกันได้ด้วย เช่น

```
Vehicle myCar = new Vehicle();
Vehicle yourCar = new Vehicle();
```

ต่อไปนี้เราจะใช้คำสั่งในรูปแบบนี้เป็นหลักเพราะกะทัดรัดกว่า

เนื่องจากเราใช้ชื่อคลาสในการประกาศตัวแปรอ้างอิง และคำสั่งก็มีรูปแบบเหมือนการประกาศตัวแปรพื้นฐานแบบอื่นๆ ดังนั้นบางทีเราอาจกล่าวว่า `myCar` และ `yourCar` เป็นตัวแปรชนิด `Vehicle` หรือบ่อยครั้งเราอาจเรียก `myCar` หรือ `yourCar` ว่าเป็นอินสแตนท์ของคลาส `Vehicle` ได้ด้วย แต่ขอให้ระลึกไว้ว่ามันคือตัวแปรอ้างอิงที่ใช้ชี้อินสแตนท์ของคลาส `Vehicle` มิใช่ตัวอินสแตนท์ของคลาส `Vehicle` ตัวแปรอ้างอิงไม่ได้ผูกติดกับอินสแตนท์ที่มันชี้อยู่ตลอดไป มันอาจเปลี่ยนไปชี้อินสแตนท์อื่นได้ด้วย ลองพิจารณาตัวอย่างต่อไปนี้

โปรแกรม 9 - 2 **: Reference.java**

```
class Vehicle {

  int numberOfWheels;
  boolean hasEngine;
  void run(){
```

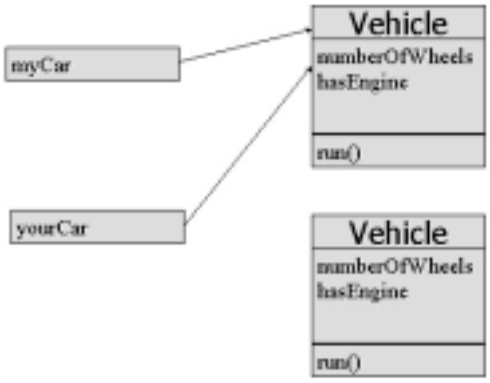
```
        System.out.println("I am running");
  }
}

public class Reference {
  public static void main (String[] args) {
    Vehicle myCar = new Vehicle();
    Vehicle yourCar = new Vehicle();
    System.out.println("myCar points to " + myCar);
    System.out.println("yourCar points to " + yourCar);

    yourCar = myCar;    // (1)
System.out.println("yourCar now points to " + yourCar);
    myCar = null;     // (2)
    System.out.println("myCar points to " + myCar);
  }
}
```

ผลของการรันโปรแกรมเป็นดังภาพ

```
C:\java> java Reference
myCar points to Vehicle@111f71
yourCar points to Vehicle@273d3c
yourCar points to Vehicle@111f71
myCar points to null
```

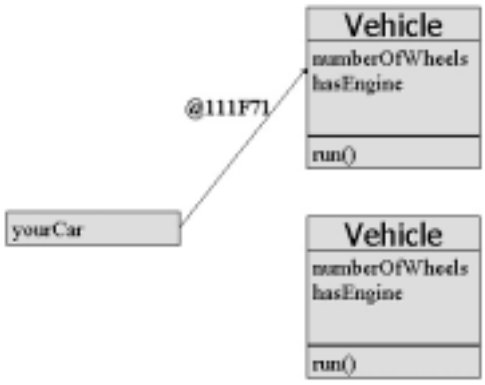
โปรแกรมเริ่มต้นด้วยการสร้างตัวแปรอ้างอิง `myCar` และ `yourCar` ซึ่งกำหนดให้ชี้ไปยังอินสแตนท์ของ `Vehicle` ที่ตำแหน่ง `111f71` และ `273d3c` ตามลำดับ

ในบรรทัด `(1)` มีการกำหนดค่าใหม่ให้ตัวแปรอ้างอิง `yourCar` มีค่าเท่ากับค่าปัจจุบันของ `myCar` คือ `111f71` ผลที่ได้คือตัวแปรอ้างอิง `yourCar` จะชี้ไปยังอินสแตนท์เดียวกับ `myCar` ดังภาพ



รูปที่ 9-2 ตำแหน่งในแรม

เราสามารถทำให้ตัวแปรอ้างอิงไม่ชี้อะไรเลยได้ด้วยการกำหนดค่าของตัวแปรอ้างอิงให้เท่ากับ `null` ในบรรทัด (2) เรากำหนดค่า `null` ให้ตัวแปร `myCar` ตอนนี้ตัวแปรอ้างอิง `myCar` ไม่ชี้อินสแตนท์อะไรเลย



รูปที่ 9—3 ตำแหน่งในแรม

`null` เป็นคำเฉพาะในภาษาจาวาและต้องเขียนด้วยตัวพิมพ์เล็กเท่านั้น

ปกติเวลาสร้างอินสแตนท์ใหม่ขึ้นมาเราก็จะประกาศตัวแปรอ้างอิงขึ้นมาชี้ด้วยเสมอ อินสแตนท์ที่ไม่มีตัวแปรอ้างอิงใดชี้อยู่จะเสี่ยงต่อการถูกทำลาย ตัวอย่างเช่น อินสแตนท์ที่เราสร้างขึ้นมาเฉยๆ โดยไม่กำหนดตัวแปรอ้างอิงให้ หรืออินสแตนท์ที่เคยมีตัวแปรอ้างอิงแต่เรา

จับตัวแปรอ้างอิงนั้นให้เท่ากับ `null` เสีย ทุกระยะของการรันโปรแกรม จาวาเวอร์ชัวน์แมทชีนจะตรวจสอบดูว่ามีอินสแตนท์ใดบ้างที่เข้าข่ายนี้ มันจะทำลายอินสแตนท์เหล่านั้นเสียเพื่อคืนแรมให้กับระบบ เราไม่สามารถทำนายพฤติกรรมการทำลายของจาวาเวอร์ชัวน์แมทชีนได้ บางครั้งมันลุกขึ้นมาตรวจสอบมันแล้วพบอินสแตนท์ที่เข้าข่ายมันอาจจะทำลายหรือไม่ทำลายก็ได้ แต่ข้อดีก็คือ หน้าที่การทำลายอินสแตนท์เป็นของจาวาเวอร์ชัวน์แมทชีน นักเขียนโปรแกรมไม่ต้องสนใจ

เคล็ดลับ เวลาศึกษาเรื่องคลาสและวัตถุ ควรให้ความสำคัญกับเรื่องของการใช้เนื้อที่ในแรม เพราะจะทำให้เข้าใจได้ดียิ่งขึ้น แม้ว่าการจัดการแรมจะเป็นหน้าที่ของจาวาเวอร์ชัวน์แมทชีนก็ตาม

10

ตัวแปรคลาส

คลาส `Vehicle` มีตัวแปรคลาสสองตัวเป็นสมาชิกคือ `numberOfWheels` และ `hasEngine` ตัวแปรคลาสมีไว้แสดงคุณสมบัติของวัตถุ

ตัวแปรคลาสจะมีกี่ตัวก็ได้ และจะเป็นตัวแปรชนิดใดก็ได้ ตั้งแต่ตัวแปรพื้นฐานจนถึงตัวแปรอ้างอิง ที่ผ่านมาจะเห็นได้ว่าเราประกาศตัวแปรคลาสไว้เฉยๆ ในคลาสโดยไม่มีการกำหนดค่าให้ ที่จริงแล้วเราสามารถกำหนดค่าของตัวแปรคลาสไว้ในคลาสได้ด้วย เช่น

โปรแกรม 10 - 1 : Vehicle.java

```
class Vehicle {

        int numberOfWheels = 4;
        boolean hasEngine =  true;
        void run(){
                System.out.println("I am running");
        }
}
```

ค่าที่เรากำหนดให้จะเป็นค่าเริ่มต้นของตัวแปรนั้นๆ ตอนที่มีการสร้างอินสแตนท์ของวัตถุนั้นขึ้น แต่ถ้าไม่กำหนดค่าไว้ในคลาส จาวาเวอร์ชัวน์แมทชีนจะกำหนดค่าให้เอง ค่าที่กำหนดให้เราเรียกว่า ค่าปกติของตัวแปรคลาส มีดังนี้

ตารางที่ 10-1 ค่าปกติของตัวแปรคลาส

ชนิดของตัวแปร	ค่าปกติ
ตัวแปรตัวเลข	0
ตัวแปรตรรก	false
ตัวแปรตัวอักษร	\u0000
ตัวแปรอ้างอิง	null

ตัวแปรคลาสประกาศไว้ในคลาส และไม่เหมือนกับตัวแปรที่ผ่านๆ มาตอนที่เรียนเรื่องตัวแปรพื้นฐาน ตัวแปรเหล่านั้นประกาศและใช้งานภายในแมธธอส ตัวแปรที่เราประกาศไว้ในแมธธอสเราเรียกว่า **ตัวแปรท้องถิ่น**

ตัวแปรท้องถิ่นไม่มีค่าปกติ จึงไม่ปลอดภัยเหมือนตัวแปรคลาส เพราะถ้าเราประกาศตัวแปรท้องถิ่นไว้เฉยๆ โดยไม่ได้กำหนดค่าไว้ โปรแกรมจะหยุดชะงักถ้ามีการเอาตัวแปรท้องถิ่นตัวนั้นไปใช้งาน เช่น

โปรแกรม 10 - 2: TestVariable.java

```
public class TestVariable {
  public static void main(String[]args){
        char c;
        System.out.println(c); // (1) Error
  }
}
```

ถ้ารันโปรแกรมนี้โปรแกรมจะหยุดชะงักที่บรรทัด `(1)` เพราะมันไม่รู้ว่าตัวแปร `c` มีค่าเป็นเท่าไร

สรุปก็คือ เวลาประกาศตัวแปรคลาสจะกำหนดค่าเริ่มต้นหรือไม่ก็ได้ เพราะถ้าไม่กำหนดจาวาเวอร์ชัวน์แมทชีนจะกำหนดค่าปกติให้ แต่ในกรณีของตัวแปรท้องถิ่นควรกำหนดค่าเริ่มต้นทันทีที่ประกาศเพื่อป้องกันโปรแกรมหยุดชะงักโดยไม่ตั้งใจ

ตัวแปรคลาสที่เห็นในคลาสเป็นเพียงแค่นิยามท่านั้น ตัวแปรคลาสจะเกิดขึ้นจริงๆ ก็ต่อเมื่อมีการสร้างอินสแตนท์ของคลาสขึ้นมาใช้งาน ที่อยู่ของตัวแปรคลาสจะอยู่ในที่ว่างในแรมที่กันไว้ให้สำหรับอินสแตนท์นั้นๆ อินสแตนท์แต่ละอินสแตนท์ที่สร้างขึ้นจะมีตัวแปรคลาสเป็นของมันเองหนึ่งชุด ไม่มีการใช้ร่วมกันข้ามอินสแตนท์ และตัวแปรคลาสจะตายไปพร้อมกับอินสแตนท์นั้นๆ เวลาที่จาวาเวอร์ชัวน์แมทชีนทำลายอินสแตนท์

สำหรับตัวแปรท้องถิ่น จะเกิดขึ้นจริงก็ต่อเมื่อมีการเรียกใช้งานแมธธอสนั้น และจะตายทันทีที่แมธธอสรันเสร็จ ตัวแปรท้องถิ่นถูกอ้างถึงได้เฉพาะภายในแมธธอสที่มันอยู่เท่านั้น ดังนั้นในคลาสเดียวกันชื่อของตัวแปรคลาส อาจซ้ำกับตัวแปรท้องถิ่นบางตัวที่อยู่ในแมธธอสได้ ตัวอย่างเช่น

โปรแกรม 10 - 3 : TestClass.java

```
class TestClass {
        int i;
        void x(){
                int i;
        }
}
```

ย้อนกลับไปดูโปรแกรม 9-1 ในบทที่แล้ว อินสแตนท์ `myCar` และ `yourCar` ต่างก็มีตัวแปร `numberOfWheels` และ `hasEngine` เป็นของตัวเอง ไม่เกี่ยวข้องกันและไม่จำเป็นต้องมีค่าเท่ากัน รถยนต์ของผมอาจเป็นรถบรรทุกมี 6 ล้อ ส่วนรถยนต์ของคุณอาจเป็นรถเก๋งมี 4 ล้อ เราสามารถเปลี่ยนค่าของตัวแปรคลาสภายหลังได้ และการอ้างถึงตัวแปรคลาสเหล่านั้นทำได้โดยใช้ชื่ออินสแตนท์ตามด้วยจุดตามด้วยชื่อตัวแปร ลองดูตัวอย่างต่อไปนี้

โปรแกรม 10 - 4 : BuildACar.java

```
class Vehicle {

  int numberOfWheels = 4;
  boolean hasEngine = true;
  void run(){
        System.out.println("I am running");
  }

}

public class BuildACar {
  public static void main (String[] args) {

    Vehicle myCar = new Vehicle(); // (1)
    Vehicle yourCar = new Vehicle(); // (2)
    myCar.numberOfWheels = 6;     // (3)
    yourCar.numberOfWheels = 4;       // (4)
    System.out.println("My car has " + myCar.numberOfWheels + "
wheels.");
    System.out.println("Your car has " + yourCar.numberOfWheels + "
wheels.");
   }
```

```
}
```

ลองคอมไพล์และรันโปรแกรมข้างต้นจะได้ผลเป็นดังนี้

```
C:\java> java BuildACar
My car has 6 wheels.
Your car has 4 wheels.
```

บรรทัด (1) (2) เป็นการสร้างอินสแตนท์สองอินสแตนท์ชื่อ myCar และ yourCar ตามลำดับ แต่ละอินสแตนท์จะเป็นอิสระจากกันและอยู่คนละที่ในแรม แต่ละอินสแตนท์จะมีตัวแปรอินสแตนท์ numberOfWheels และ hasEngine ของตัวเองแถมมาให้ทันทีที่อินสแตนท์ถูกสร้างขึ้นโดยที่เราไม่จำเป็นต้องสร้าง ค่าเริ่มต้นของ numberOfWheels จะเป็น 0 ส่วน hasEngine จะเป็นเท็จ เพราะเป็นค่าปกติของตัวแปรคลาส

บรรทัด (3) และ (4) เป็นการกำหนดค่าให้กับตัวแปรอินสแตนท์ numberOfWheels ของอินสแตนท์ทั้งสอง เราใช้ชื่อตัวแปรอ้างอิงตามด้วยจุดนำหน้าชื่อตัวแปร เพื่อแยกแยะความแตกต่างว่าเป็นตัวแปรของอินสแตนท์ไหน จากนั้นโปรแกรมจะแสดงค่าของตัวแปรทั้งสองออกหน้าจอ สังเกตว่าค่าของตัวแปรอินสแตนท์เป็นของใครของมัน และไม่จำเป็นต้องมีค่าเท่ากัน

ตัวแปรสแตรติค

เราสามารถสร้างตัวแปรคลาสที่เป็นของคลาสเองจริงๆ ไม่มีอินสแตนท์ใดยึดความเป็นเจ้าของ แต่เป็นของกลางที่อินสแตนท์ทุกอินสแตนท์ของคลาสแบ่งกันใช้ เราเรียกตัวแปรคลาสนี้ว่า ตัวแปรสแตรติค ซึ่งประกาศได้โดยการใช้คำสั่ง static ตัวอย่างเช่น

โปรแกรม 10 - 4 : BuildACar.java

```
class Vehicle {

  int numberOfWheels ;
  boolean hasEngine ;
  static int numberOfCars; // (1)

  void run(){
        System.out.println("I am running");
  }
```

```
}

public class BuildACar {
  public static void main (String[] args) {

    Vehicle myCar = new Vehicle();
    myCar.numberOfCars++;// (2)
    myCar.numberOfWheels = 6;

    Vehicle yourCar = new Vehicle();
    yourCar.numberOfCars++;// (3)
    yourCar.numberOfWheels = 4;
    System.out.println("My car has " +   myCar.numberOfWheels + "
wheels.");
    System.out.println("Your car has " + yourCar.numberOfWheels + "
wheels.");
    System.out.println("There are " + Vehicle.numberOfCars + " cars in
the world."); // (4)
  }
}
```

ผลการรันโปรแกรมเป็นดังนี้

```
C:\java> java BuildACar
My car has 6 wheels.
Your car has 4 wheels.
There are 2 cars in the world.
```

ในโปรแกรมนี้คลาส `Vehicle` มีตัวแปรคลาสเพิ่มขึ้นหนึ่งตัวคือ `numberOfCars` ในบรรทัด `(1)` ตัวแปรตัวนี้ประกาศให้เป็นตัวแปรสแตรติค ซึ่งใช้นับจำนวนรถยนต์ที่ผลิตขึ้น

ทุกครั้งที่มีการสร้างอินสแตนท์ให้คลาส `Vehicle` จะมีการเพิ่มค่าของตัวแปร `numberOfCars` ทีละหนึ่งเช่นในบรรทัด `(2)` และ `(3)`

เมื่อแสดงค่าของ `numberOfCars` ออกที่หน้าจอในบรรทัด `(4)` จะพบว่ามีค่าเป็น 2 เพราะทั้งคำสั่ง `myCar.numberOfCars++` และ `yourCar.numberOfCars++` ต่างก็ไปเพิ่มค่าตัวแปรตัวเดียวกันเพราะตัวแปรสแตรติคมีแค่ชุดเดียวตลอดการรันโปรแกรม ผลที่ได้คือตัวแปร `numberOfCars` มีค่าเป็น 2

การอ้างถึงตัวแปรสแตรติคจะใช้ชื่อของคลาสหรือชื่อของอินสแตนท์ก็ได้ ดังนั้น `myCar.numberOfCars`, `yourCar.numberOfCars` และ `Vehicle.numberOfCars` คือตัวแปรตัวเดียวกันทั้งสิ้น

ตัวแปรสแตรติคจะเกิดขึ้นเพียงครั้งเดียวตอนโหลดคลาส และคงอยู่ตลอดไปจนกว่าคลาสจะตายไป ค่าของตัวแปรสแตรติคเปลี่ยนได้ แต่จะมีค่าเดียวเท่านั้นไม่ว่าจะมีการสร้างอินสแตนท์ของคลาสนั้นกี่อินสแตนท์ ส่วนมากเราสร้างตัแปรสแตรติคขึ้นมาเพื่อใช้เป็นตัวนับจำนวนอินสแตนท์ที่เราสร้างขึ้นสำหรับคลาสนั้นๆ

ตัวแปรถาวร

เราสามารถสร้างตัวแปรคลาสที่มีค่าถาวรเปลี่ยนแปลงไม่ได้อีก ด้วยการใช้คำสั่ง `final` นำหน้า เช่น

```
final float PI = 3.14159;
```

ในตัวอย่างเป็นการกำหนดค่าคงตัว `PI` ในวิชาเรขาคณิต ปกติแล้วเรานิยมเปลี่ยนไปใช้อักษรพิมพ์ใหญ่ทั้งหมดในการตั้งชื่อตัวแปรถาวร ทั้งที่เป็นแค่ความนิยมเท่านั้น

ตัวแปรถาวรต้องมีการกำหนดค่าด้วย และการกำหนดค่าทำได้แค่ครั้งเดียว ค่านี้จะเปลี่ยนไม่ได้อีกตลอดการรันโปรแกรม

ทั้งตัวแปรคลาสปกติและตัวแปรสแตรติคสามารถประกาศให้เป็นตัวแปรถาวรได้ นอกจากนี้ตัวแปรท้องถิ่นก็ประกาศให้เป็นตัวแปรถาวรได้ด้วย

แต่ตัวแปรท้องถิ่นประกาศให้เป็นตัวแปรสแตรติคไม่ได้

11

อะเรย์

อะเรย์ คือ เซตของตัวแปรชนิดเดียวกัน ซึ่งสมาชิกของอะเรย์อาจเป็นตัวแปรพื้นฐานหรือตัวแปรอ้างอิงก็ได้ จำนวนสมาชิกของอะเรย์มีขนาดแน่นอน และสมาชิกของอะเรย์แต่ละตัวจะมีลำดับประจำตัวอยู่

อะเรย์ในภาษาจาวาเป็นวัตถุ ดังนั้นจึงเป็นการดีที่จะกล่าวถึงอะเรย์ในบทนี้ เพื่อจะได้เห็นตัวอย่างของการเอาแนวคิดเรื่องวัตถุไปใช้จริง อย่างไรก็ตามอะเรย์เป็นวัตถุแบบพิเศษ จึงมีวิธีการใช้งานและคำสั่งที่ไม่เหมือนกับวัตถุทั่วไปนัก

อะเรย์เป็นวัตถุ ดังนั้นต้องมีการประกาศตัวแปรอ้างอิง และสร้างอินสแตนท์ การประกาศอะเรย์หรือการประกาศตัวแปรอ้างอิงแบบอะเรย์ ทำได้ดังตัวอย่าง

```
int[] i;
int[] a, b;
```

คำสั่งข้างต้นเป็นการประกาศอะเรย์ชื่อ `i` ซึ่งมีสมาชิกของอะเรย์เป็นตัวแปรประเภท `int` สมาชิกของอะเรย์เดียวกันต้องเป็นตัวแปรประเภทเดียวกันเสมอ คำสั่งในการประกาศอะเรย์ใช้สัญลักษณ์เครื่องหมายวงเล็บเหลี่ยม ซึ่งอาจวางอยู่ต่อท้ายชื่อชนิดของตัวแปรสมาชิกหรือวางอยู่ต่อท้ายชื่อตัวแปรอะเรย์ก็ได้ดัง เช่น

```
int i[];
int a[], b[];
int x[], y;
```

ในกรณีที่เราวางวงเล็บไว้ต่อท้ายชื่ออะเรย์ ต้องวางไว้ท้ายชื่ออะเรย์ทุกตัวในบรรทัด มิฉะนั้นตัวที่ไม่มีวงเล็บต่อท้ายจะกลายเป็นตัวแปรธรรมดาไป ตัวอย่างข้างต้นตัวแปร y ไม่ใช่อะเรย์ แต่เป็นตัวแปร int ธรรมดา

ขั้นตอนต่อไปคือการกำหนดสร้างตัวแปรอะเรย์ ซึ่งก็คล้ายๆ กับการสร้างอินสแตนท์ของอะเรย์ แต่คำสั่งอาจดูแตกต่างกับการสร้างอินสแตนท์ของวัตถุปกติเล็กน้อย เช่น

```
int[] n
n = new n[10];
```

เราใช้คำสั่ง new ในการสร้างตัวแปรอะเรย์ และระบุจำนวนสมาชิกไว้ภายในวงเล็บก้ามปู อย่างในกรณีนี้ อะเรย์ n จะมีสมาชิกเป็นตัวแปรจำนวนเต็มจำนวนทั้งสิ้น 10 ตัวแปร เมื่อจาวาเวอร์ชัวน์แมทชีนพบคำสั่งในการสร้างอะเรย์ มันจะทำการกันเนื้อที่ไว้ให้ในแรมสำหรับเก็บตัวแปรซึ่งในกรณีนี้คือ ตัวแปรแบบจำนวนเต็มสิบตัว เราอาจมอง n เป็นวัตถุที่มีตัวแปรคลาส 10 ตัวเป็นตัวแปร int ทั้งหมดก็ได้

เราสามารถยุบคำสั่งในการประกาศ และคำสั่งในการสร้างเป็นคำสั่งเดียวได้ดังตัวอย่าง

```
int[] n = new n[10];
```

ต่อไปนี้เราจะใช้รูปแบบคำสั่งแบบนี้ในการประกาศและสร้างอะเรย์เป็นหลัก เพราะมีความกระชับกว่า

เมื่อสร้างตัวแปรอะเรย์เสร็จแล้วก็ถึงขั้นตอนของการกำหนดค่า เราใช้เครื่องหมายปีกกาในการกำหนดค่า คำสั่งในการกำหนดค่าเป็นคำสั่งที่รวมการประกาศ และการสร้างเอาไว้ด้วยเสมอ ไม่มีคำสั่งกำหนดค่าอย่างเดียว

```
int[] b = { 1, 4, 12 ,2 , 1, 4 }
```

คำสั่งนี้เป็นทั้งการประกาศ การสร้าง และการกำหนดค่าให้กับอะเรย์ b ในคำสั่งเดียว อะเรย์ b จะมีสมาชิกเป็นตัวแปรจำนวนเต็ม 6 ตัว มีค่าของตัวแปรแต่ละตัวเป็น 1, 4, 12, 2 , 1, 4 ตามลำดับ

ถ้าเราต้องการอ้างถึงตัวแปรที่เป็นสมาชิกของอะเรย์ เราใช้สัญลักษณ์ b[0], b[1], b[2], b[3], b[4], b[5] แทนสมาชิกตัวแรกจนถึงตัวสุดท้ายตามลำดับ สังเกตว่าเลขดรรชนีเริ่มจาก 0 เสมอ ดังนั้นสมาชิกตัวสุดท้ายของอะเรย์จะมีเลขดรรชนีเท่ากับจำนวนสมาชิกทั้งหมดของอะเรย์ลบด้วยหนึ่ง ตัวอย่างการอ้างถึงก็เช่น

โปรแกรม 11 - 1 : TestArray.java

```
public class TestArray {
        public static void main (String[] args) {
        int[] b = { 1, 4, 12 ,2 , 1, 4 }
        int k = b[0] + b[1];

        }
}
```

อะเรย์เป็นวัตถุที่สามารถสร้างได้ทันทีโดยไม่ต้องนิยามคลาสขึ้นมาก่อน ในกรณีนี้ `k` จะมีค่าเท่ากับ `5` เพราะ `b[0]` มีค่าเป็น1 ส่วน `b[1]` มีค่าเป็น 4

ในกรณีที่เราไม่ได้ใช้คำสั่งกำหนดค่าอะเรย์ เราสามารถกำหนดอะเรย์ทีหลังได้ด้วยการกำหนดค่าให้กับทีละสมาชิกเป็นตัวๆ ไป เช่น

```
char c[] = new char[3];
c[0] = 'k';
c[1] = 'b';
c[2] = 'x';
```

และถ้าเราไม่กำหนดค่าให้สมาชิกของอะเรย์แต่ละตัว จาวาเวอร์ชัวน์แมทชีนจะกำหนดค่าปกติให้เอง เพราะสมาชิกของอะเรย์คือตัวแปรคลาสของอะเรย์

ตัวแปรอะเรย์ทุกตัวเมื่อถูกสร้างขึ้น จาวาเวอร์ชัวน์แมทชีนจะแถมตัวแปรคลาสเป็นตัวแปรจำนวนเต็มอีกตัวหนึ่งมาให้ซึ่งมีค่าเท่ากับจำนวนสมาชิกของอะเรย์ตัวนั้น (เปลี่ยนค่าภายหลังไม่ได้ เพราะอะเรย์ต้องมีจำนวนสมาชิกคงที่) ตัวแปรตัวนั้นมีชื่อว่า `length` เช่น

```
int[] x = new x[8];
System.out.println(x.length);
```

ในกรณีนี้จะได้เลข 8 ออกหน้าจอ ข้อน่าสังเกตคือ อะเรย์เป็นวัตถุที่มีแต่ตัวแปรคลาส ไม่มีแมธธอส

ที่ผ่านมาทั้งหมดอะเรย์มีแค่หนึ่งมิติ อะเรย์สามารถมีได้มากกว่าหนึ่งมิติคล้ายๆ กับเป็นแมทริกซ์นั้นเอง ตัวอย่างของอะเรย์สองมิติเป็นดังนี้

```
double[][] aMatrix = {
        {1.0, 0.0, 0.0},
   {0.0, 1.0, 3.0},
   {2.0, 1.0, 0.0},
   {0.0, 1.5, 1.5}
};
```

จำนวนวงเล็บเหลี่ยมที่อยู่ท้าย double บอกจำนวนมิติของอะเรย์ชื่อ aMatrix ซึ่งมี 4 แถว แถวหนึ่งหนึ่งมีสมาชิก 3 ตัว หรือมี 3 คอลัมน์นั้นเอง บางทีเราเรียกว่า aMatrix เป็นอะเรย์สองมิติแบบ 4 x 3

การประกาศอะเรย์หลายมิติ ตำแหน่งของเครื่องหมายวงเล็บเหลี่ยมจะอยู่ตรงไหนก็ได้ ระหว่างชื่อชนิดของตัวแปรสมาชิกกับชื่อของตัวแปร ขอเพียงแต่ให้นับได้เท่ากับจำนวนมิติที่เราต้องการเช่น

```
char[][] c;
char c[][];
char[] c[];
```

เวลามองอะเรย์สองมิติ ให้มองว่าเป็นอะเรย์ซ้อนอะเรย์ เช่นในกรณีของ aMatrix ให้มองเป็นอะเรย์หนึ่งมิติที่มีสมาชิกเป็นอะเรย์ 4 อะเรย์ สมาชิกแต่ละตัวก็มีสมาชิกเป็นตัวแปร double อยู่ภายในอีก 3 ตัว ถ้าเราต้องการอ้างถึงตัวแปร double ที่อยู่ในอะเรย์ตัวที่สองและเป็นสมาชิกตัวที่สามของอะเรย์ เราใช้สัญลักษณ์

```
aMatrix[1][2]
```

ซึ่งในกรณีนี้มีค่าเป็น 3.0 อย่าลืมว่าเลขดรรชนีเริ่มจาก 0 มิใช่ 1 ดังนั้นแถวที่สองมีดรรชนีเป็น 1 และคอลัมน์ที่สามมีดรรชนีเป็น 2

เราสามารถอ้างถึงอะเรย์ที่ซ้อนอยู่ในอะเรย์ได้ด้วย เช่นถ้าต้องการอ้างถึงแถวที่สามทั้งแถว เราใช้สัญลักษณ์

```
aMatrix [2]
```

ซึ่งในกรณีนี้มีค่าเป็น {2.0,1.0,0.0} แต่ในทางตรงกันข้ามเราไม่สามารถอ้างถึงเป็นคอลัมน์ได้ เพราะสมาชิกแต่ละตัวในคอลัมน์อยู่ในอะเรย์ต่างแถวกัน

ที่จริงแล้วอะเรย์ในแต่ละแถวไม่จำเป็นต้องมีจำนวนสมาชิกเท่ากันก็ได้ เช่น

```
short[] [] s = {
        {3, 2, 2},
        {2},
        {1, 0, 2},
        {1, 2, 1, 1, 1}
};
```

ในกรณีนี้ s เป็นอะเรย์สองมิติ มีสมาชิกเป็นอะเรย์แบบ short สี่ตัว ซึ่งมีสมาชิกเป็นตัวแปร short อยู่ภายใน 3, 1, 3 และ 5 ตัวตามลำดับ คำสั่งคำสั่งนี้เป็นแค่คำสั่งหนึ่งคำสั่ง

สังเกตได้จากเครื่องหมาย ; มีแค่อันเดียว แต่เรานิยมเขียนเป็นหลายๆ บรรทัดโดยแยกแต่ละแถวของอะเรย์ไว้ในแต่ละบรรทัดเพื่อความสะดวกในการมองเท่านั้น

ถ้าเราต้องการประกาศและสร้างตัวแปร s แต่ยังไม่ต้องการกำหนดค่า รูปแบบของคำสั่งควรเป็นดังนี้

```
short[][] s = new short[4][];
```

นั้นคือในกรณีที่จำนวนสมาชิกในแต่ละแถวไม่เท่ากันเราไม่ต้องระบุจำนวนคอลัมน์เวลาสร้างอะเรย์ แต่ในทางตรงกันข้ามรูปแบบคำสั่งต่อไปนี้ไม่ถูกต้องตามหลักภาษาจาวา

```
short[][] s = new short[][3];
```

เนื่องจากเราไม่อาจสร้างอะเรย์ที่ประกอบด้วยสมาชิกที่เรายังไม่ทราบจำนวนแถวได้

12

แมธธอส

คลาส Vehicle มีแมธธอสอยู่หนึ่งแมธธอสคือ run()ซึ่งใช้นิยามอาการที่รถยนต์ทุกคันจะต้องวิ่งได้ ส่วนตัวของแมธธอส run() คือกลุ่มของคำสั่งที่เป็นเสมือนการวิ่งของรถยนต์ ซึ่งในที่นี้คือคำสั่งให้แสดงผลออกนอกจอว่า I am running เพราะคอมพิวเตอร์ไม่สามารถสร้างรถยนต์ที่วิ่งได้จริงๆ ให้เรา

ส่วนตัวของแมธธอสจะเป็นกลุ่มคำสั่งอะไรก็ได้ในภาษาจาวา และสามารถเขียนต่อๆ กันไปได้เรื่อยๆ เมื่อใดก็ตามที่เราต้องการให้จาวาเวอร์ชัวน์แมทชีนทำสิ่งที่เราต้องการ เราสามารถทำได้โดยเขียนคำสั่งเหล่านั้นไว้ในแมธธอส จากนั้นก็สร้างอินสแตนท์ของคลาสนั้นขึ้นมาแล้วเรียกแมธธอสอินสแตนท์ออกมาใช้งาน ดังตัวอย่างต่อไปนี้

โปรแกรม 12 - 1 : TestACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
                System.out.println("I am running");
        }

}

public class TestACar {
        public static void main (String[] args) {
```

```
                Vehicle myCar = new Vehicle();        // (1)
                myCar.run();                          // (2)
        }
}
```

คำสั่งในบรรทัด (1) คือการสร้างอินสแตนท์ของคลาส Vehicle ชื่อ myCar คำสั่งในบรรทัด (2)คือคำสั่งเรียกอินสแตนท์แมธธอสของ myCar ที่ชื่อ run()ออกมาทำงาน เราใช้ชื่อของตัวแปรอ้างอิงตามด้วยจุดนำหน้าชื่อแมธธอสในการอ้างถึงแมธธอสของอินสแตนท์ สิ่งที่ได้ก็คือคำว่า I am running ที่จะพิมพ์ออกที่หน้าจอ

สังเกตว่าเรานิยามแมธธอส run() ไว้ในคลาส Vehicle แต่พอจะใช้งานมันเราไม่สามารถเรียกมันออกมาใช้ได้โดยตรงเพราะคลาสเป็นเพียงนิยามไม่มีตัวตน อินสแตนท์ของคลาสเท่านั้นที่มีตัวตนเป็นวัตถุจับต้องได้ในภาษาจาวา เราจึงต้องสร้างอินสแตนท์ของคลาส Vehicle ขึ้นมาตัวหนึ่งก่อน และเรียก run() จากอินสแตนท์นั้นทางอ้อม

ภายในนิยามของแมธธอสในคลาสเดียวกันสามารถเรียกใช้ตัวแปรคลาสและแมธธอสของคลาสนั้นได้ด้วยตามใจชอบ ตัวอย่างเช่น

โปรแกรม 12 - 2 : Vehicle.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
                System.out.println("I am running");
        }

    void checkup() {                          // (1)
        if (numberOfWheels < 4)
                System.out.println("I cannot run.");
        else
                run();
    }
}
```

คลาส Vehicle ในตัวอย่าง มีแมธธอสตัวใหม่ชื่อ checkup() ในบรรทัด (1) แมธธอสนี้ทำการตรวจสภาพรถยนต์ว่าวิ่งได้หรือไม่ โดยการนับจำนวนล้อ ถ้าจำนวนล้อน้อยกว่า 4

โปรแกรมจะแสดงข้อความว่า `I cannot run` แต่ถ้าจำนวนล้อมีมากกว่าหรือเท่ากับ 4 โปรแกรมจะเรียกแมธธอส `run()`

สังเกตการเรียกใช้ตัวแปรคลาสและแมธธอสภายในแมธธอส `checkup()` ใช้ชื่อของตัวแปรคลาสและแมธธอสได้เลยเพราะอยู่ในภายในคลาสเดียวกัน ไม่มีชื่ออินสแตนท์มาเกี่ยวข้อง

การส่งผ่านตัวแปรเข้าไปในแมธธอสและออกจากแมธธอส

บางครั้งเราต้องการส่งผ่านตัวแปรเข้าไปในแมธธอสหรือออกจากแมธธอส

การส่งผ่านตัวแปรเข้าไปในแมธธอสทำได้ดังตัวอย่างต่อไปนี้

โปรแกรม 12 - 3 : DriveACar.java

```
class Vehicle {

  int numberOfWheels;
  boolean hasEngine;

  void run(int mile){  // (1)
    System.out.println("I am running on the " + mile + "th mile." );
                         // (2)
  }
}

public class DriveACar {
  public static void main (String[] args) {

    Vehicle myCar = new Vehicle();
    for (int i = 1;i <= 5; i++) {
       myCar.run(i);     //  (3)
    }
  }
}
```

ในบรรทัด `(1)` มีการส่งผ่านตัวแปรชื่อ `mile` ซึ่งเป็นตัวแปร `int` เข้าไปในแมธธอส การส่งผ่านตัวแปรทำได้โดยระบุชื่อชนิดของตัวแปร ตามด้วยชื่อตัวแปร ในวงเล็บที่อยู่หลังชื่อของแมธธอส และเราเรียกตัวแปรที่ส่งผ่านเข้าไปในแมธธอสว่า **พารามิเตอร์** ของแมธธอส

เราสามารถอ้างถึงชื่อนี้ที่ใดก็ได้ภายในบล๊อคปีกกาของแมธธอส ในที่นี้ในบรรทัด (2) เรานำตัวแปร mile ไปใช้แสดงข้อความออกนอกจอว่ารถของเรากำลังวิ่งอยู่ที่เลขไมล์ที่เท่าไร

ในบรรทัด (3) เราเรียกแมธธอส run()สิบครั้ง ด้วยการใช้คำสั่งวนลูป for ตัวแปร i ในคำสั่ง for ถูกส่งผ่านไปยังแมธธอส run() ดังนั้นตัวแปรชื่อ mile ที่อยู่ในแมธธอส run() จะมีค่าเท่ากับ i ทุกครั้งที่เรียกแมธธอส แต่ i มีค่าเพิ่มขึ้นเรื่อยๆ จาก 1 ถึง 5 ทุกครั้งที่วนลูป ผลที่ได้เมื่อรันโปรแกรมนี้ก็คือรถยนต์ myCar จะวิ่งเป็นระยะทางสิบไมล์ และเมื่อใดที่วิ่งได้ครบหนึ่งไมล์จะแสดงระยะทางที่วิ่งได้ออกมา ดังภาพ

```
C:\java> java DriveACar
I am running on the 1th mile.
I am running on the 2th mile.
I am running on the 3th mile.
I am running on the 4th mile.
I am running on the 5th mile.
```

เวลาเรียกแมธธอสที่มีการส่งผ่านตัวแปร เราจะส่งผ่านค่าคงตัวหรือตัวแปรเข้าไปในแมธธอสก็ได้ ถ้าเราส่งตัวแปรเข้าไปในแมธธอสอย่างเช่นในกรณีตัวอย่าง เราส่งตัวแปร i เข้าไปในแมธธอส run() แมธธอส run() จะรับค่าคงตัวที่เก็บอยู่ในตัวแปร i ไปกำหนดให้ตัวแปร mile ซึ่งเป็นตัวแปรภายในแมธธอส run() ค่าของตัวแปร i จะไม่ถูกเปลี่ยนแปลงแต่อย่างใด ลองพิจารณาโปรแกรมต่อไปนี้

โปรแกรม 12 - 4 : TestMethod.java

```
class A {
        void x(int i){                  // (1)
                i = 10;
        }
}

public class TestMethod {
        public static void main (String[] args) {
                A a = new A();
                a.x(5);  // (2)
                int j = 6;
                a.x(j);  // (3)
                System.out.println("j = " + j);

        }
```

```
}
```

คลาส A ในโปรแกรมนี้มีแมธธอส x() เป็นสมาชิก ในบรรทัด (1) แมธธอส x() รับค่าตัวแปร i เข้าไปแล้วเปลี่ยนค่าของตัวแปรเสียใหม่เป็น 10

ในบรรทัด (2) มีการเรียกใช้แมธธอส x() โดยส่งค่าคงตัว 5 เข้าไปในแมธธอส

ในบรรทัด (3) มีการเรียกใช้แมธธอส x() อีกเหมือนกัน แต่ส่งผ่านตัวแปร j ซึ่งมีค่าเป็น 6 เข้าไปในแมธธอส แม้ว่าค่าของตัวแปร j จะถูกส่งผ่านไปยังตัวแปร i ซึ่งถูกกำหนดค่าให้กลายเป็น 10 ในแมธธอส แต่ตัวแปร j จะยังคงมีค่าเป็น 6 เหมือนเดิมไม่เปลี่ยนแปลง ดังจะเห็นได้จากผลการรันในภาพ

```
C:\java> java TestMethod
j = 6
```

ตัวแปรภายในแมธธอสสามารถถูกส่งออกมาภายนอกได้ด้วย ลองพิจารณาตัวอย่างโปรแกรมต่อไปนี้

โปรแกรม 12 - 5 : DriveACar.java

```
class Vehicle {

  int numberOfWheels;
  boolean hasEngine;

  boolean run(int mile){                      // (1)
    System.out.println("I am running on the " + mile + "th mile." );
    return (mile < 10);             // (2)
  }
}

public class DriveACar {
  public static void main (String[] args) {
    Vehicle myCar = new Vehicle();
    int i = 0;
    do {                            // (3)
       ++i;
    } while (myCar.run(i));         // (4)
  }
}
```

ในบรรทัด (1) เราเปลี่ยนคำว่า void เป็น boolean เมื่อใดก็ตามที่ต้องการส่งผ่านตัวแปรออกนอกแมธธอส เราจะเปลี่ยนจากคำว่า void หน้าชื่อแมธธอสเป็นชื่อชนิดของตัวแปรที่ต้องการส่งออก ส่วนค่าของตัวแปรนั้นเรากำหนดไว้ในบล๊อคปีกกาของแมธธอส โดยใช้คำสั่ง return ตัวอย่างเช่นในบรรทัด (2) ตัวแปรที่ส่งผ่านออกมาจะมีค่าเท่ากับค่าความจริงของวลี mile < 10

ในบรรทัด (3) เราเปลี่ยนคำสั่งการวนลูปจากคำสั่ง for มาเป็นคำสั่ง do-while ซึ่งคราวนี้เราสร้างตัวแปรนับจำนวนครั้งในการวนลูปขึ้นมาเอง เพราะคำสั่ง do-while ไม่สามารถสร้างได้เองเหมือนอย่างคำสั่ง for ตัวแปร i จะถูกเพิ่มขึ้นทีละหนึ่งจากค่าเริ่มต้นคือ 0 ทุกครั้งที่วนลูป และ โปรแกรมจะหลุดจากลูปก็ต่อเมื่อ ค่าความจริงในวงเล็บหลังคำสั่ง while เป็นเท็จ (ดูในบรรทัด (4)) ซึ่งในกรณีนี้ค่าความจริงในวงเล็บหลังคำสั่ง while ก็คือ myCar.run(i) ซึ่งก็คือค่าตัวแปร boolean ที่แมธธอส run() ส่งผ่านออกมา ค่าค่านี้จะเป็นจริงก็ต่อเมื่อ i มีค่าไม่เกิน 10 ผลที่ได้ก็คือรถยนต์ของเราจะวิ่งแค่ 25 ไมล์เท่านั้น อีกนัยหนึ่ง เรากำจัดการใช้งานรถยนต์ของเราไม่ให้วิ่งเกิน 10 ไมล์ด้วยการใช้การส่งผ่านตัวแปรออกนอกแมธธอส

ผลของการรันโปรแกรมข้างต้นเป็นดังนี้

```
C:\java> java TestMethod
I am running on the 1th mile.
I am running on the 2th mile.
I am running on the 3th mile.
I am running on the 4th mile.
I am running on the 5th mile.
I am running on the 6th mile.
I am running on the 7th mile.
I am running on the 8th mile.
I am running on the 9th mile.
I am running on the 10th mile.
```

การส่งผ่านตัวแปรเข้าไปในแมธธอส และการส่งผ่านตัวแปรออกจากแมธธอสสามารถส่งตัวแปรอ้างอิงได้ด้วย ตัวอย่างเช่น

โปรแกรม 12 - 6 : DriveACar.java

```
class Vehicle {
        int numberOfWheels;
```

```
        boolean hasEngine;
        void run(){
                System.out.println("I am running");
        }

}

class A {                                       //      (1)
        void x(Vehicle v) {
                v.numberOfWheels = 10;
        }
}

public class DriveACar {
        public static void main (String[] args) {
                Vehicle myCar = new Vehicle();
                A a = new A();
                a.x(myCar); //  (2)
                System.out.println(myCar.numberOfWheels);

        }
}
```

ในโปรแกรมนี้มีคลาสคลาสใหม่ชื่อคลาส A ในบรรทัด (1) คลาสนี้มีแมธธอสชื่อ x() ซึ่งส่งผ่านตัวแปรอ้างอิง v ซึ่งเป็นตัวแปรอ้างอิงของอินสแตนท์ของคลาส Vehicle สิ่งที่แมธธอส x() ทำก็คือเซตจำนวนล้อของ v ให้เท่ากับ 10

สิ่งที่น่าสังเกตก็คือในบรรทัด (2) มีการส่งผ่านตัวแปรอ้างอิง myCar เข้าไปในแมธธอส x() แม้ว่าการส่งผ่านตัวแปรเข้าไปในแมธธอสจะเป็นแต่เพียงการส่งผ่านค่า แต่อินสแตนท์ที่ myCar ชี้อยู่จะได้รับอิทธิพลของการเซตค่าจำนวนล้อในแมธธอส x() ด้วย ซึ่งต่างกับการส่งผ่านตัวแปรทั่วไปซึ่งค่าของตัวแปรที่ถูกส่งผ่านจะไม่ได้รับอิทธิพลใดๆ

ที่เป็นเช่นนี้ก็เพราะค่าของตัวแปรที่ส่งผ่านเข้าไปในแมธธอสในกรณีของตัวแปรอ้างอิงก็คือเลขชี้ตำแหน่งในแรม ซึ่งเมื่อตัวแปร v รับค่าของ myCar เข้าไป ตัวแปรอ้างอิง v จึงชี้ไปยังอินสแตนท์เดียวกันกับ myCar ในแรม ทำให้เมื่อมีการเปลี่ยนค่าของตัวแปร numberOfWheels จึงกระทบ myCar ด้วย ผลการรันโปรแกรมจึงเป็นดังภาพ

```
C:\java> java DriveACar
10
```

เคยกล่าวให้ทราบไปแล้วว่า อะเรย์ในภาษาจาวาเป็นวัตถุ ดังนั้นการส่งผ่านอะเรย์เข้าไปในแมธธอสก็จะกระทบค่าเดิมของอะเรย์ที่ถูกส่งผ่านด้วย เช่นเดียวกับในกรณีของอินสแตนท์

แมธธอสซ้อนแมธธอส

เราสามารถเรียกแมธธอสจากข้างในตัวของมันเองได้ เราเรียกโปรแกรมที่มีการเรียกแมธธอสซ้อนแมธธอสว่า **โปรแกรมรีเคอร์ซีฟ** ตัวอย่างที่น่าสนใจได้แก่การใช้โปรแกรมรีเคอร์ซีฟในการคำนวณค่าแฟกตอเรียล

ค่าแฟกตอเรียลของจำนวนเต็มตัวหนึ่งๆ ใช้สัญลักษณ์แทนด้วย `x!` เช่น ค่าแฟกตอเรียลของ 10 แทนด้วย `10!` ความหมายของมันคือผลคูณของจำนวนเต็มตั้งแต่ 1 จนถึงเลขตัวนั้น เช่น `10!` มีค่าเท่ากับ `1x2x3x4x5x6x7x8x9x10` เป็นต้น โปรแกรมตัวอย่างต่อไปนี้เป็นโปรแกรมหาค่า `12!`

โปรแกรม 12 - 7 : Factorial.java

```
class Number {

  long factorialValue(int i){       //  (1)
    if (i == 1)           //  (2)
      return 1;
    else                        //  (3)
        return i*factorialValue(i-1);
  }
}

public class Factorial {
  public static void main (String[] args) {
    int n = 12;
    System.out.println( n + "! = " + new Number().factorialValue(n));
        //   (4)
  }
}
```

คลาส `Number` มีแมธธอสชื่อ `factorialValue()` ซึ่งมีไว้สำหรับหาค่าแฟกตอเรียลโดยเฉพาะ แมธธอส `factorialValue()` คืนค่า 1 ถ้าตัวแปรส่งผ่าน `i` มีค่าเป็น 1 แต่ถ้า `i` มีค่าเป็นอื่นจะคืนค่า `i*factorValue(i-1)`

ในบรรทัด (4) เราเรียกแมธธอส factorialValue() โดยส่งผ่านค่า n = 12 เข้าไป factorialValue() พบว่าค่า i ไม่ใช่ 1 ดังนั้นมันควรจะคืนค่า i*factorialValue(i-1) หรือ 12*factorialValue(11) แต่มันไม่รู้ว่า factorialValue(11) มีค่าเท่าไร มันจึงเรียกตัวเองด้วยการส่งผ่านค่า 11 เข้าไป ซึ่งก็จะเจอปัญหาเดิมอีกเพราะ factorialValue(11) จะคืนค่า 11*factorialValue(10) มันจะเรียกตัวเองอีกไปเรื่อยๆ จนกว่า i จะเป็น 1 มันจึงหลุดออกจากการเรียกตัวเองได้ ซึ่งมันจะหาค่าย้อนกลับมาจนถึง 12 อีกครั้ง ได้ผลเป็นดังภาพ

```
C:\java> java Factorial
12! = 479001600
```

แมธธอสที่มีชื่อซ้ำกัน

ตัวแปรคลาสสองตัวห้ามมีชื่อซ้ำกัน แต่ในกรณีของแมธธอสสามารถมีชื่อซ้ำกันได้ เราเรียกว่า **การโอเวอร์โหลดแมธธอส** ลองดูตัวอย่างต่อไปนี้

โปรแกรม 12 - 8 : DriveACar.java

```
class Vehicle {

  int numberOfWheels;
  boolean hasEngine;

  void run(int mile){                  // (1)
    System.out.println("I am running on the " + mile + "th mile." );

  }

  void run(float mile){                 // (2)
    System.out.println("I am running on the " + mile + "th mile." );

  }
}

public class DriveACar {
  public static void main (String[] args) {
        Vehicle myCar = new Vehicle();
        myCar.run(10);          // (3)
        myCar.run(10.0F);           // (4)
  }
}
```

โปรแกรมนี้มีแมธธอสชื่อ run() เหมือนกันสองแมธธอสที่บรรทัด (1) และ (2) แต่ไม่มีปัญหาเพราะตัวแปรที่ส่งผ่านไม่เหมือนกัน แมธธอสในบรรทัด (1) ส่งผ่านตัวแปร mile ที่เป็นจำนวนเต็ม ส่วนบรรทัด (2) ส่งผ่านตัวแปรทศนิยม คอมไพล์เลอร์จะใช้ตัวแปรที่ส่งผ่านเป็นตัวแยกแยะความแตกต่าง ผลที่ได้จะเป็นดังภาพ

```
C:\java> java DriveACar
I am running on the 10th mile.
I am running on the 10.0th mile.
```

ในบรรทัด (3) เราเรียกแมธธอส run() โดยส่งผ่านเลขจำนวนเต็ม ดังนั้นจาวาเวอร์ชัวน์แมทชีนจะรันแมธธอสในบรรทัด (1) ส่วนในบรรทัด (4) เป็นการเรียกแมธธอส run() โดยส่งผ่านเลขทศนิยม จาวาจะรันแมธธอสในบรรทัด (2) สังเกตผลลัพธ์ที่ออกหน้าจอว่าจะไม่เหมือนกัน

ดังนั้นเราสามารถสร้างแมธธอสที่มีชื่อซ้ำกันได้ตราบใดที่การส่งผ่านตัวแปรของแต่ละแมธธอสไม่เหมือนกัน ตัวอย่างต่อไปนี้เป็นการโอเวอร์โหลดแมธธอสรันที่เป็นไปได้

```
void run() { /*  */ }
void run(int mile) { /*  */ }
void run(long mile) { /*  */ }
void run(int mile, boolean x) { /*  */ }
int run(int mile, long mile) { /*  */ }
long run(int mile, long mile, boolean x) { /*  */ }
```

แต่ห้ามโอเวอร์โหลดแมธธอสที่มีตัวแปรส่งผ่านในวงเล็บเหมือนกัน แต่ตัวแปรส่งกลับต่างกัน เพราะคอมไพล์เลอร์จะแยกความแตกต่างไม่ออกเวลาที่เจอแมธธอสนี้ที่อื่นในโปรแกรม ตัวอย่างต่อไปนี้เป็นแมธธอสสองแมธธอสที่อยู่ร่วมกันไม่ได้

```
void run(int mile) { /*  */ }
int run(int x) { /*  */ }          // Error
```

ในกรณีที่ตัวแปรส่งผ่านในแมธธอสหนึ่งเป็นจำนวนเต็มแบบ int อีกแมธธอสหนึ่งเป็นจำนวนเต็มแบบ long เวลาเรียกใช้แมธธอสต้องใส่ L ต่อท้ายตัวส่งผ่านเสมอเพื่อให้ทราบว่าเป็นการส่งตัวแปรแบบ long ด้วย มิฉะนั้นจาวาเวอร์ชัวน์แมทชีนจะคิดว่าต้องการให้เรียกแมธธอสที่มีตัวแปรส่งผ่านแบบ int แม้ว่าปกติแล้วค่าคงตัวจำนวนเต็มที่ไม่มี L ต่อท้ายจะใช้ได้กับตัวแปรแบบ long ก็ตาม ตัวอย่างเช่น ถ้าเรามีแมธธอสสองแมธธอสนี้ในคลาสเดียวกัน

```
boolean run(int mile) { /*  */ }
boolean run(long mile) { /*  */ }
```

เวลาเรียกใช้ต้องเรียกต่างกันดังนี้

```
myCar.run(10);
myCar.run(10L);
```

ถ้าตัวแปรส่งผ่านเป็น `float` กับ `double` ก็ต้องแยกความแตกต่างในทำนองเดียวกัน

คำสั่ง this

คงยังจำได้ว่าตัวแปรคลาส กับ ตัวแปรท้องถิ่นในแมธธอสในคลาสเดียวกัน สามารถมีชื่อซ้ำกันได้ เพราะตัวแปรคลาสเกิดขึ้นเมื่อมีการสร้างอินสแตนท์ แต่ตัวแปรท้องถิ่นเกิดขึ้นเมื่อมีการเรียกใช้แมธธอสและตายไปทันทีที่แมธธอสรันเสร็จ จาวาจึงแยกแยะความแตกต่างระหว่างตัวแปรทั้งสองตัวได้

แต่บางครั้งก็เกิดการสับสน คำสั่ง `this` จึงมีไว้แยกแยะความแตกต่างให้ชัดเจนไปเลย ตัวอย่างเช่น

โปรแกรม 12 - 9 : TestThis.java

```
class TestThis {

  int a;                              // (1)
  float b;                            // (2)
  char c;                             // (3)

          void x(int a, float b,char i){      // (4)
           char c;                            // (5)
           this.a = a;                // (6)
           c = i;                             // (7)
           this.y();                  // (8)
           y();                               // (9)
        }

          void y() {                 // (10)
           System.out.println("Hello World");
          }
        }
```

คลาสนี้มีตัวแปรคลาสสามตัวได้แก่ a,b และ c ในบรรทัด (1)(2)(3) แมธธอส x() มีการส่งผ่านตัวแปรที่มีชื่อเหมือนตัวแปรคลาส a และ b ในบรรทัด (4) อีกทั้งยังประกาศตัวแปรภายในแมธธอสชื่อ c ซึ่งซ้ำกับตัวแปรคลาสอีก ในบรรทัด (5)

คำสั่งในบรรทัด (6) ใช้คำสั่ง this ตามด้วยจุดนำหน้า a เป็นการชี้เฉพาะเจาะจงลงไปว่าหมายถึงตัวแปรคลาส a ที่ประกาศไว้ในบรรทัด (1) คำสั่งนี้เป็นการกำหนดค่าตัวแปรคลาส a ให้มีค่าเท่ากับตัวแปรส่งผ่าน a ซึ่งอยู่หลังเครื่องหมายเท่ากับ ตัวแปรส่งผ่านไม่มีคำว่า this นำหน้า

คำสั่งในบรรทัด (7) เป็นการอ้างถึงตัวแปร c ที่ประกาศไว้ในบรรทัด (5) เพราะไม่มี this นำหน้า สรุปก็คือ ถ้ามีการใช้ชื่อซ้ำกัน การใช้ชื่อตัวแปรเฉยๆ จะถือว่าเป็นตัวแปรภายในแมธธอส ถ้าต้องการอ้างถึงตัวแปรคลาสต้องระบุ this ด้วยเสมอ แต่ถ้าไม่มีการใช้ชื่อซ้ำกัน ไม่ต้องระบุ this ก็ได้เหมือนกันที่เราเคยทำมาในอดีต

คำสั่ง this ใช้ระบุเฉพาะเจาะจงได้ด้วยว่าหมายถึงแมธธอสในคลาสเดียวกันดังที่ใช้ในบรรทัด (8) แต่เนื่องจากไม่มีความสับสนในกรณีของแมธธอสอยู่แล้ว (เพราะไม่มีแมธธอสท้องถิ่น) การระบุ this จึงไม่จำเป็นสำหรับการเรียกแมธธอส บรรทัด (8) กับ บรรทัด (9) จึงมีความหมายเหมือนกัน

แมธธอสสแตรติค

แมธธอสสแตรติค คือ แมธธอสที่มีแค่ชุดเดียวเท่านั้นต่อหนึ่งคลาส และจะเกิดขึ้นเมื่อตอนที่คลาสนั้นถูกโหลดเข้าไปในแรม คล้ายๆ กับตัวแปรสแตรติค ข้อดีของแมธธอสสแตรติคก็คือ เราสามารถเรียกใช้แมธธอสสแตรติคได้โดยไม่ต้องมีการสร้างอินสแตนท์ โดยมากแล้วเราสร้างแมธธอสสแตรติคไว้เป็นแมธธอสสนับสนุน เช่น การหาค่าทางคณิตศาสตร์ต่างๆ ตัวอย่างต่อไปนี้เป็นการสร้างแมธธอสสแตรติคเอาไว้ใช้ช่วยหาค่ายกกำลังสาม

โปรแกรม 12 - 10 : TestMethod.java

```
class Arithmetic {
        static int triplePower(int i) {
                return i*i*i;
        }
```

```
}

public class TestMethod {
  public static void main (String[] args) {
        System.out.println( “10 raised to the power of three is “ +
Arithmetic.triplePower(10) +”.”);
        System.out.println( “20 raised to the power of three is “ +
Arithmetic.triplePower(20) +”.”);
        System.out.println( “30 raised to the power of three is “ +
Arithmetic.triplePower(30) +”.”);
  }
}
```

ผลการรันโปรแกรมข้างต้นเป็นดังนี้

```
C:\java> java TestMethod
10 raised to the power of three is 1000.
20 raised to the power of three is 8000.
30 raised to the power of three is 27000.
```

คลาส Arithmetic มีแมธธอสสแตรติคหนึ่งตัวชื่อ triplePower() ซึ่งคืนค่ายกกำลังสามของตัวแปรที่ส่งเข้ามา แมธธอสนี้ถูกนำไปใช้หาค่ายกกำลังสามของ 10 20 และ 30 ในแมธธอส main()

เราสามารถกำหนดให้แมธธอสในคลาสเป็น แมธธอสสแตรติค ได้ด้วยการใช้คำสั่ง static นำหน้า เมื่อเวลาจะเรียกใช้แมธธอสสแตรติคเราเรียกใช้ได้เลยโดยเรียกชื่อคลาสตามด้วยจุดตามด้วยชื่อแมธธอสโดย ไม่ต้องมีการสร้างอินสแตนท์ก่อน

นี่เป็นเหตุผลที่ทำไมจึงมีคำว่า static นำหน้าแมธธอส main() เพราะทำให้เราไม่ต้องสร้างอินสแตนท์ของคลาสหลักของโปรแกรมของเรา เราเริ่มรันโปรแกรมของเราด้วยการรันแมธธอส main() ได้เลย เพราะมันเป็นแมธธอสสแตรติค

ภายในตัวแมธธอสสแตรติค ห้ามมีการอ้างถึงตัวแปรคลาสหรือแมธธอสที่ไม่สแตรติคเป็นอันขาด เพราะแมธธอสสแตรติคทำงานได้โดยที่ไม่ต้องมีการสร้างอินสแตนท์ แต่ตัวแปรคลาสและแมธธอสที่ไม่สแตรติคจะเกิดขึ้นหลังการสร้างอินสแตนท์แล้วเท่านั้น

ถ้าจำเป็นต้องมีการอ้างถึงตัวแปรคลาสหรือแมธธอสที่ไม่สแตรติคจริงๆ ภายในแมธธอสสแตรติค สามารถทำได้โดยการสร้างอินสแตนท์ขึ้นชั่วคราว ดังตัวอย่างต่อไปนี้

โปรแกรม 12 - 11 : A.java

```
class A {
        int x;
        static int y;
        void m() { };
        static void n() { };
        static void o() {

                int j = y;
                n();
                int i = new A().x;
                new A().m();

  }
}
```

ในโปรแกรมนี้คลาส A มีตัวแปรคลาสสองตัวคือ x และ y แต่ y เป็นตัวแปรสแตรติค มีแมธธอสสามแมธธอสคือ m() n() และ o() โดยที่ n() และ o() สแตรติค

แมธธอส o() เรียกใช้งานตัวแปร y และแมธธอส n() ได้เลยเพราะเป็นสแตรติคเหมือนกับตน ส่วน x และ m() ต้องเรียกผ่านการสร้างอินสแตนท์ชั่วคราวขึ้นมา

นอกจากนี้ยังห้ามใช้คำสั่ง this ในแมธธอสสแตรติคด้วย เพราะคำว่า this แทนชื่อของอินสแตนท์ แต่แมธธอสแตรติคทำงานโดยไม่มีอินสแตนท์

13

คอนสตรัคเตอร์

คงจำได้ว่าถ้าไม่มีการกำหนดค่าเริ่มต้นของตัวแปรคลาสไว้ เวลาสร้างอินสแตนท์ จาวาเวอร์ชัวน์แมทชีน จะกำหนดค่าปกติให้ การกำหนดค่าเริ่มต้นทำได้ด้วยการกำหนดเข้าไปเลยตอนนิยามคลาส ดังตัวอย่างในโปรแกรม 10 –1

จาวามีแมธธอสพิเศษชนิดหนึ่งชื่อว่า **คอนสตรัคเตอร์** ซึ่งเป็นแมธธอสที่มีไว้สำหรับการกำหนดค่าเริ่มต้นให้กับตัวแปรอินสแตนท์โดยเฉพาะ คอนสตรัคเตอร์เป็นทางเลือกอีกทางหนึ่งของการกำหนดค่าให้ตัวแปรคลาส ลองพิจารณาการกำหนดค่าเริ่มต้นให้ตัวแปรคลาสด้วยการใช้คอนสตรัคเตอร์ดังตัวอย่างข้างล่างนี้

โปรแกรม 13 - 1 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

   Vehicle() {                    // (1)
                numberOfWheels = 6;
                hasEngine = true;
        }

        void run(){
                System.out.println("I am running");
        }
```

```
}

public class BuildACar {
  public static void main (String[] args) {

    Vehicle myCar = new Vehicle();
    System.out.println("My car has  " + myCar.numberOfWheels + "
wheels.");
    System.out.println("That my car has an engine is " +
myCar.hasEngine + ".");
  }
}
```

แมธธอสคอนสตรัคเตอร์คือแมธธอสในบรรทัด (1)แมธธอสคอนสตรัคเตอร์ต้องมีชื่อเหมือนชื่อคลาสเสมอ แต่ตามด้วย () เพื่อให้รู้ว่าเป็นแมธธอส และต้องไม่มีคำว่า void หรือชื่อชนิดของตัวแปรใดๆ นำหน้าชื่อคอนสตรัคเตอร์

ภายในบล๊อคปีกกาของคอนสตรัสเตอร์เราใส่คำสั่งกำหนดค่าของตัวแปรอินสแตนท์ลงไป เมื่อไรก็ตามที่มีการสร้างอินสแตนท์ใหม่ของคลาส Vehicle ด้วยคำสั่ง

```
Vehicle myCar = new Vehicle();
```

คำสั่งกำหนดค่าตัวแปรอินสแตนท์ที่อยู่ในคอนสตรัคเตอร์จะทำงานทันที

อันที่จริงคอนสตรัคเตอร์ก็เหมือนกับแมธธอสทั่วไป คำสั่งที่อยู่ในบล๊อคปีกกาไม่จำเป็นต้องเป็นคำสั่งกำหนดค่าตัวแปรคลาสเสมอไป จะเป็นคำสั่งอย่างอื่นก็ได้ คำสั่งเหล่านี้จะทำงานทุกครั้งที่มีการสร้างอินสแตนท์ใหม่ให้คลาส ตัวอย่างเช่น

โปรแกรม 13 - 2 : BuildACar.java

```
class Vehicle {

   int numberOfWheels;
   boolean hasEngine;

   Vehicle() {
    numberOfWheels = 6;
    hasEngine = true;
    System.out.println("A car has been built."); //(1)
   }

   void run(){
    System.out.println("I am running");
   }
}
```

```
public class BuildACar {
  public static void main (String[] args) {

    Vehicle myCar = new Vehicle();
    System.out.println("My car has  " + myCar.numberOfWheels + "
wheels.");
    System.out.println("That my car has an engine is " +
myCar.hasEngine + ".");
  }
}
```

ในบรรทัด (1) เราเติมคำสั่งให้โปรแกรมแสดงผลออกนอกจอว่า A car has been built ดังนั้นคำสั่งนี้จะถูกใช้งานเมื่อมีการสร้างอินสแตนท์ myCar ขึ้น ซึ่งเกิดขึ้นก่อนการแสดงจำนวนล้อ ดังภาพข้างล่างนี้

```
C:\java> java BuildACar
A car has been built.
My car has 6 wheels.
That my car has an engine is true.
```

คอนสตรัคเตอร์เป็นแมธธอส ดังนั้นคอนสตรัคเตอร์สามารถส่งผ่านตัวแปรได้ด้วย เช่น

โปรแกรม 13 - 4 : BuildACar.java

```
class Vehicle {

  int numberOfWheels;
  boolean hasEngine;

  Vehicle() {
    numberOfWheels = 6;
    hasEngine = true;
    System.out.println("A car has been built.");
  }

   Vehicle(int number, boolean engine) {               // (1)
     numberOfWheels = number;
     hasEngine = engine;
     System.out. println("A car has been built.");
   }

   void run(){
       System.out.println("I am running");
   }
}

public class BuildACar {
  public static void main (String[] args) {

    Vehicle myCar = new Vehicle(4,true);     //(2)
```

```
    System.out.println("My car has  " + myCar.numberOfWheels + "
wheels.");
    System.out.println("That my car has an engine is " +
myCar.hasEngine + ".");
  }
}
```

ในตัวอย่างนี้เราเพิ่มคอนสตรัคเตอร์ที่มีการส่งผ่านตัวแปรในบรรทัด (1) ซึ่งเป็นการโอเวอร์โหลดคอนสตรัสเตอร์ที่มีอยู่เดิม

ในบรรทัด (2) เราสร้างอินสแตนท์โดยใช้คอนสตรัคเตอร์ตัวใหม่ ซึ่งมีการส่งผ่านตัวแปรด้วย ผลที่ได้ก็คือตัวแปรอินสแตนท์ทั้งสองจะมีค่าเหมือนกับตัวแปรที่เราส่งผ่าน ดังในภาพ

```
C:\java> java BuildACar
A car has been built.
My car has 4 wheels.
That my car has an engine is true.
```

รถยนต์มี 4 ล้อแทนที่จะมี 6 ล้อ

เราเรียกคอนสตรัคเตอร์ที่ไม่มีการส่งผ่านตัวแปรใดๆ ว่า **คอนสตรัสเตอร์ปกติ** และเรียกคอนสตรัคเตอร์ที่มีการส่งผ่านตัวแปรว่า **คอนสตรัคเตอร์โอเวอร์โหลด**

คำสั่ง this()

ภายในคอนสตรัคเตอร์เราสามารถเรียกคอนสตรัคเตอร์ตัวอื่นในคลาสเดียวกันได้ด้วยการใช้คำสั่ง this() ดังตัวอย่างต่อไปนี้

โปรแกรม 13 - 5 : TestThis.java

```
class A {
        int a;                          // (1)
        A() {
                a = 0;
                System.out.println("Default Constructor");
        }
        A(int i) {    // (2)
                this();
                a = i;
```

```
                System.out.println("Constructor 1");
        }

        A(int i,int j) {  // (3)
                this(i+j);
                System.out.println("Constructor 2");
        }
}

public class TestThis {
  public static void main(String[] args) {
    System.out.println("Instantiate x");
    A x = new A();
    System.out.println("variable a is " + x.a);

    System.out.println("Instantiate y");
    A y = new A(5);
    System.out.println("variable a is " + y.a);

    System.out.println("Instantiate z");
    A z = new A(5,6);
    System.out.println("variable a is " + z.a);
  }
}
```

คลาส A มีสามคอนสตรัคเตอร์ คอนสตรัสเตอร์แรกเป็นคอนสตรัสเตอร์ปกติในบรรทัด (1) กำหนดค่าให้ตัวแปรคลาส a เป็นศูนย์ คอนสตรัคเตอร์ที่หนึ่งในบรรทัด (2) มีการรับตัวแปร i และ มีการเรียกคอนสตรัคเตอร์ปกติด้วยคำสั่ง this() ก่อนที่จะเปลี่ยนค่าตั้งต้นของ a ให้เท่ากับตัวแปร i ส่วนคอนสตรัคเตอร์ตัวสุดท้ายคือคอนสตรัคเตอร์ที่สองในบรรทัด (3) มีการเรียกคอนสตรัสเตอร์ที่หนึ่งด้วยคำสั่ง this(i+j) มันจะส่งผ่านค่า i+j ไปให้ a และเนื่องจากมันเรียกคอนสตรัคเตอร์ที่หนึ่งซึ่งเรียกคอนสตรัคเตอร์ปกติ คอนสตรัคเตอร์ปกติจึงถูกเรียกไปด้วย ลองพิจารณาผลการรันดังภาพ

```
C:\java> java TestThis
Instatiate x
Default Constructor
variable a is 0
Instatiate y
Default Constructor
Constructor 1
variable a is 5
Instatiate z
Default Constructor
Constructor 1
Constructor 2
variable a is 11
```

จำไว้เสมอว่าคำสั่ง `this()` ใช้ได้ภายในคอนสตรัคเตอร์เท่านั้นและต้องเป็นคำสั่งแรกสุดเสมอ

14

ตัวแปรสตริง

ถึงตอนนี้เราเข้าใจความหมายของคลาสและวัตถุในภาษาจาวาพอสมควรแล้ว เพื่อให้เห็นภาพของการนำไปใช้งานมากขึ้น มาลองรู้จักคลาสคลาสหนึ่งที่มีมาให้อยู่แล้วในภาษาจาวา คือคลาส String

คงยังจำได้ว่าค่าคงตัวในภาษาจาวามีหลายประเภท แต่ละประเภทจะมีชนิดของตัวแปรที่สอดคล้องกับมันอยู่ เช่นค่าคงตัวที่เป็นตัวเลขจำนวนเต็มแบบ 4 ไบต์ จะเข้ากันได้กับตัวแปร int หรือค่าคงตัวแบบตัวอักษร จะเข้ากันได้กับตัวแปรแบบ char แต่มีค่าคงตัวอยู่ประเภทหนึ่งที่เรายังไม่เคยกล่าวถึงตัวแปรที่สอดคล้องกับมัน นั้นคือ ค่าคงตัวแบบข้อความ ตัวแปรที่สอดคล้องกับค่าคงตัวแบบข้อความในภาษาจาวาถูกสร้างขึ้นจากคลาสที่ชื่อ String นี่เอง

การประกาศตัวแปรอ้างอิงสำหรับคลาส String ทำได้ดังตัวอย่างต่อไปนี้

```
String string1;
```

สมมติว่าเราต้องการให้อินสแตนท์ของคลาส String เก็บค่าคงตัวแบบข้อความว่า “Hello World” เราทำได้ด้วยการเรียกคอนสตรัคเตอร์ดังนี้

```
string1 = new String(“Hello World”);
```

คอนสตรัคเตอร์ของคลาส String มีการส่งผ่านตัวแปรแบบค่าคงตัว ซึ่งก็คือค่าคงตัวแบบข้อความที่เราต้องการเก็บนั้นเอง คอนสตรัคเตอร์แบบไม่มีการส่งผ่านตัวแปรของคลาส String ก็มีเหมือนกันดังนี้

```
string1 = new String();
```

ในกรณีนี้เราจะได้อินสแตนท์ของคลาส String ที่ไม่ได้เก็บข้อความอะไรเลย
มีวิธีกำหนดค่าอีกแบบหนึ่งซึ่งเป็นที่นิยมใช้มากกว่าแม้จะผิดหลักของการกำหนดค่าตัวแปรคลาส คือการจับตัวแปรอ้างอิงแบบสตริงให้เท่ากับค่าคงตัวแบบข้อความเลย ดังนี้

```
string1 = “Hello World”;
```

ตัวแปรอ้างอิงไม่น่าจะจับให้เท่ากับค่าคงตัวได้ แต่กรณีนี้เป็นกรณีพิเศษสำหรับตัวแปรสตริงโดยเฉพาะ
ลองดูตัวอย่างการใช้งานตัวแปรสตริงดังต่อไปนี้

โปรแกรม 14 - 1 : TestString.java

```
public class TestString {
        public static void main(String[] args) {
                String s = “Hello World”; // (1)
                System.out.println(s);

        }
}
```

โปรแกรมนี้ให้ผลเหมือนโปรแกรม Hello World ทุกประการ เพียงแต่คราวนี้เราใช้ตัวแปรสตริงแทนการใช้ค่าคงตัวแบบข้อความในการแสดงผลออกหน้าจอ

ในบรรทัด (1) จะเห็นได้ว่าเราใช้งานคลาส String ราวกับว่ามันเป็นตัวแปรชนิดหนึ่งชื่อสตริงเลย ถึงตอนนี้คุณอาจสงสัยว่าทำไมจาวาจึงไม่สร้างตัวแปรแบบสตริงในลักษณะเดียวกับตัวแปรแบบอื่นๆ ทำไมต้องสร้างเป็นคลาสด้วย ข้อดีอย่างหนึ่งของการสร้างให้เป็นคลาสก็คือเราสามารถสร้างแมธธอสขึ้นมาช่วยการจัดการตัวแปรแบบสตริง คลาสสตริงมีแมธธอสต่อไปนี้พ่วงมาด้วย

int length()

แมธธอส `length()` คืนค่าเป็นตัวแปรจำนวนเต็มซึ่งมีค่าเท่ากับความยาวของค่าคงตัวแบบข้อความที่สตริงเก็บอยู่

`char charAt(int index)`

แมธธอส `charAt()` รับค่าเป็นตัวเลขจำนวนเต็มแล้วคืนค่าเป็นตัวแปรแบบอักษรซึ่งเท่ากับตัวอักษรในลำดับที่เท่ากับตัวเลขจำนวนเต็มตัวนั้นในข้อความ

ลองดูตัวอย่างการใช้งานแมธธอสทั้งสองดังต่อไปนี้

โปรแกรม 14 - 2 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
    String s = “Hello World”; // (1)
    System.out.println(“S is “ + s.length() + “.letter long.”);
    System.out.println(“the fourth letter is “ + s.charAt(3) + “.”);

  }
}
```

ตัวอย่างนี้เป็นการวัดความยาวของข้อความด้วยการใช้แมธธอส `length()` และหาว่าตัวอักษรตัวที่สี่ของข้อความคือตัวอะไร สังเกตว่าจำนวนเต็มที่ส่งผ่านเข้าไปในแมธธอส `charAt` มีค่าเป็น 3 เนื่องจากอักษรตัวแรกจะเริ่มนับเป็นตำแหน่งที่ 0 แทนที่จะเป็นตำแหน่งที่ 1

`boolean equals(String s)`
`boolean equalsIgnoreCase(String s)`

แมธธอส `equals()` รับค่า สตริงตัวอื่นเข้าไปแล้วตรวจสอบว่ามีข้อความเหมือนกับสตริงที่เราใช้อยู่หรือเปล่า ในขณะที่แมธธอส `equalsIgnoreCase()` จะตรวจสอบโดยไม่สนใจเรื่อตัวพิมพ์เล็กตัวพิมพ์ใหญ่ ลองดูตัวอย่างการใช้งานดังตัวอย่างต่อไปนี้

โปรแกรม 14 - 3 : TestString.java

```
        public class TestString {
          public static void main(String[] args) {
            String s1 = “Hello World”;
            String s2 = new String(“Hello World”);
            String s3 = “hello world”;
            System.out.println(s1.equals(s2));  // (1)
System.out.println(s1.equalsIgnoreCase(s3); // (2)
            System.out.println(s1 == s2); // (3)
          }
        }
```

วลีในบรรทัด (1) ให้ค่าเป็น true เพราะตัวแปร s1 และ s2 เก็บข้อความเหมือนกัน ส่วนวลีในบรรทัด (3)ก็ให้ค่าเป็น true ด้วย เพราะแม้จะเขียนตัวพิมพ์เล็กใหญ่ต่างกันเราใช้แมธธอส equalsIgnoreCase() ในการเปรียบเทียบ

ในบรรทัด (3) เราเปรียบเทียบด้วยการใช้เครื่องหมาย == ผลที่ได้คือ false เพราะเครื่องหมาย == จะให้ผลเป็น true ก็ต่อเมื่อตัวแปรอ้างอิงทางซ้ายชี้อินสแตนท์เดียวกันกับตัวแปรอ้างอิงที่อยู่ทางขวา แม้ว่า s1 กับ s2 จะเก็บข้อความเดียวกัน แต่เป็นคนละอินสแตนท์กัน อยู่ในที่ต่างกันในแรม ดังนั้นผลการเปรียบเทียบจะเป็น false

ตอนนี้คุณอาจสงสัยว่าแมธธอส equals ที่จริงแล้วสืบทอดมาจากคลาส Object มันควรจะให้ผลเหมือนเครื่องหมาย == แต่ในกรณีนี้กลับมีความแตกต่างกันอยู่ ที่เป็นเช่นนี้เป็นเพราะคลาส String มีการโอเวอร์รายด์แมธธอส equals() นั่นเอง

int compareTo(String s)

แมธธอส compareTo() เปรียบเทียบลำดับในพจนานุกรมระหว่างข้อความในสตริง s กับตัวมันเอง ถ้าข้อความเหมือนกันจะคืนค่า 0 ออกมา แต่ถ้าน้อยกว่า 0 แสดงว่าสตริงนี้มาก่อนสตริง s ในพจนานุกรม และคืนค่ามากกว่า 0 ถ้าสตริงนี้มาหลังสตริง s ในพจนานุกรม ลองดูตัวอย่างต่อไปนี้

โปรแกรม 14 - 4 : TestString.java

```
public class TestString {
        public static void main(String[] args) {
                String s1 = “abba”;
                String s2 = “abab”;
                System.out.println(s1.compareTo(s2));
        }
}
```

String toUpperCase()
String toLowerCase()

แมธธอส toUpperCase() คืนค่าสตริงตัวเดิม แต่เปลี่ยนให้เป็นตัวพิมพ์ใหญ่ทั้งหมด ส่วนแมธธอส toLowerCase() คืนค่าที่เป็นตัวพิมพ์เล็ก

String concat(String s)

แมธธอส concat คืนค่าของสตริงแต่ต่อท้ายด้วยค่าของสตริง s หนึ่งอีกนัยหนึ่งก็คือให้ผลเหมือนกันเครื่องหมาย + นั่นเอง

String trim()

แมธธอส trim() คือค่าเดิมของสตริง โดยที่ถ้ามีช่องว่างที่ต้นข้อความหรือท้ายข้อความจะทำการตัดออก

String substring(int startIndex)
String substring(int startIndex, int endIndex)

แมธธอส subString() คือค่าเดิมของสตริงโดยตัดตัวอักษรที่อยู่หน้าตัวอักษรตั้งแต่ตำแหน่งที่ startIndex ออก แมธธอสนี้มีการโอเวอร์โหลดด้วย กล่าวคือถ้าระบุค่า endIndex ด้วยจะคืนค่าเฉพาะข้อความที่อยู่ระหว่างตำแหน่ง startIndex ถึง endIndex

ลองดูตัวอย่างการใช้งานแมธธอสเหล่านี้ดู

โปรแกรม 14 - 5 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
    String s1 = “ Hello World”; // (1)
    String s2 = “ 2000”; // (2)
    System.out.println(s1.toUpperCase());
    System.out.println(s1.toLowerCase());
    System.out.println(s1.concat(“ 2000 ”));
    System.out.println(s1.concat(“ 2000 “).trim());
    System.out.println(s1.substring(2,5));
  }
}
```

โปรดสังเกตช่องว่างหน้าค่าคงตัวแบบข้อความในบรรทัด (1) (2)

ผลการรันโปรแกรมข้างต้นเป็นดังนี้

```
C:\java> java TestString
 HELLO WORLD
 hello world
 Hello World 2000
Hello World 2000
ell
```

สังเกตว่าแมธธอส substring(2,5) จะรวมตัวอักษรตำแหน่ง 2 แต่กลับตัดตัวอักษรตำแหน่ง 5 ออก

```
int indexOf(int ch)
int indexOf(int ch, int fromIndex)
int indexOf(String s)
int indexOf(String s, int fromIndex)
```

แมธธอส indexOf() คืนค่าตำแหน่งของตัวอักษร ch ในสตริง และทันทีที่หาเจอเป็นครั้งแรกมันจะหยุดหาทันที ถ้าไม่พบมันจะคืนค่า –1 เราสามารถกำหนดให้เริ่มหาตั้งแต่ตำแหน่ง fromIndex แทนที่จะหาตั้งแต่ต้นได้ด้วย นอกจากนี้เราอาจให้หาข้อความก็ได้ด้วยการส่งค่าตัวแปร s

```
int lastIndexOf(int ch)
int lastIndexOf(int ch, int fromIndex)
int lastIndexOf(String s)
int lastIndexOf(String s, int fromIndex)
```

แมธธอส indexOf() เริ่มหาจากต้นจนจบ แต่แมธธอส lastIndexOf() จะหาจากท้ายมาต้น

```
String replace(char old, char new)
```

แมธธอส replace() คืนค่าเดิมของสตริงโดยเปลี่ยนตัวอักษร old ทุกตัวในข้อความเป็น new

ลองดูตัวอย่างการใช้งานดังต่อไปนี้

โปรแกรม 14 - 6 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
        String s1 = “Hello Hello”;
        System.out.println(s1.indexOf(‘o’));
        System.out.println(s1.indexOf(“lo”,7));
        System.out.println(s1.lastIndexOf(‘e’));
        System.out.println(s1.replace(‘o’,’a’));
  }
}
```

ผลการรันโปรแกรมเป็นดังนี้

```
C:\java> java TestString
4
9
7
Hella Hella
```

static String valueOf(int i)
static String valueOf(long l)
static String valueOf(float f)
static String valueOf(double d)
static String valueOf(boolean b)
static String valueOf(char c)

แมธธอส `valueOf()` เปลี่ยนค่าของตัวแปรในวงเล็บให้เป็นตัวแปรสตริง สังเกตว่าแมธธอสนี้เป็นแมธธอสสแตรติคดังนั้นสามารถนำไปใช้ได้โดยไม่ต้องสร้างอินสแตนท์ของคลาส `String`

static String valueOf(Object obj)

แมธธอส `valueOf()` สามารถคืนค่าที่เป็นชื่อของคลาส `obj` ได้ด้วย

static String valueOf(char[] character)

ถ้าเรามีอะเรย์ของ `char` เราสามารถใช้แมธธอส `valueOf()` เปลี่ยนให้เป็นสตริงได้ โดยมันจะจับตัวอักษรมาเรียงต่อกันตามลำดับในอะเรย์

ลองดูตัวอย่างการใช้งาน

โปรแกรม 14 - 7 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
        boolean b1 = false;
        char ch1 = 'x';
        Object o = new Object();
        char[] ch2 = {'a','b','b','a'};
        System.out.println(String.valueOf(5));
        System.out.println(String.valueOf(5L));
        System.out.println(String.valueOf(5.0));
        System.out.println(String.valueOf(b1));
        System.out.println(String.valueOf(ch1));
        System.out.println(String.valueOf(o));
        System.out.println(String.valueOf(ch2));
  }
}
```

```
C:\java> java TestString
5
5
5.0
false
x
java.lang.Object@273d3c
abba
```

คลาส StringBuffer

ถ้าสังเกตดูให้ดีจะเห็นว่าแมธธอสในคลาส String ทำหน้าที่แค่ส่งผ่านสตริงที่ถูกดัดแปลงออกมาให้เท่านั้น ไม่ได้ดัดแปลงตัวสตริงเองแต่อย่างใด ตัวอย่างเช่น

โปรแกรม 14 - 8 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
        String s = “Hello World”;
        System.out.println(s.substring(1,10)); // (1)
        System.out.println(s); // (2)
  }
}
```

ในบรรทัด (1) เป็นการแสดงค่าของสตริง s ที่ตัดหัวตัดหางออกอย่างละหนึ่งตัวอักษร ผลที่ได้คือ ello Worl แต่เมื่อแสดงค่าของตัวแปร s ในบรรทัด (2) ใหม่จะเห็นว่ามีค่าเป็น Hello World อยู่เหมือนเดิม

คลาส StringBuffer คล้ายกับคลาส String เพียงแต่เป็นตัวแปรสตริงแบบที่มีแก้ไขค่าแบบถาวร คอนสตรัคเตอร์ของคลาส StringBuffer มีดังนี้

StringBuffer(String s)
ตัวแปร s คือข้อความที่ต้องการให้เก็บไว้ในตัวแปร StringBuffer

StringBuffer()
แบบนี้จะได้ข้อความว่างเปล่า แต่ขนาดของมันจะไม่เป็น 0 แต่เป็น 16

StringBuffer(int length)
แบบนี้ได้ข้อความว่างเปล่าเหมือนกัน แต่กำหนดขนาดได้ด้วยตัวแปร length

ข้อแตกต่างที่สำคัญของคลาส StringBuffer กับคลาส String อีกอันหนึ่งก็คือ ขนาดของ StringBuffer ไม่จำเป็นต้องเท่ากับขนาดของข้อความ และสามารถเพิ่มหรือลดขนาดได้ เราสามารถตรวจสอบขนาดของ StringBuffer ได้ด้วยแมธธอส capacity()

int capacity()
แมธธอส capacity()คืนค่าขนาดปัจจุบันของ StringBuffer

ลองดูตัวอย่างต่อไปนี้

โปรแกรม 14 - 9 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
        StringBuffer s1 = new StringBuffer();// (1)
        System.out.println(s1);
        System.out.println(s1.length());
        System.out.println(s1.capacity());
        StringBuffer s2 = new StringBuffer("Hello"); // (2)
        System.out.println(s2);
        System.out.println(s2.length());
        System.out.println(s2.capacity());
        //StringBuffer s3 = "Hello";  // (3) Error
  }
}
```

ผลการรันเป็นดังนี้

```
C:\java> java TestString

0
16
Hello
5
21
```

ในบรรทัด (1) เราสร้างตัวแปร s1 โดยไม่เก็บค่าอะไรเลย ผลที่ได้คือขนาดของข้อความเท่ากับ 0 แต่ขนาดของตัวมันเองจริงๆ เท่ากับ 16 ซึ่งเป็นค่าปกติของมัน

ในบรรทัด (2) เราสร้างตัวแปร s2 โดยให้เก็บข้อความ Hello ผลที่ได้คือขนาดข้อความจะเป็น 5 แต่ขนาดของตัวมันเองจะเท่ากับค่าปกติบวกด้วยความยาวของข้อความที่สั่งให้มันเก็บ หรือเท่ากับ 21

ในบรรทัด (3) เราพยายามสร้างตัวแปร s3 โดยใช้วิธีเดียวกับเวลาสร้างตัวแปรสตริง วิธีนี้ใช้ไม่ได้กับตัวแปร StringBuffer

เหตุที่ตัวแปร StringBuffer มีขนาดปกติ 16 และจะเพิ่มขนาดเมื่อเก็บข้อความลงไป เป็นเพราะถ้ามีการเปลี่ยนข้อความที่เก็บภายหลัง ข้อความใหม่อาจมีขนาดไม่เท่าเดิม จาวาจึงเผื่อที่ว่างไว้อีก 16 ที่ว่าง เพื่อจะได้ไม่ต้องกันที่ในแรมเพิ่มเติมอีกในกรณีที่ข้อความใหม่มีขนาดใหญ่กว่าเดิมไม่เกิน 16 เพราะการกันที่ในแรมเพิ่มเติมภายหลังเป็นเรื่องที่ไม่น่าสนุก เพราะที่ในแรมที่เพิ่มขึ้นอาจไม่อยู่ติดกับที่เดิมทำให้ประสิทธิภาพในการเข้าถึงลดลง

ต่อไปนี้เป็นแมธธอสที่น่าสนใจของคลาส StringBuffer จำไว้ว่าถ้ามีการเปลี่ยนแปลงข้อความในStringBuffer ผ่านแมธธอส ข้อความจะถูกเปลี่ยนแบบถาวร

```
StringBuffer append(String s)
StringBuffer append(char c)
StringBuffer append(char[] c, int offset, int len)
StringBuffer append(boolean b)
StringBuffer append(int i)
StringBuffer append(long l)
StringBuffer append(float f)
StringBuffer append(double d)
```

แมธธอสเหล่านี้จะเพิ่มข้อความในวงเล็บเข้าไปท้ายข้อความที่มีอยู่แล้วใน StringBuffer

```
StringBuffer insert(int offset, String s)
StringBuffer insert(int offset, char c)
StringBuffer insert(int offset, char[] c)
StringBuffer insert(int offset, boolean b)
StringBuffer insert(int offset, int i)
StringBuffer insert(int offset, long l)
StringBuffer insert(int offset, float f)
StringBuffer insert(int offset, double d)
```

แมธธอสเหล่านั้นเหมือนกับ append แต่แทนที่จะต่อท้ายกลับแทรกเข้าไปในตำแหน่งที่เท่ากับ offset

```
StringBuffer deleteCharAt(int index)
StringBuffer delete(int start, int end)
```

แมธธอส deleteCharAt() ลบตัวอักษรในตำแหน่ง index ออก ส่วนแมธธอส delete() ลบออกตั้งแต่ตำแหน่ง start จนถึง end แต่ไม่รวม end

```
StringBuffer reverse()
```

แมธธอส reverse() กลับตัวอักษรที่มีอยู่เดิมใหม่จากหลังมาหน้า

ลองดูตัวอย่างการใช้งานดังต่อไปนี้

โปรแกรม 14 - 10 : TestString.java

```
public class TestString {
  public static void main(String[] args) {
    StringBuffer s1 = new StringBuffer("banana split");// (1)
    s1.delete(4,12); // "bana"
    s1.append(42);  // "bana42"
    s1.insert(4,"na");  // "banana42"
    s1.setCharAt(0,'s');  //sanana42"
    s1.reverse(); "24ananas"
  }
}
```

```
char charAt(int index)
```

char setCharAt(int index, char ch)
แมธธอส charAt() ส่งค่าตัวอักษรในตำแหน่ง index กลับ ส่วนแมธธอส setCharAt() เปลี่ยนค่าตัวอักษรในตำแหน่ง index ด้วย ch

ถ้าเราต้องการส่งค่าของ StringBuffer ให้ตัวแปรสตริงเราทำได้โดยใช้แมธธอส toString()

String toString()
แมธธอส toString() ส่งค่าของข้อความออกมาในรูปตัวแปรสตริง ตัวอย่างการใช้งานก็เช่น

```
String str = strBuff.toString();
```

สตริง args

ตอนนี้คุณคงพอจะรู้แล้วว่าแมธธอส main() มีการส่งผ่านอะเรย์ชื่อ **args** เป็นอะเรย์ของตัวแปรสตริง ตัวแปร args มีไว้รับค่าพารามิเตอร์อะไรก็ตามที่อยู่หลังชื่อโปรแกรมในคำสั่ง java ตัวอย่างเช่น

โปรแกรม 14 - 11 : TestString.java

```
public class TestString {
        public static void main(String[] args) {
                System.out.println(args[0]);
                System.out.println(args[1]);
                System.out.println(args[2]);
        }
}
```

ลองรันโปรแกรมข้างต้นโดยมีการส่งผ่านข้อความเข้าไปในตัวแปร args ดังนี้

```
C:\java> java TestString Hello World 2000
Hello
World
2000
```

นั้นคืออะเรย์ args จะเก็บข้อความหลังชื่อคลาสในคำสั่ง java ซึ่งข้อความที่ตามมานี้จะมีกี่ตัวก็ได้ เราอาจนำข้อความเหล่านี้ไปใช้ในโปรแกรมของเราเพื่อประโยชน์ใดๆ ก็ได้

15

คลาสสำหรับตัวแปรพื้นฐาน

ในภาษาจาวาตัวแปรพื้นฐานอื่นๆ นอกจากตัวแปรสตริงไม่ได้เป็นวัตถุ แต่จาวาก็จะมีคลาสที่ทำงานได้เหมือนกับมันอยู่ด้วย ชื่อของคลาสเหล่านี้จะเหมือนกับชื่อชนิดของตัวแปรแต่ขึ้นต้นด้วยอักษรตัวใหญ่เพราะเป็นชื่อคลาส การใช้งานของคลาสเหล่านี้ก็คล้ายๆ กับคลาส String

คอนสตรัคเตอร์ของสำหรับคลาสเหล่านี้ จะรับค่าของตัวแปรนั้นๆ ตัวอย่างเช่น

```
Integer  intObj = new Integer(2000);
Double douObj = new Double(3.56);
Character  charObj = new Character(‘a’);
Boolean    booObj = new Boolean(false);
```

นอกจากนี้ยังมีคอนสตรัคเตอร์โอเวอร์โหลดอีกอันหนึ่งซึ่งรับค่าของตัวแปรในรูปของข้อความได้ด้วย เช่น

```
Integer  intObj = new Integer(“2000”);
Double douObj = new Double(“3.56”);
Character  charObj = new Character(“a”);
Boolean    booObj = new Boolean(“false”);
```

สิ่งที่ได้จากการใช้งานตัวแปรพื้นฐานในรูปของคลาสก็คือแมธธอสช่วยเหลือที่มีมาให้ ที่น่าสนใจมีดังต่อไปนี้

ถ้าต้องการส่งผ่านค่าจากตัวแปรเหล่านี้ไปเป็นตัวแปรพื้นฐานใช้แมธธอส typeValue() โดยแทนคำว่า type ด้วยชื่อของตัวแปร เช่น

```
int i = intObj.intValue();
char c = charObj.charValue();
```

ถ้าต้องการส่งผ่านค่าจากตัวแปรเหล่านี้ไปเป็นตัวแปรสตริงใช้แมธธอส toString() เช่น

```
String intString = intObj.toString();
String charString = charObj.toString();
```

และถ้าต้องการส่งผ่านค่าตัวแปรสตริงไปเป็นตัวแปรพื้นฐาน คลาสของตัวแปรพื้นฐานเหล่านี้จะมีแมธธอสสแตรติคที่ทำหน้าที่เหล่านี้ให้คือแมธธอส parseType() เช่น

```
double d = Double.parseDouble("3.14");
boolean b = Boolean.parseBoolean("false");
```

แมธธอสเหล่านี้อ้างถึงได้ทันที ไม่จำเป็นต้องมีการสร้างอินสแตนท์ของ Double หรือ Boolean ก่อน เนื่องจากแมธธอสข้างต้นเหล่านี้เป็นแมธธอสสแตรติค

คลาสของตัวแปรพื้นฐานเหล่านี้มีตัวแปรถาวรที่เป็นตัวแปรสแตรติค ที่แทนค่าสูงสุดและต่ำสุดที่เป็นไปได้ของแต่ละตัวแปรพื้นฐานด้วยดังนี้

```
Integer.MIX_VALUE
Integer.MAX_VALUE
Double.MIX_VALUE
Double.MAX_VALUE
Byte.MIX_VALUE
Byte.MAX_VALUE
Long.MIX_VALUE
Long.MAX_VALUE
Short.MIX_VALUE
Short.MAX_VALUE
Float.MIX_VALUE
Float.MAX_VALUE
Character.MIN_VALUE
Character.MAX_VALUE
Boolean.TRUE
Boolean.FALSE
```

สำหรับ Boolean ไม่มีค่าสูงสุดต่ำสุด จึงมีแต่ค่าที่เป็นไปได้สองค่าคือ true และ false แทน ลองดูตัวอย่างการใช้งาน

โปรแกรม 15 - 1 : TestWrapper.java

```
public class TestWrapper {
        public static void main(String[] args) {
                double d = Double.MAX_VALUE;
```

```
                System.out.println(d);
        }
}
```

```
C:\java> java TestWrapper
1.7976931348623157E308
```

เคล็ดลับ ที่ตัวแปรเหล่านี้ถูกกำหนดให้เป็นตัวแปรสแตรติคเพราะต้องการให้สามารถอ้างถึงได้ทันที โดยไม่ต้องมีการสร้างอินสแตนท์ขึ้นมาก่อน และที่ตัวแปรเหล่านี้ถูกกำหนดให้เป็นตัวแปรถาวรเป็นเพราะค่าของมันตายตัว

ถ้าต้องการเปรียบเทียบค่าระหว่างสองอินสแตนท์ใช้แมธธอส `equals()` เช่น

โปรแกรม 15 - 2 : TestWrapper.java

```
public class TestWrapper {
        public static void main(String[] args) {
                Character c1 = new Character('a');
                Character c2 = new Character('a');
                System.out.println(c1.equals(c2));

        }
}
```

คลาส `Character` มีแมธธอสสำหรับตรวจสอบค่าที่น่าสนใจอีกดังนี้คือ

```
static boolean isLowerCase(char ch)
static boolean isUpperCase(char ch)
static boolean isTitleCase(char ch)
static boolean isDigit(char ch)
static boolean isLetter(char ch)
static boolean toUpperCase(char ch)
static boolean toLowerCase(char ch)
static boolean toTitleCase(char ch)
```

ตัวอย่างเช่น ถ้าต้องการตรวจสอบว่าตัวอักษรเป็นตัวพิมพ์เล็กหรือไม่ ถ้าใช่ให้เปลี่ยนเป็นตัวพิมพ์ใหญ่ ก็ทำได้ดังนี้

```
char c = 'a';
if (Character.isLowerCase(c)) c = Character.toUpperCase(c);
```

16

คลาส Math

คลาส **Math** เป็นคลาสที่จาวามีมาให้สำหรับการจัดการทางคณิตศาสตร์

คลาสนี้มีค่าคงตัวที่น่าสนใจอยู่สองตัวคือ

```
Math.E
Math.PI
```

ซึ่งเท่ากับค่าล็อกธรรมชาติ และค่าพายนั่นเอง

แมธธอสที่น่าสนใจสำหรับคลาสนี้ได้แก่

```
static int abs(int i)
static long abs(long l)
static float abs(float f)
static double abs(double d)
```

ซึ่งคืนค่าสัมบูรณ์ของตัวเลขในวงเล็บ

```
static double ceil(double d)
```

คืนค่าเท่ากับจำนวนเต็มที่ใกล้เคียงค่าในวงเล็บที่สุด แต่มากกว่าค่าในวงเล็บ หรือก็คือการปัดเศษขึ้นนั่นเอง

```
static double floor(double d)
```

คือการปัดลง

```
static int round(float f)
static long round(double d)
```

คือการปัดเศษตามหลักมาตรฐาน กล่าวคือถ้าต่ำกว่า .5 ก็ปัดลง นอกนั้นปัดขึ้น

static int min(int a, int b)
static long min(long a, long b)
static float min(float a, float b)
static double min(double a, double b)

คืนค่าที่น้อยกว่าระหว่าง a หรือ b

static int max(int a, int b)
static long max(long a, long b)
static float max(float a, float b)
static double max(double a, double b)

คืนค่าที่น้อยกว่าระหว่าง a หรือ b

static double pow(double d1, double d2)

คืนค่า d1 ยกกำลัง d2

static double exp(double d)

คืนค่า e ยกกำลัง d

static double log(double d)

คืนค่าล็อกธรรมชาติของ d

static sqrt(double d)

คืนค่ายกกำลังสองของ d

static double sin(double d)
static double cos(double d)
static double tan(double d)

คืนค่าตรีโกณมิติของ d

static double random()

คืนค่าตัวเลขทศนิยมที่สุ่มมาจาก 0.0 ถึง 1.0 บนการกระจายปกติ

ลองดูตัวอย่างการใช้งานแมธธอสสแตรติคเหล่านี้

โปรแกรม 16 - 1 : TestMath.java

```
public class TestMath {
        public static void main(String[] args) {
                double d = -5.456;
                System.out.println(d);
                d = Math.abs(d);
                System.out.println(d);
                d = Math.ceil(d);
                System.out.println(d);
                double e = Math.random();
                System.out.println(e);
```

```
                e = Math.random();
                System.out.println(e);

        }
}
```

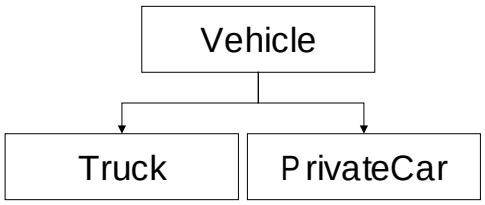
```
C:\java> java TestMath
-5.456
5.456
6.0
0.9652705070022861
0.34808306194739037
```

17

การสืบทอด

เราสามารถสร้างคลาสใหม่โดยอาศัยคลาสที่มีอยู่แล้วเป็นต้นแบบ เราเรียกคลาสใหม่ว่า **สับคลาส** ของคลาสเก่า และเรียกคลาสเก่าว่า **ซูปเปอร์คลาส** ของคลาสใหม่ ตัวแปรคลาสและแมธธอสจะได้รับการสืบทอดไปยังสับคลาสโดยอัตโนมัติ เหมือนกับการสืบทอดลักษณะทางพันธุกรรมจากพ่อไปสู่ลูก

ในบทที่แล้วเราสร้างได้คลาส `Vehicle` ขึ้นมา คลาส `Vehicle` นิยามสิ่งที่ใช้ตัดสินว่าวัตถุที่จัดว่าเป็นรถยนต์ต้องมีคุณสมบัติและพฤติกรรมอะไรบ้าง คราวนี้ถ้าเราต้องการสร้างนิยามที่มีความเฉพาะเจาะจงมากขึ้น เช่นนิยามของรถบรรทุก นิยามของรถเก๋ง เราอาจสร้างคลาสใหม่ชื่อ `Truck` กับ `PrivateCar` ซึ่งนิยามคุณสมบัติและพฤติกรรมที่มีเฉพาะแต่ในรถบรรทุกและรถเก๋ง



รูปที่ 17-1 ลำดับการสืบทอดคุณลักษณะ

แทนที่เราจะสร้างคลาสทั้งสองขึ้นมาใหม่ตั้งแต่ต้น เราอาจใช้คลาส `Vehicle` เป็นต้นแบบ เพราะทั้งรถบรรทุก และรถเก๋งย่อมต้องมีล้อ มีเครื่อง และวิ่งได้ จากนั้นค่อยเติมคุณสมบัติและพฤติกรรมอื่นๆ ที่มีเฉพาะในรถบรรทุก หรือรถเก๋งเข้าไป เราทำเช่นนี้ได้โดยการสร้างสับคลาสของซูปเปอร์คลาส `Vehicle` สองตัวชื่อ `Truck` และ `PrivateCar` ดังตัวอย่างต่อไปนี้

โปรแกรม 17 - 1 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
                System.out.println("I am running.");
        }

}

class Truck extends Vehicle { // (1)

        float maximumLoad;            // (2)

        void load(float weight) {     // (3)
                if (weight <= maximumLoad)
System.out.println("I am carrying a " + weight + "-pound load.");
        }
}

class PrivateCar extends Vehicle {    // (4)
```

```
        int numberOfPassengers;              // (5)

        void playCD() {                              // (6)
                System.out.println("CD is playing.");
        }
}

public class BuildACar {
  public static void main(String[] args) {
        Truck isuzu = new Truck();
        PrivateCar toyota = new PrivateCar();
        isuzu.numberOfWheels = 6;
        toyota.numberOfWheels = 4;
        isuzu.maximumLoad = 1500.00F;
        toyota.numberOfPassengers = 5;
        isuzu.load(700.0F);
        isuzu.run();
        toyota.playCD();
        toyota.run();
  }
}
```

ในบรรทัด (1),(4) เราประกาศคลาสใหม่ชื่อ Truck และ PrivateCar ซึ่งเป็นสับคลาสของคลาส Vehicle ที่มีอยู่แล้ว โดยเราใช้คำสั่ง extends ตามด้วยชื่อซูปเปอร์คลาส สิ่งที่ได้มาโดยอัตโนมัติคือทั้งคลาส Truck และ PrivateCar จะมีตัวแปรชื่อ noOfWheels, hasEngine และมีแมธธอสชื่อ run() พ่วงมาด้วยทันที

จากนั้นเราสามารถสร้างตัวแปรคลาสและแมธธอสเพิ่มเติมได้ เพื่อใช้เป็นสิ่งที่บอกลักษณะที่มีเฉพาะแต่ในรถบรรทุกหรือรถเก๋ง ในบรรทัด(2)เราสร้างตัวแปร maximumLoad ให้คลาส Truck เพื่อใช้ในระบุน้ำหนักบรรทุกสูงสุดของรถบรรทุก และสร้างแมธธอส load() ในบรรทัด (3) เพราะสิ่งที่รถบรรทุกต้องทำได้คือการบรรทุกของ

ในบรรทัด (5) เราสร้างตัวแปร numberOfPassengers ให้คลาส PrivateCar เพื่อใช้ระบุจำนวนผู้นั่ง รถเก๋งควรมีเครื่องเล่นซีดีด้วย เพื่อให้เกิดความแตกต่างกับรถบรรทุก ดังนั้นเราจึงสร้างแมธธอส playCD() ในบรรทัด (6)

คำสั่งในแมธธอส main() เป็นการสร้างอินสแตนท์ให้กับคลาส Truck และ PrivateCar อย่างละหนึ่งอินสแตนท์ สังเกตว่าอินสแตนท์ทั้งสองมีตัวแปร numberOfWheels และแมธธอส run()ทั้งที่ไม่ได้สร้างไว้ในคลาส ที่เป็นเช่นนี้เพราะคลาสทั้งสองสืบทอดคุณลักษณะของคลาส Vehicle ผลการรันโปรแกรมข้างต้นจะเป็นดังภาพ

```
C:\java> java BuildACar
I am carrying a 700.0-pound load.
I am running.
CD is playing.
I am running.
```

เราสามารถสร้างสับคลาสของสับคลาสได้ด้วย ดังนั้นคุณลักษณะของคลาสสามารถสืบทอดต่อกันไปได้เรื่อยๆ ไม่มีที่สิ้นสุด คลาสคลาสหนึ่งมีได้หลายสับคลาสเช่นในกรณีของคลาส Vehicle มีสองสับคลาสคือ Truck และ PrivateCar แต่คลาสคลาสหนึ่งในภาษาจาวามีได้แค่ซูปเปอร์คลาสเดียว กล่าวคือเมื่อ Truck สืบทอดคุณลักษณะจากซูปเปอร์คลาส Vehicle แล้ว คลาส Truck ไม่สามารถสืบทอดคุณลักษณะจากคลาสอื่นได้อีก คำสั่ง extends จึงตามด้วยชื่อซูปเปอร์คลาสได้แค่ซูปเปอร์คลาสเดียวเท่านั้น

คลาสนามธรรม

คลาสคือนิยามของวัตถุ เราสร้างวัตถุขึ้นจากการสร้างอินสแตนท์ของคลาส แต่ในภาษาจาวาเราสามารถสร้างคลาสที่ไม่อนุญาตให้นำไปสร้างอินสแตนท์ได้ด้วย เราเรียกคลาสเหล่านั้นว่า **คลาสนามธรรม** ลองพิจารณาตัวอย่างต่อไปนี้

โปรแกรม 17 - 2 : TestAbstract.java

```
abstract class A {            //  (1)

}

public class TestAbstract {
        public static void main (String[] args) {
                // A a = new A();  //  Error (2)

        }
}
```

ในบรรทัด (1) เป็นการนิยามคลาส A ซึ่งกำหนดให้เป็นคลาสนามธรรมด้วยการใช้คำสั่ง abstract นำหน้าคำสั่ง class ผลที่ได้ก็คือ ห้ามสร้างอินสแตนท์ของคลาส A ดังในบรรทัด (2)

คุณอาจสงสัยว่าทำไมต้องมีการห้ามไม่ให้มีการสร้างอินสแตนท์ของคลาส ในเมื่อจุดประสงค์ของการสร้างคลาสก็คือการสร้างอินสแตนท์ ในบางกรณีนักเขียนโปรแกรมต้องการลดขนาดของโปรแกรมลงเวลาที่ต้องสร้างคลาสหลายๆ คลาสที่มีคุณสมบัติคล้ายๆ กัน นักเขียนโปรแกรมจะสร้างคลาสนามธรรมขึ้นมาก่อน คลาสนามธรรมจะบรรจุส่วนที่ซ้ำกันของคลาสเหล่านั้นเอาไว้ จากนั้นนักเขียนโปรแกรมก็จะสร้างคลาสเหล่านั้นด้วยการสืบทอดคลาสนามธรรมแล้วเติมรายละเอียดปลีกย่อยที่แตกต่างลงไปทีหลัง วิธีนี้โปรแกรมจะมีขนาดเล็กลงเพราะไม่มีการนิยามตัวแปรคลาส และแมธธอสที่ซ้ำๆ กัน แต่เนื่องจากคลาสนามธรรมมีไว้เพียงเพื่อจุดประสงค์ในการลดขนาดโปรแกรม นักเขียนโปรแกรมจึงระวังไม่ให้ใครเอาคลาสนามธรรมไปใช้ด้วยการห้ามมิให้อินสแตนท์คลาสนามธรรม

ที่ผ่านมาเราสร้างคลาส `Vehicle` ขึ้นมาโดยประกาศตัวแปรคลาส `numberOfWheels` `hasEngine` และแมธธอส `run()` ซึ่งเป็นลักษณะร่วมของรถยนต์ทุกชนิด จากนั้นเราก็สร้างคลาส `Truck` และ `PrivateCar` ให้สืบทอดคลาส `Vehicle` ซึ่งจะได้ตัวแปรคลาสและแมธธอสมาโดยอัตโนมัติ จากนั้นก็นิยามตัวแปรและแมธธอสอื่นที่เป็นลักษณะเฉพาะของรถแต่ละชนิดในตัวของคลาสนั้นๆ เองทีหลัง ในกรณีถ้านักเขียนโปรแกรมไม่ต้องการให้ใครสร้างอินสแตนท์ `Vehicle` นักเขียนโปรแกรมสามารถทำได้โดยกำหนดให้คลาส `Vehicle` เป็นคลาสนามธรรม

คลาสถาวร

คลาสถาวร คือ คลาสที่ไม่อนุญาตให้นำไปใช้สืบทอด

เราสามารถกำหนดให้คลาสของเราเป็นคลาสถาวรได้ด้วยการใช้คำสั่ง `final` นำหน้าคำสั่ง `class` ดังตัวอย่างต่อไปนี้

โปรแกรม 17 - 3 : TestFinal.java

```
final class A {                //  (1)

}
/*
class B extends A {            //  (2)
```

```
}
*/
public class TestAbstract {
        public static void main (String[] args) {
                A a = new A();  //  (3)

        }
}
```

คลาส A ในบรรทัด (1) ถูกกำหนดให้เป็น คลาสถาวร การสร้างคลาส B ในบรรทัด (2) จึงทำไม่ได้เพราะคลาส B สืบทอดคลาส A

ประโยชน์ของคำสั่งถาวรคือ การป้องกันไม่ให้นักเขียนโปรแกรมคนอื่นเอาคลาสของเราไปสืบทอดโดยไม่บอกกล่าว เพราะอาจทำให้เกิดการสับสน

ตอนที่เรากล่าวถึงคลาสนามธรรมจะเห็นได้ว่า คลาสนามธรรม มักใช้เป็นต้นแบบสำหรับคลาสอื่น ดังนั้นจึงไม่มีเหตุผลที่จะกำหนดให้คลาสนามธรรมเป็นคลาสถาวร เพราะคลาสนามธรรมจะมีประโยชน์ก็ต่อเมื่อเรานำมันไปสืบทอด ด้วยเหตุนี้ คำสั่ง final และคำสั่ง abstract จะใช้นำหน้าคลาสเดียวกันไม่ได้

คลาส String เป็นคลาสถาวร คุณไม่สามารถสร้างคลาสที่สืบทอดคลาส String ได้

การโอเวอร์รายด์แมธธอส

เราสามารถสร้างแมธธอสที่มีชื่อเหมือนแมธธอสของซูปเปอร์คลาสในสับคลาสได้ด้วย เราเรียกว่า **การโอเวอร์รายด์** แมธธอสใหม่จะแทนที่แมธธอสเก่า ตัวอย่างเช่น

โปรแกรม 17 - 4 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){                                  // (1)
                System.out.println("I am running");
        }

}
```

```
class PrivateCar extends Vehicle {

  int numberOfPassengers;

  void playCD() {
        System.out.println("CD is playing.");
  }

  void run() {                        // (2)
        System.out.println("I am a running car.");
  }
}

public class BuildACar {
  public static void main(String[] args) {

        PrivateCar toyota = new PrivateCar();
        toyota.run();                 // (3)
  }
}
```

ในบรรทัด (2) เราสร้างแมธธอส run() ซึ่งมีชื่อซ้ำกับแมธธอส run() ในบรรทัด (1) ถ้าลองรันโปรแกรมนี้ดูจะพบว่าจาวาเวอร์ชัวน์แมทชีนจะเลือกรันแมธธอส run() ตัวใหม่ ดังในภาพ

```
C:\java> java BuildACar
I am a running car.
```

ข้อความที่ปรากฏคือคำว่า I am a running car แทนที่จะเป็น I am running เหมือนเคย

การโอเวอร์รายด์แมธธอส นอกจากจะต้องใช้ชื่อแมธธอสชื่อเดิมแล้ว การส่งผ่านตัวแปรจะต้องเหมือนกันด้วย ในบทที่แล้วเรารู้จักการโอเวอร์โหลดแมธธอส ขอให้สังเกตว่าการโอเวอร์โหลดกับการโอเวอร์รายด์มีความหมายต่างกัน

ในกรณีของตัวแปรคลาส เราสามารถสร้างตัวแปรคลาสชื่อเดิมในสับคลาสได้เหมือนกัน แต่ตัวแปรตัวใหม่นี้จะไม่ทับตัวแปรตัวเก่าที่นิยามไว้ในซูปเปอร์คลาส เรายังสามารถอ้างถึงตัวแปรในซูปเปอร์คลาสได้อยู่ด้วยการใช้คำว่า super ตามด้วยจุดนำหน้าชื่อตัวแปรดังตัวอย่างต่อไปนี้

โปรแกรม 17 - 5 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        Vehicle() {                          // (1)
                numberOfWheels = 6;
                hasEngine = true;
        }

        void run(){
                System.out.println("I am running");
        }

}

class PrivateCar extends Vehicle {

  int numberOfPassengers;
  int numberOfWheels;                        // (2)

  PrivateCar() {                                     // (3)
    numberOfPassengers = 5;
    numberOfWheels = super.numberOfWheels - 2;  // (4)
  }

  void playCD() {
        super.run();                   //  (5)
        System.out.println("CD is playing.");
  }

  void run() {                               // (6)
        System.out.println(I am a running car.");
  }

}

public class BuildACar {
  public static void main(String[] args) {

        PrivateCar toyota = new PrivateCar();
        System.out.println(toyota.numberOfWheels);
        toyota.playCD();
  }
}
```

บรรทัด (1) คือ คอนสตรัคเตอร์ของ Vehicle ซึ่งเรากำหนดค่าเริ่มต้นให้ตัวแปร numberOfWheels เป็น 6 ตัวแปร numberOfWheels พร้อมทั้งค่าของมันจะสืบทอดไปยังสับคลาส PrivateCar

บรรทัด (2) เราสร้างตัวแปรคลาสชื่อเดิมให้กับคลาส PrivateCar แต่ในคอนสตรัคเตอร์ของ PrivateCar ในบรรทัด (3) เรายังสามารถอ้างถึงตัวแปร numberOfWheels ที่นิยามไว้ในคลาส Vehicle ได้อยู่ด้วยการ
ใช้ super. นำหน้า โดยในกรณีนี้เรานำมันมากำหนดค่าให้กับ numberOfWheels ตัวใหม่ เมื่อรันโปรแกรมจะได้ค่าของ numberOfWheels ตัวใหม่เป็น 4 เพราะถูกหักออก 2 ดังภาพ

```
C:\java> java BuildACar
4
I am running.
CD is playing.
```

สังเกตว่าเราใช้คำว่า super เรียกตัวแปรคลาสของซูปเปอร์คลาสได้เฉพาะในตัวสับคลาสเท่านั้น ไม่มีวิธีเรียกตัวแปรคลาสของซูปเปอร์คลาสนอกตัวสับคลาส เช่นการเรียกผ่านอินสแตนท์

เช่นเดียวกัน เราสามารถใช้คำว่า super ในการเรียกแมธธอสของซูปเปอร์คลาสที่ถูกโอเวอร์รายด์ไปแล้วได้ อย่างเช่นในบรรทัด (6) เราโอเวอร์รายด์แมธธอส run() ไปแล้ว แต่เรายังสามารถเรียกใช้งานแมธธอสรันตัวเก่าได้ในบรรทัด (5)ด้วยการใช้คำว่า super นำหน้า ผลที่ได้คือจาวาเวอร์ชัวน์แมทชีนจะรันแมธธอส run() ของซูปเปอร์คลาสในแมธธอส playCD() ผลที่ได้จึงเป็นดังภาพข้างต้น

การใช้คำสั่ง super เรียกแมธธอสของซูปเปอร์คลาสทำได้เฉพาะภายในตัวสับคลาสเท่านั้น ไม่มีวิธีเรียกแมธธอสของซูปเปอร์คลาสภายในนอกตัวแมธธอส

ข้อสังเกตเกี่ยวกับ super ก็คือว่า super ใช้เรียกตัวแปรและแมธธอสของซูปเปอร์คลาสที่อยู่ถัดขึ้นไปหนึ่งระดับเท่านั้น ถ้ามีการสร้างสับคลาสมากกว่าหนึ่งระดับเราไม่มีวิธีเรียกตัวแปรและแมธธอสของซูปเปอร์ที่อยู่ถัดขึ้นไปกว่าซูปเปอร์คลาสที่อยู่ในระดับติดกับตัวสับคลาสได้ ตัวอย่างเช่น

โปรแกรม 17 - 6 : TestSuper.java

```
class A  {
        void x() {                          // (1)
        System.out.print("I am in A.");
        }
```

```
}
class B extends A {
        void x() {                          // (2)
        System.out.println("I am in B.");
        }
}
class C extends B {
        void x() {                          // (3)
        System.out.println("I am in C.");
        }
        void y() {
                super.x();                   // (4)
        }
}
public class TestSuper {
        public static void main(String[] args) {
                C c = new C();
                c.y();                       // (5)
        }
}
```

คลาส C สืบทอดคลาส B ซึ่งสืบทอดคลาส A มาก่อน ทั้งสามคลาสมีแมธธอสชื่อ x() อยู่ทั้งสิ้น หรืออีกนัยหนึ่งก็คือ x() ถูกโอเวอร์รายด์มาโดยตลอด ในบรรทัด (4) แมธธอส y() มีการเรียกแมธธอส x() โดยใช้ super นำหน้า ผลที่ได้คือเมื่อรันแมธธอส y() ในบรรทัด (5) จะได้แมธธอสในคลาส B ที่นิยามไว้ในบรรทัด (2) เพราะคลาส B เป็นซูปเปอร์คลาสตัวที่ใกล้ชิดคลาส c มากที่สุด ดังในภาพ

```
C:\java> java BuildACar
I am in B.
```

แมธธอสถาวร

แมธธอสถาวร คือ แมธธอสที่กำหนดให้สับคลาสของมันไม่สามารถนำมันไปโอเวอร์รายด์ได้ เราประกาศให้แมธธอสเป็นแมธธอสถาวรได้โดยใช้คำสั่งว่า final ตัวอย่างเช่น

```
final void x() {
```

```
}
```

เคล็ดลับ แมธธอสถาวร คือ แมธธอสที่โอเวอร์รายด์ไม่ได้
ตัวแปรถาวร คือตัวแปรที่เปลี่ยนค่าอีกไม่ได้
คลาสถาวร คือ คลาสที่คลาสอื่นสืบทอดไม่ได้
ทั้งสามกรณีแม้ว่าจะใช้คำสั่ง `final` ในการประกาศเหมือนกัน แต่จุดมุ่งหมายต่างกัน

แมธธอสนามธรรม

แมธธอสนามธรรม คือ แมธธอสที่มีแต่ชื่อ ไม่มีนิยาม เราใช้คำว่า `abstract` นำหน้าชื่อแมธธอส และแทนที่จะมีบล๊อคปีกกาต่อท้ายชื่อ กลับมีแต่เครื่องหมาย `;` เท่านั้น ตัวอย่างเช่น

โปรแกรม 17 - 7 : TestAbstract.java

```
abstract class A {
        void x() { }
abstract y();
}

class B extends A {
        y() { };

}
```

คลาสที่มีแมธธอสนามธรรมต้องอยู่ในคลาสนามธรรมเท่านั้น และสับคลาสที่สืบทอดคลาสนี้ต้องโอเวอร์รายด์แมธธอสนามธรรมทั้งหมดที่มีอยู่ในคลาสนามธรรมนั้นเพื่อเป็นการสร้างนิยามให้แมธธอสเหล่านั้น หรือมิฉะนั้นสับคลาสเหล่านั้นต้องถูกกำหนดให้เป็นคลาสนามธรรมด้วย เพื่อรอให้สับคลาสในรุ่นต่อไปมานิยามแมธธอสนามธรรมเหล่านั้นให้ ดังในตัวอย่าง คลาส `B` ต้องมีแมธธอส `y()` มิฉะนั้นคอมไพล์เลอร์จะไม่ยอมให้ผ่าน นอกเสียจาก

จะประกาศให้คลาส B เป็นคลาสนามธรรมเพื่อรอให้คลาสอื่นมาสืบทอดแมธธอสนามธรรม y() ต่อไป

เช่นเดียวกันกับกรณีของคลาส แมธธอสนามธรรมไม่สามารถถูกกำหนดให้เป็นถาวรแมธธอสได้ และถาวรแมธธอสก็ไม่สามารถถูกกำหนดให้เป็นแมธธอสนามธรรมได้

การเรียกคอนสตรัคเตอร์ของซูปเปอร์คลาส

ในคอนสตรัคเตอร์นอกจากเราจะใช้คำสั่ง this() ในการเรียกคอนสตรัสเตอร์ตัวอื่นในคลาสเดียวกันแล้ว เรายังสามารถใช้คำสั่ง super() ในการเรียกคอนสตรัคเตอร์ของซูปเปอร์คลาสได้ด้วย ลองพิจารณาตัวอย่างต่อไปนี้

โปรแกรม 17- 8 : TestSuper.java

```
class A  {
        int a;                          // (1)
        A() {                             // (2)
          System.out.println("A : Default Constructor");
        }
        A(int a) {                            // (3)
          this.a = a;
          System.out.println("A : Constructor No. 1");
        }
}
class B extends A {
        B() {                            // (4)
        System.out.println("B : Default Constructor");
        }
        B(int b) {
        super(b);                       // (5)
        System.out.println("B : Constructor No. 1");
       }
}

public class TestSuper {
        public static void main(String[] args) {
                B b = new B();          // (6)
                B c = new B(3);                 // (7)
        }
}
```

ผลการรันโปรแกรมข้างต้นเป็นดังในภาพ

```
C:\java> java TestSuper
A : Default Constructor
B : Default Constructor
A : Constructor No. 1
B : Constructor No. 1
```

ในโปรแกรมนี้คลาส B สืบทอดคลาส A ทั้งสองคลาสมีคอนสตรัคเตอร์สองคอนสตรัสเตอร์ คือคอนสตรัคเตอร์ปกติที่ไม่มีการส่งผ่านตัวแปร และคอนสตรัคเตอร์ที่มีการส่งผ่านตัวแปรหนึ่งตัว

ในบรรทัด (5) คอนสตรัคเตอร์ที่มีการส่งผ่านตัวแปรของคลาส B ใช้คำสั่ง super(b) เพื่อเรียกคอนสตรัคเตอร์ที่มีการส่งผ่านตัวแปรของซูปเปอร์คลาสของมันคือคลาส A

โปรแกรมเริ่มจากสร้างอินสแตนท์ของคลาส B ที่ชื่อ b ในบรรทัด (6) ด้วยการเรียกคอนสตรัคเตอร์แบบปกติในบรรทัด (4) คุณอาจแปลกใจเมื่อดูที่ผลการรัน เพราะคอนสตรัคเตอร์ที่ถูกเรียกก่อนคอนสตรัคเตอร์ปกติของคลาส B คือคอนสตรัคเตอร์ปกติของคลาส A ที่เป็นเช่นนี้เป็นเพราะความจริงแล้วทุกๆ คอนสตรัคเตอร์เวลาคอมไพล์ คอมไพล์เลอร์จะแอบใส่คำสั่ง super() ไว้ที่บรรทัดแรก ทำให้มีการเรียกคอนสตรัคเตอร์ของซูปเปอร์คลาสก่อนที่จะทำอะไรอย่างอื่นในคอนสตัคเตอร์ของตัวเอง ลองพิจารณาโปรแกรมข้างล่างนี้

```
Class A {
        A() {}
}

Class B extends A {
        B() {}
}
```

เวลาคอมไพล์ คอมไพล์จะแอบใส่คำสั่ง super() ลงไปดังนี้

```
Class A{
        A() { super(); }
}

Class B extends A {
        B() { super(); }
```

```
}
```

ดังนั้นจะมีการรันคอนสตรัคเตอร์ของซูปเปอร์คลาสก่อนจะรันของตัวเอง และถ้ามีการสืบทอดมากกว่าหนึ่งระดับ จะมีการเรียกคอนสตรัคเตอร์ย้อนต้นขึ้นไปเรื่อยๆ และจะรันคอนสตรัคเตอร์ของซูปเปอร์คลาสบนสุดก่อนเรียงไล่ลำดับลงมาจากบนลงล่าง

ในบรรทัด (7) มีการเรียกคอนสตรัคเตอร์แบบมีการส่งผ่านตัวแปรของคลาส B ในบรรทัด (5) ในกรณีนี้เราใส่คำสั่ง super(b)ซึ่งเป็นการเรียกคอนสตรัคเตอร์ของซูปเปอร์คลาสแบบมีตัวแปรส่งผ่าน คอมไพล์เลอร์จะไม่แอบใส่คำสั่ง super() อีก โปรแกรมจะเรียกคอนสตรัคเตอร์ในบรรทัด (3) แล้วค่อยรันคอนสตรัสเตอร์ในบรรทัด (5) ดังที่เห็นในผลการรัน

คำสั่ง super() มีใช้ได้ในคอนสตรัคเตอร์เท่านั้น และต้องเป็นคำสั่งแรกสุดด้วย ซึ่งก็หมายความว่าคำสั่ง super() ไม่สามารถอยู่ร่วมกับคำสั่ง this() ได้ เพราะต่างก็ต้องเป็นคำสั่งแรกสุดของคอนสตรัคเตอร์

ในกรณีที่ซูปเปอร์คลาสไม่มีการนิยามคอนสตรัคเตอร์ปกติแบบไม่มีตัวแปรส่งผ่านเอาไว้ แต่มีการนิยามคอนสตรัคเตอร์แบบมีตัวแปรส่งผ่าน คอนสตรัคเตอร์ของสับคลาสของมันจำเป็นต้องเรียกคอนสตรัคเตอร์ของซูปเปอร์คลาสแบบมีตัวแปรส่งผ่านด้วยคำสั่ง super() มิฉะนั้นเวลาคอมไพล์ คอมไพล์เลอร์จะแอบใส่คำสั่ง super() ลงไป ทำให้คอมไพล์ไม่ผ่านเพราะคอนสตรัคเตอร์ปกติไม่มีในซูปเปอร์คลาส ตัวอย่างเช่น

```
Class A {
        A(int a) {}
}
Class B extends A {
        B() {
super(a);   // (1)
        }
}
```

คำสั่งบรรทัด (1) ต้องมีไว้เสมอ จะละไม่ได้เป็นอันขาด

เคล็ดลับ เพื่อป้องกันปัญหาข้างต้นผู้รู้บางคนบอกว่า หลักการเขียนคลาสที่ดีต้องนิยามคอนสตรัคเตอร์ปกติไว้ด้วยเสมอ แม้ว่าจะไม่มีคำสั่งอะไรอยู่ในนั้นเลยก็ตาม และฝึกให้เป็นนิสัย

การแปลงรูป

ตัวแปรอ้างอิงของซูปเปอร์คลาสสามารถใช้ชี้สับคลาสได้ด้วย ดังตัวอย่างต่อไปนี้

โปรแกรม 17 - 9 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
                System.out.println("I am running");
        }

}

class Truck extends Vehicle {

        float maximumLoad;

        void load(float weight) {
                if (weight <= maximumLoad)
        System.out.println("I am carrying a " + weight + "-pound
load.");
        }
}

class PrivateCar extends Vehicle {

        int numberOfPassengers;

        void playCD() {                                // (6)
                System.out.println("CD is playing.");
        }
}

public class BuildACar {
        public static void main(String[] args) {
                Vehicle v = new Vehicle();    // (1)
                Truck isuzu = new Truck();    // (2)
                PrivateCar toyota = new PrivateCar();// (3)
                v = isuzu; // (4)
                toyota = isuzu; // (5) Error
                isuzu = new Vehicle(); // (6) Error

        }
}
```

โปรแกรมนี้มีซูปเปอร์คลาสชื่อ `Vehicle` ซึ่งเป็นแม่แบบของสับคลาส `Truck` และ `PrivateCar`

ในแมธธอส main() เราประกาศตัวแปรอ้างอิงของทั้งสามคลาสในบรรทัด (1) (2) (3) ตามลำดับ จากนั้นเราเปลี่ยนให้ตัวแปรอ้างอิง v มีค่าเท่ากับตำแหน่งในแรมของตัวแปรอ้างอิง isuzu ในบรรทัด (4) ซึ่งสามารถทำได้เพราะตัวแปรอ้างอิงของซูปเปอร์คลาสสามารถใช้ชี้อินสแตนท์ของสับคลาสได้ คุณสมบัตินี้มีเหตุผลตามหลักสามัญสำนึกเพราะ ตัวแปรอ้างอิง v มีไว้ชี้รถยนต์ ดังนั้นมันควรจะสามารถใช้ชี้รถบรรทุกได้ด้วย เพราะรถบรรทุกก็เป็นรถยนต์

ในบรรทัด (5) เราพยายามกำหนดค่าของตัวแปรอ้างอิง toyota ให้ชี้ไปที่ตำแหน่งของอินสแตนท์ของรถบรรทุก คอมไพล์เลอร์จะฟ้องความผิดพลาดออกมาเพราะตัวแปรอ้างอิง toyota ไม่ใช่ตัวแปรอ้างอิงรถยนต์ และไม่ใช่ตัวแปรอ้างอิงรถบรรทุก จึงไม่สามารถใช้ชี้อินสแตนท์ของรถบรรทุกได้

ในบรรทัด (6) เราพยายามกำหนดค่าของตัวแปรอ้างอิง isuzu ให้ชี้อินสแตนท์ของคลาส Vehicle ที่สร้างขึ้นมาใหม่ คอมไพล์เลอร์จะฟ้องความผิดพลาดออกมาเพราะตัวแปรอ้างอิง isuzu เป็นตัวแปรอ้างอิงที่ใช้ชี้รถบรรทุกเท่านั้น ไม่ควรนำมาชี้อินสแตนท์ของ Vehicle

สรุปก็คือตัวแปรอ้างอิงของซูปเปอร์คลาสสามารถชี้อินสแตนท์ของสับคลาสได้ด้วย แต่ตัวแปรอ้างอิงของสับคลาสไม่สามารถใช้ชี้อินสแตนท์ของซูปเปอร์คลาสหรืออินสแตนท์ของคลาสที่มีซูปเปอร์คลาสเดียวกันได้ และแน่นอนถ้าจะลองนำไปชี้คลาสอื่นๆ ที่ไม่มีความเกี่ยวข้องกันเลยก็ยิ่งไม่ได้ใหญ่

การที่ตัวแปรอ้างอิงของซูปเปอร์คลาสสามารถชี้อินสแตนท์ของสับคลาสได้ด้วย เป็นการเพิ่มความยืดหยุ่นในการเขียนโปรแกรม ประโยชน์ที่เห็นได้ชัดที่สุดเพราะมีการนำไปใช้กันอย่างแพร่หลายก็คือ การส่งผ่านอินสแตนท์เข้าไปในแมธธอส ดังจะเห็นได้จากตัวอย่างต่อไปนี้

โปรแกรม 17 - 10 : BuildACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
```

```
                System.out.println("I am running");
        }

}

class Truck extends Vehicle {

        float maximumLoad;

        void load(float weight) {
                if (weight <= maximumLoad)
        System.out.println("I am carrying a " + weight + "-pound
load.");
        }
}

class PrivateCar extends Vehicle {

        int numberOfPassengers;

        void playCD() {
                System.out.println("CD is playing.");
        }
}

class Mechanic {
    void removeWheels(Vehicle v) { // (1)
                v.numberOfWheels = 0;
        }
}

public class BuildACar {

  public static void main(String[] args) {
        Truck isuzu = new Truck();    // (2)
        PrivateCar toyota = new PrivateCar();// (3)
        Mechanic robert = new Mechanic(); //  (4)
        robert.removeWheels(isuzu); //  (5)
        robert.removeWheels(toyota); // (6)
        System.out.println("isuzu has " + isuzu.numberOfWheels + "
wheels.");
        System.out.println("toyota has " + toyota.numberOfWheels + "
wheels.");
        }
}
```

สิ่งใหม่ในโปรแกรมนี้คือคลาส `Mechanic` ซึ่งก็คือช่างเครื่อง ช่างเครื่องในโปรแกรมนี้มีความสามารถหนึ่งอย่างคือการสามารถถอดล้อรถยนต์ได้ ดังจะเห็นได้จากแมธธอสชื่อ `removeWheels()` ในบรรทัด `(1)` แมธธอส `removeWheels()` มีการส่งผ่านตัวแปรหนึ่งตัว ได้แก่ตัวแปรอ้างอิง `v` ซึ่งเป็นตัวแปรอ้างอิงที่ใช้ชี้รถยนต์

ในบรรทัด (2) (3) ในแมธธอส main() เราสร้างอินสแตนท์ของรถบรรทุก และรถเก๋งขึ้นมา เราต้องการถอดล้อรถทั้งสองคันนี้ เราจึงสร้างอินสแตนท์ของช่างเครื่องขึ้นมาชื่อ robert ในบรรทัด (4)

ในบรรทัด (5) (6) เราสามารถเรียกแมธธอส removeWheels() ของโรเบิร์ตขึ้นมาถอดล้อของทั้งรถบรรทุกและรถเก๋งได้เพราะตัวแปรอ้างอิง v ที่ส่งผ่านเข้าไปในแมธธอส removeWheels() ใช้ชี้ได้ทั้งอินสแตนท์ของรถบรรทุกและรถเก๋ง ถ้าไม่มีคุณสมบัติการแปรรูปเช่นนี้ในภาษาจาวา เราคงต้องเขียนแมธธอสในการถอดล้อขึ้นมาเฉพาะสำหรับรถแต่ละประเภท ทำให้โปรแกรมมีความซ้ำซ้อนไม่กระชับ นี่คือประโยชน์ที่สำคัญมากของการแปรรูป

ผลการรันโปรแกรมเป็นดังภาพข้างล่าง รถทั้งสองคันถูกถอดล้อ

```
C:\java> java BuildACar
isuzu has 0 wheels.
toyota has 0 wheels.
```

การแปรรูปเป็นคุณสมบัติที่มีประโยชน์ อย่างไรก็ตามปัญหาจะเกิดขึ้นในกรณีที่มีการโอเวอร์รายด์แมธธอส ลองพิจารณาตัวอย่างต่อไปนี้

โปรแกรม 17 - 11 : TestACar.java

```
class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        Vehicle() {
                numberOfWheels = 10;   // (1)
        }

        void run(){                            //  (2)

                System.out.println("I am running");
        }

}

class PrivateCar extends Vehicle {

        int numberOfWheels;
        int numberOfPassengers;
```

```
        PrivateCar() {                // (3)
        numberOfWheels = 4;
        }

        void playCD() {                       // (4)
System.out.println("CD is playing.");
        }

        void run() {                  // (5)
          System.out.println("I am a running private car.");
        }
}

public class BuildACar {
  public static void main(String[] args) {
        Vehicle v = new PrivateCar(); // (6)
        v.playCD(); // (7) Error
        v.run();       // (8)
        System.out.println("v has " + v.numberOfWheels + " wheels.");
               // (9)

  }
}
```

ในบรรทัด (5) สับคลาส PrivateCar มีการโอเวอร์รายด์แมธธอส run() ของซูปเปอร์คลาส Vehicle

โปรแกรมเริ่มต้นที่แมธธอส main() ในบรรทัด (6) ด้วยการประกาศตัวแปรอ้างอิงชื่อ v โดยกำหนดให้ชี้อินสแตนท์ของคลาส PrivateCar เมื่อเรียกแมธธอส playCD() ในบรรทัด (7) คอมไพลเลอร์จะฟ้องความผิดพลาดออกมาเพราะถึงแม้ว่าตัวแปรอ้างอิง v จะชี้อินสแตนท์ของคลาส PrivateCar แต่ตัวมันเองเป็นที่รู้จักในฐานะของตัวชี้อินสแตนท์ของคลาส Vehicle ซึ่งไม่มีแมธธอสชื่อ playCD() อยู่

แต่ในกรณีที่สับคลาสมีการโอเวอร์รายด์แมธธอสของซูปเปอร์คลาส คอมไพลเลอร์ย่อมรู้จักแมธธอสนั้นเพราะมันพบชื่อแมธธอสในซูปเปอร์คลาส ดังนั้นโปรแกรมจะคอมไพล์ผ่าน เช่นในกรณีของแมธธอส run() ที่ถูกเรียกในบรรทัด (8) อีกทั้งเมื่อเวลารันโปรแกรมจาวาเวอร์ชัวน์แมทชีนจะรู้จักรันแมธธอสในสับคลาสให้ด้วย ผลที่ได้ก็คือโปรแกรมข้างต้นจะรันแมธธอส run() ในบรรทัด (5) แทนที่จะเป็นบรรทัด (2) ดังในภาพ

```
C:\java> java TestACar
I am running a private car.
v has 10 wheels.
```

แต่ในกรณีของตัวแปรอินสแตนท์ สถานการณ์จะกลับกับกรณีของแมธธอส ถ้ามีการประกาศตัวแปรชื่อเดิมในสับคลาสแทนที่จาวาเวอร์ชัวน์แมทชีนจะรู้จักตัวแปรตัวใหม่ในสับคลาส มันจะใช้ตัวแปรตัวเก่าในซูปเปอร์คลาส หรืออีกนัยหนึ่ง จาวาเวอร์ชัวน์แมทชีนจะดูชนิดของตัวแปรอ้างอิงเป็นสำคัญในกรณีของตัวแปรอินสแตนท์ ส่วนในกรณีของแมธธอสอินสแตนท์จาวาเวอร์ชัวน์แมทชีนจะดูชนิดของอินสแตนท์เป็นสำคัญ ดังในตัวอย่างมีการกำหนดค่าเริ่มต้นให้ `numberOfWheels` เป็น 10 ในคอนสตรัคเตอร์ของซูปเปอร์คลาส `Vehicle` ในบรรทัด `(1)` และกำหนดค่าเริ่มต้นให้ `numberOfWheels` เป็น 4 ในคอนสตรัคเตอร์ของสับคลาส `PrivateCar` ในบรรทัด `(3)` พอเรียกตัวแปร `numberOfWheels` ออกมาแสดงค่าในบรรทัด `(9)`ผลที่ได้คือ 10 แทนที่จะเป็น 4

ในบางกรณีเราอาจเคยกำหนดตัวแปรอ้างอิงสำหรับซูปเปอร์คลาสให้ชี้อินสแตนท์ของสับคลาส แล้วต้องการใช้อินสแตนท์นั้นในการกำหนดค่าให้กับตัวแปรอ้างอิงสำหรับสับคลาสตัวหนึ่ง เราสามารถทำได้ด้วยการแคสในลักษณะเดียวกันกับการแคสตัวแปรพื้นฐาน ลองดูตัวอย่างต่อไปนี้

โปรแกรม 17 - 12 : TestACar.java

```
class Vehicle {

        int numberOfWheels=10;
        boolean hasEngine;

        void run(){

                System.out.println("I am running");
        }

}

class PrivateCar extends Vehicle {

        int numberOfWheels=4;
        int numberOfPassengers;

        void playCD() {
          System.out.println("CD is playing.");
        }

        void run() {
          System.out.println("I am a running private car.");
        }

}
```

```
public class TestACar {
  public static void main(String[] args) {
        Vehicle v = new PrivateCar();         // (1)
        PrivateCar p = (PrivateCar) v;        // (2)
  p.playCD();
  p.run();
  System.out.println("P has " + p.numberOfWheels + " wheels.");

  }
}
```

ในบรรทัด (1) เราประกาศตัวแปรอ้างอิงแบบ Vehicle ขึ้นมาแล้วกำหนดให้ชี้อินสแตนท์ของคลาส PrivateCar จากนั้นในบรรทัด (2) เราก็นำตัวแปรอ้างอิงมาใช้กำหนดค่าให้ตัวแปรอ้างอิง p ซึ่งเป็นตัวแปรอ้างอิงสำหรับอินสแตนท์ของคลาส PrivateCar แต่เนื่องจาก v เป็นตัวแปรอ้างอิงชนิด Vehicle เราจึงต้องทำการแคสให้กลายเป็น PrivateCar ด้วยการเติม (PrivateCar) ไว้หน้า v เพื่อมิให้คอมไพล์เลอร์เข้าใจผิดและฟ้องความผิดพลาดออกมา

ตัวแปรอ้างอิง p สามารถรันแมธธอสในคลาส PrivateCar และเรียกตัวแปรอินสแตนท์ของ PrivateCar ได้เหมือนปกติทุกประการ

18

แพจเกจ

ซอร์สโค้ดภาษาจาวาประกอบด้วยนิยามของคลาสตั้งแต่หนึ่งคลาสขึ้นไปเขียนเรียงต่อกันไปเรื่อยๆ โดยที่ต้องมีคลาสหนึ่งคลาสในซอร์สโค้ดที่มีชื่อเหมือนชื่อไฟล์และมีแมธธอสชื่อ `main()` อยู่ เวลารันโปรแกรมจาวาเวอร์ชัวน์แมทชีนจะมองหาคลาสคลาสนี้แล้วเริ่มรันจากแมธธอส `main()`

โปรแกรมที่มีขนาดใหญ่ จะประกอบด้วยคลาสจำนวนมาก อาจมีมากเป็นร้อยหรือเป็นพัน เราสามารถจัดหมวดหมู่ของคลาสให้เป็นระเบียบและง่ายต่อการใช้งานได้ด้วยการแตกซอร์สโค้ดของโปรแกรมหนึ่งโปรแกรมให้เป็นหลายซอร์สโค้ด ในกรณีนี้ซอร์สโค้ดไม่จำเป็นต้องมีแมธธอส `main()`อยู่ ยกเว้นซอร์สโค้ดที่มีคลาสคลาสแรกที่เราใช้เรียกโปรแกรมหลักของเรา เราเรียกซอร์สโค้ดไฟล์อื่นๆ ที่ประกอบด้วยคลาสตั้งแต่หนึ่งคลาสขึ้นไปแต่ไม่มีคลาสที่มีแมธธอส `main()`อยู่ว่า **แพจเกจ**

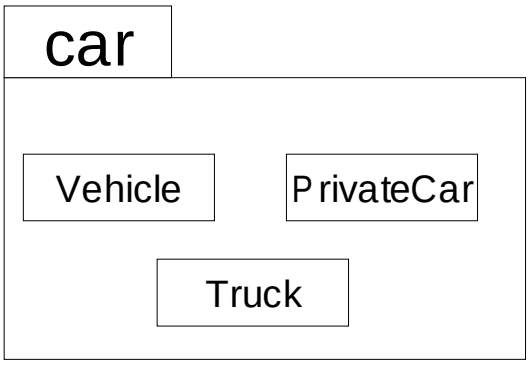
ลองพิจารณาโปรแกรมต่อไปนี้ซึ่งเป็นโปรแกรมที่เราเคยผ่านมาแล้ว

โปรแกรม 18 - 1 : BuildACar.java

```
class Vehicle {
        int numberOfWheels;
```

```
        boolean hasEngine;

        void run(){
                System.out.println("I am running");
        }

}

class Truck extends Vehicle {

        float maximumLoad;

        void load(float weight) {
                if (weight <= maximumLoad)
        System.out.println("I am carrying a " + weight + "-pound
load.");
        }
}

class PrivateCar extends Vehicle {

        int numberOfPassengers;

        void playCD() {
                                        System.out.println("CD is
playing.");
        }
}

class Mechanic {
    void removeWheels(Vehicle v) { // (1)
                v.numberOfWheels = 0;
        }
}

public class BuildACar {
  public static void main(String[] args) {
        Truck isuzu = new Truck();    // (2)
        PrivateCar toyota = new PrivateCar();// (3)
        Mechanic robert = new Mechanic(); //  (4)
  }
}
```

โปรแกรมนี้ประกอบด้วยคลาสห้าคลาส คลาสสี่คลาสแรกเป็นคลาสที่สนับสนุนคลาสสุดท้าย คือคลาส `BuildACar` ซึ่งเป็นคลาสหลักที่มีแมธธอสชื่อ `main()` อยู่



รูปที่ 18 - 1 แพจเกจ car

สมมติว่าเราต้องการย้ายคลาสสามคลาสแรกไปอยู่ในแพจเกจชื่อ car เราสามารถทำได้โดยการสร้างไฟล์สามไฟล์ต่อไปนี้

โปรแกรม 18 - 2 : Vehicle.java

```
package car;

public class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
                System.out.println("I am running");
        }

}
```

โปรแกรม 18 - 3 : Truck.java

```
package car;

public class Truck extends Vehicle {

  float maximumLoad;

  void load(float weight) {
        if (weight <= maximumLoad)
```

```
        System.out.println("I am carrying a " + weight + "-pound
load.");
  }
}
```

โปรแกรม 18 - 4 : PrivateCar.java

```
package car;
public class PrivateCar extends Vehicle {
        int numberOfPassengers;
        void playCD() {
                                System.out.println("CD is
playing.");
        }
}
```

ในบรรทัดแรกสุดของทั้งสามไฟล์มีคำสั่ง

```
package car;
```

คำสั่งนี้เป็นการบอกว่าคลาสที่อยู่ในซอร์สโค้ดนี้ถูกจัดหมวดหมู่ให้อยู่ในแพจเกจชื่อว่า car คำสั่งนี้ต้องเป็นคำสั่งแรกสุดเสมอในไฟล์แพจเกจ และการตั้งชื่อแพจเกจเรานิยมใช้ตัวอักษรภาษาอังกฤษพิมพ์เล็กทั้งหมด

ข้อสังเกตอีกอย่างหนึ่งก็คือ คราวนี้เราต้องเพิ่มคำสั่ง public ไว้หน้าคลาสทั้งสามคลาส และตั้งชื่อไฟล์ให้เหมือนชื่อคลาส เพราะคลาสในแพจเกจเป็นคลาสที่สนับสนุนซอร์สโค้ดอื่นในโปรแกรม ดังนั้นจึงต้องประกาศให้คลาสในแพจเกจเป็นคลาสสาธารณะ

คำว่า public สามารถใช้กำกับหน้าคลาสอะไรก็ได้เพื่อกำหนดให้คลาสนั้นเป็น **คลาสสาธารณะ** คลาสสาธารณะคือคลาสที่สามารถถูกอ้างถึงได้ในแพจเกจอื่น ปกติแล้วคลาสใดๆ สามารถอ้างถึงคลาสอื่นได้เฉพาะที่อยู่ในแพจเกจเดียวกันเท่านั้น คำสั่ง public ทำให้คลาสในแพจเกจอื่นๆ สามารถเรียกคลาสนั้นๆ ได้ด้วย สังเกตว่าคลาสที่แมธธอส main() อยู่ต้องประกาศเป็นคลาสสาธารณะเสมอ เพราะมันจะถูกจาวาเวอร์ชัวน์แมทชีนเรียกตอนเริ่มโปรแกรม

กลับมาที่แพจเกจ car ของเราต่อ เวลาคอมไพล์ทั้งสามไฟล์นี้ต้องเริ่มคอมไพล์จากไฟล์ Vehicle.java ก่อน เพราะคลาส Truck และ PrivateCar มีการอ้างถึงชื่อของคลาส Vehicle ดังนั้น ก่อนจะคอมไพล์ Truck.java และ PrivateCar.java ได้คลาส Vehicle จะต้องไปรออยู่ก่อนแล้วในแพจเกจ car ดังนั้นการคอมไพล์คลาสทั้งสามต้องทำตามลำดับดังนี้

```
C:\java> javac –d C:\java Vehicle.java
C:\java> javac –d C:\java Truck.java
C:\java> javac –d C:\java PrivateCar.java
```

สังเกตว่าคราวนี้เราเติมพารามิเตอร์ –d C:\java เข้าไปในคำสั่งด้วย พารามิเตอร์ –d เป็นพารามิเตอร์ที่ใช้บอกว่าเมื่อคอมไพล์เสร็จแล้วให้เก็บไฟล์ .class ไว้ที่ไหน ซึ่งในกรณีนี้ให้เก็บไว้ที่ C:\java ถ้าลองใช้คำสั่ง dir ดูจะพบโฟล์เดอร์ชื่อ car ใต้ C:\java และถ้าลองเข้าไปดูในโฟล์เดอร์ car ก็จะพบไฟล์ Vehicle.class Truck.class และ PrivateCar.class อยู่ ตอนนี้ถือว่าคลาสทั้งสามอยู่ภายใต้แพจเกจ car เรียบร้อยแล้ว

นั้นคือเวลาสร้างแพจเกจ คอมไพล์เลอร์จะสร้างโฟลเดอร์ที่มีชื่อเหมือนแพจเกจ แล้วเก็บคลาสที่เป็นสมาชิกของแพจเกจเอาไว้ใต้โฟลเดอร์นั้น ไฟล์เหล่านี้เป็นไฟล์นามสกุล .class ดังนั้นจึงเป็นคลาสที่คอมไพล์ไว้แล้ว เวลานำไปใช้งานคอมไพล์เลอร์สามารถดึงไปใช้งานได้เลยไม่ต้องคอมไพล์ใหม่

ทีนี้สมมติว่าเราจะสร้างโปรแกรมส่วนที่เหลือให้เสร็จสมบูรณ์ เราสามารถอ้างถึงคลาสในแพจเกจได้ทันที ไม่ต้องเขียนซ้ำอีก ไฟล์โปรแกรมของเราจึงเหลือแค่สองคลาสสุดท้ายที่เราไม่ได้เลือกเอาไปไว้ในแพจเกจ

โปรแกรม 18 - 5 : BuildACar.java

```
class Mechanic {
    void removeWheels(car.Vehicle v) {   // (1)
                v.numberOfWheels = 0;
        }
}

public class BuildACar {
  public static void main(String[] args) {
        car.Truck isuzu = new car.Truck();    //(2)
        car.PrivateCar toyota = new car.PrivateCar(); //(3)
        Mechanic robert = new Mechanic();
```

```
  }
}
```

สังเกตสิ่งที่เปลี่ยนไปก็คือ ทุกครั้งที่มีการอ้างถึงคลาสที่อยู่ในแพจเกจ เราต้องระบุชื่อของแพจเกจก่อนแล้วตามด้วยจุด ดังในบรรทัด (1) (2) (3)

เวลาจะคอมไพล์ไฟล์โปรแกรมที่มีการใช้คลาสในแพจเกจที่สร้างไว้แล้ว จำเป็นที่จะต้องบอกคอมไพล์เลอร์ด้วยว่าแพจเกจถูกเก็บไว้ที่ไหน เราใช้พารามิเตอร์ –classpath ในการระบุที่อยู่ของแพจเกจ ซึ่งในกรณีนี้เราเก็บแพจเกจไว้ที่ C:\java\car คำสั่งในการคอมไพล์จึงเป็นดังนี้

```
C:\java> javac –classpath C:\java BuildACar.java
```

ถ้าเราไม่ต้องการเขียนชื่อแพจเกจทุกครั้งที่มีการอ้างถึงคลาสในแพจเกจ เราทำได้โดยใช้คำสั่ง import ดังนี้

โปรแกรม 18 - 6 : BuildACar.java

```
import car.Vehicle;
import car.Truck;
import car.PrivateCar;

class Mechanic {
    void removeWheels(Vehicle v) {   // (1)
                v.numberOfWheels = 0;
        }
}

public class BuildACar {
  public static void main(String[] args) {
    Truck isuzu = new Truck();        //(2)
    PrivateCar toyota = new PrivateCar(); //(3)
    Mechanic robert = new Mechanic();
  }
}
```

ที่ต้นโปรแกรมเราใช้คำสั่ง import ตามด้วยชื่อเต็มของคลาสในแพจเกจ car ที่ต้องมีการอ้างถึงในโปรแกรมของเราทุกคลาส คราวนี้เราก็ไม่จำเป็นต้องระบุชื่อแพจเกจทุกครั้งที่มีการอ้างถึงคลาสเหล่านี้ในโปรแกรม ดังในบรรทัด (1) (2) (3)

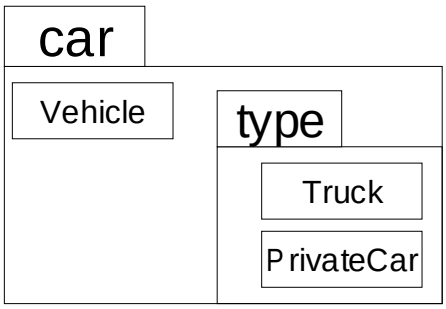
เรายังสามารถย่อคำสั่ง import ได้อีก เนื่องจากทุกคลาสที่เราอิมพอร์ตเข้ามาอยู่ในแพจเกจเดียวกันคือแพจเกจ car ดังนั้นเราสามารถอิมพอร์ตทุกคลาสในแพจเกจ car ได้ด้วยคำสั่งคำสั่งเดียวดังบรรทัด (1) ในโปรแกรมข้างล่างนี้

โปรแกรม 18 - 7 : BuildACar.java

```
import car.*; // (1)


class Mechanic {
    void removeWheels(Vehicle v) {
                v.numberOfWheels = 0;
        }
}

public class BuildACar {
        public static void main(String[] args) {
                Truck isuzu = new Truck();
                PrivateCar toyota = new PrivateCar();
                Mechanic robert = new Mechanic();

        }
}
```

ผลที่ได้ของการสร้างแพจเกจก็เหมือนกับการคอมไพล์ไฟล์ที่มีคลาสทั้งห้าคลาสอยู่ในไฟล์เดียว แต่มีข้อดีคือ ถ้าเราต้องการเขียนโปรแกรมอื่น ที่ต้องใช้คลาส Vehicle Truck หรือ PrivateCar ด้วย เราสามารถ import คลาสเหล่านั้นเข้ามาในซอร์สโค้ดของเราได้ทันทีโดยที่ไม่ต้องเขียนนิยามของคลาสเหล่านี้ซ้ำอีกในซอร์สโค้ดของเรา คอมไพล์เลอร์จะดึงคลาสเหล่านั้นที่คอมไพล์ไว้แล้วไปใช้ได้ทันที

สับแพจเกจ



รูปที่ 18-2 แพจเกจ car และ สับแพจเกจ type

การเก็บคลาสไว้ในแพจเกจสามารถจัดเป็นหมวดหมู่แบบต้นไม้ได้ด้วย ตัวอย่างเช่น แทนที่เราจะจัดคลาส `Vehicle` `Truck` และ `PrivateCar` ไว้ในแพจเกจเดียวกัน เราอาจจัดไล่ระดับกล่าวคือ ให้ `Vehicle` อยู่ในแพจเกจ `car` ส่วน `Truck` และ `PrivateCar` อยู่ในแพจเกจ `type` ซึ่งอยู่ใต้แพจเกจ `car` อีกที เราเรียกแพจเกจ `type` ว่าเป็น **สับแพจเกจ** ของแพจเกจ `car` ถ้าเราต้องการจัดคลาสในลักษณะที่กล่าวมานี้ ซอร์สโค้ดที่เราสร้างจะเป็นดังนี้

โปรแกรม 18 - 8 : Vehicle.java

```
package car;

public class Vehicle {

        int numberOfWheels;
        boolean hasEngine;

        void run(){
                System.out.println("I am running");
        }

}
```

โปรแกรม 18 - 9 : Truck.java

```
package car.type;

import car.Vehicle;
```

```
public class Truck extends Vehicle {

        float maximumLoad;

        void load(float weight) {
                if (weight <= maximumLoad)
        System.out.println("I am carrying a " + weight + "-pound
load.");
        }
}
```

โปรแกรม 18 – 10 : PrivateCar.java

```
package car.type;

import car.Vehicle;

public class PrivateCar extends Vehicle {

        int numberOfPassengers;

        void playCD() {
                System.out.println("CD is playing.");
        }
}
```

ไม่มีอะไรเปลี่ยนแปลงในไฟล์ Vehicle.java เพราะเราเลือกให้คลาส Vehicle อยู่ในแพจเกจ car เหมือนเดิม

ในไฟล์ Truck.java และ PrivateCar.java เราเปลี่ยนชื่อของแพจเกจเป็น car.type ซึ่งหมายความว่าแพจเกจนี้ชื่อแพจเกจ type และเป็นสับแพจเกจของแพจเกจ car

เราต้อง import คลาส car.Vehicle ด้วย เพราะคราวนี้คลาส Vehicle อยู่คนละแพจเกจกับคลาส Truck และ PrivateCar แล้ว แม้ว่า type จะเป็นสับแพจเกจของ car แต่ไม่ได้หมายความว่าแพจเกจ type จะเข้าถึงแพจเกจ car ได้ ลองพิจารณาคำสั่งต่อไปนี้

```
import car.*;
```

คำสั่งนี้อิมพอร์ตเฉพาะ car.Vehicle เท่านั้น ไม่รวม car.type.Truck และ car.type.PrivateCar ถ้าต้องการอิมพอร์ตทั้งสามคลาสควรเขียนเป็น

```
import car.*;
import car.type.*;
```

หรืออีกนัยหนึ่งก็คือ สับแพจเกจไม่มีความสัมพันธ์ใดๆ ทั้งสิ้นกับแพจเกจแม่ของมัน การจัดหมวดหมู่โดยแบ่งเป็นสับแพจเกจนั้นเป็นไปเพื่อการจัดระเบียบของซอร์สโค้ดเท่านั้น และคลาสที่อยู่ในแพจเกจเดียวกันไม่จำเป็นต้องมีความเกี่ยวข้องกันแต่ประการใด

คราวนี้เวลาคอมไพล์ให้ใช้คำสั่งดังนี้

```
C:\java> javac –d C:\java Vehicle.java
C:\java> javac –d C:\java –classpath C:\java Truck.java
C:\java> javac –d C:\java –classpath C:\java PrivateCar.java
```

สังเกตว่าคราวนี้เราต้องระบุ `classpath` ให้ไฟล์ `Truck.java` และ `PrivateCar.java` ด้วย เพราะไฟล์ทั้งสองมีการอิมพอร์ตคลาส `Vehicle`

ส่วนเวลาคอมไพล์โปรแกรมก็ทำได้เหมือนเดิม อย่าลืมอิมพอร์ตแพจเกจ `car.type` ด้วยดังบรรทัด `(1)` ข้างล่างนี้

โปรแกรม 18 - 11 : BuildACar.java

```
import car;
import car.type.*; // (1)


class Mechanic {
    void removeWheels(Vehicle v) {
                v.numberOfWheels = 0;
        }
}

public class BuildACar {

        public static void main(String[] args) {
                Truck isuzu = new Truck();
                PrivateCar toyota = new PrivateCar();
                Mechanic robert = new Mechanic();

        }
}
```

เวลาคอมไพล์ก็สั่งดังนี้

```
C:\java> javac –classpath C:\java BuildACar.java
```

นักเขียนโปรแกรมที่ทำงานจริงๆ อาจใช้แพจเกจที่คนอื่นเขียนขึ้นช่วยในการเขียนโปรแกรมของตัวเอง ดังนั้นโปรแกรมหนึ่งๆ อาจมีการอิมพอร์ตคลาสจากแพจเกตจำนวนมาก ประโยชน์อย่างหนึ่งของการเก็บคลาสไว้เป็นแพจเกจก็คือการป้องกันปัญหาในกรณีที่มีการตั้งชื่อคลาสซ้ำๆ กัน ซึ่งโดยทั่วไปจะนิยมตั้งชื่อแพจเกจตามชื่อบริษัท เช่น แพจเกจที่เขียนโดยบริษัท Oracle อาจตั้งชื่อแพจเกจหลักว่า com.oracle แล้วสร้างสับแพจเกจจำนวนมากเท่าไรก็ได้ตามใจชอบ เช่น com.oracle.util, com.oracle.sql, com.oracle.sql.query เป็นต้น

สมมติว่าในสับแพจเกจ com.oracle.util มีคลาสชื่อ ToString อยู่ แต่บังเอิญนักเขียนโปรแกรมคนเดียวกันใช้แพจเกจของบริษัท BEA ชื่อ com.bea ซึ่งมีคลาสชื่อ ToString อยู่ เราต้องเขียนชื่อเต็มของคลาสทุกครั้งเพื่อเป็นการแยกแยะความแตกต่างแม้ว่าเราจะอิมพอร์ตแล้วก็ตาม ตัวอย่างเช่น

โปรแกรม 18 - 12 : MyProgram.java

```
import com.oracle.util.*;
import com.bea.*;

public class MyProgram {
  public static void main(String[] args) {
    com.oracle.util.ToString tostr1  = new com.oracle.util.ToString();

    com.bea.ToString tostr2 = new com.bea.ToString();
  }
}
```

แพจเกจมาตรฐาน

ในภาษาจาวามีแพจเกจมาตรฐานกลุ่มหนึ่งซึ่งเก็บคลาสพื้นฐานของภาษาจาวาเอาไว้ให้นักเขียนโปรแกรมเมอร์ภาษาจาวาอิมพอร์ตไปใช้ได้เลย ตัวอย่างของแพจเกจมาตรฐานที่ควรรู้จักได้แก่

java.lang

แพจเกจนี้เป็นแพจเกจพื้นฐานที่สุดของภาษาจาวา ตัวอย่างคลาสในแพจเกจนี้ได้แก่ คลาส Math คลาส String คลาสของตัวแปรพื้นฐาน คลาสเกี่ยวกับเอ็กซ์เซฟชั่น และคลาสเกี่ยว

กับเทรด แพจเกจนี้เป็นแพจเกจพิเศษเพราะเวลาต้องการใช้งานคลาสในแพจเกจนี้ไม่ต้องใช้คำสั่ง import สามารถเรียกใช้คลาสในแพจเกจได้โดยตรง

java.util
แพจเกจนี้เก็บคลาสเกี่ยวกับการจัดการวันที่ โซนเวลา การจัดการข้อมูล การจัดการภาษาต่างประเทศ เวลาต้องการใช้งานคลาสในแพจเกจนี้ต้องใช้คำสั่ง import java.util.*;

java.io
แพจเกจนี้เก็บคลาสเกี่ยวกับการเขียนอ่านดิสก์ การรับค่าคีย์บอร์ด และการแสดงผลออกนอกจอ เวลาต้องการใช้งานคลาสในแพจเกจนี้ต้องใช้คำสั่ง import java.io.*;

java.awt
แพจเกจนี้เก็บคลาสเกี่ยวกับการสร้าง GUI ที่เป็นกราฟฟิก เวลาต้องการเรียกใช้งานต้องใช้คำสั่ง import java.awt.*;

java.awt.event
แพจเกจนี้เก็บคลาสเกี่ยวกับการรับคำสั่งจากเมาส์หรือคีย์บอร์ด แพจเกจนี้เป็นสับแพจเกจของแพจเกจ java.awt แต่เวลาใช้งานต้องเรียกด้วยคำสั่งต่างหากคือ java.awt.event.*; เพราะเครื่องหมาย * ไม่นับรวมสับแพจเกจดังที่ได้กล่าวไปแล้ว

javax.swing
แพจเกจนี้เก็บคลาสเกี่ยวกับการสร้าง GUI เช่นเดียวกับ java.awt

ถ้าต้องการทราบว่าคลาสในแพจเกจเหล่านี้มีอะไรบ้าง สามารถค้นคว้าได้จากเวบไซต์ http://java.sun.com การรวมคลาสและแมธธอสหลายๆ อันไว้ในแพจเกจนี้ก็คล้ายๆ กับการจัดระเบียบ API เป็นกลุ่มๆ นั้นเอง

แพจเกจของคุณเอง

โปรแกรมที่มีขนาดใหญ่ๆ ต้องใช้นักเขียนโปรแกรมเป็นสิบเป็นร้อยคน การจะสื่อสารและเชื่อมต่อผลงานระหว่างกันอย่างมีประสิทธิภาพนั้นควรแบ่งงานออกเป็นส่วนย่อยๆ และกระจายให้นักเขียนโปรแกรมแต่ละคนไปรับผิดชอบ เช่น นักเขียนโปรแกรมคนที่หนึ่งรับผิดชอบส่วนติดต่อกับผู้ใช้ นักเขียนโปรแกรมอีกคนหนึ่งรับผิดชอบส่วนจัดการฐานข้อมูล ในขณะที่นักเขียนโปรแกรมอีกคนรับผิดชอบส่วนที่เกี่ยวกับโปรแกรมช่วยเหลือ เป็นต้น

เพื่อให้การเชื่อมงานแต่ละส่วนเข้าด้วยกันทำได้ง่าย นักเขียนโปรแกรมแต่ละคนจะเขียนโปรแกรมในส่วนที่ตัวเองรับผิดชอบเป็นคลาส และเก็บคลาสเหล่านั้นไว้ในแพจเกจ จากนั้นก็ประชาสัมพันธ์ออกไปให้เพื่อนร่วมงานทราบว่า แพจเกจของตนชื่ออะไร มีคลาสและแมธธอสอะไรให้ใช้บ้าง เพื่อนร่วมงานแค่รู้ว่าจะเรียกแมธธอสในคลาสเหล่านั้นได้อย่างไร และคลาสเหล่านั้นทำอะไรได้บ้างก็พอ เพื่อนร่วมงานจะไม่สนใจว่าคลาสและแมธธอสเหล่านั้นสร้างขึ้นได้อย่างไร หรือมีเนื้อหาข้างในเป็นอย่างไรบ้าง เพราะงานส่วนที่พวกเขารับผิดชอบก็น่าปวดหัวพออยู่แล้ว

วิธีการแบบนี้ลดความน่าปวดหัวเวลาทำงานร่วมกันลง ทุกคนที่สร้างคลาสจะซ่อนรายละเอียดในการสร้างคลาสเอาไว้ และประชาสัมพันธ์ให้คนอื่นทราบแต่เฉพาะ วิธีการเรียกใช้คลาสเหล่านั้นเท่านั้น ซึ่งก็คือ คลาสนั้นชื่ออะไร มีตัวแปรคลาสอะไร มีแมธธอสอะไร ทำอะไรได้ เวลาเรียกต้องส่งผ่านตัวแปรอะไรเข้าไป แล้วจะได้ตัวแปรอะไรกลับออกมา ข้อมูลเหล่านี้เราเรียกว่า **คอนแทร็คของคลาสหรือแมธธอส** ส่วนรายละเอียดภายในคลาสที่เราซ่อนไว้ไม่ให้คนอื่นเห็นเรียกว่า **อิมพลีเมนท์เทชั่นของคลาส หรือแมธธอส**

ตัวอย่างเช่น นักเขียนโปรแกรมคนหนึ่งเขียนแมธธอสสแตรติคอันหนึ่งขึ้นมา ดังนี้

```
public static int  squareRoot(int i) {
        return i * i;
}
```

คอนแทร็คของแมธธอสนี้คือ

```
public static int squareRoot(int i)
```

อิมพลีเมนท์เทชั่นของแมธธอสนี้คือ

```
{ return i*i; }
```

พูดง่ายๆ คอนแทร็คของแมธธอสก็คือส่วนหัวของแมธธอส และ อิมพลีเมนท์เทชั่นของแมธธอส ก็คือ ส่วนตัว เวลานักเขียนโปรแกรมคนนี้ต้องการจะบอกคนเพื่อนร่วมงาน เขาอาจเขียนเป็นคู่มือแพจเกจของเขา ซึ่งจะมีแค่ข้อความ

```
public static int squareRoot(int)
```

และอาจมีคำอธิบายว่าแมธธอสนี้ใช้ทำอะไร แค่นี้ผู้อ่านคู่มือก็รู้แล้วว่าจะเรียกแมธธอสนี้ทำอะไรและจะเรียกแมธธอสนี้ได้อย่างไร ส่วนอิมพลีเมนท์เทชั่นของแมธธอสนั้นผู้อ่านไม่สนใจ

19

ตัวกำกับตัวแปรคลาสและแมธธอส

ปกติแล้วคลาสที่เราสร้างขึ้นมาจะสามารถถูกอ้างถึงได้ภายในตัวมันเอง และภายในคลาสอื่นๆ ที่อยู่ในแพจเกจเดียวกัน ถ้าเราต้องการให้คลาสในแพจเกจอื่นๆ อ้างถึงคลาสที่เราสร้างขึ้นได้ด้วยเราจะประกาสคลาสนั้นให้เป็น คลาสสาธารณะ ด้วยการใช้คำสั่ง `public`

เราสามารถกำหนดขอบเขตของการเข้าถึงตัวแปรคลาส และ แมธธอส ได้ในทำนองเดียวกันกับการใช้คำสั่ง `public` ในกรณีของคลาส แต่คำสั่งสำหรับตัวแปรคลาส และ แมธธอส มีมากกว่าแค่คำสั่ง `public` เราเรียกคำสั่งเหล่านี้ว่า **ตัวกำกับตัวแปรคลาส และแมธธอส**

คำสั่ง Private

โดยปกติแล้วทั้งตัวแปรคลาส และแมธธอส สามารถถูกอ้างถึงได้ทั้งในและนอกคลาส ลองพิจารณาตัวอย่างต่อไปนี้

โปรแกรม 19 - 1 : TestACar.java

```
class Vehicle {
```

```
        int numberOfWheels;
        boolean hasEngine;

        Vehicle() {                             // (1)
                numberOfWheels = 10;
                hasEngine = true;
                run();
        }

        void run(){                             // (2)
          numberOfWheels = 4;
          if (numberOfWheels >= 4) {
        System.out.println("I am running");
          }
        }

}

class Truck extends Vehicle {

        float maximumLoad;

        Truck() {                               // (3)
                numberOfWheels = 6;
                hasEngine = true;
                run();
        }

    void load(float weight) {
                if (weight <= maximumLoad)
System.out.println("I am carrying a " + weight + "-pound load.");
    }
}


public class TestACar {
        public static void main (String[] args) {

                Vehicle myCar = new Vehicle();
                if (myCar.hasEngine) {                // (4)
                        myCar.run();                    // (5)
                }
                Truck t = new Truck();
        }
}
```

คลาส `Vehicle` มีตัวแปร `numberOfWheels` `hasEngine` และแมธธอส `run()` ซึ่งสามารถถูกนำไปใช้ที่ไหนก็ได้ภายในคลาส `Vehicle` เช่นในคอนสตรัคเตอร์ในบรรทัด `(1)` หรือในตัวแมธธอส `run()` เองในบรรทัด `(2)` นอกจากนี้ยังถูกนำไปใช้ได้ภายในคลาส `Truck` ซึ่งเป็นสับคลาสของ `Vehicle` เช่นในคอนสตรัคเตอร์ในบรรทัด `(3)` รวมทั้งถูกนำไปใช้ได้ในคลาส `TestACar` ซึ่งไม่ได้สืบทอดคลาส `Vehicle` โดยการประกาศอินสแตนท์ของคลาส

Vehicle และเรียกผ่านอินสแตนท์ดังในบรรทัด (4) และ (5) กล่าวคือสามารถนำไปใช้ได้ทุกที่ในซอร์สไฟล์

เราสามารถบังคับให้ตัวแปรคลาสและแมธธอสถูกนำไปใช้ได้เฉพาะในคลาสที่มันเป็นเจ้าของเท่านั้นได้ด้วยการใช้ คำสั่ง private นำหน้า การเขียนโปรแกรมเชิงวัตถุที่ดีควรกำหนดตัวแปรคลาสให้เป็น private เสมอ ถ้าคลาสอื่นๆ ต้องการใช้งานตัวแปรคลาสนั้นๆ ก็ให้เขียนแมธธอสขึ้นมาไว้สำหรับตั้งค่าโดยเฉพาะ เราสามารถปรับปรุงโปรแกรมข้างต้นของเราให้เป็นโปรแกรมที่ดีตามหลักของการเขียนโปรแกรมเชิงวัตถุได้ดังตัวอย่างข้างล่างนี้

โปรแกรม 19 - 2 : TestACar.java

```
class Vehicle {

        private int numberOfWheels;   // (1)
        private boolean hasEngine;    // (2)

        Vehicle() {                           // (3)
                numberOfWheels = 10;
                hasEngine = true;
                run();
        }

        void setNumberOfWheels(int i) {  // (4)
                numberOfWheels = i;
        }

        void setHasEngine(boolean b) {  // (5)
                hasEngine = b;
        }

        int getNumberOfWheels() {       // (6)
                return numberOfWheels;
        }

        boolean getHasEngine() {        // (7)
                return hasEngine;
        }

        private boolean isReady() {     //  (8)
                return (numberOfWheels = 4 && hasEngine)
        }

        void run(){                           // (9)
                numberOfWheels = 4;
                if (isReady()) {
                        System.out.println("I am running");
                }
        }

}
```

```
class Truck extends Vehicle {

        float maximumLoad;

        Truck() {                               // (10)
                setNumberOfWheels(6);
                setHasEngine(true);
                run();
        }

    void load(float weight) {
                if (weight <= maximumLoad)
        System.out.println("I am carrying a " + weight + "-pound
load.");
    }

}


public class TestACar {
  public static void main (String[] args) {

        Vehicle myCar = new Vehicle();
        if (myCar.getHasEngine()) {              // (11)
        myCar.run();
        }
        Truck t = new Truck();
  }
}
```

ในบรรทัด (1) และ (2) ตัวแปรคลาส numberOfWheels และ hasEngine ถูกกำหนดให้เป็นตัวแปร private ตัวแปรทั้งสองยังคงถูกนำไปใช้ได้ภายในคลาส Vehicle ดังในบรรทัด (3) และ (9) แต่ถ้าต้องการอ้างถึงตัวแปรเหล่านี้นอกคลาส Vehicle ต้องอ้างถึงโดยการผ่านแมธธอสในบรรทัด (4) (5) (6) และ (7)

ตัวอย่างเช่นคอนสตรัคเตอร์ในบรรทัด (10) ต้องเซตค่าตัวแปรคลาสจึงใช้แมธธอส setNumberOfWheels() และ setHasEngine() โดยส่งผ่านค่าคงตัวที่ต้องการเข้าไป

ในบรรทัด (11) แมธธอส main() ต้องการรู้ค่าของตัวแปร hasEngine จึงใช้แมธธอส getHasEngine() ซึ่งส่งค่าของตัวแปร hasEngine ออกมาแทนที่จะเข้าถึงตัวแปร hasEngine โดยตรง

การตั้งชื่อแมธธอสสำหรับการเข้าถึงตัวแปร private นี้นิยมใช้คำว่า set และ get ตามด้วยชื่อของตัวแปรนั้นๆ ในกรณีที่ต้องการเซตค่าและทราบค่าตามลำดับ ทั้งนี้เป็นเพียงความนิยมเท่านั้น ไม่จำเป็นต้องตั้งชื่อแบบนี้เสมอไป

ในกรณีของแมธธอส เราสามารถกำหนดให้เป็น private ได้ด้วย การเขียนโปรแกรมเชิงวัตถุที่ดีจะกำหนดให้แมธธอสใดที่มีประโยชน์เฉพาะในคลาสเป็น private เสมอ ตัวอย่างเช่นในบรรทัด (8) isReady() เป็นแมธธอสที่เขียนขึ้นใช้เฉพาะสำหรับการเช็คความเรียบร้อยของรถยนต์ก่อนออกวิ่ง ดังนั้นจึงมีที่ใช้เฉพาะในแมธธอส run() ในบรรทัด (9) เท่านั้น เราจึงกำหนดค่าแมธธอส isReady() เป็น private ในขณะที่ตัวแมธธอส run() เองมีประโยชน์นอกคลาส Vehicle จึงไม่กำหนดให้เป็น private

การที่ตัวแปรคลาสและแมธธอสถูกกำจัดให้ใช้งานได้เฉพาะแต่ในคลาสด้วยการกำหนดให้เป็น private นี้อย่าสับสนกับการโอเวอร์รายด์ สับคลาสของซูปเปอร์คลาสยังคงสามารถโอเวอร์รายด์ตัวแปรและแมธธอสของซูปเปอร์คลาสได้เสมอไม่ว่าตัวแปรและแมธธอสเหล่านั้นจะถูกกำหนดให้เป็น private หรือไม่

อย่างไรก็ตามการกำหนดตัวแปรคลาสให้เป็น private ก็มีความเสี่ยงอยู่เหมือนกัน เพราะถ้าเราต้องการสร้างสับคลาสของคลาสนั้นในอนาคต และสับคลาสนั้นต้องมีการใช้ตัวแปรคลาสที่เป็น private ที่สืบทอดมา จะมีปัญหาเพราะตัวแปรคลาส private ไม่สามารถถูกกล่าวถึงได้เลยในสับคลาส เราต้องกลับไปแก้ไขซูปเปอร์คลาสอีก ซึ่งเป็นเรื่องไม่น่าพึงประสงค์สำหรับการเขียนโปรแกรมที่ดี

จำไว้ว่าสับคลาสจะสืบทอดตัวแปรคลาสและแมธธอสทั้งหมดของซูปเปอร์คลาสยกเว้นตัวแปรคลาสและแมธธอสที่ประกาศให้เป็น private

คำสั่ง public

เราได้เรียนรู้มาแล้วว่าเราสามารถกำหนดให้ตัวแปรคลาสและแมธธอสถูกอ้างถึงได้เฉพาะในคลาสเท่านั้นด้วยการใช้คำสั่ง private

แต่ถ้าเราไม่กำหนดอะไรเลย ตัวแปรคลาสและแมธธอสจะถูกอ้างถึงได้ทั้งในคลาส และนอกคลาส แต่ต้องเป็นคลาสที่อยู่ในแพจเกจเดียวกันเท่านั้น ถ้าต้องการให้ตัวแปรคลาสและแมธธอสถูกอ้างถึงนอกแพจเกจได้ด้วย เราต้องใช้คำสั่ง public

ทั้งนี้คลาสนั้นต้องเป็น public ด้วยมิฉะนั้นคำสั่ง public สำหรับตัวแปรคลาสและแมธธอสก็ไม่มีประโยชน์อะไร

แมธธอส main() เป็นแมธธอสหนึ่งที่ต้องระบุให้เป็น public เสมอ

คำสั่ง protected

คำสั่ง protected เป็นคำสั่งที่อยู่ตรงกลางระหว่างการไม่ระบุอะไรเลยให้ตัวแปรคลาสและแมธธอส กับการระบุให้เป็น public ตัวแปรคลาสและแมธธอสที่ถูกระบุให้เป็น protected จะถูกอ้างถึงได้ทั้งในคลาส นอกคลาสในแพจเกจเดียวกัน รวมทั้งสับคลาสของคลาสนั้นๆ นอกแพจเกจ คลาสที่ไม่ใช่สับคลาสและอยู่นอกแพจเกจเดียวกันเท่านั้นที่อ้างถึงไม่ได้

ทั้งนี้คลาสนั้นต้องเป็น public ด้วยมิฉะนั้นคำสั่ง protected สำหรับตัวแปรคลาสและแมธธอสก็ไม่มีประโยชน์อะไร

เคล็ดลับ ตามหลักการเขียนโปรแกรมเชิงวัตถุที่เข้มงวด ตัวแปรคลาสควรประกาศเป็น private คลาสใดที่จะอ้างถึงหรือเปลี่ยนค่าของตัวแปรคลาสต้องเข้าถึงผ่านแมธธอสที่สร้างไว้ให้โดยเฉพาะ แต่ถ้าคลาสนั้นมีแนวโน้มที่จะถูกสืบทอดในอนาคต ให้กำหนดเป็น protected แทน เพราะสับคลาสมักต้องอ้างถึงตัวแปรคลาสของซูปเปอร์คลาส