SE-ED COMPUTER BOOK developer

การเขียนโปรแกรม Java และ Android

เขียนโปรแกรมด้วยฟีเจอร์ใหม่ล่าสุดของ Java New API และ Android Studio

- การเขียนภาษา Java โดยใช้ IntelliJ IDEA Community
- การพัฒนา Mobile App ด้วย Android Studio
- ครอบคลุมทุกองค์ประกอบสำคัญของระบบ Android

บัญชา ปะสีละเตสัง

ผู้เขียนหนังสือขายดีระดับ Best Seller ด้าน Programming

DOWNLOAD

ดาวน์โหลดโค้ดหนังสือเล่มนี้ได้ที่
http://www.developerthai.com

ซีเอ็ด

การเขียนโปรแกรม Java และ Android

โดย บัญชา ปะสีละเตสัง

ข้อมูลทางบรรณานุกรมของหอสมุดแห่งชาติ

บัญชา ปะสีละเตสัง.

การเขียนโปรแกรม Java และ Android.--กรุงเทพฯ : ซีเอ็ดยูเคชั่น, 2559.
684 หน้า.
1. การเขียนโปรแกรม (คอมพิวเตอร์). 2. จาวา (ภาษาคอมพิวเตอร์).
I. ชื่อเรื่อง

005.133

Barcode (e-book) : 9786160841691

ผลิตและจัดจำหน่ายโดย

บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน)
SE-EDUCATION PUBLIC COMPANY LIMITED

เลขที่ 1858/87-90 ถนนเทพรัตน แขวงบางนาใต้ เขตบางนา กรุงเทพฯ 10260
โทรศัพท์ 0-2826-8000

[หากมีคำแนะนำหรือติชม สามารถติดต่อได้ที่ comment@se-ed.com]

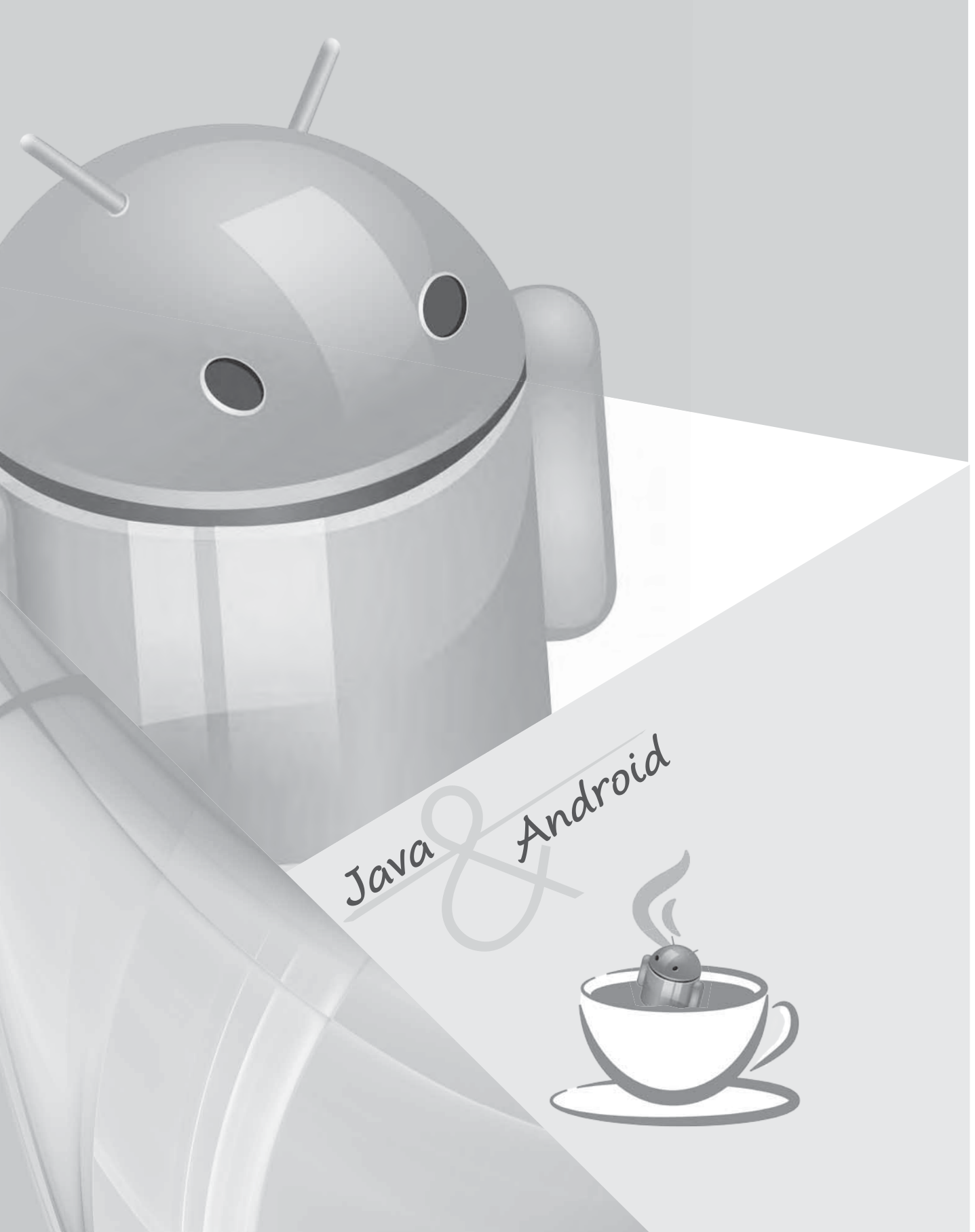
คำนำ

Java เป็นภาษาคอมพิวเตอร์ที่มีผู้ใช้งานมากที่สุดในปัจจุบัน เนื่องจากการมีโครงสร้างที่สละสลวย เรียนรู้ง่าย และโดยเฉพาะอย่างยิ่งการทำงานข้าม Platform ได้ ส่งผลให้ Java ถูกนำไปใช้อย่างแพร่หลายทั้งในภาคธุรกิจ อุตสาหกรรม รวมถึงเทคโนโลยีสื่อสารต่างๆ และปัจจัยสำคัญอย่างหนึ่งที่ผลักดันให้มีผู้สนใจภาษานี้กันอย่างล้นหลาม ก็คือ การที่ Java ถูกเลือกใช้เป็นเครื่องมือในการพัฒนาแอปพลิเคชั่นบนระบบ Android ซึ่งเป็นระบบปฏิบัติการสำหรับอุปกรณ์เคลื่อนที่ที่มีผู้ใช้งานมากที่สุด และสามารถศึกษาทดลองบนคอมพิวเตอร์ทั่วไปที่เป็นระบบวินโดวส์ได้ นอกจากนี้ยังสามารถเคลื่อนย้ายแอปไปยังเครื่องจริงได้โดยตรง ดังนั้นการพัฒนาแอปสำหรับ Android จึงมีผู้สนใจเรียนรู้กันเป็นจำนวนมาก

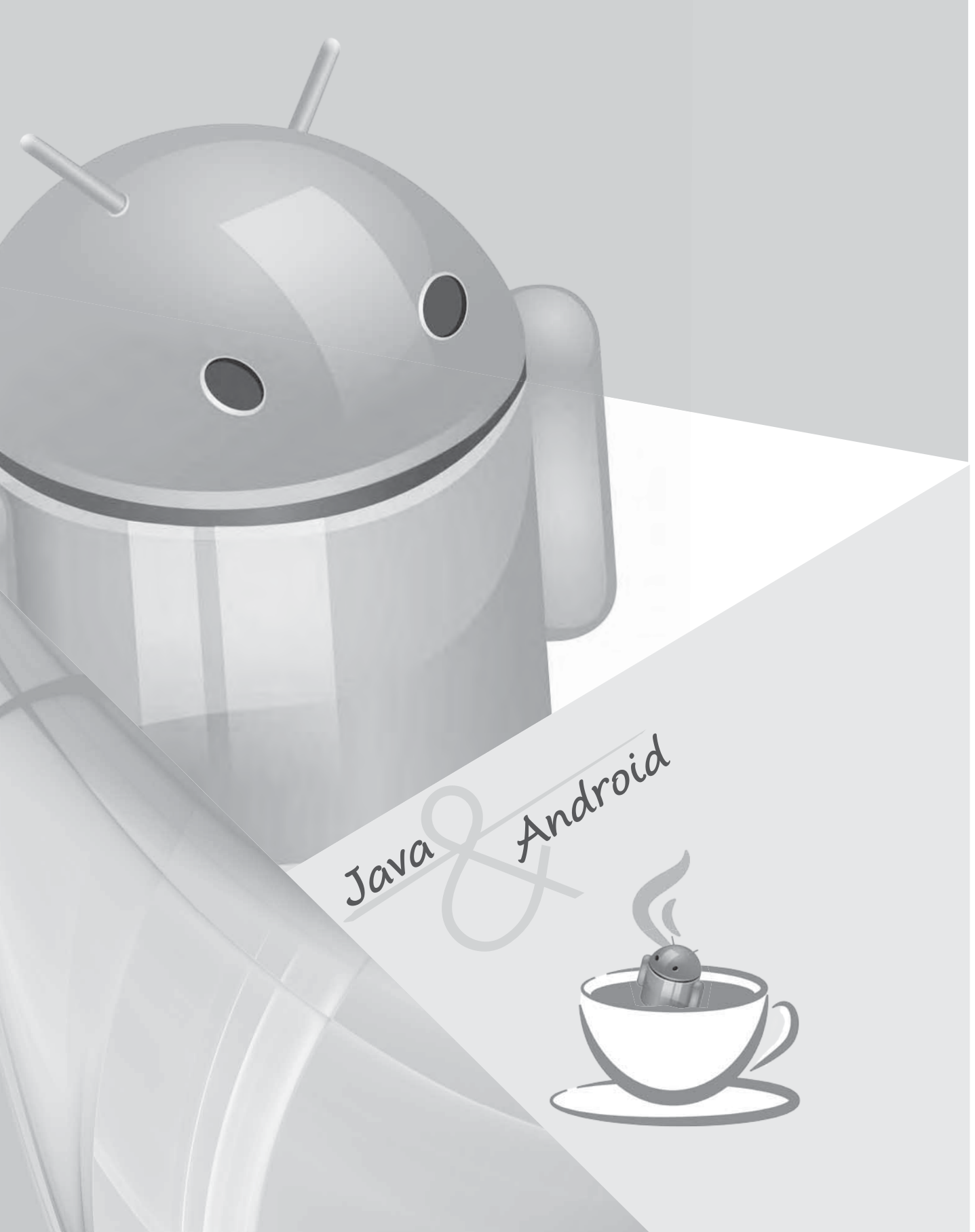
ดังนั้นเพื่อให้เราสามารถศึกษาทั้งการเขียนโปรแกรม Java และ Android ควบคู่กันไปได้อย่างต่อเนื่อง ผู้เขียนจึงได้รวบรวมเนื้อหาสองอย่างนี้มาไว้ในหนังสือเล่มนี้ทั้งหมด โดยเริ่มจากพื้นฐานสำคัญที่เราต้องรู้เกี่ยวกับ Java ไปจนถึงการพัฒนาแอปด้วย Android Studio ซึ่งองค์ประกอบที่สำคัญและจำเป็นต้องรู้ก็จะมีไว้ให้อย่างครบถ้วน ผู้เขียนก็หวังว่าหนังสือเล่มนี้จะช่วยให้ผู้อ่านสามารถเขียนโปรแกรม Java และ Android เพื่อพัฒนาแอปพลิเคชั่นที่สามารถใช้งานได้จริงตามต้องการ

บัญชา ปะสีละเตสัง

banchar_pa@yahoo.com
facebook.com/DeveloperThai


Java & Android

สารบัญ


Java & Android

1

Java Basic & Data Type

Java เป็นภาษาคอมพิวเตอร์ที่ถูกนำไปใช้พัฒนาซอฟแวร์กันอย่างแพร่หลาย ไม่ว่าจะเป็นทางด้านธุรกิจ อุตสาหกรรม หรือเทคโนโลยีสารสนเทศ และโดยเฉพาะอย่างยิ่งการพัฒนาแอปพลิเคชั่นบนระบบ Android ก็ใช้ภาษา Java เป็นเครื่องมือในการพัฒนาเช่นกัน ดังนั้นก่อนที่เราจะก้าวไปสู่การเป็นนักพัฒนาแอปพลิเคชั่นบน Android ก็จำเป็นต้องเขียนโปรแกรมภาษา Java ให้ได้ก่อน โดยในบทนี้จะกล่าวถึงสิ่งที่เราควรต้องรู้ตั้งแต่เริ่มแรก เช่น การติดตั้งเครื่องมือและการเซตค่าที่จำเป็นต่างๆ รวมไปถึงองค์ประกอบของภาษาที่สำคัญในเบื้องต้น เพื่อเป็นพื้นฐานสำหรับการนำไปใช้งานในบทต่อๆ ไป

เกี่ยวกับภาษาจาวา

Java เป็นภาษาที่สร้างขึ้นโดย James Gosling ให้กับบริษัท Sun Microsystems และได้เริ่มใช้งานมาตั้งแต่ปี ค.ศ. 1995 ซึ่งชื่อของภาษานี้ได้มาจากการใช้โปรแกรมสุ่มเลือกคำ เมื่อคำว่า “Java” ถูกเลือก รูปถ้วยกาแฟก็ถูกใช้เป็นโลโก้ด้วย เนื่องจากคำว่า Java ในภาษาอังกฤษหมายถึง “กาแฟ” นั่นเอง

จุดเด่นของจาวา

ปัจจุบันจาวากลายเป็นภาษาคอมพิวเตอร์ที่มีผู้ใช้งานมากที่สุดอีกภาษาหนึ่งของโลก ซึ่งจุดเด่นบางประการที่ทำให้จาวาได้รับความนิยมอย่างกว้างขวาง มีดังนี้คือ

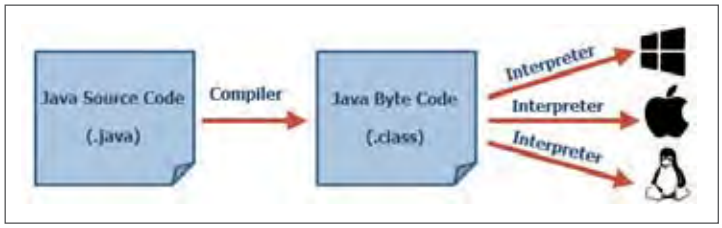
- จาวามีลักษณะโครงสร้างคล้ายกับ C/C++ แต่มีรูปแบบที่สละสลวยและเรียนรู้ได้ง่ายกว่า
- จาวาจะทำงานอยู่บน Java Virtual Machine (JVM) ซึ่งสามารถนำไปติดตั้งในระบบหรืออุปกรณ์ที่หลากหลาย ดังนั้นจึงสามารถทำงานข้ามแพลตฟอร์มได้ (Platform Independent) ถ้าหากว่าอุปกรณ์เหล่านั้นมี JVM ติดตั้งอยู่ ตามแนวทางของจาวาที่ว่า **Write Once Run Anywhere**
- จาวาเป็นภาษาที่ยึดหลักการเขียนโปรแกรมแบบ OOP อย่างเข้มงวด ดังนั้นเราจึงสามารถใช้ประโยชน์จากข้อกำหนดต่างๆ ของ OOP เพื่อพัฒนาแอปพลิเคชั่นได้อย่างเต็มประสิทธิภาพ
- จาวาสามารถแก้ไขข้อบกพร่องหลายอย่างในภาษา C++ เช่น Pointer, Garbage Collection, การจัดการหน่วยความจำ รวมถึงหลักการด้าน OOP ของ C++ ที่อาจก่อให้เกิดปัญหาด้วย
- จาวามีความคงทน (Robust) ในการทำงานมากกว่า เนื่องจากจาวามีกลไกการป้องกันข้อผิดพลาดต่างๆ ที่อาจเกิดขึ้นจนเป็นสาเหตุให้แอปพลิเคชั่นต้องหยุดการทำงานลงกลางครันได้
- จาวาถูกใช้ในธุรกิจและอุตสาหกรรมต่างๆ มากมาย เช่น การเงินการธนาคาร, สื่อสารโทรคมนาคม, ไอที, อิเล็กทรอนิกส์, มัลติมีเดีย, เกม ฯลฯ
- จาวานั้นถูกเลือกให้เป็นภาษาในการพัฒนาแอปพลิเคชั่นบน Android ดังนั้นผู้ที่จะศึกษาการเขียนโปรแกรมสำหรับระบบแอนดรอยด์ จึงจำเป็นต้องมีพื้นฐานการเขียนจาวามาก่อนด้วย

อย่างไรก็ตาม แม้ว่า Sun Microsystems จะเป็นเจ้าของภาษาจาวามาตั้งแต่เริ่มแรก แต่ปัจจุบันบริษัทนี้ได้ถูกซื้อกิจการไปโดยบริษัท Oracle ทำให้จาวาจึงกลายเป็นลิขสิทธิ์ของ Oracle ไปด้วย ดังนั้นข้อมูลต่างๆ เกี่ยวกับจาวาจึงถูกนำไปรวมให้เป็นส่วนหนึ่งของเว็บไซต์ Oracle ไปด้วย

หลักการประมวลผลของจาวา

ตามปกติแล้ว โปรแกรมที่พัฒนาขึ้นมาจากแพลตฟอร์ม (Platform) หนึ่งก็จำกัดการใช้งานอยู่แต่บนแพลตฟอร์มนั้นเท่านั้น ไม่สามารถนำไปใช้ข้ามแพลตฟอร์มได้ ภาษาจาวาจึงได้ถูกสร้างขึ้นเพื่อแก้ปัญหานี้ โดยโปรแกรมที่เขียนด้วยจาวาไม่ว่าบนแพลตฟอร์มใดก็ตาม จะสามารถนำไปใช้งานบนแพลตฟอร์มอื่นๆ ได้ทันที แต่การจะทำงานแบบข้ามแพลตฟอร์มได้ ต้องอาศัยการประมวลผลคำสั่งใน 2 รูปแบบร่วมกันคือ

- **การคอมไพล์ (Compile)** โดยโค้ดเขียนเป็นภาษาจาวา (มีส่วนขยายเป็น **.java**) จะถูกคอมไพล์ออกมาเป็น Java byte code (มีส่วนขยายเป็น **.class**) ซึ่งเป็นโค้ดมาตรฐานในการทำงานข้ามแพลตฟอร์มของจาวา
- **การแปลคำสั่ง (Interpret)** เมื่อนำ Java byte code นั้นไปรันบนแพลตฟอร์มใดก็ได้ที่มี Java Virtual Machine สำหรับแพลตฟอร์มนั้นอยู่ Java byte code นั้นจะถูกแปลคำสั่ง (Interpret) ให้ทำงานตามที่กำหนด



การติดตั้ง JDK และ IntelliJ

จาวานั้นก็เหมือนกับโปรแกรมภาษาทั่วไป ที่เราต้องนำมาติดตั้งลงไปในระบบนั้นๆ ก่อนจึงจะสามารถใช้งานได้ โดยชุดเครื่องมือในการพัฒนาจาวาเราเรียกว่า Java Development Kit (JDK) นี่คือสิ่งแรกที่เราต้องติดตั้งลงไป แต่เนื่องจาก JDK นั้นจะไม่มี Editor สำหรับเขียนโค้ดมาให้ ดังนั้นในหนังสือเล่มนี้ผู้เขียนจะเลือกใช้ IntelliJ เนื่องจากทาง Google ได้นำ IntelliJ ไปพัฒนาต่อยอดเป็น Android Studio (จากเดิมที่ใช้ Eclipse) ดังนั้นหากเราคุ้นเคยกับ IntelliJ มาก่อนแล้ว เมื่อไปพัฒนาแอปบน Android Studio ก็ไม่ต้องเสียเวลาไปเริ่มต้นใหม่หรือปรับตัวอะไรมาก เพราะสามารถเรียนรู้เพิ่มเติมเฉพาะส่วนที่เกี่ยวกับแอนดรอยด์ได้เลย สำหรับแนวทางการติดตั้ง JDK และ IntelliJ มีดังนี้

การติดตั้ง JDK

การติดตั้ง JDK มีขั้นตอนดังต่อไปนี้

1. เราสามารถดาวน์โหลด JDK ได้ที่เว็บไซต์ https://java.com/en/download/ โดยให้เลือกรุ่น **Standard Edition (JSE)** และเลือกชนิดไฟล์ติดตั้งให้ตรงกับระบบปฏิบัติการที่เราใช้งานอยู่ ซึ่งเวอร์ชั่นล่าสุดในขณะที่เขียนหนังสือเล่มนี้คือ Java 8
2. จากนั้นก็ให้ติดตั้งลงไปเหมือนโปรแกรมทั่วๆ ไป ซึ่งหลังการติดตั้ง หากจะใช้ตามแนวทางของหนังสือเล่มนี้ก็ไม่จำเป็นต้องเซตค่าใดๆ

การติดตั้ง IntelliJ

IntelliJ เป็นโปรแกรมประเภท IDE (Integrated Development Environment) ที่สร้างโดยบริษัท JetBrains ซึ่งเป็นบริษัทที่มีความเชี่ยวชาญในการสร้าง IDE สำหรับภาษาคอมพิวเตอร์ต่างๆ มากมาย จุดเด่นของ IntelliJ คือรูปร่างหน้าตาที่ดูเรียบร้อย สวยงาม นอกจากนี้ยังมีเครื่องมือมาช่วยทั้งในการเขียนโค้ดและการออกแบบต่างๆ อย่างครบถ้วน สมบูรณ์แบบ แล้วยังมีรุ่นที่สามารถนำมาใช้งานได้ฟรีอีกด้วย โดยขั้นตอนการติดตั้ง IntelliJ มีดังต่อไปนี้

1. เราสามารถดาวน์โหลด IntelliJ ได้จากเว็บไซต์ https://www.jetbrains.com/idea/download โดยให้เลือกรุ่น **Community Edition** ซึ่งเป็นรุ่นที่สามารถนำมาใช้ได้ฟรี
2. หลังจากนั้นก็ติดตั้งลงไปเหมือนกับโปรแกรมทั่วๆ ไป

การสร้างและจัดการโปรเจ็กต์บน IntelliJ

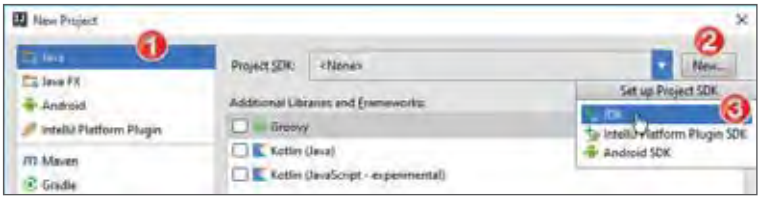
การจะเปิดเข้าสู่ IntelliJ ได้ ต้องเป็นการสร้างโปรเจ็กต์ใหม่ หรือเปิดโปรเจ็กต์ที่สร้างเอาไว้แล้ว โดยหลักการที่เราควรทราบในเบื้องต้นมีดังนี้

การสร้างโปรเจ็กต์ใหม่

ขั้นตอนการสร้างโปรเจ็กต์ใหม่ มีดังต่อไปนี้

1. ให้เปิดเข้าสู่ IntelliJ เมื่อปรากฏหน้าจอ Welcome ให้เลือก Create New Project

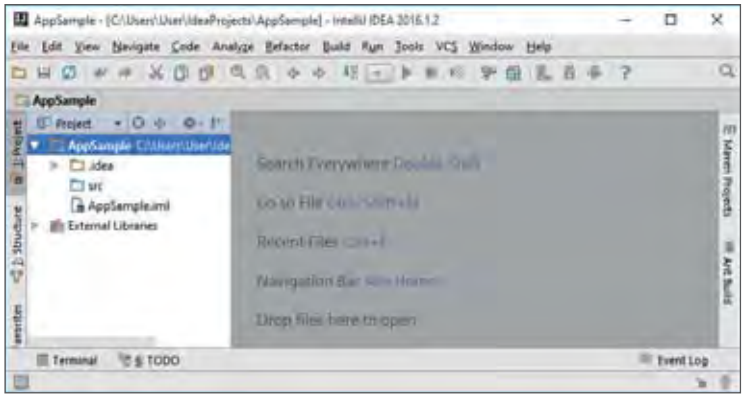
2. กรณีที่เราเริ่มใช้งาน IntelliJ เป็นครั้งแรก ต้องเซตตำแหน่งที่ติดตั้ง JDK เอาไว้ โดยกรอบด้านซ้ายเลือก **Java** แล้วคลิกที่ **ปุ่ม New...** แล้วเลือก **JDK** ตามลำดับ



3. จากนั้นเปิดไปยังตำแหน่งที่เราตั้งจาวาเอาไว้ แล้วให้เลือกโฟลเดอร์ jdk_xxx (ไม่ใช่ jre_xxx) ซึ่งการเซตตำแหน่ง JDK นี้ จะทำเฉพาะเมื่อสร้างโปรเจ็กต์บน IntelliJ เป็นครั้งแรกเท่านั้น แล้วครั้งต่อๆ ไป เราไม่ต้องเซตค่านี้อีก เพราะโปรแกรมจะจดจำตำแหน่งนี้ไว้ให้เราโดยอัตโนมัติ ยกเว้นแต่ว่า เราจะเปลี่ยนตำแหน่งการติดตั้ง JDK ใหม่ หรืออัปเกรดเป็นเวอร์ชั่นใหม่

4. ขั้นตอนถัดไปคลิกปุ่ม Next แล้วจากนั้นจะเป็นการกำหนดชื่อโปรเจ็กต์ และตำแหน่งในการจัดเก็บ ซึ่งในที่นี้จะใช้ตำแหน่งดีฟอลต์ของ IntelliJ (ปกติจะเป็น C:\Users\user\IdeaProjects)

5. หลังจากนั้นก็จะเข้าสู่หน้าจอหลักของ IntelliJ ดังภาพ

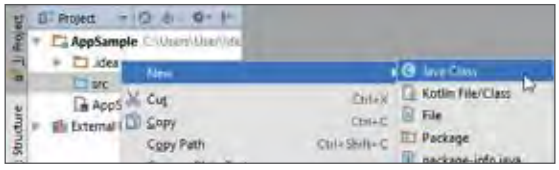


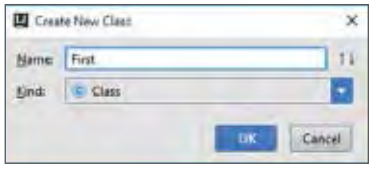
ถ้าเราต้องการปิดเฉพาะโปรเจ็กต์ โดยไม่ปิดโปรแกรม IntelliJ (อาจเพื่อสร้างหรือเปิดโปรเจ็กต์ใหม่) ให้เลือกที่เมนู **File > Close Project**

การสร้างคลาสใหม่

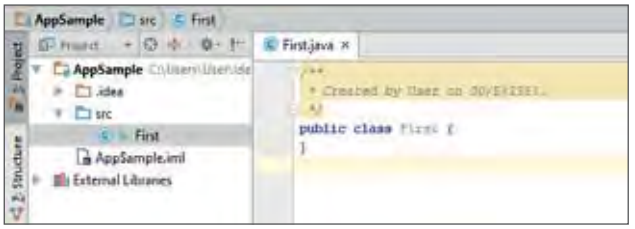
หากจะเขียนโค้ดจาวา ต้องเริ่มต้นที่การสร้างคลาส (Class) ขึ้นมาก่อน สำหรับรายละเอียดเกี่ยวกับคลาสจะอธิบายเพิ่มเติมในภายหลัง ให้รู้วิธีการล่วงหน้าเอาไว้ก่อน โดยทำตามขั้นตอนต่อไปนี้

1. ให้คลิกขวาตรงโฟลเดอร์ src แล้วเลือก **New > Java Class**
2. กำหนดชื่อคลาส แล้วคลิกปุ่ม OK





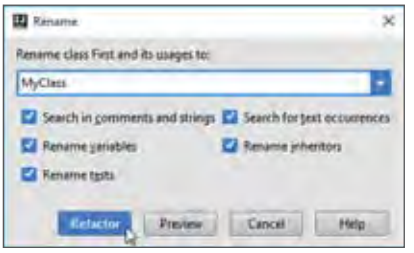
3. ลักษณะคลาสที่โปรแกรมสร้างให้เป็นดังนี้



การลบและเปลี่ยนชื่อคลาส

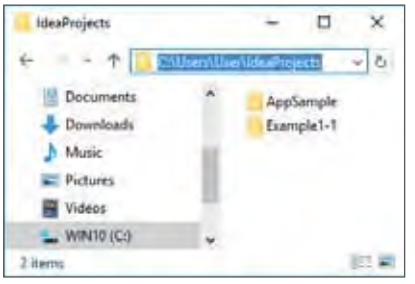
สำหรับคลาสที่สร้างขึ้น หากต้องการลบหรือเปลี่ยนชื่อ ก็ให้ทำดังนี้คือ

- **การลบคลาส** ให้คลิกขวาที่ชื่อคลาสตรงโฟลเดอร์ src แล้วเลือก **Delete** ได้เลย
- **การเปลี่ยนชื่อคลาส** จะเปลี่ยนที่โค้ดโดยตรงเลยไม่ได้ เนื่องจากมีส่วนที่เกี่ยวข้องอีกหลายแห่งที่อ้างอิงชื่อคลาสที่เราได้สร้างขึ้น ทั้งนี้หากเราจะเปลี่ยนชื่อคลาส ให้คลิกขวาที่ชื่อคลาส > **เลือก Refactor > Rename** จากนั้นก็กำหนดชื่อคลาสใหม่ตามที่ต้องการ ซึ่งหลังการเปลี่ยนชื่อโปรแกรม IntelliJ จะไปเปลี่ยนส่วนต่างๆ ทั้งหมด ที่อ้างถึงคลาสชื่อเดิมให้โดยอัตโนมัติ

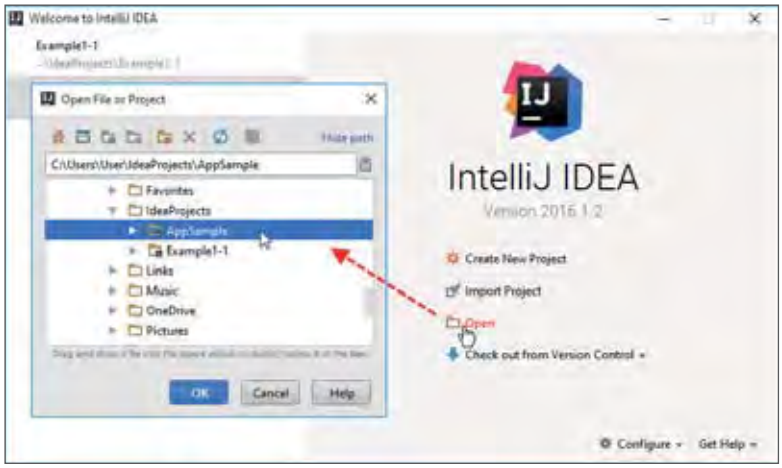


การลบและเปิดโปรเจ็กต์เก่า

โปรเจ็กต์ที่เราสร้างขึ้นมานี้ โดยปกติแล้ว IntelliJ จะเก็บไว้ที่ **C:\Users\user\IdeaProjects** ซึ่งแต่ละโปรเจกต์ก็จะถูกแยกเก็บไว้คนละโฟลเดอร์ ดังภาพขวามือ ถ้าเราจะลบโปรเจ็กต์ใด ก็ให้เปิดมาที่นี่แล้วลบโฟลเดอร์ของโปรเจ็กต์นั้นทิ้งไป



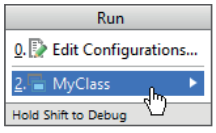
สำหรับโปรเจ็กต์ที่เราได้สร้างเอาไว้แล้ว ถ้าต้องการเปิดเพื่อพัฒนาต่อ หรือปรับปรุงแก้ไข ก็ให้เปิดเข้าสู่ IntelliJ IDEA (หรือถ้าเปิดโปรเจ็กต์เดิมอยู่ก็ให้ปิดโปรเจ็กต์นั้นก่อน) ซึ่งเมื่อปรากฏหน้าจอ Welcome to IntelliJ จะมีลิสต์แสดงรายชื่อโปรเจ็กต์ที่เคยเปิดล่าสุดจำนวนหนึ่งมาให้เราเลือกดังภาพในหน้าถัดไป แต่ถ้าโปรเจ็กต์ที่เราต้องการเปิดไม่มีแสดงอยู่ในลิสต์ ให้คลิกที่เมนู Open จากนั้นก็ให้เปิดเข้าไปยังตำแหน่งที่จัดเก็บโปรเจ็กต์เอาไว้ ตามค่าดีฟอลต์คือ C:\Users\user\IdeaProjects ดังที่กล่าวมาแล้ว ต่อไปดับเบิลคลิกที่ชื่อโปรเจ็กต์ที่ต้องการ



การรันโปรแกรม

การรัน (Run) ก็คือการสั่งให้โปรแกรมประมวลผลโค้ดที่เราเขียนขึ้นและทำงานตามที่กำหนด โดยเราสามารถสั่งรันโปรแกรมได้ดังนี้คือ

- **การรันครั้งแรก** ให้คลิกที่เมนู **Run > Run...** (หรือกด <Alt + Shift + F10>) หลังจากนั้นจะปรากฏรายชื่อสิ่งที่จะรัน ก็ให้เราเลือก Main Class ของโปรเจ็กต์ (คลาสที่มีเมธอด main)



- **การรันครั้งต่อๆ ไป** สามารถคลิกที่ไอคอน ▶ บนทูลบาร์ได้เลย (หรือกด <Shift + F10>) หากไม่ปรากฏไอคอนนี้บนทูลบาร์ ให้เลือกที่เมนู **View > Toolbar**

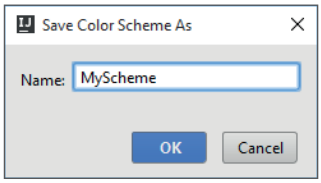
การปรับแต่ง Editor ของ IntelliJ

ลักษณะสีและฟอนต์ของโค้ดที่เขียนก็อาจมีผลต่อความรู้สึกในการทำงานของเราด้วย นอกจากนี้ยังต้องเลือกรูปแบบที่สามารถรองรับอักขระในภาษาไทยได้ โดยมีแนวทางการปรับแต่งดังต่อไปนี้

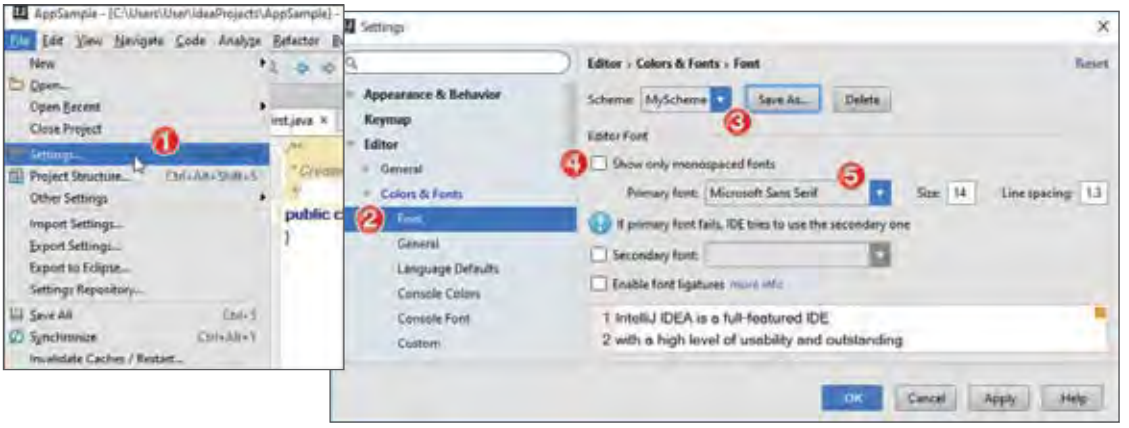
การกำหนดชนิดฟอนต์ของ Editor

ชนิดฟอนต์ที่ IntelliJ เลือกเป็นค่าดีฟอลต์ไว้ให้นั้น ไม่ใช่ฟอนต์ที่รองรับภาษาไทย จึงอาจมีปัญหาในการแสดงข้อความที่เป็นภาษาไทยได้ ก็ให้เรากำหนดชนิดฟอนต์ใหม่ดังนี้

1. ที่เมนูของ IntelliJ เลือก **File > Settings...**
2. ในกรอบด้านซ้ายเลือก **Editor > Colors & Fonts > Font**



3. ถ้าเรายังไม่เคยปรับแต่งใดๆ มาก่อนต้องสร้าง Scheme ที่เป็นของเราเองก่อน โดยตรงช่องการกำหนด Scheme ให้คลิกปุ่ม **Save As...** แล้วตั้งชื่อ Scheme อะไรก็ได้ เช่น MyScheme หลังจากสร้าง Scheme ขึ้นมาแล้ว ต้องเลือก Scheme ที่เราสร้างนั้นด้วย
4. ให้เอาเช็กตรงตัวเลือก **Show only monospaced fonts** ออก
5. จากนั้นตรงช่อง **Primary font** ให้เลือกชนิดฟอนต์ที่เราต้องการ และควรเลือกฟอนต์ที่รองรับภาษาไทย เช่น ในที่นี้ผู้เขียนเลือก Microsoft Sans Serif ขนาด 14 และ Line Spacing 1.3

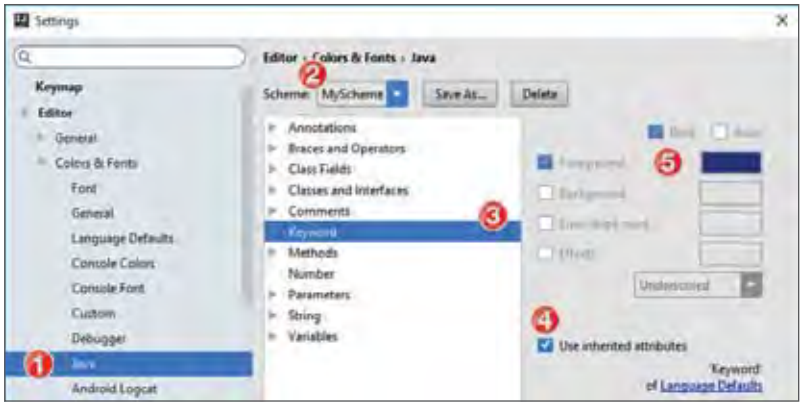


การกำหนดสีและรูปแบบฟอนต์ของ Editor

ในส่วนของการกำหนดสีนั้น ผู้อ่านต้องรู้จักองค์ประกอบในการเขียนจาวามาบ้างแล้ว เช่น คีย์เวิร์ด สตริง คลาส เมธอด ฯลฯ จึงจะพอแยกได้ว่า อันไหนควรกำหนดสีอะไร แต่หากยังไม่รู้จักองค์ประกอบเหล่านี้มาก่อน ก็ดูผ่านๆ ให้ทราบหลักการไปก่อน แล้วหลังจากที่เราคุ้นเคยกับการเขียนโปรแกรมดีแล้ว ค่อยมาปรับแต่งในภายหลังก็ได้ โดยการกำหนดสีของโค้ด มีขั้นตอนดังต่อไปนี้

1. ไปที่เมนู **File > Settings...** แล้วเลือกที่ **Editor > Colors & Fonts > Java**
2. ถ้าเราได้สร้าง **Scheme** สำหรับปรับแต่งฟอนต์เอาไว้แล้ว ก็ให้เลือก Scheme นั้น แต่ถ้ายังไม่เคยสร้าง Scheme มาก่อน ก็ให้ทำเช่นเดียวที่กล่าวไว้ในหัวข้อการกำหนดฟอนต์ (Scheme ของทั้งสีและฟอนต์ต้องเป็นอันเดียวกัน)
3. ให้คลิกเลือกจากลิสต์ว่าเราต้องการกำหนดลักษณะฟอนต์ขององค์ประกอบใด เช่น Class, Method, Field, Keyword เป็นต้น
4. บางกรณี ถ้าจะเลือกรูปแบบใหม่ ต้องเอาเช็กที่ตัวเลือก **Use inherited attributes** ออกด้วย

5. เช็กเลือกตรง **Foreground** แล้วคลิกกำหนดสีที่ต้องการได้เลย นอกจากนี้เราอาจเลือกหรือยกเลิกการทำตัวหนา (Bold) หรือตัวเอียง (Italic) เพิ่มเติมก็ได้



การกำหนดชนิดฟอนต์ของ Console

Console ในที่นี้คือหน้าจอสำหรับแสดงผลการรันโปรแกรม ทั้งนี้เราควรกำหนดฟอนต์ของ Console เป็นชนิดที่รองรับภาษาไทย เผื่อมีกรณีที่ต้องการแสดงข้อความเป็นภาษาไทยด้วย ดังขั้นตอนต่อไปนี้

1. ให้เปิดเข้าสู่หน้าจอ Settings แล้วเลือกที่โหนด **Editor > Colors & Fonts > Console Font**
2. เลือก **Scheme** ที่ได้สร้างเอาไว้
3. เอาเช็กตรง Show only monospaced fonts ออก
4. เลือกชนิดฟอนต์ที่ต้องการ ซึ่งควรเป็นฟอนต์ที่แสดงผลภาษาไทยได้ เช่น Microsoft Sans Serif แล้วกำหนดขนาด (Size) และ Line Spacing ตามที่ต้องการ

การเขียนโปรแกรมด้วยจาวา

การเขียนโปรแกรม ไม่ว่าจะเป็นระดับพื้นฐานหรือระดับสูง ก็จำเป็นต้องใช้องค์ประกอบพื้นฐานบางอย่างเหมือนกัน สำหรับในภาษาจาวา ก็มีสิ่งที่เราควรรู้จักในเบื้องต้นมีดังต่อไปนี้

คลาส

เนื่องจากจาวานั้นเป็นภาษาแบบ Object Oriented Programming (OOP) ซึ่งทุกอย่างจะถูกกำหนดไว้ภายในคลาส (Class) โดยรูปแบบอย่างง่ายของคลาสมีดังนี้ (รายละเอียดทั้งหมดอยู่ในบทที่ 3 แต่ในบทนี้ จะแนะนำให้รู้จักคร่าวๆ ไว้ก่อน เนื่องจากเป็นองค์ประกอบที่จำเป็นต้องมี)

```
class ชื่อคลาส {

}
```

- **class** เป็นคีย์เวิร์ดที่บ่งบอกว่าเป็นคลาส
- **ชื่อคลาส** ควรกำหนดให้สื่อความหมายว่าเป็นคลาสเกี่ยวกับอะไร

ตัวอย่างการสร้างคลาสที่ชื่อ First

```
class First {

}
```

บล็อก { }

ในจาวาจะใช้วงเล็บ {...} ในการกำหนดจุดเริ่มต้นและจุดสิ้นสุดของบล็อกการทำงานในแต่ละส่วน เช่น คลาส, เมธอด หรือการทำงานตามเงื่อนไขและลูปต่างๆ นอกจากนี้เรายังนำ {...} มาใช้ในอีกหลายกรณี นอกจากนี้ภายในบล็อก {...} อาจมีบล็อกย่อยๆ ซ้อนลงไปอีกตามความซับซ้อนของโปรแกรม เช่น

```
class Example {
   public static void main(String[] args) {
      for(...) {
         if(...) {
            ...
         } else {
            ...
         }
      }
   }
}
```

การแทรกคำอธิบาย

การแทรกคำอธิบาย (Comment) ไว้ในโค้ด จะช่วยให้อ่านโค้ดได้เข้าใจง่ายยิ่งขึ้น รวมถึงกรณีที่เรากลับมาแก้ไขโค้ดในภายหลัง ก็จะช่วยให้เราทราบรายละเอียดของโค้ดตรงนั้นได้เร็วกว่า ทั้งนี้การแทรกคำอธิบายในจาวาทำได้ดังนี้

การใช้สัญลักษณ์ //

สำหรับคำอธิบายเพียงบรรทัดเดียว โดยโปรแกรมจะถือว่าตั้งแต่เครื่องหมาย // ไปจนถึงสิ้นสุดบรรทัดเป็นคำอธิบายทั้งหมด จะไม่นำมาพิจารณาในโปรแกรม อาจเขียนไว้ในคลาสหรือนอกคลาสก็ได้ เช่น

```
//โปรแกรมนี้เขียนเมื่อวันที่ 1 มิถุนายน 2559
class Employee {
   private String name;    //ชื่อ
   private int age;        //อายุ
}
```

การใช้สัญลักษณ์ /* */

กรณีที่คำอธิบายของเรานั้นยาวหลายบรรทัด การใช้ // หลายๆ ครั้งอาจไม่สะดวกนัก เราอาจเปลี่ยนมาใช้ /*...*/ แทนได้ โดยโปรแกรมจะถือว่าตั้งแต่สัญลักษณ์ /* เป็นต้นไป จะเป็นคำอธิบายทั้งหมด จนกว่าจะเจอสัญลักษณ์ */ จึงจะถือว่าเป็นการสิ้นสุดคำอธิบาย เช่น

```
/* โปรแกรมนี้เขียนโดย นายจาวา
   เมื่อวันที่ 1 มิถุนายน 2559
   สงวนลิขสิทธิ์
*/
class Employee {
   private String name;    /* ชื่อ */
   private int age;        /* อายุ */
}
```

เครื่องหมายสิ้นสุดคำสั่ง

ในจาวาเราจะใช้เครื่องหมาย Semicolon (;) เป็นตัวแสดงจุดสิ้นสุดในแต่ละคำสั่ง หากเราไม่ใส่โปรแกรมจะถือว่าเป็นคำสั่งเดียวกันไปตลอด แม้จะอยู่คนละบรรทัดก็ตาม ทำให้เกิดข้อผิดพลาดได้ เช่น

```
int a = 10;
int b = a + 10;
String str = "Java";
```

หลังจากเติม ; สามารถนำคำสั่งอื่นมาต่อท้ายได้เลยแต่ไม่นิยมทำกัน เนื่องจากจะอ่านโค้ดยาก เช่น

```
int a = 10; int b = a + 10; String str = "Java";
```

Main Class

หากคลาสนั้นเป็นคลาสหลักของแอปพลิเคชั่นที่ต้องเรียกขึ้นมาทำงานเป็นลำดับแรก คลาสนั้นจะต้องประกอบไปด้วยเมธอด main() เสมอ หรือเรียกว่า Main Class ซึ่งลักษณะของเมธอด main() เป็นดังนี้

```
class Sample {
   public static void main(String[] args) {
      //คำสั่งต่างๆ
   }
}
```

คีย์เวิร์ดในภาษาจาวา

คีย์เวิร์ด (Keyword) คือคำที่ใช้เป็นคำสั่งสำหรับควบคุมการทำงานของโปรแกรม เราต้องไม่นำคำเหล่านี้ไปตั้งเป็นชื่อตัวแปรหรือชื่อสมาชิกใดๆ ของคลาส โดยคีย์เวิร์ดทั้งหมดของจาวามีดังนี้

abstract	default	if	outer	this
boolean	do	implements	package	throw
break	double	import	private	throws
byte	else	inner	protected	transient
byvalue	extends	instanceof	public	try
case	final	int	rest	var
cast	finally	interface	return	void
catch	float	long	short	volatile
char	for	native	static	while
class	future	new	super	
const	generic	null	switch	
continue	goto	operator	synchronized	

การแสดงผลด้วย print() และ println()

การศึกษาจาวาขั้นพื้นฐาน ส่วนใหญ่เราจะแสดงผลและติดต่อกับผู้ใช้ในแบบของข้อความทั้งหมด ดังนั้นสิ่งที่เรารู้จักในเบื้องต้นก็คือ การแสดงข้อความออกไปที่คอนโซล โดยในจาวามีคำสั่งที่ใช้ในการแสดงผลที่ควรรู้จักในเบื้องต้นดังนี้คือ

System.out.print(ข้อมูล)	ใช้ในการแสดงข้อมูลที่กำหนดออกไปที่คอนโซล
System.out.println(ข้อมูล)	ใช้ในการแสดงข้อมูลเช่นเดียวกัน แต่หลังจากการแสดงข้อมูล เคอร์เซอร์จะถูกเลื่อนไปอยู่บรรทัดถัดไป หรือขึ้นบรรทัดใหม่ให้โดยอัตโนมัตินั่นเอง

สำหรับข้อมูลที่จะกำหนดให้แก่ทั้งสองคำสั่งนี้ หากเป็นตัวเลข ก็เขียนลงไปโดยตรงได้เลย แต่หากเป็นข้อความ (สตริง) ให้เขียนไว้ในเครื่องหมาย “...” เช่น

```
System.out.print(12345);
System.out.println("Java & Android");
```

ข้อแตกต่างระหว่าง print() และ println() จะอยู่ที่ตำแหน่งเคอร์เซอร์หลังข้อมูลที่กำหนด หากใช้ print() เคอร์เซอร์จะอยู่ต่อท้ายข้อมูลที่แสดง ถ้าเราใช้คำสั่งต่อเนื่องกันหลายครั้ง ข้อมูลทั้งหมดจะถูกวางเรียงต่อเนื่องในบรรทัดเดียวกันไปเรื่อยๆ เช่น

```
System.out.print("Java ");
System.out.print("Programming ");
System.out.print("Using IntelliJ");
//ลักษณะผลลัพธ์คือ Java Programming Using IntelliJ
```

แต่ถ้าเปลี่ยนมาใช้ println() จะได้ผลดังนี้

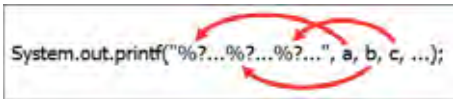
```
System.out.println("Java ");
System.out.println("Programming ");
System.out.println("Using IntelliJ");
//ลักษณะผลลัพธ์คือ
Java
Programming
Using IntelliJ
```

การแสดงผลด้วย printf()

คำสั่งนี้ใช้ในการจัดรูปแบบ (format) ข้อมูลที่จะแสดงผลโดยนำสัญลักษณ์ต่างๆ มาใช้ แต่ในที่นี้จะกล่าวถึงเพียงลักษณะพื้นฐาน เพื่อนำไปใช้ประโยชน์ในการแทรกข้อมูลลงในสตริงเท่านั้น โดยมีรูปแบบดังนี้

```
System.out.printf("สตริงที่จะแสดงผล", ข้อมูล1, ข้อมูล2, ...)
```

เราสามารถแทรกข้อมูลลงในสตริงกี่ครั้งก็ได้ แต่ต้องจัดเรียงลำดับให้ถูกต้อง ทั้งนี้ตำแหน่งที่จะแทรกข้อมูลลงไป ต้องแทนด้วยสัญลักษณ์ตามชนิดข้อมูล ซึ่งสัญลักษณ์ที่น่าสนใจมีดังนี้คือ



%s	(string) ใช้ในการแทรกข้อมูลที่เป็นข้อความ หรืออักขระ (char) หรือตัวเลขทั้งจำนวนเต็มและทศนิยม
%d	(decimal) ใช้ในการแทรกข้อมูลที่เป็นเลขจำนวนเต็มเท่านั้น เช่น 1234
%f	(float) ใช้ในการแทรกข้อมูลที่เป็นเลขทศนิยมเท่านั้น เช่น 12.34
%.?f	ใช้ในการแทรกเลขทศนิยม โดย ? นั้นให้ระบุจำนวนตำแหน่งทศนิยมที่ต้องการ เช่น %.2f หมายถึงให้แสดงเป็นเลขทศนิยม 2 ตำแหน่ง
%,d %,f	ใช้ในกรณีที่ต้องการให้มีเครื่องหมาย , คั่นหลักพันด้วย โดย %,d ใช้กับเลขจำนวนเต็ม และ %,f ใช้กับเลขทศนิยม
%,.?f	ใช้ในกรณีที่ต้องการให้มีเครื่องหมาย , คั่นหลักพันและมีจำนวนตำแหน่งทศนิยมตามที่ต้องการ เช่น %,.3f
%n หรือ \n	สำหรับการขึ้นบรรทัดใหม่ โดยจะตัดสตริงตั้งแต่สัญลักษณ์ %n หรือ \n เป็นต้นไปจนจบสตริง ไปขึ้นบรรทัดใหม่ ซึ่งเราไม่ต้องกำหนดข้อมูลใดๆ ให้กับสัญลักษณ์นี้ (ปกติเรานิยมใช้ \n มากกว่า %n)

การใช้ %d และ %f ต้องให้ตรงกับชนิดข้อมูลที่จะนำมาแทรกในสตริง มิฉะนั้นจะเกิดข้อผิดพลาดขึ้นได้ เช่น ถ้าจะแทรกข้อมูล 123.45 ต้องใช้ %f เท่านั้น ในกรณีที่เลขที่จะนำมาแทรกนั้นอาจจะเป็นไปได้ทั้งจำนวนเต็มและทศนิยม เราอาจใช้ %s แทน %d และ %f ก็ได้ สำหรับแนวทางการนำไปใช้ เช่น

```
System.out.printf("ขอเชิญหมายเลข %d ที่ช่องบริการ %d ค่ะ", 50, 2);
//ผลลัพธ์คือ ขอเชิญหมายเลข 50 ที่ช่องบริการ 2 ค่ะ
System.out.printf("ผมชื่อ %s  อายุ %d ปี หนัก %f กิโลกรัม", "สมชาย", 40, 70.5);
//ผลลัพธ์คือ ผมชื่อ สมชาย อายุ 40 ปี หนัก 70.5 กิโลกรัม
System.out.printf("Java กำเนิดขึ้นในปี %d ขณะนี้ตรงกับปี %d \n แสดงว่า Java มีอายุ %d ปี",
                  1995, 2016, (2016-1995));
//Java กำเนิดขึ้นในปี 1995 ขณะนี้ตรงกับปี 2016
//แสดงว่า Java มีอายุ 21 ปี
```

ชนิดข้อมูลพื้นฐานในจาวา

ในการจัดเก็บหรือนำข้อมูลไปใช้งาน เราต้องทราบชนิดของข้อมูลนั้นก่อน เพื่อจะได้นำไปใช้งานอย่างถูกต้อง เนื่องจากข้อมูลแต่ละชนิดจะใช้เนื้อที่ในการจัดเก็บไม่เท่ากัน รวมทั้งมีวิธีการประมวลผลที่แตกต่างกันไปด้วย ซึ่งชนิดของข้อมูลพื้นฐาน (Primitive Data Type) ในจาวา แบ่งเป็นประเภทย่อยได้ดังนี้คือ

- **ข้อมูลประเภทเลขจำนวนเต็ม**

ชนิดข้อมูล	ช่วงข้อมูล	อักขระต่อท้าย
byte	8-bit signed integral type: -128 ถึง 127	
short	16-bit signed integral type: -32768 ถึง 32767	
int	32-bit signed integral type: -2,147,483,648 ถึง 2,147,483,647	
long	64-bit signed integral type: -9,223,372,036,854,775,808 ถึง 9,223,372,036,854,775,807	L หรือ l

- **ข้อมูลประเภทเลขทศนิยม**

ชนิดข้อมูล	ช่วงข้อมูล	อักขระต่อท้าย
float	single-precision floating point type: 1.4E-45 ถึง 3.4028235E38	F หรือ f
double	double-precision floating point type: 4.9E-324 ถึง 1.7976931348623157E308	D หรือ d

- **ข้อมูลประเภทจริงเท็จ**

ชนิดข้อมูล	ช่วงข้อมูล	อักขระต่อท้าย
boolean	ค่าที่เป็นไปได้ มี 2 อย่างคือ true หรือ false	

- **ข้อมูลประเภทอักขระ**

ชนิดข้อมูล	ช่วงข้อมูล	อักขระต่อท้าย
char	เก็บข้อมูลที่เป็นอักขระเพียง 1 ตัว	
String	เก็บข้อมูลที่เป็นข้อความที่อาจมีอักขระได้มากกว่า 1 ตัว	

หมายเหตุ

เนื่องจากข้อมูลบางชนิดนั้นมีค่าอยู่ในช่วงที่ซ้อนทับกัน เช่น ตัวเลข 1000000 อาจจัดให้เป็นชนิด int หรือ long ก็ได้ แต่เนื่องจากวิธีการประมวลผลข้อมูลแต่ละชนิดจะแตกต่างกัน ดังนั้นในบางกรณีเราจำเป็นต้องระบุให้แน่ชัดว่าเป็นข้อมูลชนิดใด โดยเติมอักขระต่อท้ายตัวเลข เช่น 1000000L แสดงว่าเป็นชนิด long เป็นต้น และในจาวาไม่ถือว่า String เป็นชนิดข้อมูลพื้นฐานแบบ Primitive Data Type เหมือนในบางภาษา เพราะว่า String เป็นคลาส แต่โดยทั่วไปเรามักจะใช้งานเหมือนกับข้อมูลพื้นฐานทั่วๆ ไป ดังนั้นในที่นี้จึงนำมากล่าวรวมเอาไว้ด้วย แต่ขอให้รู้ไว้ว่า String นั้นไม่จัดเป็นชนิดข้อมูลพื้นฐาน

การกำหนดตัวแปร

ตัวแปร (Variable) หมายถึงสัญลักษณ์ที่เราใช้ในการอ้างอิงถึงข้อมูลแต่ละอย่าง โดยข้อมูลที่จะนำมากำหนดให้แก่ตัวแปรต้องตรงกับชนิดข้อมูลที่เราได้ระบุไว้กับตัวแปรนั้น รวมถึงการจะนำข้อมูลไปใช้งานก็ต้องกระทำผ่านตัวแปรนี้ ซึ่งรายละเอียดที่เราควรรู้ในเบื้องต้นเกี่ยวกับตัวแปรมีดังนี้

หลักเกณฑ์ที่สำคัญในการตั้งชื่อตัวแปร

การตั้งชื่อตัวแปรในจาวา มีหลักเกณฑ์ที่สำคัญดังต่อไปนี้คือ

- ต้องไม่ซ้ำกับคีย์เวิร์ดของภาษาจาวา
- ต้องขึ้นต้นด้วยตัวอักษร หรือเครื่องหมาย $ หรือเครื่องหมาย _ เช่น myVar, $num, value_
- สามารถใช้ตัวเลขร่วมได้ แต่ห้ามใช้ตัวเลขเป็นตัวขึ้นต้น เช่น num1, year2000
- การเขียนด้วยตัวพิมพ์เล็กใหญ่ต่างกัน เช่น abc, ABC, Abc ถือว่าไม่เหมือนกัน
- ในจาวานิยมตั้งชื่อตัวแปรให้ขึ้นต้นด้วยตัวพิมพ์เล็ก แล้วถ้าเป็นการนำหลายๆ คำมารวมกันเป็นชื่อตัวแปรเดียวกัน ให้เขียนติดกัน แล้วให้ตัวอักษรขึ้นต้นของแต่ละคำเป็นตัวพิมพ์ใหญ่ ยกเว้นคำแรก เช่น numProducts, isValidInput, numDayOfMonth แต่หลักเกณฑ์นี้ไม่ใช่ข้อบังคับ เราสามารถเขียนในรูปแบบตัวพิมพ์ที่เราต้องการก็ได้

ตัวอย่างการตั้งชื่อตัวแปรที่**ถูกต้อง** เช่น	ตัวอย่างการตั้งชื่อตัวแปรที่**ไม่ถูกต้อง** เช่น
x, firstname, num1, pricePerUnit, _isValid, freeDiskSpace, totalMemoryAvailable	price/unit (เพราะมีอักขระ/) 1stName (เพราะเริ่มต้นด้วยตัวเลข) int (เพราะซ้ำกับคีย์เวิร์ด)

การประกาศตัวแปร

สำหรับการประกาศตัวแปรในจาวา จะระบุชนิดหรือประเภทของข้อมูลของตัวแปรนั้นก่อน แล้วตามด้วยชื่อตัวแปร ในรูปแบบดังนี้

ชนิดข้อมูล ชื่อตัวแปร;

```
int x;
double result;
String str;
```

หากข้อมูลเป็นชนิดเดียวกันเราสามารถประกาศรวมในชนิดข้อมูลเดียวกันได้ เช่น

```
int x, y, z;        // เมื่อ x, y, z เป็นตัวแปรชนิด int เหมือนกัน
String a, b, c;     // เมื่อ a, b, c เป็นตัวแปรชนิด String เหมือนกัน
```

ตัวแปรที่กำหนดชนิดข้อมูลอย่างใดอย่างหนึ่งให้กับมันแล้ว จะเปลี่ยนชนิดข้อมูลเป็นอย่างอื่นอีกไม่ได้ หรือกล่าวได้ว่า เราจะประกาศตัวแปรชื่อซ้ำกันไม่ได้ แม้จะกำหนดข้อมูลคนละชนิดก็ตาม เช่น

```
int x;
double x;          //Error!
```

การกำหนดค่าให้กับตัวแปร

การกำหนดค่าให้แก่ตัวแปรสำหรับข้อมูลแต่ละชนิดจะมีรูปแบบที่แตกต่างกัน ซึ่งสามารถจำแนกตามชนิดข้อมูลได้ดังนี้

ข้อมูลชนิดตัวเลข

- ในการกำหนดค่าที่เป็นข้อมูลชนิดตัวเลขทั้งหมด สามารถใส่ตัวเลขลงไปได้เลย เพราะปกติตัวเลขทั้งหมดนั้นจะเขียนติดกันอยู่แล้ว แต่ต้องไม่มีเครื่องหมาย “,” คั่นหลักพันอยู่ด้วย เช่น

```
byte x;
x = 123;                                    //อาจประกาศตัวแปรไว้ก่อน แล้วค่อยมากำหนดค่าทีหลัง
int y = 456;                                //หรืออาจประกาศพร้อมกำหนดค่าให้แก่ตัวแปร
double z = 456.789;
int a = 108, b = 1009;                      //ถ้าเป็นข้อมูลชนิดเดียวกันอาจประกาศในคำสั่งเดียวกันได้
double m = 1.23, n = 4.56, p, q, r, s;      //ข้อมูลชนิดเดียวกันแต่กำหนดค่าให้กับตัวแปรเพียงบางตัว
```

- ต้องกำหนดค่าให้อยู่ในช่วง MIN_VALUE - MAX_VALUE ของข้อมูลชนิดนั้น (ดูจากหัวข้อที่แล้ว) เช่น ข้อมูลชนิด byte ต้องกำหนดค่าระหว่าง -128 ถึง 127 มิฉะนั้นจะเกิดข้อผิดพลาดขึ้นได้
- หากนำตัวเลขที่มีทศนิยมไปกำหนดให้แก่ตัวแปรประเภทจำนวนเต็ม **จะเกิดข้อผิดพลาด** เช่น

```
int x = 123.45;       //Error
short y = 10E8;       //Error
```

- แม้ว่าเราจะสามารถกำหนดค่าที่เป็นจำนวนเต็มให้แก่ตัวแปรประเภททศนิยมได้ แต่ค่านั้นจะถูกแปลงเป็นเลขทศนิยม เช่น

```
double d = 123;
System.out.println(d);   //123.0
```

- กรณีตัวแปรชนิด long และ float เราอาจจะใส่อักขระ L และ F ต่อท้าย เพื่อเป็นการแยกความแตกต่างกับชนิด int และ double ตามลำดับ เช่น

```
long a = 1234567L;
float b = 3.14159F;
```

ข้อมูลชนิดบูลีน

ข้อมูลชนิดบูลีนจะเป็นได้เพียงค่า true (จริง) หรือ false (เท็จ) อย่างใดอย่างหนึ่ง ซึ่ง true และ false นี้ก็เป็นคีย์เวิร์ดของจาวาอยู่แล้ว ดังนั้นจึงสามารถนำไปกำหนดให้แก่ตัวแปรได้เลย เช่น

```
boolean a = true;
boolean b = false;
```

ข้อมูลชนิด char

ข้อมูลชนิดนี้จะกำหนดเป็นอักขระใดๆ เพียง 1 ตัว ซึ่งเราต้องเขียนอักขระนั้นไว้ในเครื่องหมาย ‘ ’

```
char grade = 'A';
char size = 'M';
char v1 = 'a', v2 = 'e', v3 = 'i', v4 = 'o', v5 = 'u';
```

ข้อมูลชนิดสตริง

สตริง เป็นการนำอักขระแต่ละตัวมาวางเรียงต่อกัน ซึ่งเราต้องกำหนดจุดเริ่มต้นและสิ้นสุดของมันด้วยเครื่องหมาย “ ” เช่น

```
String str = "Java for Android";
```

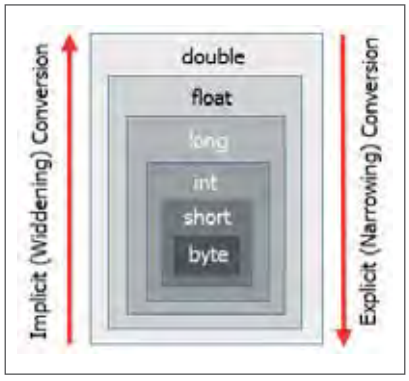
เครื่องหมาย “...” ในการกำหนดสตริงนั้น ต้องอยู่ในบรรทัดเดียวกัน จะแยกสตริงที่กำหนดไว้ในเครื่องหมาย “...” อันเดียวกันไปไว้คนละบรรทัดไม่ได้ เช่น กรณีต่อไปนี้**จะเกิดข้อผิดพลาด**

```
String slogan = "Write Once
                 Run Anywhere";    //Error!
```

ถ้าเราไม่สะดวกที่จะเขียนสตริงที่ยาวๆ ไว้ในเครื่องหมาย "..." เดียวกันทั้งหมด ให้แยกเป็นสตริงย่อยๆ แล้วนำมาต่อกันด้วยเครื่องหมาย + เช่น

```
String slogan = "Write Once " +
                "Run Anywhere";
```

การแปลงชนิดข้อมูล



เนื่องจากข้อมูลแต่ละชนิดนั้นจะมีช่วงในการจัดเก็บข้อมูลที่ต่างกัน ซึ่งสามารถเรียงลำดับตามขนาดของชนิดข้อมูลจากมากไปน้อยคือ **double > float > long > int > short > byte**

แต่ในบางกรณีเราอาจจำเป็นต้องเปลี่ยนชนิดข้อมูลจากชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง หรือการทำ Type Conversion ซึ่งแบ่งเป็น 2 ลักษณะคือ Implicit Conversion และ Explicit Conversion ดังรายละเอียดต่อไปนี้

Implicit Conversion (Widening)

Implicit Conversion หรือ Widening Conversion **เป็นการแปลงจากชนิดข้อมูลที่มีขนาดเล็กกว่า ไปเป็นชนิดที่มีขนาดใหญ่กว่า** เช่น แปลงจาก short -> int หรือจาก int -> long เป็นต้น ดังนั้นการแปลงจึงสามารถทำได้โดยอัตโนมัติ ไม่ต้องอาศัยการกระทำใดๆ เข้ามาช่วย เช่น

```
byte b = 100;
int i = b;
short x = 1000;
long y = x;
float f = 123.45F;
double d = f;
```

Explicit Conversion (Narrowing)

Explicit Conversion หรือ Narrowing Conversion **เป็นการแปลงจากชนิดข้อมูลที่มีขนาดใหญ่กว่า ไปเป็นชนิดที่มีขนาดเล็กกว่า** เช่น แปลงจาก long -> int หรือจาก double -> int เป็นต้น เนื่องจากการแปลงแบบนี้ผิดจากหลักการทั่วไป จึงต้องอาศัย**การคาสต์ (Cast)** เข้ามาช่วย ดังนี้

```
ชนิดข้อมูล ตัวแปร = (ชนิดข้อมูล)ค่าที่ต้องแปลง
```

เช่น ถ้าเราจะคาสต์จากชนิด long -> int ก็อาจกำหนดโค้ดดังนี้

```
long a = 9999L;
int b = (int)a;   //b = 9999
```

ทั้งนี้หากแปลงจากเลขทศนิยมไปเป็นจำนวนเต็ม ทศนิยมจะถูกตัดทิ้งเหลือแค่จำนวนเต็ม เช่น

```
double d = 123.45;
int i = (int)d;   //i = 123
```

การกำหนดค่าคงที่

ค่าคงที่ (Constant) เป็นการกำหนดค่าให้กับตัวแปรด้วยค่าใดค่าหนึ่ง และต้องใช้ค่านั้นไปตลอด ไม่สามารถเปลี่ยนค่าเป็นอย่างอื่นๆ ได้ โดยวางคีย์เวิร์ด final ไว้หน้าชนิดข้อมูล ดังรูปแบบต่อไปนี้

```
final ชนิดข้อมูล ชื่อค่าคงที่  =  ค่าที่กำหนด;
```

ในภาษาจาวา เพื่อให้แยกความแตกต่างระหว่างค่าคงที่และตัวแปรได้ง่ายขึ้น เรานิยมเขียนชื่อค่าคงที่ให้เป็นตัวพิมพ์ใหญ่ทั้งหมด ถ้านำหลายคำมาใช้ร่วมกัน ให้เชื่อมแต่ละคำด้วยเครื่องหมาย “_” เช่น

```
final float PI = 3.141F;
final short PRICE_PER_UNIT = 1000;
final double LATITUDE, LONGITUDE;
LATITUDE = 100.12345;
LONGITUDE = 80.67890;
```

อักขระ Escape

อักขระ Escape เป็นกลุ่มอักขระพิเศษในจาวา ซึ่งถูกกำหนดขึ้นมาเพื่อใช้งานในกรณีเฉพาะ โดยอักขระกลุ่มนี้จะต้องวางเครื่องหมาย Backslash(\) ไว้ข้างหน้าเสมอ ซึ่งตัวที่น่าสนใจ มีดังนี้คือ

\n	สำหรับการขึ้นบรรทัดใหม่	\”	สำหรับแทรกเครื่องหมาย “
\t	สำหรับการเว้นระยะ 1 แท็บ	\’	กำหนดค่าให้ข้อมูลชนิด char เป็นเครื่องหมาย ‘ เช่น char c = ‘\’’;
\\	เมื่อเขียนเครื่องหมาย \ ไว้ในสตริง		

แนวทางการนำอักขระ Escape ไปใช้ เช่น

```
System.out.print("Press \"Enter\" to continue");
//แสดงข้อความ => Press "Enter" to continue
System.out.print("บรรทัดที่ 1\nบรรทัดที่ 2\nบรรทัดที่ 3" + "\n" + "บรรทัดที่ 4" + "\nบรรทัดที่ 5");
//แสดงข้อความ => บรรทัดที่ 1...บรรทัดที่ 5 อยู่คนละบรรทัด
System.out.print("วันที่\tยอดขายสินค้าชนิดที่ 1\tยอดขายสินค้าชนิดที่ 2\tยอดขายสินค้าชนิดที่ 3");
//เว้นระยะ 1 แท็บ => วันที่   ยอดขายสินค้าชนิดที่ 1     ยอดขายสินค้าชนิดที่ 2     ยอดขายสินค้าชนิดที่ 3
String path = "C:\\temp\\images\\bird.png";
```

การรับข้อมูลทางคีย์บอร์ด

การรับข้อมูลทางคีย์บอร์ด จะทำให้ผู้ใช้สามารถกำหนดค่าของข้อมูลได้ตามต้องการ โดยวิธีที่นิยมใช้กันมากที่สุดในปัจจุบัน คือการใช้คลาส Scanner ซึ่งเริ่มใช้ตั้งแต่ Java 5 เป็นต้นมา ดังแนวทางต่อไปนี้

- เขียนคำสั่ง **import java.util.Scanner** ไว้ส่วนบนสุด (ก่อนบรรทัดกำหนดชื่อคลาส)
- สร้างอินสแตนซ์ของคลาสดังนี้

```
Scanner scan = new Scanner(System.in);
```

- เลือกใช้คำสั่งอันใดอันหนึ่งต่อไปนี้เพื่ออ่านตามชนิดข้อมูลที่ต้องการ คือ
  - อ่านข้อมูลที่เป็นสตริง ให้ใช้คำสั่ง next()
  - อ่านข้อมูลที่เป็นเลขทศนิยม ให้ใช้ nextDouble() หรือ nextFloat()
  - อ่านข้อมูลที่เป็นเลขจำนวนเต็ม ให้ใช้ nextByte(), nextShort(), nextInt(), nextLong()

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("กรุณาใส่ข้อมูล >> ");
        int in = scan.nextInt();
    }
}
```

การสร้างเลขสุ่ม

เลขสุ่ม ก็คือสร้างขึ้นมาโดยวิธีการสุ่ม (Random) ซึ่งจะได้ค่าที่ไม่แน่นอน เนื่องจากเลขสุ่มนั้นเป็นสิ่งสำคัญที่เราต้องนำไปใช้ประกอบการศึกษาเนื้อหาในหลายๆ บท ดังนั้นจึงควรรู้จักวิธีการสร้างแบบง่ายๆ เอาไว้ก่อน เพื่อเป็นพื้นฐานสำหรับนำไปใช้งานในบทต่อๆ ไป ดังแนวทางต่อไปนี้

- การสร้างเลขสุ่มในจาวา ทำได้ 2 วิธีคือ ใช้คำสั่ง Math.random() และอีกวิธีคือใช้คลาส Random แต่ในบทนี้จะแนะนำแค่การใช้ Math.random() ก่อน ส่วนคลาส Random จะกล่าวถึงในบทที่ 5
- เมื่อจะใช้เลขสุ่ม ให้สร้างตัวแปรชนิด double (ชนิด float ไม่ได้) เพื่อมารับค่าจาก Math.random() ดังโค้ดถัดไป โดย**เลขสุ่มที่ได้จะมีค่าระหว่าง 0-1** เช่น 0.123456

```
double rnd = Math.random();
```

- หากเราต้องการเลขจำนวนเต็มอาจคูณด้วย 10 หรือ 100 หรืออื่นๆ ขึ้นอยู่กับว่าเราต้องการเลขกี่หลัก แล้วใช้การคาสต์ให้เป็นจำนวนเต็ม เพราะชนิด double มีขนาดใหญ่กว่าจำนวนเต็ม เช่น

```
double rnd = Math.random();
byte a = (byte)(rnd * 10);     //เมื่อต้องการเลขหลักเดียว จะได้เลขจำนวนเต็มระว่าง 0-9
short b = (short)(rnd * 100); //จะได้เลขจำนวนเต็มระว่าง 0-99 (ถ้าสุ่มได้ 0.0xxx จะได้เลขหลักเดียว)
int c = (int)(rnd * 10000);
```

2

Operator & Control Statement

การที่จะเขียนโปรแกรมให้ทำงานได้ตามที่เราต้องการนั้น จำเป็นต้องใช้คำสั่งเพื่อควบคุมให้เกิดการประมวลผลไปตามลำดับ เช่น การกำหนดเงื่อนไขและการใช้ลูปแบบต่างๆ เป็นต้น นอกจากนี้ยังต้องรู้จักการใช้เครื่องหมายหรือโอเปอเรเตอร์ในแต่ละประเภท สำหรับดำเนินการกับข้อมูลหรือตัวแปร เพื่อที่จะนำผลลัพธ์ที่ได้ ไปใช้งานอื่นๆ ต่อไป

โอเปอเรเตอร์

โอเปอเรเตอร์ (Operator) คือสัญลักษณ์ที่ใช้ในการประมวลผลทางคณิตศาสตร์หรือทางตรรกะ สำหรับในจาวา เราสามารถแบ่งโอเปอเรเตอร์ได้หลายกลุ่ม ดังนี้

โอเปอเรเตอร์สำหรับการคำนวณ

+	บวก	%	โอเปอเรเตอร์นี้เรียกว่า Modulus เป็นการหารแบบเอาเฉพาะเศษ เช่น 8 % 3 = 2

-	ลบ	++	เป็นการเพิ่มค่าตัวแปรขึ้นไปอีก 1 เช่น
			x = 10;
			x++; //x = 11

*	คูณ	--	เป็นการลดค่าตัวแปรลงอีก 1 เช่น
			x = 10;
			x--; //x = 9

/	หาร		

ตัวอย่าง Example 2-1 ธนบัตรที่ตู้ ATM จ่ายให้เราได้มี 3 ชนิดคือ 100, 500 และ 1000 เมื่อเรากดเงินตามจำนวนที่ต้องการ ให้คำนวณหาจำนวนธนบัตรแต่ละชนิดที่เครื่องจะจ่ายให้เรา โดยให้เริ่มจากชนิดที่มีค่ามากสุดที่เป็นไปได้เสมอ ซึ่งมีแนวทางดังนี้

1	ให้เริ่มจากชนิด 1000 บาทก่อน โดยนำจำนวนที่จะถอนไปหารด้วย 1000 แล้วเอาเฉพาะจำนวนเต็ม ซึ่งจะได้ค่าเป็นจำนวนธนบัตรชนิด 1000	**int** w = 2800; **int** b1000 = (**int**) (w / 1000); //2
2	หาเศษที่เหลือจากการหารด้วย 1000 เพื่อนำไปหาธนบัตรชนิด 500	**int** r = w % 1000; //800
3	นำเศษที่ได้ในข้อ 2 ไปหารด้วย 500 แล้วเอาเฉพาะจำนวนเต็มเพื่อคำนวณหาจำนวนธนบัตรชนิด 500	**int** b500 = (**int**)(r / 500); //1
4	คำนวณหาเศษจากการหารด้วย 500 เพื่อนำไปหาธนบัตรชนิด 100	r = r % 500; //300
5	นำเศษที่ได้ในข้อ 4 ไปหารด้วย 100 แล้วเอาเฉพาะจำนวนเต็มเพื่อคำนวณหาจำนวนธนบัตรชนิด 100	**int** b100 = (**int**)(r / 100); //3
6	ดังนั้นธนบัตรที่จะได้คือ b1000 = 2, b500 = 1, b100 = 3	

ตัวอย่างนี้อาจจะยากนิดหน่อย แต่ขอให้ลองทำดูก่อน โดยให้สร้างโปรเจ็กต์และคลาสดังนี้

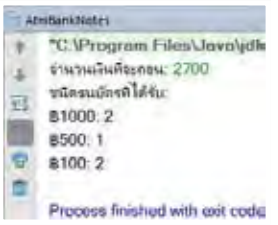
```
import java.util.Scanner;

public class AtmBankNotes {
    public static  void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("จำนวนเงินที่จะถอน: ");
        int withdraw = scan.nextInt();
        int b1000 = (int)(withdraw / 1000);

        int remain = withdraw % 1000;
        int b500 = (int)( remain / 500);

        remain = remain % 500;
        int b100 = (int)(remain / 100);

        System.out.println("ชนิดธนบัตรที่ได้รับ:");
        System.out.println("฿1000: " + b1000);
        System.out.println("฿500: " + b500);
        System.out.println("฿100: " + b100);
        scan.close();
    }
}
```



โอเปอเรเตอร์สำหรับการกำหนดค่า

โอเปอเรเตอร์สำหรับการกำหนดค่า (Assignment) เป็นเครื่องหมายสำหรับการกำหนดค่าของตัวแปรทางด้านซ้ายของเครื่องหมาย ด้วยค่าที่อยู่ทางด้านขวาของเครื่องหมาย ซึ่งโอเปอเรเตอร์ในกลุ่มนี้มีดังนี้

=	เท่ากับ เช่น x = 10;
+=	บวกค่าเดิมด้วย เช่น x = 5; x += 10; ซึ่งมีค่าเท่ากับ x = x + 10 = 5 + 10 ผลลัพธ์ที่ได้คือ x = 15 หากใช้กับข้อมูลชนิดสตริงก็จะเป็นการนำสตริงที่ระบุไปต่อท้ายสตริงเดิม เช่น a = “Write Once ”; b = “Run Anywhere”; a += b; ผลลัพธ์ที่ได้คือ a = “Write Once Run Anywhere”;
-=	ลบค่าเดิมด้วย เช่น x = 20; x -= 15; ซึ่งมีค่าเท่ากับ x = x - 10 = 20 - 15 ผลลัพธ์ที่ได้คือ x = 5
*=	คูณค่าเดิมด้วย เช่น x = 5; x *= 2; มีค่าเท่ากับ x = x * 2 ผลลัพธ์ที่ได้คือ x = 10
/=	หารค่าเดิมด้วย เช่น x /= 10 ซึ่งมีค่าเท่ากับ x = x /10;
%=	หารแบบ modulus ค่าเดิม ด้วย เช่น x % = 10 ซึ่งมีค่าเท่ากับ x = x % 10;

การใช้ ++ และ -- แบบ Prefix กับ Postfix

การวางเครื่องหมาย ++ หรือ -- ไว้ด้านหน้า (Prefix) หรือหลังตัวแปร (Postfix) หากตัวแปรนั้นอยู่เดี่ยวๆ ค่าที่ได้จะไม่ต่างกัน เช่น

```
int x = 10;    //ลักษณะต่อไปนี้เทียบกับค่าเดิมคือ 10
x++;           //x = 11
++x;           //x = 12
x--;           //x = 11
--x;           //x = 10
```

แต่หากนำไปกระทำกับค่าอื่นๆ ด้วย ผลที่ได้อาจแตกต่างกันไป เช่น ลองพิจารณา 2 กรณีต่อไปนี้

`x = 10;`
`y = 20;`
`y += ++x;  //x = 11, y = 31`
	`x = 10;`
	`y = 20;`
	`y += x++;  //x = 11, y = 30`

y มีค่าเท่ากับ (20 + 1) + 10 = 31 โดยจะเพิ่มค่า x ไปอีก 1 ก่อน แล้วค่อยนำไปบวกกับ y	y มีค่าเท่ากับ 20 + 10 = 30 โดยจะบวกค่า x เดิมกับ y ก่อน แล้วค่อยเพิ่มค่า x ขึ้นไปอีก 1

หลักการที่กล่าวมานี้ พิจารณาจากลำดับความสำคัญของโอเปอเรเตอร์ ที่จะกล่าวถึงในลำดับต่อไป

โอเปอเรเตอร์สำหรับการเปรียบเทียบ

โอเปอเรเตอร์สำหรับการเปรียบเทียบ (Comparison) เป็นเครื่องหมายสำหรับการเปรียบเทียบเพื่อหาค่าความจริงระหว่างข้อมูล 2 อย่าง ซึ่งผลที่ได้จะเป็นชนิด boolean ทั้งนี้หากเปรียบเทียบแล้วเป็นจริงจะได้ผลลัพธ์เป็น true แต่หากเป็นเท็จจะได้ค่า false โดยมีโอเปอเรเตอร์ดังต่อไปนี้

==	ใช้ในการเปรียบเทียบว่าเท่ากันหรือไม่	<	น้อยกว่า
!=	ไม่เท่ากับ	>=	มากกว่าหรือเท่ากับ
>	มากกว่า	=<	น้อยกว่าหรือเท่ากับ

ลักษณะการใช้งาน เช่น

```
boolean a = 10 > 5;       //a = true เพราะ 10 มากกว่า 5 เป็นจริง
boolean b = (9 >= 10);    //c = false เพราะ 9 ไม่ได้มากกว่า 10
```

การเปรียบเทียบสตริง จะใช้โอเปอเรเตอร์เหล่านี้ไม่ได้ เพราะในจาวาสตริงจะเป็นออบเจ็กต์ ถ้าจะเปรียบเทียบสตริง ในเบื้องต้นนี้ผู้เขียนแนะนำให้ใช้เมธอด eqauls() มาต่อท้ายตัวแปรชนิดสตริงเพื่อเปรียบเทียบไปก่อน ส่วนรายละเอียดเพิ่มเติมจะกล่าวถึงในบทที่ 5 เช่น

```
String login = "...";
boolean b = login.equals("admin");  //ตัวแปร login เป็นคำว่า admin หรือไม่ ถ้าใช่จะคืนค่า true
```

โอเปอเรเตอร์ทางตรรกะ

โอเปอเรเตอร์ทางตรรกะ (Logic) เป็นการเปรียบระหว่าง 2 เงื่อนไขซึ่งแต่ละเงื่อนไขที่นำมาเปรียบเทียบจะต้องมีค่าเป็น true หรือ false อย่างใดอย่างหนึ่ง และผลลัพธ์สุดท้ายที่ได้ก็จะเป็น true หรือ false อย่างใดอย่างหนึ่งเช่นกัน ซึ่งโอเปอเรเตอร์ในกลุ่มนี้มีดังนี้

&&	ถ้าทั้งสองเงื่อนไขเป็นจริงทั้งคู่ ผลที่ได้จะเป็นจริง นอกนั้นเป็นเท็จหมด
||	ถ้าทั้งสองเงื่อนไขเป็นเท็จทั้งคู่ ผลที่ได้จะเป็นเท็จ นอกนั้นเป็นจริงหมด
^	ถ้าเงื่อนไขแรกกับเงื่อนไขหลังเหมือนกัน (จริง-จริง/เท็จ-เท็จ) ผลที่ได้จะเป็นเท็จ นอกนั้นเป็นจริง
!	จะเป็นค่าตรงข้ามกับเงื่อนไขที่กำหนด

การเปรียบเทียบทางตรรกะนี้สำคัญมากในการเขียนโปรแกรม ซึ่งเราต้องได้ใช้งานกันไปตลอด ดังนั้นขอให้ทำความเข้าใจหลักการตรรกะการเปรียบเทียบนี้ให้ดี เช่นกรณีต่างๆ ต่อไปนี้

```
boolean b;
b = (1 < 2) && (3 < 4);    //b = true เพราะเป็นจริงทั้ง 2 เงื่อนไข (true && true = true)
b = (1 < 2) && (3 == 4);   //b = false เพราะเป็นจริงเพียงเงื่อนไขเดียว (true && false = false)
b = (1 > 2) && (3 > 4);    //b = false เพราะเป็นเท็จทั้ง 2 เงื่อนไข (false && false = false)
```

```
b = (1 > 2) || (3 > 4);    //b = false เพราะเป็นเท็จทั้งสองเงื่อนไข (false || false = false)
b = (1 < 2) || (3 == 4);   //b = true เพราะมี 1 เงื่อนไขที่เป็นจริง (true || false = true)
b = (1 < 2) || (3 < 4);    // b = true เพราะเป็นจริงทั้ง 2 เงื่อนไข (true || true = true)
```

```
b = !(1 < 2);              //b = false เนื่องจาก !(true) = false
b = !(1 > 2);              //b = true เนื่องจาก !(false) = true
b = !((1 < 2) || (3 < 4)); //b = false เนื่องจาก !(true || false) = !(true) = false
b = !(1 < 2) && !(3 == 4); //b = false เนื่องจาก !(true) && !(false) = false && true = false
```

```
boolean x, y;
x = ...
y = !x;                    //ถ้า x เป็น true ค่า y จะเป็น false แต่ถ้า x เป็น false ค่า y จะเป็น true
```

โอเปอเรเตอร์แบบ Ternary

โอเปอเรเตอร์แบบ Ternary เป็นการเปรียบเทียบโดยใช้หลักการที่ว่า ถ้าเงื่อนไขตรงกับที่กำหนดผลลัพธ์จะเป็นอย่างไร แต่ถ้าเงื่อนไขไม่ตรงจะได้ผลลัพธ์เป็นอะไร รูปแบบการใช้งานคือ

```
(เงื่อนไข)? a : b
```

หมายความว่า หากเงื่อนไขตรงตามที่กำหนด ผลลัพธ์จะเป็น a แต่หากไม่ใช่ผลลัพธ์จะเป็น b เช่น

```
boolean x = (1 > 0)? true : false;   //x จะมีค่าเป็น true เนื่องจาก 1 > 0 นั้นเป็นจริง
String s = (x % 2 == 0)? "Even": "Odd";
//หาก x หารด้วย 2 ลงตัว (เศษเป็น 0) ค่า s จะเท่ากับ "Even" แต่หากหารไม่ลงตัว s จะเท่ากับ "Odd"
```

```
//ถ้าซื้อครบทุก 50 บาท จะได้แสตมป์ 1 ดวง แต่มีข้อกำหนดว่าจำนวนแสตมป์สูงสุดที่ได้ต้องไม่เกิน 10 ดวง
//ดังนั้นถ้าได้มากกว่า 10 ก็ให้แค่ 10 ดวง แต่ถ้าได้น้อยกว่า 10 ก็ให้ตามจำนวนที่ได้
int numStamps = (int)(total/50);
numStamps = (numStamps > 10)? 10 : numStamps;
```

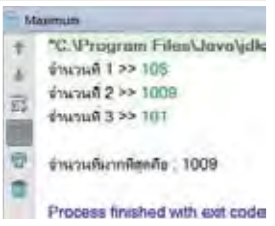
ตัวอย่าง Example 2-2 เป็นการหาจำนวนที่มีค่ามากที่สุดจากจำนวนทั้งหมดที่รับเข้ามา ดังแนวทางต่อไปนี้

1. รับจำนวนแรกเข้ามา แล้วตั้งค่าสูงสุดให้เท่ากับจำนวนแรกไว้ก่อน
2. รับจำนวนที่สองเข้ามา แล้วใช้ Ternary Operator เปรียบเทียบว่า จำนวนที่สองมีค่ามากกว่าค่าสูงสุดเดิมหรือไม่ ถ้ามากกว่าให้ตั้งค่าจำนวนนั้นเป็นค่าสูงสุดแทน
3. รับจำนวนที่สามเข้ามา แล้วใช้ Ternary Operator เปรียบเทียบทีละจำนวนเช่นเดิม ถ้าจำนวนใดมีค่ามากกว่าค่าสูงสุดเดิม ก็ให้ตั้งจำนวนนั้นเป็นค่าสูงสุดแทน จนเมื่อเปรียบเทียบครบทุกจำนวนก็จะได้ค่าสูงสุดของจำนวนทั้งหมด (เรารับจำนวนที่ สี่, ห้า, ... เข้ามาอีกได้)

```
import  java.util.Scanner;

public class Maximum {
    public static  void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int max;
        System.out.print("จำนวนที่ 1 >> ");
        int n1 = scan.nextInt();
        max = n1;                       //ให้ค่าสูงสุดเท่ากับจำนวนแรกไว้ก่อน
        System.out.print("จำนวนที่ 2 >> ");
        int n2 = scan.nextInt();
        max = (n2 > max)? n2 : max; //ถ้า n2 มากกว่า max เดิม ให้คืนค่า n2 มิฉะนั้นคืนค่า max เดิม
        System.out.print("จำนวนที่ 3 >> ");
        int n3 = scan.nextInt();
        max = (n3 > max)? n3 : max;

        System.out.println("\nจำนวนที่มากที่สุดคือ : " + max);
        scan.close();
    }
}
```



โอเปอเรเตอร์ในการเลื่อนบิต

โอเปอเรเตอร์ในการเลื่อนบิต เป็นการจัดการข้อมูลในระดับบิต ซึ่งเมื่อเกิดการเลื่อนบิต จะทำให้ค่าของข้อมูลนั้นเปลี่ยนแปลงไปจากเดิม โดยโอเปอเรเตอร์ที่น่าสนใจในกลุ่มนี้คือ

x << n	เป็นการเลื่อนบิตของ x ไปทางซ้ายจำนวน n ครั้ง ซึ่งจะทำให้ค่าของ x กลายเป็น $x * 2^n$
x >> n	เป็นการเลื่อนบิตของ x ไปทางขวาจำนวน n ครั้ง ซึ่งจะทำให้ค่าของ x กลายเป็น $x / 2^n$

ลักษณะการนำไปใช้ เช่น

```
int a, x = 10;
a = x << 4;
System.out.print(x);    //a = 10*2^4 = 10*16 = 160

int b, y = 20;
b = y >> 2;
System.out.println(y);  //b = 20/2^2 = 20/4 = 5
```

ลำดับความสำคัญของโอเปอเรเตอร์

ในการคำนวณอาจจะต้องใช้โอเปอเรเตอร์ร่วมกันมากกว่า 1 อย่าง จึงอาจเกิดปัญหาว่าเราจะนำโอเปอเรเตอร์ตัวใดมาพิจารณาก่อน เพราะการวางตำแหน่งหรือจัดกลุ่มที่ต่างกัน ผลลัพธ์ที่ได้ก็อาจแตกต่างกันไปด้วย สำหรับโอเปอเรเตอร์ทั้งหมดที่เราได้ศึกษามา สามารถจัดลำดับความสำคัญได้ดังนี้

1	( )	**6**	== , !=
2	++ , -- (วางไว้หน้าตัวแปร)	**7**	&&
3	*, /, %	**8**	||
4	+, -	**9**	=, +=, -=, /=, %=
5	<, <=, >, >=	**10**	++, -- (วางไว้หลังตัวแปร)

โอเปอเรเตอร์ที่มีลำดับความสำคัญเท่ากัน โอเปอเรเตอร์ที่มาก่อนจะถูกนำมาใช้ในการคำนวณก่อน เช่น ลองดูการใช้โอเปอรเตอร์และผลลัพธ์ที่ได้จากโค้ดต่อไปนี้

```
a = 2 * 100 / 10;         //a = 20 เพราะเท่ากับ (2  * 100)  / 10
b = 100 / 10 * 2          //b = 20 เพราะเท่ากับ (100  /  10)  * 2
c = 100 * 10 + 1;         //c = 1001 เพราะเท่ากับ (100 * 10) + 1
d = 2 * 100 / 10 + 30;    //d = 50 เพราะเท่ากับ ((2 * 100) / 10) + 30
e = 1 > 2 && 3 < 4 || 5 == 6 //e = false => (false && true) || false = false || false

int x, y = 5, z = 10;
x = y++ * --z;            //x = 5 * (10 - 1) = 45
//เนื่องจาก y++ มีลำดับความสำคัญต่ำสุด ดังนั้นการประมวลผลด้วย * และกำหนดค่าด้วย = จึงเกิดขึ้นก่อน y++
```

ในทางปฏิบัติเพื่อหลีกเลี่ยงข้อผิดพลาดจากลำดับความสำคัญของโอเปอเรเตอร์ ควรใช้วงเล็บในการจัดแบ่งกลุ่มให้ชัดเจน

การกำหนดค่าแบบเชื่อมโยง

การกำหนดค่าแบบเชื่อมโยง (Associative) เป็นการกำหนดค่าแบบต่อเนื่องให้กับหลายตัวในคำสั่งเดียวกัน ซึ่งมักใช้ในกรณีที่การกำหนดค่านั้นมีผลเกี่ยวเนื่องกันระหว่างตัวแปร โดยให้เราเริ่มพิจารณาค่าของตัวแปรจาก **ขวาสุดไปยังซ้ายสุด** เช่น

```
int a, b, c, d;
a = b = c = d = 100;  //ค่าของทั้งตัวแปร a, b, c และ d จะเป็น 100

int a, b, c = 100;
a = b = c += 50;      //ค่าของทั้งตัวแปร a, b และ c จะเป็น 150
```

```
int a, b = 10, c = 30;
a = b *= c += 20;     //c = 50, b = 10*50, a = b
```

การคำนวณที่มีผลกระทบต่อชนิดข้อมูล

การคำนวณตัวเลขที่มีชนิดข้อมูลต่างกัน ผลลัพธ์ที่ได้จะมีชนิดข้อมูลเหมือนกับตัวเลขที่มีชนิดข้อมูลใหญ่ที่สุดที่เรานำมาคำนวณ เช่น ลองพิจารณาโค้ดต่อไปนี้ (ดูการเปรียบเทียบขนาดข้อมูลจากบทที่ 1)

```
byte b = 10;
short s = 100;
ชนิดข้อมูล r = b + s;     //r มีชนิดข้อมูลเป็น short เนื่องจากเป็นชนิดข้อมูลใหญ่ที่สุดของเลขที่นำมาคำนวณ

int i = 5555;
short s = 3000;
double d = 10.5;
ชนิดข้อมูล r = i / d * s; //r มีชนิดข้อมูลเป็น double เนื่องจากเป็นชนิดข้อมูลใหญ่ที่สุดของเลขที่นำมาคำนวณ
```

จากหลักการตรงนี้ เป็นสิ่งที่เราต้องคำนึงถึงในการเขียนโปรแกรม เพราะบางครั้งอาจก่อให้เกิดข้อผิดพลาดที่เราคาดไม่ถึงก็เป็นได้ เช่น กรณีต่อไปนี้

```
int x = 123;
short y = 456;
short z = x + y;
```

โค้ดข้างบนนี้จะเกิดข้อผิดพลาด เพราะผลลัพธ์ x + y จะต้องเป็นข้อมูลชนิด int แต่เรากำหนดชนิดข้อมูลของ z เป็น short ซึ่งมีขนาดเล็กกว่า int สำหรับแนวทางการแก้ปัญหามีดังนี้

- อาจใช้วิธีกำหนดชนิดข้อมูลผลลัพธ์ ให้เท่ากับหรือใหญ่กว่าชนิดข้อมูลที่ใหญ่ที่สุดที่นำมาคำนวณ

```
short a = 1234;
int b = 5678;
int c = a * b;
//long c = a * b;  ← แบบนี้ก็ได้ เพราะ long ใหญ่กว่า int อยู่แล้ว
```

- อาจใช้วิธีคาสต์ชนิดข้อมูลที่ใหญ่กว่า ให้เป็นชนิดเดียวกันหรือเล็กกว่าชนิดข้อมูลของผลลัพธ์ เช่น

```
long a = 1234L;
int b = 567;
int c = ((int)a) * b; //ปกติผลลัพธ์ต้องเป็นชนิด long แต่เราเปลี่ยน long เป็น int ให้ตรงกับชนิดของ c
```

การตรวจสอบเงื่อนไขด้วย if

คำสั่ง if จะเป็นการตรวจสอบว่าเงื่อนไขนั้นตรงตามที่ระบุหรือไม่ ทั้งนี้เราสามารถกำหนดรูปแบบการทำงานได้ว่าหากตรงกับเงื่อนไขจะให้ทำอะไร และหากไม่ตรงกับเงื่อนไขจะต้องทำอย่างไร โดยเงื่อนไขแบบ if มีการใช้งานในหลายรูปแบบ ดังนี้

การใช้คำสั่ง if

ใช้สำหรับการตรวจสอบเงื่อนไขก่อนการทำตามคำสั่ง หากตรงกับที่กำหนด จะทำตามคำสั่งที่อยู่ภายในบล็อก { } โดยมีรูปแบบดังนี้

```
if(เงื่อนไข) {
   คำสั่งต่างๆ ถ้าเงื่อนไขตรงกับที่กำหนด (จะมีกี่คำสั่งก็ได้)
}
```

สำหรับการกำหนดเงื่อนไขต้องเขียนไว้ในวงเล็บเสมอ และสิ่งที่จะนำมากำหนดเป็นเงื่อนไขของ if ต้องให้ผลลัพธ์ออกมาเป็น true หรือ false เท่านั้น เช่น

```
int withdraw = ...;
if(withdraw > 20000) {
   System.out.println("จำนวนเงินที่จะถอนต้องไม่เกิน 20000 บาทเท่านั้นค่ะ");
}

if((withdraw % 100) != 0) {
   System.out.println("จำนวนเงินที่จะถอนต้องเป็นจำนวนเต็มร้อยเท่านั้นค่ะ");
}
```

```
boolean first = ...          //true หรือ false
if(first == true) {          //หรือเขียนอีกแบบเป็น  if(first) {
   first = false;
}
```

```
boolean gameOver = ...       //true หรือ false
if(gameOver == false) {      //หรือเขียนอีกแบบเป็น  if(!gameOver) {
   ...
}
```

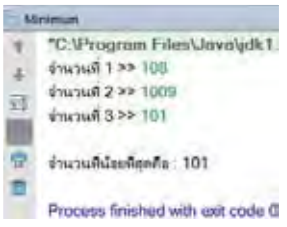
```
String login = "...";
if(login.equals("admin")) {
   ...
}
```

ตัวอย่าง Example 2-3 จากตัวอย่างที่แล้ว (2-2) เราหาค่ามากที่สุดโดยใช้ Ternary Operator ในการเปรียบเทียบ แต่ในตัวอย่างนี้ เราจะเปลี่ยนมาหาค่าน้อยที่สุดจากจำนวนที่รับเข้ามาโดยใช้ if ในการเปรียบเทียบแทน แต่หลักการนั้นยังเหมือนเดิม เพียงแต่เปลี่ยนจากค่ามากที่สุดเป็นน้อยที่สุด

```
import  java.util.Scanner;

public class Minimum {
    public static  void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int min;
        System.out.print("จำนวนที่ 1 >> ");
        int n1 = scan.nextInt();
        min = n1;

        System.out.print("จำนวนที่ 2 >> ");
        int n2 = scan.nextInt();
        if(n2 < min) {
            min = n2;
        }
        System.out.print("จำนวนที่ 3 >> ");
        int n3 = scan.nextInt();
        if(n3 < min) {
            min = n3;
        }
        System.out.println("\nจำนวนที่น้อยที่สุดคือ : " + min);
        scan.close();
    }
}
```



คำสั่ง if...else

คำสั่ง if...else ใช้ในกรณีที่เราต้องการกำหนดทางเลือกอื่นในการทำงาน หากเงื่อนไขไม่ตรงกับที่ระบุใน if โดยมีรูปแบบดังนี้

```
if(เงื่อนไข) {
   คำสั่งต่างๆ ถ้าเงื่อนไขตรงกับที่กำหนด
} else {
   คำสั่งต่างๆ ถ้าเงื่อนไขไม่ตรงกับที่กำหนด
}
```

ลักษณะการเปรียบเทียบ เช่น

```
int x = ...
if(x % 2 == 0) {
   System.out.print(x + " เป็นเลขคู่");    //ถ้าเลขนั้นหารด้วย 2 ลงตัวแสดงว่าเป็นเลขคู่
} else {
   System.out.print(x + " เป็นเลขคี่");    //มิฉะนั้น (ถ้าหารด้วย 2 ไม่ลงตัว) แสดงว่าเป็นเลขคี่
}
```

```
boolean isLeapYear = false;
int numDaysInYear;
if(!isLeapYear) {                        //หรือเท่ากับ if(isLeapYear == false) {
   numDaysInYear = 365;
} else {
   numDaysInYear = 366;
}
```

การใช้คำสั่ง else if

ในกรณีที่มีการตรวจสอบหลายๆ เงื่อนไข ทำให้ไม่สะดวกต่อการใช้ if หลายๆ ครั้ง เราสามารถใช้ else if เข้ามาช่วยตรวจสอบในแต่ละเงื่อนไขได้ โดยใช้รูปแบบดังต่อไปนี้

```
if(เงื่อนไขที่ 1) {
   คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 1
} else if(เงื่อนไขที่ 2) {
   คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 2
} else if(เงื่อนไขที่ 3 ) {
   คำสั่งต่างๆ ถ้าตรงกับเงื่อนไขที่ 3
...
} else { //กรณีอื่นๆ (ถ้ามี)
   คำสั่งต่างๆ ถ้าไม่ตรงกับเงื่อนไขใดๆ เลย
}
```

ลักษณะการใช้งาน เช่น

```
byte manUnitedGoals = ...
byte liverpoolGoals = ...

if(manUnitedGoals > liverpoolGoals) {        // ถ้าประตูของ ManU มากกว่า Liverpool
   System.out.print("ManUnited ชนะ!");
} else if(manUnitedGoals < liverpoolGoals) { // ถ้าประตูของ ManU น้อยกว่า Liverpool
   System.out.print("Liverpool ชนะ!");
} else {
   System.out.print("เสมอกัน");
}
```

ทั้งนี้จะมีการตรวจสอบเงื่อนไขด้วย else if กี่ครั้งก็ได้ และเราสามารถนำการตรวจสอบเงื่อนไขทั้ง if, else if และ else มาใช้ร่วมกันได้ แต่ต้องวาง else ไว้ในลำดับท้ายสุด เช่น

```
double balance = 500000.00;       //ยอดคงเหลือ
double withdraw = ...             //จำนวนที่จะถอน
if(balance < withdraw) {          //ยอดเงินคงเหลือ ต้องมากกว่าจำนวนที่จะถอน จึงจะถอนได้
   System.out.print("ยอดเงินคงเหลือไม่เพียงพอ");
} else if(withdraw > 20000) {     //ถ้ากดเกิน 20000
   System.out.print("ถอนได้ไม่เกิน 20000 บาท");
} else if(withdraw % 100 != 0) {  //ถ้าไม่ใช่จำนวนเต็มร้อย
   System.out.print("ต้องเป็นจำนวนเต็มร้อยเท่านั้น");
} else {                          //ถ้าไม่ตรงตามเงื่อนไขทั้งหมด แสดงว่าถอนได้
   balannce -= withdraw;
}
```

ถ้าเงื่อนไขสามารถเป็นไปได้หลายกรณี โปรแกรมจะเลือกทำเพียงเงื่อนไขแรกที่ตรวจสอบว่าตรงกับที่กำหนดเท่านั้น ส่วนเงื่อนไขต่อๆ ไปจะถูกข้ามไปเลย ลองดูลักษณะโค้ดต่อไปนี้

```
int score = 75;
char grade;

if(score >= 90) {
   grade = 'A';
} else if(score >= 80) {
   grade = 'B';
} else if(score >= 70) {
   grade = 'C';
} else if(score >= 60) {
   grade = 'D';
} else {
   grade = 'F';
}
```

จากที่กำหนด score = 75 แสดงว่าตรงกับทั้งเงื่อนไข (score >= 70) และ (score >= 60) แต่เนื่องจากเราวางเงื่อนไข (score >= 70) เอาไว้ก่อน ดังนั้นเฉพาะเงื่อนไข (score >= 70) เท่านั้นที่จะถูกเลือก และหลังจากที่ทำตามเงื่อนไขนี้เสร็จก็ข้ามการตรวจสอบเงื่อนไขส่วนนี้ไปเลย

ตัวอย่าง Example 2-4 เปลี่ยนหน่วยของขนาดไฟล์ให้เหมาะสม โดยรับข้อมูลเข้ามาในหน่วยไบต์ แล้วเปรียบเทียบว่าสามารถเปลี่ยนเป็นหน่วยใหญ่ที่สุดหน่วยใดได้ โดยจำนวนไบต์ของแต่ละหน่วยเป็นดังตาราง

หน่วย	จำนวนไบต์
KB	1024
MB	1048576
GB	1073741824


```
import java.util.Scanner;

public class FileSizeConversion {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("ขนาดของไฟล์ (ไบต์) >> ");
        double size = scan.nextDouble();
        String unit;

        if(size >= 1073741824) {      //ถ้ามีขนาด 1073741824 ขึ้นไปให้แปลงเป็นหน่วย GB
            size /= 1073741824;
            unit = "GB";
        } else if(size >= 1048576) {  //ถ้ามีขนาด 1048576 ขึ้นไปให้แปลงเป็นหน่วย MB
            size /= 1048576;
            unit = "MB";
        } else if(size >= 1024) {     //ถ้ามีขนาด 1024 ขึ้นไปให้แปลงเป็นหน่วย KB
            size /= 1024;
            unit = "KB";
        } else {                      //มิฉะนั้น (ต่ำกว่า 1024) ก็ได้เป็นหน่วย Byte เช่นเดิม
            unit = (size > 1) ? "Bytes" : "Byte";
        }
        System.out.printf("แปลงเป็นขนาดที่เหมาะสมได้ประมาณ: %.2f %s\n", size, unit);
        scan.close();
    }
}
```

การใช้คำสั่ง if ซ้อนกัน

สาเหตุที่ต้องใช้คำสั่ง if ซ้อนกัน (nest if) ก็เนื่องจาก การทำตามคำสั่งของบางเงื่อนไขนั้น ต้องขึ้นอยู่กับเงื่อนไขอื่นๆ ด้วย ซึ่งการใช้ if ซ้อนกันนี้ เงื่อนไขแรกสุดต้องตรงตามที่กำหนด เงื่อนไขที่อยู่ในลำดับถัดไปจึงจะถูกประมวลผล โดยมีรูปแบบดังต่อไปนี้

```
if(เงื่อนไข 1){          ถ้าเงื่อนไขที่ 1 เป็นจริง
   ...
   if(เงื่อนไข 2){       ถ้าเงื่อนไขที่ 2 เป็นจริง
      ...
      if(เงื่อนไข n){    ถ้าเงื่อนไขที่ n เป็นจริง
         ...
      }                  สิ้นสุดเงื่อนไขที่ n
   }                     สิ้นสุดเงื่อนไขที่ 2
}                        สิ้นสุดเงื่อนไขที่ 1
```

บล็อกของ if ที่อยู่ชั้นใน อาจจะไม่ได้ซ้อนใน if โดยตรง แต่อาจซ้อนอยู่ใน else if หรือ else ก็ได้ ทั้งนี้ขึ้นกับวิธีการพิจารณาลำดับเงื่อนไขเป็นหลัก เช่น การเปรียบเทียบเพื่อหาผู้ชนะในเกมเป่ายิ้งฉุบ

```
if(player1.equals("hammer")) {
   if(player2.equals("hammer")) {
      System.out.print("เสมอกัน");
   } else if(player2.equals("paper")) {
      System.out.print("player 2 ชนะ");
   } else if(player2.equals("scissors")) {
      System.out.print("player 1 ชนะ");
   }
} else if(player1.equals("paper")) {
   if(player2.equals("hammer")) {
      System.out.print("player 1 ชนะ");
   } else if(player2.equals("paper")) {
      System.out.print("เสมอกัน");
   } else if(player2.equals("scissors")){
      System.out.print("player 2 ชนะ");
   }
} else if(player1.equals("scissors")) {
   ...
}
```

เทคนิคการตรวจสอบหลายเงื่อนไข

ถ้ามีหลายเงื่อนไขที่เราต้องตรวจสอบก่อนทำตามคำสั่ง จากหัวข้อที่แล้วเราจะต้องใช้ if ซ้อนกัน หรือไม่ก็ใช้ else if หลายครั้ง แต่บางกรณีเราสามารถรวมเอาเงื่อนไขเหล่านั้นมาตรวจสอบด้วย if เพียงครั้งเดียว โดยใช้โอเปอเรเตอร์ทางตรรกะในการเชื่อมโยงระหว่างเงื่อนไข ทั้งนี้หากเราพิจารณาการใช้ร่วมกับ if ก็สามารถสรุปหลักการสำคัญได้ดังนี้

- ใช้โอเปอเรเตอร์ && ถ้าต้องการให้เงื่อนไขทั้งหมดเป็นจริงจึงจะทำตามคำสั่ง
- ใช้โอเปอเรเตอร์ || ถ้าต้องการให้เพียงเงื่อนไขอันใดอันหนึ่งเป็นจริงก็จะทำตามคำสั่ง

เช่น ลองพิจาณาจากโค้ดต่อไปนี้ (ไม่ขอนำขั้นตอนการประกาศชนิดตัวแปรมาแสดง)

```
//ถ้าต้องการให้ตัวแปร age ต้องมีค่าระหว่าง 12-20 จึงจะทำตามคำสั่ง
if((age >= 12) && (age <= 20)) {
   System.out.print("You are teen-age");
}
```

```
//ถ้าไม่ใส่ข้อมูล login หรือ password อันใดอันหนึ่ง ก็ให้แสดงคำเตือน
if(login.equals("") || password.equals("")) {
   System.out.print("ท่านต้องใส่ข้อมูลให้ครบทั้งสองช่อง");
}
```

นอกจากนี้เราสามารถตรวจสอบได้มากกว่า 2 เงื่อนไข และควรใช้วงเล็บในการแยกแต่ละกลุ่มให้ชัดเจนว่าเงื่อนไขใดจะเปรียบเทียบกับเงื่อนไขใด มิฉะนั้นอาจเกิดข้อผิดพลาดได้ เช่น

```
if((x > 0) && (x < 100) && (x % 1 > 0)) {      //x ต้องตรงกับทุกเงื่อนไขจึงจะทำตามคำสั่ง
   ...
}
```

```
//ถ้ามีเงื่อนไขว่า หากเป็นผู้ชายต้องแต่งงานแล้ว หรือถ้าเป็นผู้หญิงต้องยังโสด
if((gender.equals("male") && status.equals("married")) ||
   (gender.equals("female") && status.equals("single")) {
   ...
}
```

ตัวอย่าง Example 2-5 เงื่อนไขของปีที่เป็นปี Leap Year (เดือนกุมภาพันธ์มี 29 วัน) มี 2 อย่างคือ (ถ้าให้ตัวแปร year แทนปี ค.ศ. ที่ต้องการตรวจสอบ)

- ปีนั้นต้องหารด้วย 400 **ลง**ตัว ดังนั้นลักษณะเงื่อนไขนี้คือ

```
year % 400 == 0
```

- หรือปีนั้นต้องหารด้วย 100 **ไม่**ลงตัวแต่หารด้วย 4 ลงตัว ดังนั้นลักษณะเงื่อนไขนี้คือ

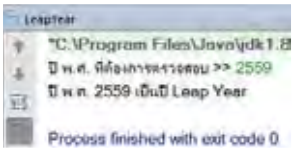
```
(year % 100 != 0) && (year % 4 == 0)
```

ทั้งสองเงื่อนไขที่กล่าวมา หากเพียงอันใดอันหนึ่งเป็นจริง (เชื่อมด้วย ||) ก็จะถือว่าเป็นปี Leap Year ให้รับข้อมูลเป็น พ.ศ. เข้ามา (ต้องลบด้วย 543 เพื่อให้เป็น ค.ศ.) แล้วตรวจสอบว่าเป็นปี Leap Year หรือไม่

```
import java.util.Scanner;

public class LeapYear {
    public static  void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("ปี พ.ศ. ที่ต้องการตรวจสอบ >> ");
        int year = scan.nextInt() ;
        System.out.print("ปี พ.ศ. " + year);
        year -= 543;

        if((year % 400 == 0) || ((year % 100 != 0) && (year % 4 ==0))) {
           System.out.println(" เป็นปี Leap Year");
        } else {
            System.out.println(" ไม่เป็นปี Leap Year");
        }
        scan.close();
    }
}
```



คำสั่ง switch...case

ใช้ในกรณีที่ค่าของข้อมูลที่เราต้องการตรวจสอบนั้น อาจเป็นไปได้หลายกรณี โดยกำหนดว่าค่าที่เป็นไปได้ในแต่ละกรณีนั้นจะต้องทำอย่างไร ซึ่งการใช้ switch นี้จะคล้ายกับการเปรียบเทียบด้วย else if แต่ในบางครั้งเราอาจไม่สะดวกที่จะใช้ else if ก็สามารถใช้ switch แทนได้ รูปแบบที่ใช้มีดังนี้

```
switch(สิ่งที่ต้องการตรวจสอบ) {
   case กรณีที่ 1:
      คำสั่ง
      break;
   case กรณีที่ 2:
      คำสั่ง
      break;
   case กรณีที่ n:
      คำสั่ง
      break;
   default: คำสั่ง ถ้าไม่ตรงกับกรณีใดเลยๆ
}
```

ลักษณะการใช้งาน เช่น

```
char size = ...;
switch(size) {
   case 'S': System.out.print("Small"); break;
   case 'M': System.out.print("Medium"); break;
   case 'L': System.out.print("Large"); break;
   default: System.out.print("Unknown Size");
}
```

ข้อควรทราบเพิ่มเติมเกี่ยวกับการใช้คำสั่ง switch...case

- ข้อมูลที่จะนำมาใช้กับ switch นั้นต้องเป็นชนิด **เลขจำนวนเต็ม** (byte, short, int, long) หรือ String หรือ char เท่านั้น จะเป็นเลขทศนิยมชนิด double หรือ float ไม่ได้
- break เป็นการสั่งให้โปรแกรมออกจากคำสั่ง switch หากเจอเงื่อนไขที่ต้องการแล้ว มิฉะนั้นโปรแกรมจะตรวจสอบเงื่อนไขถัดไปเรื่อยๆ แม้ว่าจะพบกับเงื่อนไขที่ต้องการแล้วก็ตาม
- กรณี default เราจะกำหนดหรือไม่ก็ได้ ขึ้นกับว่ามีลักษณะอื่นๆ ที่ต้องจัดการหรือไม่
- ถ้ามีหลาย case ที่ต้องใช้คำสั่งในรูปแบบเดียวกัน เราไม่จำเป็นต้องไปเขียนคำสั่งเหล่านั้นซ้ำๆ ก็ได้ แต่สามารถการนำไปเขียนไว้ที่ case ตัวสุดท้ายที่ตรงกับเงื่อนไขได้เลย เช่น

```
String m = "...";
switch(m) {
   case "Samsung":
   case "Sony":
   case "HTC": System.out.print("Android"); break;
   case "iPhone":
   case "iPad": System.out.print("iOS"); break;
   case "Surface": System.out.print("Windows"); break;
   default: System.out.print("Unknown");
}
```

โค้ดข้างบนนี้หมายความว่า หาก m มีค่าอย่างใดอย่างหนึ่งระหว่าง (Samsung, Sony, HTC) จะพิมพ์คำว่า Android ส่วนถัดมาก็พิจารณาในลักษณะเดียวกัน ซึ่งหากเราเปลี่ยนไปใช้ if จะเขียนได้เป็น

```
if(m.equals("Samsung") || m.equals("Sony") || m.equals("HTC")) {
   System.out.print("Android");
} else if(...) {
...
```

คำสั่ง for

for เป็นคำสั่งสำหรับกำหนดการทำงานแบบวงรอบหรือลูป (Loop) จากค่าเริ่มต้นจนถึงค่าสุดท้ายของตัวที่ใช้เป็นตัวนับ โดยที่ตัวนับไม่จำเป็นต้องเริ่มต้นจาก 0 และอาจเป็นการนับแบบเพิ่มค่าหรือลดค่าก็ได้ ซึ่งรูปแบบของลูปแบบ for คือ

```
for(ตัวแปร = ค่าเริ่มต้น; เงื่อนไขการวนลูป; การเปลี่ยนค่าตัวแปร) {
    คำสั่งต่างๆ
}
```

โดยโปรแกรมจะเปรียบเทียบตัวนับว่าอยู่ภายในเงื่อนไขหรือไม่ หากอยู่ภายในเงื่อนไขก็จะเปลี่ยนค่าของตัวนับเพื่อวนลูปถัดไป แต่หากตัวนับไม่อยู่ในเงื่อนไขก็จะออกจากลูปไป เช่น หากต้องการนำเลขตั้งแต่ 1-10 มาบวกกันก็เขียนเป็นลูปแบบ for ดังนี้

```
int sum = 0;
for(int i = 0; i <= 10; i++) {
   sum += i;
}
```

ข้อควรรู้เพิ่มเติมสำหรับลูปแบบ for มีดังนี้

- สำหรับวิธีการเปลี่ยนค่าของตัวนับนั้น อาจจะเป็นการเพิ่มค่า หรือลดค่าก็ได้ นอกจากนี้ก็ไม่จำเป็นต้องเพิ่มหรือลดทีละ 1 อาจเป็นค่าที่มากกว่า 1 ก็ได้ เช่น

```
for(int a = 0; a <= 50 1; a += 5)      //เพิ่มค่าทีละ 5   (a += 5)
```

```
for(int b = 100; b >= 1; b -= 10)      //ลดค่าทีละ 10   (b -= 10)
```

- ตัวนับอาจไม่ใช่เลขจำนวนเต็มเสมอไป อาจเป็นเลขจุดทศนิยมก็ได้ เช่น

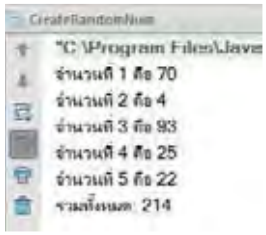
```
for(double x = 1.5; x <= 7.5; x += 0.5)
```

- **ไม่** ควรเปลี่ยนค่าของตัวนับภายในลูป เพราะอาจเกิดข้อผิดพลาดขึ้นได้ เช่น

```
for(int i = 1; i < 100; i++) {
   i += 10;                            //ไม่ควรเปลี่ยนค่า counter ในลูปแบบนี้
   ...
}
```

ตัวอย่าง Example 2-6 ใช้ลูป for สร้างเลขสุ่ม 5 จำนวน แล้วแสดงค่า พร้อมหาผลรวม

```
public class CreateRandomNum {
    public static void main(String[] args) {
        int n, sum = 0;
        for(int i = 1; i <= 5; i++) {
            n = (int)(Math.random() * 100);
            sum += n;
            System.out.printf("จำนวนที่ %d คือ %d\n", i, n);
        }
        System.out.println("รวมทั้งหมด: " + sum);
    }
}
```



คำสั่ง while

การทำงานเป็นวงรอบแบบ for ใช้ได้เฉพาะกรณีที่เราทราบค่าเริ่มต้นและค่าสุดท้ายของการวนรอบเท่านั้น ทั้งนี้หากเราไม่ทราบค่าเหล่านี้ล่วงหน้า จะไม่สามารถใช้ลูปแบบ for ได้ แต่ถ้าเราทราบเพียงเงื่อนไขบางอย่างก็อาจใช้ลูป while แทนได้ โดยลูปแบบ while จะมีการตรวจสอบเงื่อนไขก่อนวนรอบ หากตรงตามเงื่อนไขก็จะเริ่มทำตามคำสั่งในลูป ไปจนกว่าเงื่อนไขจะไม่ตรงตามที่กำหนด ซึ่งมีรูปแบบดังนี้

```
while(เงื่อนไข) {
   คำสั่งต่างๆ
}
```

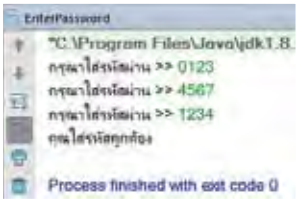
สำหรับเงื่อนไข ก็กำหนดคล้ายกับคำสั่ง if คือต้องเป็นการเปรียบเทียบที่ให้ค่าเป็น true หรือ false เช่น หากต้องการตรวจสอบว่า จะสุ่มกี่ครั้งจึงจะได้เลขที่มีค่าตั้งแต่ 0.9 ขึ้นไป ก็กำหนดโค้ดดังนี้

```
double x = 0.0;
int times = 0;
while(x < 0.9) {        //ถ้าได้ค่าที่น้อยกว่า 0.9 ต้องวนลูปต่อไป
   x = Math.random();   //ค่าเลขสุ่มที่จะได้ จะอยู่ระหว่าง 0-1
   times += 1;
}
System.out.printf("ต้องสุ่มทั้งหมด %d ครั้ง เพื่อให้ได้ค่าตั้งแต่  0.9 ขึ้นไป", times);
```

ตัวอย่าง Example 2-7 เป็นการรับรหัสผ่านจากผู้ใช้ ซึ่งหากใส่ผิดก็ให้ใส่ใหม่ไปเรื่อยๆ ดังนั้นจึงใช้ลูป while ในการรับข้อมูลไปเรื่อยๆ จนกว่าจะใส่ถูก

```
import java.util.Scanner;

public class EnterPassword {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        boolean validCode = false;
        String code = "";
        while(validCode == false) {  //ถ้าตัวแปร validCode ยังเป็น false ก็ให้วนลูปต่อไป
            System.out.print("กรุณาใส่รหัสผ่าน >> ");
            code = scan.next();
            if(code.equals("1234")) {
                validCode = true; //ถ้าใส่รหัสถูก เปลี่ยนค่า validCode เป็น true เพื่อออกจากลูป while
            }
        }
        System.out.println("คุณใส่รหัสถูกต้อง");
        scan.close();
    }
}
```



คำสั่ง do while

do...while เป็นการวนลูปที่จะตรวจสอบเงื่อนไขที่ท้ายลูป โดยจะมีการทำงานตามคำสั่งอย่างน้อย 1 ครั้ง ดังนั้นลูปแบบนี้จึงเหมาะกับงานที่ต้องมีการกระทำบางอย่างไปก่อนในครั้งแรก แล้วค่อยตรวจสอบเงื่อนไข ทั้งนี้การวนลูปจะดำเนินไปจนกว่าเงื่อนไขจะตรงกับค่าที่กำหนด ซึ่งรูปแบบของลูปนี้คือ

```
do {
    คำสั่งต่างๆ
} while(เงื่อนไข);
```

สำหรับการกำหนดเงื่อนไขก็มีหลักการเช่นเดียวกับลูปแบบ while หรือคำสั่ง if คือต้องเป็นการเปรียบเทียบที่ให้ค่าออกมาเป็น true หรือ false เท่านั้น และต้องเขียนเงื่อนไขไว้ในวงเล็บ โดยลูปแบบ do...while กับแบบ while มีข้อแตกต่างที่สำคัญดังนี้

- ลูปแบบ while จะตรวจสอบเงื่อนไขก่อนเริ่มลูป และจะวนลูป **ขณะที่** เงื่อนไข ยังตรงกับที่กำหนด
- ลูปแบบ do...while จะตรวจสอบเงื่อนไขที่ท้ายลูป นั่นแสดงว่าลูปแบบนี้จะต้องทำตามคำสั่งที่กำหนดในบล็อกของลูปอย่างน้อย 1 ครั้ง และจะวนลูป **ขณะที่** เงื่อนไขยังตรงกับที่กำหนด