เอกสารประกอบการสอน
วิชา
Object-Oriented Programming (Java)

ผศ.ดร. ธนิศา นุ่มนนท์
คณะเทคโนโลยีสารสนเทศ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

สารบัญ

บทที่ 1 ความรู้เบื้องต้นภาษาจาวา.....1

1.1 ระบบคอมพิวเตอร์เบราว์เซอร์.....1

1.2 ภาษาคอมพิวเตอร์.....2

1.3 ตัวแปลภาษา.....6

1.4 หลักการของโปรแกรมเชิงอ็อบเจกต์.....7

1.5 ประวัติภาษาจาวา.....9

1.6 เทคโนโลยีจาวา.....10

1.6.1 Java Virtual Machine.....11

1.6.2 Java Runtime Environment.....12

1.6.3 Java Development Kit.....13

1.7 จุดเด่นของภาษาจาวา.....15

1.8 แพลตฟอร์มของเทคโนโลยีจาวา.....16

1.8.1 Java Platform, Standard Edition.....18

1.8.2 Java Platform, Enterprise Edition.....18

1.8.3 Java Platform, Micro Edition.....19

1.9 โปรแกรมภาษาจาวา.....20

1.9.1 การสร้างโปรแกรมจาวาประยุกต์.....20

1.9.2 การสร้างโปรแกรมจาวาแอปเพล็ต.....22

1.10 คู่มือ Java API.....24

สรุปเนื้อหาของบท.....25

บทที่ 2 พื้นฐานโปรแกรมภาษาจาวา.....27

2.1 การเขียนโปรแกรมภาษาจาวาเชิงอ็อบเจกต์.....27

2.2 ไวยากรณ์ระดับของคำ.....29

2.2.1 คอมเม็นต์.....29

2.2.2 Identifier.....30

2.2.3 คีย์เวิร์ด.....31

2.2.4 สัญลักษณ์แยกคำ....32
2.2.5 ช่องว่าง....33
2.2.6 ข้อมูลค่าคงที่....33
2.3 ชนิดข้อมูลแบบพื้นฐาน....33
2.3.1 ชนิดข้อมูลตรรกะ....35
2.3.2 ชนิดข้อมูลตัวอักขระ....35
2.3.3 ชนิดข้อมูลตัวเลขจำนวนเต็ม....36
2.3.4 ชนิดข้อมูลตัวเลขทศนิยม....37
2.4 ตัวแปรและค่าคงที่....38
2.4.1 คำสั่งกำหนดค่า....39
2.4.2 ค่าคงที่....41
2.4.3 ขอบเขตของตัวแปรและค่าคงที่....42
2.5 ตัวดำเนินการ....45
2.5.1 ตัวดำเนินการทางคณิตศาสตร์....46
2.5.2 ตัวดำเนินการแบบสัมพันธ์....48
2.5.3 ตัวดำเนินการทางตรรกศาสตร์....50
2.5.4 ตัวดำเนินการแบบบิต....52
2.5.5 ลำดับความสำคัญของตัวดำเนินการ....53
2.6 การแปลงชนิดข้อมูล....54
2.6.1 การแปลงข้อมูลในคำสั่งกำหนดค่า....55
2.6.2 Typecasting....57
2.7 ชนิดข้อมูลแบบอ้างอิง....58
2.7.1 คลาส String....60
2.7.2 คลาส Math....62
2.8 คำสั่งอินพุตและเอาต์พุต....64
2.8.1 System.out.println()....64
2.8.2 การรับข้อมูลเข้ามาทาง Command Line....65
สรุปเนื้อหาของบท....66

บทที่ 3 โครงสร้างควบคุม.....69
3.1 คำสั่งโครงสร้างควบคุม.....69
3.2 โครงสร้างแบบเลือกทำ.....70
3.2.1 คำสั่ง if.....70
3.2.2 คำสั่ง if..else.....72
3.2.3 คำสั่ง if แบบซ้อน.....74
3.2.4 คำสั่ง switch.....78
3.3 โครงสร้างแบบทำซ้ำ.....80
3.3.1 คำสั่ง while.....81
3.3.2 คำสั่ง do..while.....83
3.3.3 คำสั่ง for.....84
3.4 โครงสร้างแบบซ้อน (Nested Structure).....86
3.4.1 คำสั่ง break, continue และ label.....88
สรุปเนื้อหาของบท.....90

บทที่ 4 หลักการเชิงอ็อบเจกต์.....93
4.1 องค์ประกอบของโปรแกรมเชิงอ็อบเจกต์.....93
4.1.1 อ็อบเจกต์.....93
4.1.2 คลาส.....94
4.1.3 คุณลักษณะ.....95
4.1.4 เมธอด.....97
4.2 การเขียนโปรแกรมเชิงอ็อบเจกต์โดยใช้ภาษาจาวา.....98
4.2.1 การประกาศคลาส.....98
4.2.2 การประกาศคุณลักษณะ.....99
4.2.3 การประกาศเมธอด.....99
4.2.4 การประกาศและสร้างอ็อบเจกต์.....101
4.2.5 การเรียกใช้สมาชิกของอ็อบเจกต์.....102
4.3 คุณลักษณะเด่นของโปรแกรมเชิงอ็อบเจกต์.....104
4.3.1 การห่อหุ้ม.....104

4.3.2 การสืบทอด 106
4.3.3 การมีได้หลายรูปแบบ.....107
4.4 การเขียนโปรแกรมเชิงอ็อบเจกต์เพื่อสร้างคลาสแบบ abstract และอินเตอร์เฟส.....109
4.4.1 คลาสแบบ abstract.....109
4.4.2 อินเตอร์เฟส.....111
4.5 แพคเก็จ.....113
4.5.1 การประกาศและใช้แพคเก็จ.....114
4.6Unified Modeling Language.....115
4.6.1 ไดอะแกรมของคลาส 116
4.6.2 ไดอะแกรมของอ็อบเจกต์.....116
4.7 ขั้นตอนการพัฒนาโปรแกรม.....117
สรุปเนื้อหาของบท.....119

บทที่ 5 การสร้างส่วนต่อประสานกราฟิกกับผู้ใช้.....121
5.1Java Foundation Class 121
5.1.1 แพคเก็จ AWT.....122
5.1.2 แพคเก็จ Swing.....124
5.2 คลาสประเภท Container 125
5.2.1 คลาส Frame.....125
5.2.2 คลาส Panel.....127
5.2.3 คลาส Dialog.....128
5.2.4 คลาส JFrame.....128
5.3 การจัดวางผังของส่วนประกอบกราฟิก.....131
5.3.1 BorderLayout.....132
5.3.2 FlowLayout.....135
5.3.3 GridLayout.....137
5.4 ส่วนประกอบกราฟิกในแพคเก็จ Swing.....138
5.4.1 คลาส JButton.....140
5.4.2 คลาส JLabel.....141

5.4.3 คลาส JTextField....144
5.4.4 คลาส JTextArea....145
5.4.5 คลาส JcheckBox และ JRadioButton....147
5.4.6 คลาส JComboBox....150
5.4.7 คลาส JList....151
5.5 การสร้างเมนู....153
5.5.1 การสร้าง JMenuBar....154
5.5.2 การสร้าง JMenu....154
5.5.3 การสร้าง JMenuItem....155
5.5.4 คลาส JCheckBoxMenuItem....157
5.5.5 การสร้างเมนูย่อย....159
5.6 คุณลักษณะของคลาส Component....161
สรุปเนื้อหาของบท....163

บทที่ 6 การเขียนโปรแกรมภาษาจาวาเชิงอ็อบเจกต์....165
6.1 เมธอด 165
6.1.1 การเรียกใช้เมธอด....165
6.1.2 การส่งผ่าน argument 166
6.1.3 การรับค่าที่ส่งกลับมา....172
6.1.4 modifier ของเมธอด....172
6.2 การเขียนโปรแกรมโดยใช้หลักการของการห่อหุ้ม....173
6.2.1 เมธอดแบบ Accessor....175
6.2.2 คีย์เวิร์ด this....178
6.3 การเขียนโปรแกรมโดยใช้หลักการของการสืบทอด....179
6.3.1 การสืบทอดที่ถูกต้อง....182
6.3.2 คีย์เวิร์ด protected 183
6.3.3 คลาสที่ชื่อ Object....184
6.3.4 คีย์เวิร์ด super 185
6.4 การมีได้หลายรูปแบบ....186

6.4.1 Dynamic Binding....186
6.4.2 การกำหนดเมธอดให้มีวิธีการที่ต่างกัน....187
6.4.3 Virtual Method Invocation....190
6.4.4 การส่งผ่าน argument ได้หลายรูป....190
6.4.5 ตัวดำเนินการ instanceof....191
6.4.6 การ Casting อ็อบเจกต์....192
6.5 Constructor....194
6.5.1 การเขียน Constructor....195
6.5.2 ขั้นตอนการทำงานของคำสั่ง new....196
6.5.3 Constructor แบบ Overloaded....197
6.5.4 เมธอด this()....198
6.5.5 เมธอ super()....199
6.5.6 ขั้นตอนการทำงานของ Constructor....200
6.6 เมธอดของคลาสที่ชื่อ Object....202
6.6.1 เมธอด toString()....202
6.6.2 เมธอด equals()....203
6.7 คลาสประเภท Wrapper....204
6.7.1 Autoboxing....205
6.8 คีย์เวิร์ดอื่นๆ ที่สำคัญ....206
6.8.1 คุณลักษณะแบบ static....206
6.8.2 เมธอดแบบ static....207
6.8.3 Static Initializer....208
6.8.4 คีย์เวิร์ด final....209
6.9 คลาสภายใน....210
6.9.1 คลาสภายในที่อยู่ภายในคลาส....210
6.9.2 คลาสภายในที่อยู่ภายในเมธอด....212
6.10 Generic Types....213
6.10.1 เมธอดแบบ Generic....214
6.11Annotation....215

6.11.1 Annotation Document....216
6.11.2 Annotation ที่ใช้โดยคอมไพเลอร์....217
สรุปเนื้อหาของบท....218

บทที่ 7 การจัดการกับเหตุการณ์กราฟิก....221
7.1 เหตุการณ์....221
7.2 AWTEvent....223
7.2.1 ActionEvent....224
7.2.2 WindowEvent....225
7.2.3 MouseEvent....225
7.2.4 ItemEvent....226
7.2.5 Event อื่นๆ....227
7.3 อินเตอร์เฟสประเภท Listener....227
7.4 การจัดการกับเหตุการณ์....230
7.4.1 การสร้างอ็อบเจกต์ของคลาสภายนอก....231
7.5 การสร้างอ็อบเจกต์ของคลาสภายใน....232
7.5.1 การสร้างอ็อบเจกต์ภายในคลาสเดียวกัน....234
7.5.2 การรับฟังเหตุการณ์หลายเหตุการณ์....235
7.5.3 คลาสประเภท Event Adapter....237
7.5.4 การสร้างคลาสแบบ anonymous....238
สรุปเนื้อหาของบท....239

บทที่ 8 อะเรย์และคอลเล็กชั่น....241
8.1 อะเรย์....241
8.2 อะเรย์ของข้อมูลชนิดพื้นฐาน....242
8.2.1 การประกาศชื่อตัวแปรอะเรย์ของข้อมูลชนิดพื้นฐาน....242
8.2.2 การสร้างตัวแปรอะเรย์ของข้อมูลชนิดพื้นฐาน....243
8.2.3 การเรียกใช้สมาชิกของอะเรย์....245
8.2.4 การกำหนดค่าเริ่มต้นให้กับสมาชิกของอะเรย์....246

8.2.5 การใช้คำสั่ง for เพื่ออ้างอิงสมาชิกของอะเรย์....247
8.2.6 ข้อผิดพลาดประเภท ArrayIndexOutOfBoundsException....248
8.3 อะเรย์ของข้อมูลชนิดคลาส....249
8.3.1 การเก็บค่าของตัวแปรอะเรย์ของข้อมูลชนิดคลาส....251
8.4 อะเรย์หลายมิติ....252
8.4.1 การเขียนโปรแกรมเพื่อจัดการกับเมตริกซ์....254
8.4.2 อะเรย์สองมิติที่มีจำนวนคอลัมน์ต่างกัน....257
8.4.3 เมธอดที่เกี่ยวข้องกับอะเรย์....259
8.5 คอลเล็กชั่น....261
8.5.1 อินเตอร์เฟส Collection....262
8.5.2 อินเตอร์เฟส Set....263
8.5.3 อินเตอร์เฟส List....264
8.5.4 อินเตอร์เฟส Map....266
8.5.5 อินเตอร์เฟส Iterator....268
8.5.6 คลาส Vector....271
8.5.7 การใช้คำสั่ง for และ Generic....272
สรุปเนื้อหาของบท....273

บทที่ 9 การจัดการกับข้อผิดพลาด....275
9.1 ข้อผิดพลาด....275
9.2 Exception....276
9.3 คำสั่ง try..catch....278
9.4 การจัดการกับข้อผิดพลาดหลายๆ ประเภท....279
9.5 บล็อก finally....281
9.6 การจัดการกับเมธอดที่ส่งอ็อบเจกต์ประเภท Exception....283
9.7 การสร้างคลาสประเภท Exception ขึ้นใหม่....285
9.7.1 การเขียนเมธอดเพื่อส่งอ็อบเจกต์ประเภท Exception....286
สรุปเนื้อหาของบท....287

บทที่ 10 คลาสอินพุตและเอาต์พุต.....289
10.1 Stream 289
10.1.1 แพคเก็จ java.io.....290
10.1.2 คลาสประเภท Byte Stream.....292
10.1.3 คลาส InputStream.....292
10.1.4 คลาส OutputStream.....294
10.1.5 คลาสประเภท Character Stream.....295
10.2 โหนดสำหรับ Stream.....296
10.3 คลาสประเภท Stream ระดับสูง.....298
10.3.1 DataInputStream และ DataOutputStream.....301
10.3.2 InputStreamReader และ OutputStreamWriter.....304
10.4 คลาส File.....305
10.4.1 คลาส RandomAccessFile.....307
10.4.2 ObjectInputStream และ ObjectOutputStream.....308
สรุปเนื้อหาของบท.....310

บทที่ 11 โปรแกรมแบบเธรด.....313
11.1 โปรแกรมแบบมัลติเธรด.....313
11.2 การสร้างคลาสแบบเธรด.....315
11.2.1 การ extends คลาสที่ชื่อ Thread.....316
11.2.2 การ implements อินเตอร์เฟส Runnable.....318
11.3 วงจรการทำงานของเธรด.....320
11.4 เมธอดของคลาส Thread.....323
11.5 Synchronization.....329
11.5.1 เมธอด wait() และ notify() 332
11.5.2 Deadlock.....334
สรุปเนื้อหาของบท.....336

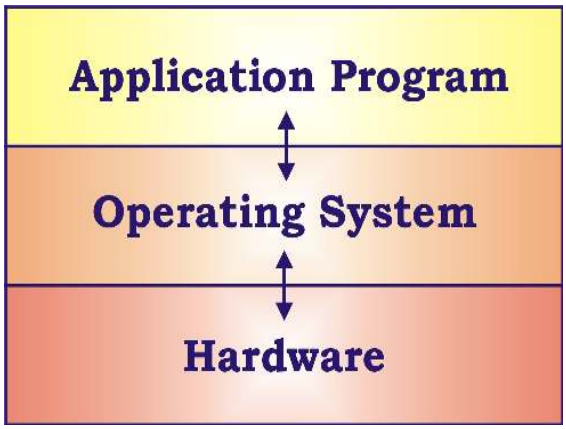
บทที่ 1 ความรู้เบื้องต้นภาษาจาวา

เนื้อหาในบทนี้เป็นการแนะนำภาษาจาวา โดยจะเริ่มจากการแนะนำหลักการของภาษาคอมพิวเตอร์โดยทั่วๆ ไป ประวัติความเป็นมาโดยย่อของภาษาจาวา ข้อแตกต่างของภาษาจาวากับภาษาคอมพิวเตอร์อื่นๆ หลักการการทำงานของโปรแกรมภาษาจาวา และความหมายของเครื่องจักรสมมุติที่ใช้ในภาษาจาวา พร้อมทั้งแนะนำเทคโนโลยีและแพลตฟอร์มต่างๆ ของภาษาจาวา แนะนำตัวอย่างการเขียนโปรแกรมจาวาประยุกต์และโปรแกรมจาวาแอปเพล็ต และในส่วนท้ายของบทได้มีการแนะนำวิธีการใช้คู่มือ Java API

1.1 ระบบคอมพิวเตอร์เบราว์เซอร์

เครื่องคอมพิวเตอร์เป็นเครื่องอิเล็กทรอนิกส์ ที่ใช้ในการคำนวณและจัดการกับระบบข้อมูล องค์ประกอบของระบบคอมพิวเตอร์โดยทั่วไปจะเป็นดังแสดงในรูปที่ 1.1 ซึ่งจะประกอบไปด้วยส่วนหลักสามส่วนคือ

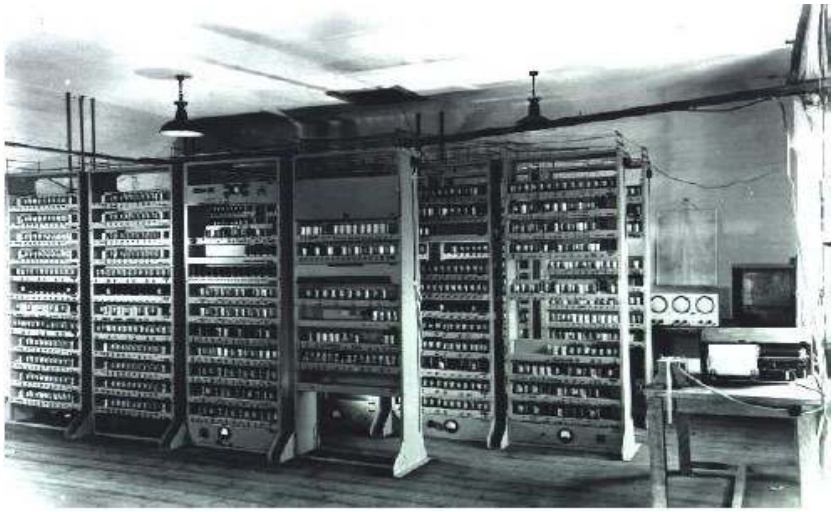
1. ฮาร์ดแวร์ (Hardware) คือส่วนประกอบที่เป็นตัวเครื่องคอมพิวเตอร์ที่ประกอบด้วยหน่วยประมวลผลกลาง (Central Processing Unit หรือ CPU) อุปกรณ์ส่วนอินพุต อุปกรณ์ส่วนเอาต์พุต หน่วยความจำ และอุปกรณ์เก็บข้อมูล
2. ระบบปฏิบัติการ (Operating System) คือระบบซอฟต์แวร์ที่ติดตั้งเพื่อให้โปรแกรมประยุกต์ต่างๆ สามารถติดต่อกับฮาร์ดแวร์ได้ ระบบปฏิบัติการที่นิยมใช้ในปัจจุบันมีอยู่หลายระบบ อาทิเช่น Windows Vista, Linux และ Solaris เป็นต้น ทั้งนี้ฮาร์ดแวร์ชนิดเดียวกันสามารถที่จะมีระบบปฏิบัติการที่แตกต่างกันได้
3. โปรแกรมประยุกต์ (Application Program) คือโปรแกรมที่ใช้งานทั่วๆ ไป เช่น โปรแกรม Word Processor, เกมส์ หรือโปรแกรมเว็บเบราว์เซอร์ (Web Browser) เป็นต้น โปรแกรมเหล่านี้จะถูกพัฒนาโดยใช้ภาษาคอมพิวเตอร์ต่างๆ อาทิเช่น ภาษาฟอร์แทรน (FORTRAN) ภาษาโคบอล (COBOL) ภาษา C# ภาษา C++ หรือภาษาจาวา เป็นต้น โปรแกรมประยุกต์จะทำงานภายใต้ระบบปฏิบัติการ ดังนั้นโปรแกรมประยุกต์ที่ทำงานบนระบบปฏิบัติการระบบใดระบบหนึ่งจะไม่สามารถนำไปใช้ในระบบ ปฏิบัติการอื่นได้



รูปที่ 1.1 องค์ประกอบของระบบคอมพิวเตอร์

<u>เกร็ดความรู้</u>

EDSAC เป็นเครื่องคอมพิวเตอร์รุ่นแรกๆของโลกสร้างขึ้นเมื่อเดือนพฤษภาคม คศ. 1949 โดยมีหน่วยความจำหลักเพียง 512 คำ (words) และสามารถคำนวณคำสั่งได้ 600 คำสั่งต่อวินาที และมีขนาดใหญ่ดังรูป



1.2 ภาษาคอมพิวเตอร์

ภาษาคอมพิวเตอร์คือคำสั่งที่นักพัฒนาโปรแกรม (programmer) พัฒนาขึ้นเพื่อสั่งให้เครื่องคอมพิวเตอร์ทำงานตามที่ต้องการ ภาษาคอมพิวเตอร์แบ่งออกเป็นสามประเภทคือ

1. ภาษาเครื่อง (Machine Language) เป็นภาษาเดียวที่เครื่องคอมพิวเตอร์สามารถเข้าใจได้ ภาษาเครื่องจะประกอบไปด้วยคำสั่งที่เป็นชุดของเลขฐานสองเช่น 01010110 ซึ่งจะถูกกำหนดโดยฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ ทั้งนี้ภาษาเครื่องจะขึ้นอยู่กับชนิดของหน่วยประมวลผลกลางของเครื่องคอมพิวเตอร์แต่ละเครื่อง โดยปกติแล้วนักพัฒนาโปรแกรมไม่สามารถที่จะพัฒนาโปรแกรมโดยเขียนภาษาเครื่องได้โดยตรง ทั้งนี้เนื่องจากเป็นภาษาที่ใช้เลขฐานสองซึ่งไม่ใช่ชุดคำสั่งที่มนุษย์จะสามารถเข้าใจได้โดยง่าย ตัวอย่างของคำสั่งภาษาเครื่องมีดังนี้

```
10110011  00011001
01111010  11010001  10010100
10011111  00011001
01011100  11010001  10010000
10111011  11010001  10010110
```

2. ภาษาแอสเซมบลี (Assembly Language) เป็นภาษาคอมพิวเตอร์ที่แทนชุดคำสั่งเลขฐานสองด้วยคำสัญลักษณ์ที่เป็นภาษาอังกฤษ เช่น 10110011 อาจแทนด้วย MOV เป็นต้น ทำให้นักพัฒนาโปรแกรมเขียนและเข้าใจโปรแกรมได้ง่ายขึ้น การทำงานของโปรแกรมภาษาแอสเซมบลี จะต้องมีการแปลภาษา แอสเซมบลีให้เป็นภาษาเครื่องก่อนจึงจะทำงานได้ แม้ภาษาแอสเซมบลีจะมีชุดคำสั่งที่เป็นคำภาษาอังกฤษ แต่นักพัฒนาโปรแกรมภาษาแอสเซมบลีจะต้องมีความเข้าใจโครงสร้างฮาร์ดแวร์ของระบบคอมพิวเตอร์จึงจะสามารถเขียนโปรแกรมได้ ดังนั้นทำให้ภาษาแอสเซมบลีไม่เป็นที่นิยมใช้ ตัวอย่างของโปรแกรมภาษาแอสเซมบลีมีดังนี้

```
MOV   0, SUM
MOV   NUM, AC
ADD   SUM, AC
STO   SUM, TOT
```

3. ภาษาระดับสูง (High-level Language) การพัฒนาโปรแกรมโดยทั่วไปจะใช้โปรแกรมภาษาคอมพิวเตอร์ระดับสูง ภาษาคอมพิวเตอร์ระดับสูงจะใช้ชุดคำสั่งที่คนทั่วไปเข้าใจได้ง่าย เหมือนการใช้คำสั่งในภาษาอังกฤษหรือการเขียนสมการคณิตศาสตร์ทั่วไป นักพัฒนาโปรแกรมไม่จำเป็นต้องเข้าใจหลักการทำงานของฮาร์ดแวร์ก็สามารถที่จะพัฒนาโปรแกรมภาษาคอมพิวเตอร์ระดับสูงได้ โปรแกรมภาษาคอมพิวเตอร์ระดับสูง จะต้องการตัวแปลภาษาที่ทำหน้าที่เปลี่ยนชุดคำสั่งให้มาเป็นภาษาแอสเซมบลีหรือภาษาเครื่องจึงจะทำงานได้ เมื่อเทียบกับภาษาแอสเซมบลีแล้ว ภาษาคอมพิวเตอร์ระดับสูงจะช่วยให้นักพัฒนาโปรแกรมพัฒนาโปรแกรมได้รวดเร็วกว่า แต่โปรแกรมที่พัฒนาขึ้นจะทำงานได้ช้ากว่า

ในปัจจุบันมีโปรแกรมภาษาคอมพิวเตอร์ระดับสูงอยู่หลายร้อยภาษา แต่ที่นิยมใช้และได้รับการยอมรับมีเพียง

ไม่กี่ภาษา ภาษาคอมพิวเตอร์ในยุคแรกที่นิยมใช้มีหลายภาษาอาทิเช่น

- ภาษาฟอร์แทรน (FORTRAN ย่อมาจาก FORmula TRANslator) พัฒนาโดยบริษัท IBM ระหว่างปี ค.ศ. 1954 ถึง ค.ศ. 1957 ภาษานี้ใช้สำหรับการพัฒนาโปรแกรมประยุกต์ด้านวิทยาศาสตร์และวิศวกรรมศาสตร์ ที่ต้องใช้ในการคำนวณสมการคณิตศาสตร์ที่ซับซ้อน ปัจจุบันภาษาฟอร์แทรนยังเป็นที่นิยมใช้ในการพัฒนาโปรแกรมด้านวิทยาศาสตร์และวิศวกรรมศาสตร์
- ภาษาโคบอล (COBOL ย่อมาจาก COmmon Business Oriented Language) พัฒนาขึ้นในปี ค.ศ. 1959 เป็นภาษาที่พัฒนาขึ้นมาเพื่อใช้ในการพัฒนาโปรแกรมประยุกต์ด้านธุรกิจและการค้า ปัจจุบันโปรแกรมที่ใช้ในด้านธุรกิจจำนวนมากอาทิเช่นโปรแกรมในสถาบันการเงิน ยังเป็นโปรแกรมที่พัฒนามาจากภาษาโคบอล
- ภาษาเบสิก (BASIC ย่อมาจาก Beginners All-purpose Symbolic Instructional Code) เป็นภาษาที่พัฒนาขึ้นโดยมีจุดประสงค์เพื่อให้ผู้เริ่มต้นพัฒนาโปรแกรมสามารถเรียนรู้และเข้าใจการพัฒนาโปรแกรมอย่างง่าย ภาษาเบสิกเป็นภาษา คอมพิวเตอร์ภาษาแรกที่ใช้ในเครื่องไมโครคอมพิวเตอร์

<u>เกร็ดความรู้</u>

การเขียนโปรแกรมภาษาฟอร์แทรนหรือภาษาโคบอลในยุคแรกจะใช้เครื่องคอมพิวเตอร์ที่อ่านโปรแกรมจาก punch card ที่ต้องใช้การ์ดหนึ่งใบต่อหนึ่งคำสั่ง ซึ่งเครื่องคอมพิวเตอร์ที่ใช่ punch card ยังมีการใช้ในการเรียนการสอนในประเทศไทยจนถึงปี พ.ศ. 2529 จนกระทั่งเครื่องคอมพิวเตอร์พีซีเข้ามาแทนที่

ภาษาคอมพิวเตอร์ที่พัฒนาขึ้นในยุคแรก ยังมีข้อจำกัดในการที่จะพัฒนาโปรแกรมขนาดใหญ่ ทั้งนี้เนื่องจากภาษาคอมพิวเตอร์เหล่านั้นขาดโครงสร้างที่ดี ทำให้การพัฒนาโปรแกรมที่มีขนาดใหญ่และมีความซับซ้อนเป็นไปได้ยาก ในช่วงต้นปี ค.ศ. 1970 จึงมีภาษาคอมพิวเตอร์ที่เป็นภาษาเชิงกระบวนการ (Procedural หรือ Structural Language) เกิดขึ้น ภาษาคอมพิวเตอร์ประเภทนี้จะมีความยืดหยุ่นในการพัฒนาโปรแกรม ทำให้สามารถแก้ไขและบำรุงรักษาได้ง่าย เนื่องจากโปรแกรมถูกแยกออกเป็นส่วนๆ ภาษาคอมพิวเตอร์ที่เป็นภาษาเชิงกระบวนการที่สำคัญคือ

- ภาษาปาสคาล (Pascal) พัฒนาโดย Nicklaus Wirth ในปี ค.ศ. 1971 โดยมีจุดประสงค์เพื่อใช้ในการสอนการเขียนโปรแกรมภาษาเชิงกระบวนการ ในมหาวิทยาลัย แต่เนื่องจากภาษาปาสคาลไม่มีคุณลักษณะที่จะสนับสนุนการพัฒนาโปรแกรมด้านธุรกิจและอุตสาหกรรมจึงไม่ได้รับความนิยมมากนัก
- ภาษาซี (C) พัฒนาขึ้นในช่วงเดียวกับภาษาปาสคาล โดยนักวิจัยที่ห้องปฏิบัติการ AT&T Bell ซึ่งได้นำเอาจุดเด่นของภาษา BCPL และภาษา B มาใช้และได้เพิ่มคุณลักษณะและชนิดข้อมูลอื่นเข้ามาด้วย เดิมภาษาซีถือว่าเป็นภาษาคอมพิวเตอร์ที่สำคัญในการพัฒนาโปรแกรมบนระบบปฏิบัติการยูนิกส์ (Unix) ซึ่งเป็นภาษาคอมพิวเตอร์ที่สามารถสร้างโปรแกรมประยุกต์ที่ทำงานได้รวดเร็วมาก เมื่อเทียบกับภาษาคอมพิวเตอร์อื่นๆ

จุดด้อยของการพัฒนาโปรแกรมภาษาเชิงกระบวนการคือ จะมีต้นทุนในการพัฒนาโปรแกรมที่ค่อนข้างสูง เนื่องจากจะต้องมีการออกแบบโปรแกรมให้แยกออกเป็นส่วนๆ ที่เรียกว่าโมดูล (module) ซึ่งจะต้องเป็นอิสระจากกัน การออกแบบให้มีความเป็นอิสระต่อกันนั้นทำได้ยาก ซึ่งหากออกแบบมาไม่ดีจะทำให้การแก้ไขและบำรุงรักษาโปรแกรมเป็นไปได้ยาก ดังนั้นจึงมีการพัฒนาภาษาคอมพิวเตอร์ในแบบที่เรียกว่า ภาษาเชิงอ็อบเจกต์ (Object-Oriented Programming) ที่พยายามให้การพัฒนาโปรแกรมเป็นการเลียนแบบการทำงานของอ็อบเจกต์ต่างๆ ในโลก ทั้งนี้เพื่อให้ง่ายต่อการพัฒนาโปรแกรมและสามารถนำโปรแกรมกลับมาใช้ใหม่ (reuse) ได้ดีกว่าภาษาเชิงกระบวนการ ภาษาคอมพิวเตอร์ที่เป็นภาษาเชิงอ็อบเจกต์ที่สำคัญคือ

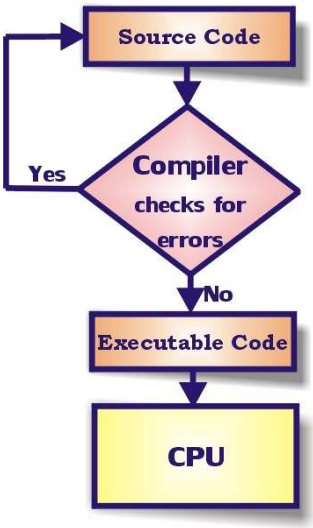
- ภาษา C++ เป็นภาษาที่พัฒนามาจากภาษาซีเมื่อต้น ค.ศ. 1980 โดยนักวิจัยที่ห้องปฏิบัติการ Bell โดยได้เพิ่มหลักการเชิงอ็อบเจกต์ขึ้นมาจากภาษาซี ดังนั้นนักพัฒนาโปรแกรมภาษา C++ สามารถที่จะพัฒนาโปรแกรมทั้งในเชิงอ็อบเจกต์และเชิงกระบวนการ (ตามแบบภาษาซี) ได้ ทำให้ปัจจุบันภาษา C++ ยังเป็นที่นิยมใช้กันอย่างแพร่หลาย
- ภาษา Smalltalk เป็นภาษาเชิงอ็อบเจกต์ที่พัฒนาโดยนักวิจัยที่ Xerox's Palo Alto Research Center (PARC) ซึ่งเป็นภาษาคอมพิวเตอร์ที่เป็นภาษาเชิงอ็อบเจกต์ อย่างแท้จริง แต่ภาษา Smalltalk ไม่ได้รับความนิยมในการนำไปใช้งานมากนักเมื่อเทียบกับภาษา C++ หรือภาษาจาวา
- ภาษาจาวาเป็นภาษาคอมพิวเตอร์ที่พัฒนาโดย นักวิจัยของบริษัทซันไมโครซิสเต็มส์ (Sun Microsystems) โดยเริ่มมีการนำมาเผยแพร่เมื่อปี ค.ศ. 1995 ปัจจุบันเป็นภาษาเชิงอ็อบเจกต์ที่เป็นที่นิยมใช้กันมากภาษาหนึ่ง ซึ่งได้รับการยอมรับในการพัฒนาโปรแกรมทางธุรกิจและอุตสาหกรรม และยังเป็นที่นิยมใช้เพื่อการศึกษาหลักการการเขียนโปรแกรมคอมพิวเตอร์โดยใช้ภาษาเชิงอ็อบเจกต์อีกด้วย นอกจากนี้ภาษาจาวายังเป็นซอฟต์แวร์แบบเปิดเผยซอร์สโค้ด (Open source code)
- ภาษา C# เป็นภาษาคอมพิวเตอร์ที่มีลักษณะคล้ายกับภาษาจาวาที่พัฒนาโดยบริษัท Microsoft โดยมีจุดประสงค์เพื่อให้นักพัฒนาโปรแกรมสามารถพัฒนาโปรแกรมเชิงอ็อบเจกต์ที่จะรันบนระบบปฏิบัติการ Windows ได้ง่ายขึ้น

1.3 ตัวแปลภาษา

ภาษาคอมพิวเตอร์ระดับสูงจะต้องการตัวแปลภาษา (Language Translator) เพื่อแปลโปรแกรมที่เขียนขึ้น หรือที่เรียกว่าซอร์ดโค้ด (Source Code) ของภาษาคอมพิวเตอร์แต่ละภาษาให้เป็นภาษาแอสเซมบลีหรือภาษาเครื่องที่เครื่องคอมพิวเตอร์สามารถเข้าใจได้ ตัวแปลภาษาแบ่งออกเป็นสองแบบคือ

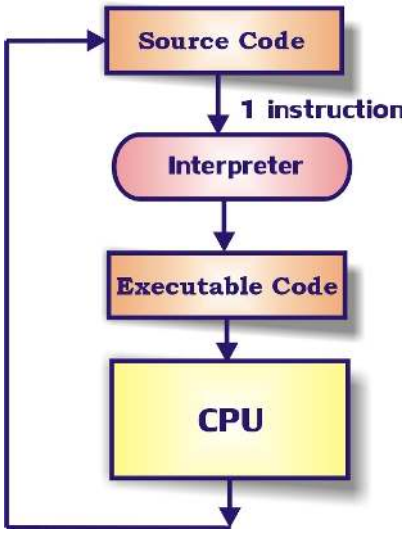
1. คอมไพเลอร์ (Compiler) ตัวแปลภาษาประเภทนี้จะแปลชุดคำสั่งในซอร์ดโค้ดทั้งหมดให้เป็นโปรแกรม executable code ดังแสดงในรูปที่ 1.2 ตัวอย่างเช่น แปลซอร์ดโค้ดภาษาซีจากโปรแกรม `Hello.c` ให้เป็นโปรแกรม executable code ที่ชื่อ `Hello.exe` โดยโปรแกรม executable code ที่ได้จากการแปลภาษาคอมพิวเตอร์ระดับสูงจะสามารถทำงานได้อย่างรวดเร็ว ทั้งนี้เนื่องจากโค้ดอยู่ในรูปของเลขฐานสองที่สามารถติดต่อกับเครื่องคอมพิวเตอร์ได้โดยตรง ตัวอย่างของภาษา คอมพิวเตอร์ที่ใช้คอมไพเลอร์ คือ ภาษาซี C++ ฟอร์แทรน และปาสคาล เป็นต้น

   คอมไพเลอร์จะสร้างโปรแกรม executable code ที่ขึ้นอยู่กับแพลตฟอร์ม (ฮาร์ดแวร์และระบบปฏิบัติการ) ดังนั้นผู้ใช้จะไม่สามารถนำโปรแกรม executable code ที่ได้จากการแปลของคอมไพเลอร์บนแพลตฟอร์มหนึ่งไปใช้บนแพลตฟอร์มอื่นๆ ได้ ในกรณีที่นักพัฒนาโปรแกรมต้องการนำโปรแกรมที่พัฒนาขึ้นไปใช้บนแพลตฟอร์มอื่นๆ นักพัฒนาโปรแกรมจะต้องทำการแปลซอร์ดโค้ด ของโปรแกรมใหม่ โดยจะต้องใช้คอมไพเลอร์ที่สร้างโปรแกรม executable code สำหรับแพลตฟอร์มที่ต้องการใช้งานนั้นๆ



รูปที่ 1.2 ขั้นตอนการทำงานของคอมไพเลอร์

2. อินเตอร์พรีตเตอร์ (Interpreter) ตัวแปลภาษาประเภทนี้จะแปลชุดคำสั่งของภาษาคอมพิวเตอร์ระดับสูงทีละคำสั่งให้เป็นโปรแกรม executable code แล้วจะสั่งให้เครื่องคอมพิวเตอร์ทำงานทันทีดังแสดงในรูปที่ 1.3 โปรแกรมภาษาคอมพิวเตอร์ที่ใช้อินเตอร์พรีตเตอร์ จะทำงานได้ช้ากว่าโปรแกรมภาษาคอมพิวเตอร์ที่ใช้คอมไพเลอร์ แต่ข้อดีของโปรแกรมภาษาคอมพิวเตอร์ที่ใช้อินเตอร์พรีตเตอร์คือจะสามารถใช้งานข้ามแพลตฟอร์มได้ ทั้งนี้เนื่องจากขั้นตอนการแปลภาษากระทำอยู่ในช่วงการรันโปรแกรม นอกจากนี้โปรแกรมอินเตอร์พรีตเตอร์จะพัฒนาได้ง่ายกว่าเนื่องจากมีขนาดเล็ก ตัวอย่างของภาษาคอมพิวเตอร์ที่ใช้อินเตอร์พรีตเตอร์คือ ภาษาเบสิก โปรล็อก (Prolog) และ Smalltalk เป็นต้น



รูปที่ 1.3 ขั้นตอนการทำงานของอินเตอร์พรีตเตอร์

1.4 หลักการของโปรแกรมเชิงอ็อบเจกต์

การเขียนโปรแกรมโดยใช้ภาษาคอมพิวเตอร์ระดับสูงในปัจจุบันจะมีแนววิธีการคิดอยู่สองรูปแบบที่สำคัญคือ แนวคิดเชิงกระบวนการ และแนวคิดเชิงอ็อบเจกต์ ดังที่กล่าวไว้ในหัวข้อที่ 1.2 การเขียนโปรแกรมโดยใช้ภาษาเชิงกระบวนการ จะเริ่มต้นจากการวิเคราะห์ปัญหาโดยพิจารณาจากกระบวนการและลำดับการทำงาน และจะพยายามแบ่งโปรแกรมออกเป็นส่วนๆ ตามฟังก์ชันของการทำงานอาทิเช่น การพัฒนาโปรแกรมระบบทะเบียนนักเรียนโดยใช้โปรแกรมเชิงกระบวนการอาจจะพิจารณาแบ่งโปรแกรมตามลำดับการทำงาน โดยอาจมีฟังก์ชันในการทำงานสำหรับการลงทะเบียนรายวิชา การชำระเงิน หรือการเพิ่มหรือถอนรายวิชา เป็นต้น

ฟังก์ชันแต่ละส่วนของโปรแกรมเชิงกระบวนการ จะมีตัวแปรที่จะส่งผ่านข้อมูลระหว่างกัน การออกแบบโปรแกรมเชิงกระบวนการที่ดีจะต้องพยายามแบ่งฟังก์ชันให้เป็นอิสระต่อกันให้มากที่สุด และต้องพยายามที่จะเขียนขั้นตอนหรือลำดับการทำงานให้สมบูรณ์ที่สุดตั้งแต่ขั้นตอนการออกแบบ จึงทำให้การปรับปรุงหรือแก้ไขโปรแกรมที่พัฒนาจากภาษาเชิงกระบวนการทำได้ยากเมื่อเทียบกับโปรแกรมที่พัฒนาจากภาษาเชิงอ็อบเจกต์ ทั้งนี้เนื่องจากกระบวนการการพัฒนาโปรแกรมนั้นไม่สามารถที่จะออกแบบโปรแกรมให้สมบูรณ์ตั้งแต่ต้นโดยไม่มีการนำกลับมาแก้ไขปรับปรุงอีกได้

การพัฒนาโปรแกรมเชิงอ็อบเจกต์จะมีแนวคิดในการแก้ปัญหา โดยมองปัญหาว่าประกอบไปด้วยอ็อบเจกต์ต่างๆ ซึ่งแนวคิดนี้จะเข้าใกล้เคียงกับธรรมชาติของมนุษย์มากที่สุด เนื่องจากมนุษย์มองสิ่งต่างๆ รอบตัวเป็นอ็อบเจกต์ทั้งที่เป็นรูปธรรม (วัตถุ) เช่น ปากกา นักศึกษา หรือใบลงทะเบียน เป็นต้น และที่เป็นนามธรรมเช่น คะแนน หรือรายชื่อวิชา เป็นต้น

การเขียนโปรแกรมเชิงอ็อบเจกต์จะเป็นขบวนการการวิเคราะห์ปัญหา โดยการจำลองคุณลักษณะและพฤติกรรมของอ็อบเจกต์ในระบบจริง ให้อยู่ในรูปของโปรแกรมคอมพิวเตอร์ ตัวอย่างเช่น การพัฒนาโปรแกรมระบบทะเบียนนักศึกษาอาจแบ่งโปรแกรมให้ประกอบด้วยอ็อบเจกต์ต่างๆ อาทิเช่น นักศึกษา ใบลงทะเบียน และรายวิชา เป็นต้น อ็อบเจกต์ชนิดนักศึกษาอาจมีคุณลักษณะต่างๆ เช่น ชื่อ รหัส และเกรดเฉลี่ย เป็นต้น และอาจมีพฤติกรรมที่นักศึกษาสามารถกระทำได้เช่น ลงทะเบียน และเพิ่มหรือถอนวิชา เป็นต้น

การพัฒนาโปรแกรมโดยใช้ภาษาคอมพิวเตอร์เชิงอ็อบเจกต์ จะทำให้ขบวนการพัฒนาโปรแกรมทำได้รวดเร็วขึ้นและสามารถปรับปรุงแก้ไขโปรแกรมได้ง่าย ซึ่งเหมาะกับการพัฒนาโปรแกรมขนาดใหญ่ที่จะต้องมีการปรับปรุงแก้ไขโปรแกรมอยู่ตลอด นอกจากนี้โปรแกรมเชิงอ็อบเจกต์ยังมีคุณลักษณะเด่นอื่นๆ อีกดังนี้

- การห่อหุ้ม (Encapsulation) เป็นคุณลักษณะที่ทำให้อ็อบเจกต์แต่ละตัวเป็นอิสระต่อกัน ซึ่งทำให้สามารถแบ่งการพัฒนาโปรแกรมออกเป็นส่วนๆ ได้ง่าย
- การสืบทอด (Inheritance) เป็นคุณลักษณะที่ทำให้สามารถนำโปรแกรมที่พัฒนาแล้วกลับมาใช้ใหม่ได้ง่ายกว่าการเขียนโปรแกรมเชิงกระบวนการ
- การมีได้หลายรูปแบบ (Polymorphism) เป็นคุณลักษณะที่ทำให้นักพัฒนาโปรแกรมสามารถเพิ่มเติมส่วนต่างๆ ของโปรแกรมได้ง่าย

1.5 ประวัติภาษาจาวา

ภาษาจาวาเป็นภาษาคอมพิวเตอร์ที่พัฒนาขึ้น โดยทีมวิจัยของบริษัทซันไมโครซิสเต็มส์ ซึ่งเริ่มต้นเมื่อปี ค.ศ. 1991 โดยทางบริษัทซันไมโครซิสเต็มส์ได้สนับสนุนให้มีการพัฒนาโครงการที่ชื่อ Green Project ที่มีจุดมุ่งหมายที่จะออกแบบภาษาคอมพิวเตอร์ขนาดเล็กเพื่อใช้กับอุปกรณ์อิเล็กทรอนิกส์ต่างๆ เช่น สวิทซ์บล็อกของเคเบิ้ลทีวี ซึ่งอุปกรณ์เหล่านี้จะมีข้อจำกัดในด้านหน่วยความจำและหน่วยประมวลผลกลางที่จะมีความแตกต่างกัน ดังนั้นภาษาคอมพิวเตอร์ที่พัฒนาขึ้นจะต้องไม่ขึ้นอยู่กับแพลตฟอร์ม (Platform Independent) ผลงานของทีมวิจัยนี้ทำให้ได้ภาษาคอมพิวเตอร์ใหม่ที่ชื่อว่า “Oak” ซึ่งมีโครงสร้างและคำสั่งคล้ายภาษา C++ ทั้งนี้เนื่องจากทีมวิจัยของบริษัทซันไมโครซิสเต็มส์มีความคุ้นเคยกับระบบปฏิบัติการยูนิกส์ซึ่งมักจะใช้ภาษา C++ ในการพัฒนาโปรแกรม ภาษา “Oak” ต่อมาได้เปลี่ยนมาเป็นชื่อ “จาวา” ทั้งนี้เนื่องจากทีมวิจัยได้ทราบภายหลังว่า ”Oak” เป็นชื่อภาษาคอมพิวเตอร์ที่มีอยู่แล้ว

ภาษาจาวาเป็นภาษาคอมพิวเตอร์เชิงอ็อบเจกต์ (Object Oriented Programming หรือเรียกย่อว่า OOP) โดยมีจุดเด่นตรงที่สามารถทำงานได้กับระบบคอมพิวเตอร์หลายแพลตฟอร์ม ทีมวิจัยของโครงการ Green Project ได้สร้างเครื่องต้นแบบที่เรียกว่า “*7” ซึ่งเป็นรีโมทคอนโทรล เพื่อสาธิตเทคโนโลยีภาษาจาวาที่ค้นคิดขึ้นมานี้เมื่อปี ค.ศ. 1992 โดยทางบริษัทซันไมโครซิสเต็มส์ได้พยายามหาผู้ซื้อเทคโนโลยีดังกล่าว แต่ก็ไม่ประสบความสำเร็จมากนัก

<u>เกร็ดความรู้</u>

James Gosling เป็นผู้ที่คิดค้นภาษาจาวาจนได้รับการยกย่องว่าเป็นบิดาของภาษาจาวา โดยเขาเกิดที่ประเทศ Canada จบการศึกษาระดับปริญญาเอกที่มหาวิทยาลัย Carnegie Mellon และเข้าทำงานที่บริษัท Sun Microsystems ตั้งแต่ปี ค.ศ. 1984 และในปี ค.ศ. 2007 เขาได้รับการยกย่องจากประเทศ Canada ให้เป็น Officer ของ Order of Canada ซึ่งเป็นเกียรติยศที่มอบให้กับพลเมืองของประเทศ

โลโก้ของภาษาจาวาจะเป็นรูปแก้วกาแฟที่มีควันร้อนระอุขึ้นมา โดยทางบริษัท Sun Microsystems ได้เปลี่ยนรูปโลโก้เดิมมาเป็นรูปล่าสุดดังที่แสดงให้เห็น

Duke เป็น mascot ของภาษาจาวา ซึ่งจะมีการใช้สัญลักษณ์ในท่าทางที่แตกต่างกัน และในงานประชุม Java One Conference ที่จัดเป็นประจำทุกปี ก็จะมีการมอบรางวัลให้แก่ซอฟต์แวร์หรือนักพัฒนาที่ใช้เทคโนโลยีจาวาและมีนวัตกรรมดีเด่นที่เรียกว่า Duke's Choice Awards

ในช่วงดังกล่าว ระบบอินเทอร์เน็ตได้เริ่มมีการใช้งานอย่างแพร่หลายมากขึ้น โดย เฉพาะการใช้งานของโปรแกรมเว็บเบราว์เซอร์ (Web Browser) เพื่อที่จะเรียกดูเอกสารในรูปแบบของไฟล์ HTML เนื่องจากระบบอินเทอร์เน็ตจะประกอบไปด้วยเครื่องคอมพิวเตอร์ที่มีแพลตฟอร์มแตกต่างกัน ทางบริษัทซันไมโครซิสเต็มส์จึงได้เห็นความจำเป็นที่จะต้องมีภาษาคอมพิวเตอร์ที่สามารถสร้างโปรแกรมที่ทำงานบนแพลตฟอร์มใดๆ ก็ได้ จึงได้นำภาษาจาวามาพัฒนาอีกครั้งหนึ่ง โดยได้พัฒนาโปรแกรมเว็บเบราว์เซอร์ที่ชื่อ Hot Java ที่สามารถรันโปรแกรมจาวาแอปเพล็ต (Java Applet) ได้เพื่อพิสูจน์การทำงานของเทคโนโลยีจาวา (Proof of Technology) และได้นำผลงานดังกล่าวมาแสดงในงาน Sun World'95 ในเดือนพฤษภาคม ปี ค.ศ. 1995

จากนั้นภาษาจาวาเริ่มเป็นที่นิยมแพร่หลายขึ้นในช่วงปลายปี ค.ศ. 1995 เมื่อบริษัท Netscape ได้พัฒนาโปรแกรมเว็บเบราว์เซอร์ Netscape 2.0 และได้รวมเอาการทำงานของ โปรแกรมจาวาแอปเพล็ต (Java Applet) เข้ามาเป็นคุณลักษณะเด่นของโปรแกรม ซึ่งบริษัทอื่นๆ อาทิเช่น IBM, Symantec, Inspire หรือ Microsoft ต่างก็ประกาศสนับสนุนการใช้งานของโปรแกรมภาษาจาวาในเวลาต่อมา ในช่วงเดียวกันทางบริษัทซันไมโครซิสเต็มส์ได้นำชุดพัฒนาโปรแกรมภาษาจาวาเวอร์ชันแรก (Java Development Kit 1.0 หรือที่เรียกย่อว่า JDK 1.0) ออกมาแจกจ่าย ต่อมาในปี ค.ศ. 1998 ทางบริษัทซันไมโครซิสเต็มส์ได้เผยแพร่โปรแกรมภาษาจาวา JDK 1.2 (แต่ต่อมาได้เปลี่ยนชื่อเป็น Java 2) และได้แยกแพลตฟอร์มในการพัฒนาโปรแกรมภาษาจาวาเป็นหลายแพลตฟอร์มคือ J2EE สำหรับการพัฒนาโปรแกรมประยุกต์บนเครื่องแม่ข่าย (Server) J2ME สำหรับการพัฒนาโปรแกรมบนเครื่องโทรศัพท์มือถือ และ J2SE สำหรับโปรแกรมมาตรฐานจาวาที่พัฒนาบนเครื่องคอมพิวเตอร์พื้นฐาน และปี ค.ศ. 2006 ทางซันไมโครซิสเต็มส์ได้เปลี่ยนชื่อ J2 ทั้งสามเป็น JavaEE, JavaME และ JavaSE ตามลำดับ

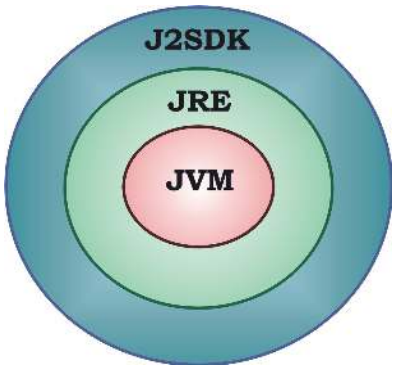
ในเดือนมกราคม ปี ค.ศ. 2010 บริษัทออราเคิล (Oracle Corporation) ซึ่งเป็นซอฟต์แวร์ที่พัฒนาโปรแกรมฐานข้อมูล และซอฟต์แวร์ประยุกต์อื่นๆ ได้เข้ามาซื้อกิจการบริษัทซันไมโครซิสเต็มส์ ทำให้ผู้พัฒนาเทคโนโลยีจาวารายสำคัญได้เปลี่ยนเป็นบริษัทออราเคิล ซึ่งได้ใช้เทคโนโลยีจาวาในซอฟต์แวร์ต่างๆ จำนวนมาก ทั้งนี้ บริษัทออราเคิลก็จะยังคงไว้ในหลักการของเทคโนโลยีจาวาที่จะให้มาตรฐานเปิด โปรแกรมเปิดรหัส (Open Source) และสามารถทำงานบนแพลตฟอร์มที่หลากหลายได้เช่นเดิม

1.6 เทคโนโลยีจาวา

เทคโนโลยีจาวาประกอบไปด้วยองค์ประกอบหลักๆ สามส่วนดังแสดงในรูปที่ 1.4 คือ

1. Java Virtual Machine (JVM) เป็นเทคโนโลยีจาวาที่ทำหน้าที่เป็นอินเตอร์พรีตเตอร์ ที่จะแปลโปรแกรมจาวาไบท์โค้ด (Java Bytecode) ให้เป็นภาษาที่เครื่องเข้าใจได้

2. Java Runtime Environment (JRE) เป็นเทคโนโลยีจาวาที่ใช้ในการรันโปรแกรมภาษาจาวาอาทิเช่น โปรแกรมจาวาประยุกต์ (Java Application) และโปรแกรมจาวาแอปเพล็ต (Java Applet) JRE จะประกอบด้วย JVM และ Java Application Programming Interface (Java API) ที่จะรวบรวมคลาสและอินเตอร์เฟสต่างๆ ที่จำเป็นต่อการใช้งานของโปรแกรมภาษาจาวา
3. Java Software Developer Kit (JDK) เป็นชุดพัฒนาโปรแกรมภาษาจาวาที่จะประกอบไปด้วย JVM ตัวแปลภาษาจาวา (Java Compiler) เครื่องมือ (tool) อื่นๆ ในการพัฒนาโปรแกรม และ API ทั้งหมดในภาษาจาวา ซึ่ง API จะเป็นมาตรฐานคำสั่งต่างๆ ของภาษาจาวา



รูปที่ 1.4 องค์ประกอบของเทคโนโลยีจาวา

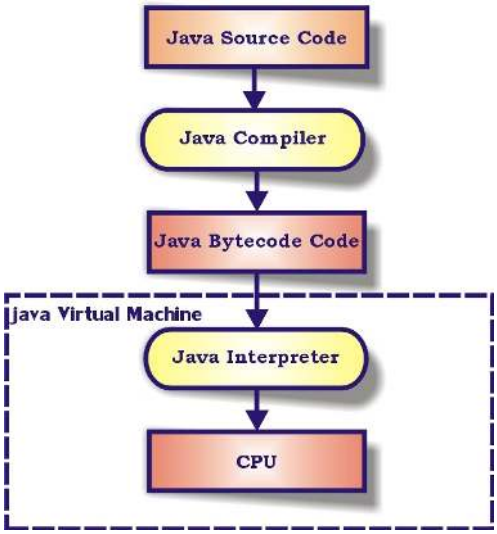
1.6.1 Java Virtual Machine

ภาษาจาวาออกแบบมาเพื่อให้สามารถประมวลผลได้กับทุกแพลตฟอร์ม โดยได้รวมหลักการของตัวแปลภาษาทั้งคอมไพเลอร์และอินเตอร์พรีตเตอร์ไว้ โดยคอมไพเลอร์ของภาษาจาวาจะทำหน้าที่แปลซอร์ดโค้ดของโปรแกรม (ชื่อไฟล์.`java`) ให้เป็นโปรแกรมไบท์โค้ด (ชื่อไฟล์.`class`) โปรแกรมไบท์โค้ดจะแตกต่างจากโปรแกรมภาษาเครื่องที่สามารถประมวลผลบนระบบปฏิบัติการได้โดยตรง แต่โปรแกรมไบท์โค้ดจะประมวลผลได้โดยผ่านอินเตอร์พรีตเตอร์ซึ่งจะแปลโปรแกรมไบท์โค้ดแล้วสั่งงานไปยังเครื่องคอมพิวเตอร์ อินเตอร์พรีตเตอร์แบบนี้มีชื่อเรียกว่า JVM (Java Virtual Machine หรือเครื่องจักรสมมุติจาวา) ซึ่งขั้นตอนการทำงานของโปรแกรมภาษาจาวาสามารถแสดงได้ดังรูปที่ 1.5

JVM จะทำหน้าที่แปลโปรแกรมไบท์โค้ดให้เป็นภาษาเครื่องที่ขึ้นอยู่กับแพลตฟอร์มโปรแกรมไบท์โค้ดที่ประมวลผลโดยใช้ JVM จะทำงานได้เร็วกว่าโปรแกรมภาษาคอมพิวเตอร์อื่นๆ ที่ใช้อินเตอร์พรีตเตอร์ปกติในการประมวลผล ทั้งนี้เนื่องจาก JVM จะมีชุดคำสั่งที่ใกล้เคียงกับชุดคำสั่งภาษาเครื่องของหน่วยประมวลผลกลางที่ใช้งาน ดังนั้น JVM จึงสามารถแปลคำสั่งของโปรแกรมไบท์โค้ดไปเป็นคำสั่งภาษาเครื่องของหน่วยประมวลผลกลางที่ใช้งานได้ง่ายกว่า นอกจากนี้โปรแกรมไบท์โค้ดจะไม่ขึ้นอยู่กับแพลตฟอร์ม ดังนั้นเราสามารถที่จะนำโปรแกรมไบท์โค้ดที่

คอมไพล์จากระบบปฏิบัติการหนึ่ง มาประมวลผลบนระบบปฏิบัติการอื่นๆ ได้ หากระบบปฏิบัติการนั้นมี JVM อยู่

JVM เป็นเครื่องจักรกลสมมุติที่ทำหน้าที่เหมือนกับระบบคอมพิวเตอร์จริง โดยมาตรฐานของ JVM ที่กำหนดโดยบริษัทซันไมโครซิสเต็มส์จะมีข้อกำหนดต่างๆ เช่นเดียวกับองค์ประกอบของระบบคอมพิวเตอร์เช่น ชุดคำสั่ง (instruction set) และชุดรีจีสเตอร์ (register set) เป็นต้น และยังมีข้อกำหนดอื่นๆ เช่น เนื้อที่ของหน่วยความจำ (memory area) สแต็ก (stack), heap, garbage collection และรูปแบบของคลาสไฟล์ (class file format) เป็นต้น ทั้งนี้ข้อกำหนดของ JVM เป็นมาตรฐานที่เปิดเผย ทำให้บริษัทต่างๆ สามารถที่จะพัฒนา JVM ขึ้นมาเองได้ โดยไม่จำกัดอยู่เพียง JVM ของบริษัทซันไมโครซิสเต็มส์เท่านั้น และล่าสุดในเดือนพฤศจิกายน ค.ศ. 2006 ทางซันไมโคร-ซิสเต็มส์ ได้เปิดเผยซอร์สโค้ดในการพัฒนาซอฟต์แวร์จาวาทั้งหมด



รูปที่ 1.5 ขั้นตอนการทำงานของโปรแกรมภาษาจาวา

JVM ที่พัฒนามาใช้ในปัจจุบันส่วนใหญ่จะเป็นซอฟต์แวร์ แต่ก็ได้มีการพัฒนา JVM ในรูปแบบของฮาร์ดแวร์ขึ้นมาบ้างแล้วเช่น Java Chip ในปัจจุบัน JVM มีใช้ในระบบ ปฏิบัติการคอมพิวเตอร์ต่างๆ โปรแกรมเว็บเบราว์เซอร์ เครื่องใช้ไฟฟ้าต่างๆ โทรศัพท์ เครื่องมือสื่อสารต่างๆ และสมาร์ตคาร์ด (Smart Card) จึงทำให้อุปกรณ์ต่างๆ เหล่านี้สามารถประมวลผลโปรแกรมที่พัฒนาโดยใช้ภาษาจาวาได้

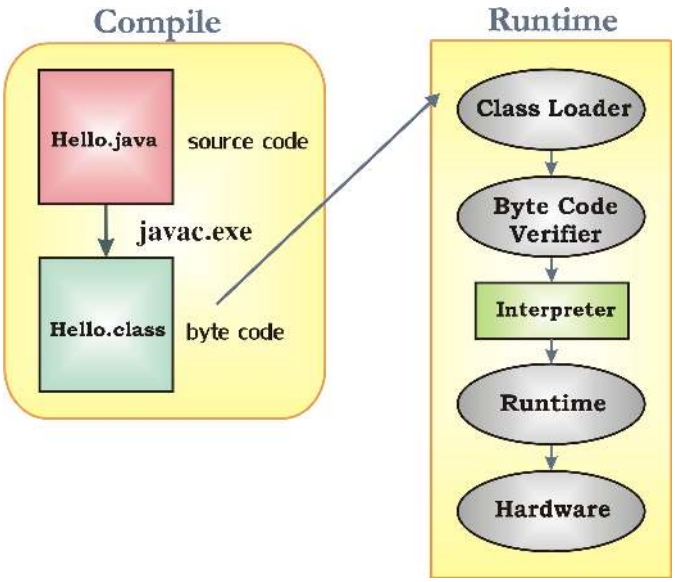
1.6.2 Java Runtime Environment

JRE จะรันโปรแกรมไบท์โค้ดที่แปลมาจาก JVM โดยจะมีขั้นตอนการทำงานสาม ขั้นตอนดังแสดงในรูปที่ 1.6 คือ

1. โหลดโปรแกรมไบท์โค้ด ขั้นตอนนี้จะเป็นการโหลดคลาสทุกคลาสที่จำเป็นต่อการรันโปรแกรม โดย

ใช้ Class Loader

2. ตรวจสอบโปรแกรมไบท์โค้ด ขั้นตอนนี้เป็นการตรวจสอบโดยใช้ Byte Code Verifier ว่าโปรแกรมไบท์โค้ดถูกต้องตามข้อกำหนดของ JVM หรือไม่ และโปรแกรมจะต้องไม่มีคำสั่งใดที่สั่งงานแล้วจะทำให้เกิดข้อผิดพลาดกับระบบ เช่น การแปลงข้อมูลที่ผิดพลาด หรือการพยายามบุกรุกเข้าสู่ระบบภายใน
3. รันโปรแกรมไบท์โค้ด ขั้นตอนนี้จะเป็นการรันโปรแกรมไบท์โค้ด โดยใช้ Runtime Interpreter



รูปที่ 1.6 ขั้นตอนการทำงานของ JRE

1.6.3 Java Development Kit

JDK คือชุดพัฒนาโปรแกรมภาษาจาวาของบริษัทซันไมโครซิสเต็มส์ ซึ่งแต่เดิมเรียกว่า JDK ทั้งนี้บริษัทซันไมโครซิสเต็มส์ได้เปลี่ยนชื่อมาเป็น Java 2 ตั้งแต่ JDK เวอร์ชัน 1.2 ชุดโปรแกรม JDK ที่ใช้ในปัจจุบันคือเวอร์ชัน 6 (สามารถ download ได้ที่ http://java.sun.com) จะประกอบไปด้วยโปรแกรมต่างๆ อาทิเช่น โปรแกรมคอมไพเลอร์ (`javac.exe`) โปรแกรมอินเตอร์พรีตเตอร์ (`java.exe`) และโปรแกรมดีบักเกอร์ เป็นต้น แต่ชุดโปรแกรม JDK จะเป็นเพียงเซ็ตย่อยของโปรแกรมประเภท Software Development Kit (SDK) ทั้งนี้เนื่องจาก JDK จะไม่มีโปรแกรมอีดิเตอร์สำหรับการเขียนซอร์ดโค้ดหรือรันโปรแกรม ดังนั้นผู้ใช้จะต้องใช้โปรแกรม SDK อื่นช่วยในการเขียนซอร์ดโค้ด

นอกจากนี้โปรแกรมที่อยู่ใน JDK จะเป็นโปรแกรมที่ต้องเรียกใช้งานผ่านทาง command line จึงไม่สะดวก

ในการใช้งาน แต่ก็มีข้อดีตรงที่ทำให้ผู้ที่เริ่มต้นในการพัฒนาโปรแกรมสามารถเข้าใจขั้นตอนต่างๆ ของการพัฒนาโปรแกรมโดยใช้ภาษาจาวาได้ดีขึ้น โดยไม่มุ่งเน้นการใช้เครื่องมือหรือคำสั่งต่างๆ ที่มีอยู่ในชุดพัฒนาโปรแกรมภาษาจาวาโปรแกรมอื่นๆ

นอกจากโปรแกรม JDK ที่พัฒนาโดยบริษัทซันไมโครซิสเต็มส์แล้ว บริษัทอื่นๆ ก็มีการพัฒนาโปรแกรม JDK ขึ้นมา โดยบางส่วนอาจใช้ซอร์โค้ดของทางซันไมโครซิสเต็มส์ ซึ่งโปรแกรม JDK เหล่านี้จะมีมาตรฐานคำสั่งที่เหมือนกันแต่อาจจะมีข้อแตกต่างในคุณลักษณะอื่นๆ เช่น คอมไพเลอร์หรือ Garbage Collection ตัวอย่างของโปรแกรม JDK อื่นๆ มีอาทิเช่น J9 ของบริษัทไอบีเอ็ม หรือ JRockit ของบริษัทออราเคิล

เกร็ดความรู้

NetBeans เป็นชุดพัฒนาโปรแกรมแบบเปิดซอร์สโค้ดของบริษัทซันไมโครซิสเต็มส์ซึ่งสามารถดาวน์โหลดมาใช้ได้จากเว็บไซต์ www.netbeans.org โปรแกรม NetBeans ถูกพัฒนาเริ่มต้นโดยกลุ่มนักศึกษาที่มหาวิทยาลัย Charles ที่ประเทศสาธารณรัฐเชค ซึ่งภายหลังก็จัดตั้งเป็นบริษัทเพื่อขายโปรแกรม NetBeans ในปี 1997 ก่อนที่จะถูกบริษัทซันไมโครซิสเต็มส์เข้ามาซื้อในปี 1999 แล้วจัดให้เป็นโปรแกรมแบบฟรีแวร์ ปัจจุบันโปรแกรม NetBeans เป็นเวอร์ชัน 6.8 (เดือนมีนาคม ปี ค.ศ. 2010)ที่สามารถใช้พัฒนาโปรแกรมจาวาได้ทุกแพลตฟอร์มทั้ง Java SE, Java ME และ Java EE รวมถึงการพัฒนา SOA และเขียนโปรแกรมภาษาอื่นๆ อาทิเช่น C/C++, PHP และ Ruby เป็นต้น

ส่วน SDK สำหรับการพัฒนาโปรแกรมภาษาจาวาที่มีอยู่ในปัจจุบันจะมีทั้งที่เป็นโปรแกรมแบบฟรีแวร์ (freeware) หรือโปรแกรมเพื่อการค้า (commercial software) อาทิเช่น

- Netbeans ของบริษัท Sun Microsystems (http://www.netbeans.org)
- Eclipse ของบริษัท IBM (http://eclipse.org)
- JDeveloper ของบริษัท Oracle (http://www.oracle.com)
- intelliJ IDEA ของบริษัท JetBrains (http://www.jetbrains.com/idea)

โปรแกรม SDK เหล่านี้จะมีเครื่องมือเพื่อช่วยในการสร้างโปรแกรมภาษาจาวาในรูปแบบต่างๆ ได้ง่ายขึ้นอาทิเช่น เครื่องมือช่วยในการออกแบบโปรแกรมกราฟิก

สำหรับชุดพัฒนาโปรแกรม JDK นั้นจำเป็นต้องมีโปรแกรมอิดีเตอร์ (Editor)เพื่อใช้ในการเขียนซอร์ดโค้ด

ของโปรแกรมภาษาจาวา ซึ่งสามารถที่จะใช้โปรแกรม Text Editor ทั่วไปได้เช่น โปรแกรม Notepad ในระบบปฏิบัติการ Windows แต่หากต้องการพัฒนาโปรแกรมให้เร็วขึ้น ผู้ใช้ควรเลือกใช้โปรแกรมอิดีเตอร์ที่ออกแบบมาเพื่อเขียนซอร์ดโค้ดของโปรแกรมภาษาจาวาโดยเฉพาะ ซึ่งโปรแกรมเหล่านี้จะทำงานคล้ายกับโปรแกรมแบบ SDK โดยผู้ใช้สามารถเรียกคำสั่งในการคอมไพล์ หรือการรันโปรแกรมที่พัฒนาขึ้นจากเมนูที่กำหนดไว้ได้ตัวอย่างโปรแกรมอิดีเตอร์แบบนี้คือ

- EditPlus ของบริษัท ES-Computing (http://www.editplus.com)
- JCreator ของบริษัท Xinox Software (http://www.jcreator.com)

1.7 จุดเด่นของภาษาจาวา

บริษัทซันไมโครซิสเต็มส์ได้ระบุถึงจุดเด่นของภาษาจาวาไว้ดังนี้

- ความง่าย (Simple) ภาษาจาวาเป็นภาษาที่ง่ายต่อการศึกษาและพัฒนาโปรแกรม ทั้งนี้เพราะภาษาจาวาพัฒนาโดยตัดข้อด้อยของภาษา C++ ออกไปอาทิเช่น เรื่องของการใช้ pointer
- ภาษาเชิงอ็อบเจกต์ (Object-Oriented) ภาษาจาวาเป็นภาษาคอมพิวเตอร์เชิงอ็อบเจกต์ที่สมบูรณ์โดยมีคุณลักษณะเด่นของโปรแกรมเชิงอ็อบเจกต์คือ การสืบทอด การห่อหุ้ม และการมีได้หลายรูปแบบ
- การกระจาย (Distributed) ภาษาจาวามีชุดคำสั่งที่เป็นแพคเก็จ (Package) ในการจัดการกับโปรโตคอล TCP/IP ทำให้สามารถพัฒนาโปรแกรมเชิงอ็อบเจกต์แบบกระจาย (Distributed Object) ผ่านระบบเครือข่ายอินเทอร์เน็ตได้ง่าย
- การป้องกันการผิดพลาด (Robust) ภาษาจาวาเป็นภาษาคอมพิวเตอร์ที่ออกแบบมาเพื่อให้โปรแกรมที่พัฒนาขึ้นมีความน่าเชื่อถือ โดยมีการตรวจสอบการผิดพลาดที่อาจเกิดขึ้นในขั้นตอนต่างๆ เช่น ขั้นตอนการคอมไพล์ และการรันโปรแกรม เป็นต้น
- ความปลอดภัย (Secure) ภาษาจาวาออกแบบมาเพื่อพัฒนาโปรแกรมบนระบบเครือข่าย และมีการกระจายการทำงานบนระบบเครือข่ายอินเทอร์เน็ต ดังนั้นจึงต้องสร้างระบบป้องกันความปลอดภัยจากไวรัสคอมพิวเตอร์ และการแก้ไขโปรแกรมจากภายนอก
- สถาปัตยกรรมกลาง (Architecture Neutral) โปรแกรมภาษาจาวาจะคอมไพล์ได้โปรแกรมไบท์โค้ด (byte code) ซึ่งสามารถทำงานบนสถาปัตยกรรมคอมพิวเตอร์ที่มีหน่วยประมวลผลกลางและระบบปฏิบัติการต่างๆ ได้
- เคลื่อนย้ายง่าย (Portable) ข้อกำหนดของภาษาจาวาจะไม่ขึ้นอยู่กับระบบคอมพิวเตอร์ใดโดยเฉพาะ ดังนั้นโปรแกรมภาษาจาวาจึงสามารถประมวลผลได้กับระบบคอมพิวเตอร์ทุกประเภท

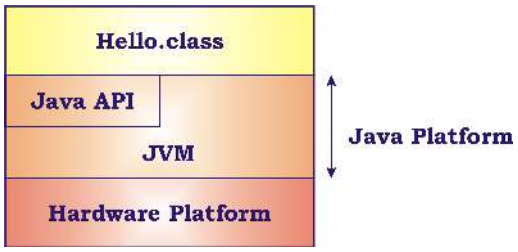
- อินเตอร์พรีต (Interpreted) ภาษาจาวาจะใช้อินเตอร์พรีตเตอร์ในการแปลโปรแกรมไบท์โค้ดให้เป็นภาษาเครื่อง ดังนั้นจึงทำให้ขบวนการการพัฒนาโปรแกรมเป็นไปได้อย่างรวดเร็ว เช่นเดียวกับภาษาคอมพิวเตอร์อื่นๆ ที่ใช้อินเตอร์พรีตเตอร์
- ประสิทธิภาพสูง (High Performance) โดยปกติโปรแกรมอินเตอร์พรีตเตอร์ที่ทำหน้าที่แปลโปรแกรมไบท์โค้ดจะทำงานช้า แต่เทคโนโลยีจาวาได้พัฒนาให้มีการแปลโปรแกรมไบท์โค้ด ในขั้นตอนการรันโปรแกรมให้เป็นภาษาเครื่องแบบทันที ทันใด (Just In Time) ที่ทำงานได้รวดเร็วเทียบเท่ากับคอมไพเลอร์ เพื่อได้โปรแกรมจาวาที่มีประสิทธิภาพในการประมวลผลสูง
- มัลติเธรด (Multithreaded) ภาษาจาวามีความสามารถที่จะประมวลผลหลายๆ งานได้พร้อมกัน
- พลวัต (Dynamic) ภาษาจาวาออกแบบมาเพื่อที่จะให้ปรับเปลี่ยนเพิ่มเติมไลบารี (library) ต่างๆ ได้ง่าย ซึ่งแตกต่างจากภาษาซี หรือ C++

1.8 แพลตฟอร์มของเทคโนโลยีจาวา

แพลตฟอร์ม (Platform) คือฮาร์ดแวร์และสภาวะแวดล้อมทางซอฟต์แวร์ (Software Environment) ที่โปรแกรมจะใช้ในการประมวลผลโดยทั่วไป

แพลตฟอร์มจะนิยามโดยพิจารณาจากองค์ประกอบของฮาร์ดแวร์และระบบปฏิบัติการที่ใช้เช่น แพลตฟอร์มของระบบปฏิบัติการ Windows XP บนเครื่องไมโครคอมพิวเตอร์ Pentium IV แต่นิยามของแพลตฟอร์มสำหรับเทคโนโลยีจาวาจะแตกต่างจากนิยามที่ใช้กันทั่วไป ทั้งนี้จะพิจารณาจากองค์ประกอบของซอฟต์แวร์เท่านั้นดังแสดงในรูปที่ 1.7 ซึ่งแพลตฟอร์มของเทคโนโลยีจาวาจะประกอบด้วย

- Java Virtual Machine (JVM)
- Java Application Programming Interface (Java API)



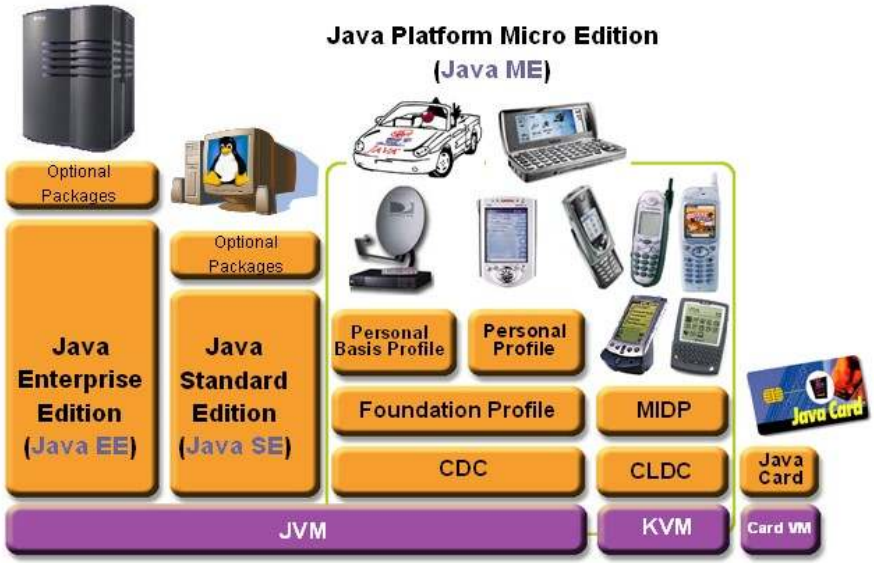
รูปที่ 1.7 แพลตฟอร์มของเทคโนโลยีจาวา

โดยที่ JVM จะเป็นส่วนฐานของแพลตฟอร์มเพื่อใช้ในการติดต่อกับแพลตฟอร์มส่วนที่เป็นฮาร์ดแวร์ ส่วน Java API เป็นชุดแพคเก็จที่รวบรวมคลาสและอินเตอร์เฟสต่างๆ ที่เป็นประโยชน์ต่อการใช้งานของโปรแกรมที่พัฒนาโดยภาษาจาวาเช่น คลาสที่เกี่ยวข้องกับส่วนต่อประสานกราฟิกกับผู้ใช้ (Graphical User Interface) เป็นต้น

บริษัทซันไมโครซิสเต็มส์ได้กำหนดแพลตฟอร์มของเทคโนโลยีจาวาให้มีสามรูปแบบคือ

- Java Platform, Standard Edition (Java SE)
- Java Platform, Enterprise Edition (Java EE)
- Java Platform, Micro Edition (Java ME)

โดยแต่ละแพลตฟอร์มจะมี JVM และ API ที่แตกต่างกัน และมุ่งเน้นที่จะใช้ในการพัฒนาโปรแกรมภาษาจาวาสำหรับระบบคอมพิวเตอร์หรืออุปกรณ์ที่แตกต่างกัน ดังแสดงในรูปที่ 1.8



รูปที่ 1.8 แพลตฟอร์มของ Java 2 ในรูปแบบต่างๆ

โปรแกรมจาวาที่ใช้บนเครื่องคอมพิวเตอร์โดยทั่วไป จะใช้แพลตฟอร์มที่เป็น Java SE ส่วนแพลตฟอร์ม Java EE มุ่งเน้นการพัฒนาโปรแกรมภาษาจาวาที่ใช้งานบนเครื่องแม่ข่าย (Server) สำหรับระบบงานในองค์กร (Enterprise) และแพลตฟอร์ม Java ME ใช้ในการพัฒนาโปรแกรมจาวาบนอุปกรณ์ขนาดเล็ก เช่น โทรศัพท์เคลื่อนที่หรือ PDA เป็นต้น

1.8.1 Java Platform, Standard Edition

แพลตฟอร์ม Java SE เป็นแพลตฟอร์มจาวามาตรฐานที่ใช้ในการพัฒนาโปรแกรมภาษาจาวาเพื่อใช้งานกับเครื่องคอมพิวเตอร์ทั่วไป Virtual Machine ที่กำหนดไว้ในแพลตฟอร์มนี้คือ JVM และมี Java API ที่กำหนดไว้ต่างๆ อาทิเช่น `java.lang, java.util, java.io` และ `java.net`

แพลตฟอร์ม Java SE ใช้ในการพัฒนาโปรแกรมจาวาที่ใช้ในเครื่องคอมพิวเตอร์ลูกข่าย โดยแบ่งออกเป็น 2 ประเภท คือ

- โปรแกรมจาวาประยุกต์ (Java Application) คือ โปรแกรมที่ใช้งานทั่วไป
- โปรแกรมจาวาแอปเพล็ต (Java Applet) คือโปรแกรมภาษาจาวาที่รันบนเว็บเบราว์เซอร์

เวอร์ชันล่าสุดของ Java SE คือเวอร์ชัน 7 (เดือนมีนาคม ค.ศ. 2010) ซึ่งผู้ใช้สามารถ download ได้ฟรีทางเว็บไซต์ที่ชื่อ java.sun.com

1.8.2 Java Platform, Enterprise Edition

แพลตฟอร์มจาวารุ่นนี้ มีจุดประสงค์เพื่อใช้พัฒนาโปรแกรมภาษาจาวาสำหรับเครื่องแม่ข่าย โดยมุ่งเน้นการพัฒนาโปรแกรมที่เป็น N-tier สำหรับงานในองค์กร โดยใช้โปรแกรม Application Server หรือโปรแกรม Web Server ที่เรียกว่าโปรแกรมจาวาแบบ Servlet โปรแกรม Application Server เป็นโปรแกรมที่ช่วยในการพัฒนาโปรแกรมภาษาจาวาที่สามารถเปลี่ยนแปลง แก้ไข และนำกลับมาใช้ใหม่ได้ง่าย โดยพัฒนาเป็นส่วนประกอบอ็อบเจกต์ (Object Component) ที่เรียกว่า Enterprise Java Bean (EJB) โปรแกรม Web Server หรือ Application Server ที่ใช้กับภาษาจาวาในปัจจุบันมีทั้งโปรแกรมที่เป็นแบบเปิดซอร์สโค้ด อาทิเช่น Tomcat, GlassFish หรือ JBoss และโปรแกรมเพื่อการค้า เช่น WebLogic หรือ WebSphere สำหรับ เวอร์ชันล่าสุดของ Java EE คือ Java EE6 (เดือนมีนาคม ค.ศ. 2010)

<u>เกร็ดความรู้</u>

การพัฒนาโปรแกมบน Web Server จะต้องมีเครื่องลูกข่ายในการแสดงผลผ่านทางอินเทอร์เน็ต ซึ่งโดยมากจะใช้โปรแกรมเว็บเบราว์เซอร์ในการแสดงผล การพัฒนาโปรแกรม Web Server จะต้องมีการติดตั้งเครื่องแม่ข่ายและโปรแกรม Web Server ที่อาจใช้เครื่องคอมพิวเตอร์แม่ข่ายที่มีประสิทธิภาพสูงและใช้ระบบปฏิบัติการสำหรับแม่ข่ายเช่น Windows Server, Linux และ Solaris เป็นต้น ภาษาที่นิยมใช้ในการพัฒนาโปรแกรมบน Web Server ที่นิยมในปัจจุบันคือ Java, PHP และ ASP.NET

สำหรับโปรแกม Web Server ที่เป็นที่นิยมตัวหนึ่งในภาษาจาวาคือ Tomcat ซึ่งเป็นโปรแกรมแบบเปิดเผยซอร์สโค้ด ที่พัฒนาโดยองค์กร Apache Software Foundation ซึ่งสามารถทำงานได้บนระบบปฏิบัติการต่างๆ อาทิเช่น Linux และ Windows โดยเวอร์ชันปัจจุบันคือ Tomcat 6.0 (เดือนมีนาคม ค.ศ. 2010)

1.8.3 Java Platform, Micro Edition

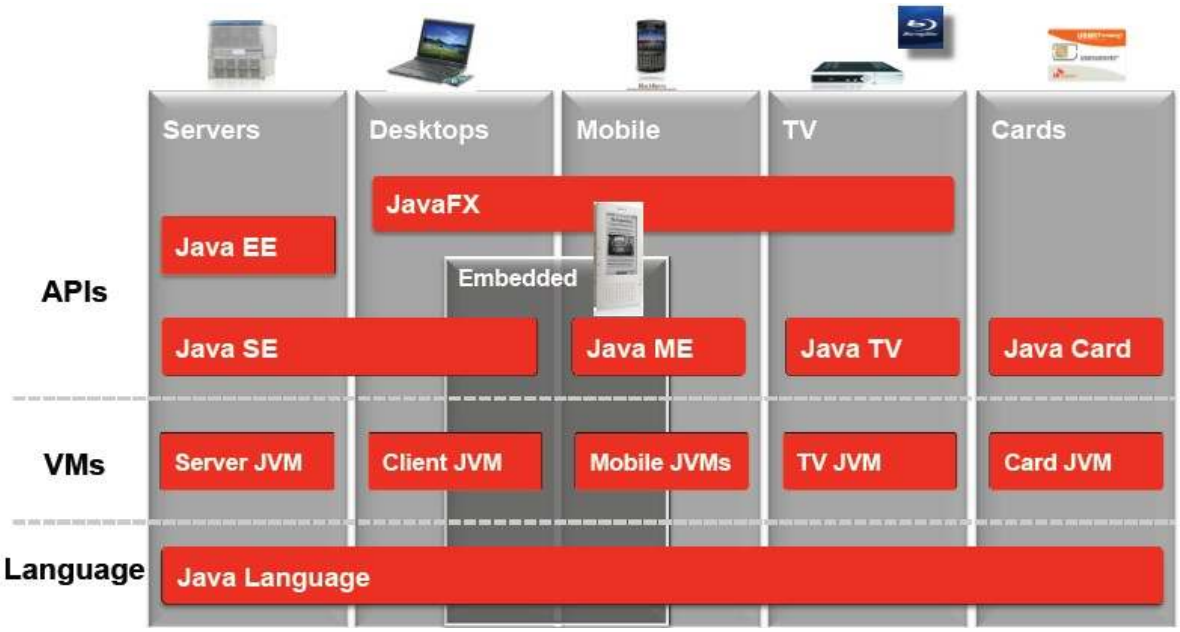
แพลตฟอร์ม Java ME เป็นแพลตฟอร์มจาวาที่ใช้ในการพัฒนาโปรแกรมภาษาจาวา เพื่อใช้งานกับอุปกรณ์ไฟฟ้า และเครื่องมือสื่อสารต่างๆ อาทิเช่น โทรศัพท์มือถือ หรือ PDA (Personal Digital Assistant) เป็นต้น แพลตฟอร์ม Java ME จะใช้ Virtual Machine ที่มีขนาดเล็กกว่า JVM ที่ใช้ในเครื่องคอมพิวเตอร์ทั่วไป ทั้งนี้เนื่องจากอุปกรณ์เหล่านั้นจะมีหน่วยความจำน้อยกว่า และจะมีหน่วยประมวลผลกลางที่มีความเร็วช้ากว่า

แต่เนื่องจากปัจจุบันมีการออกโทรศัพท์มือถือรุ่นต่างๆออกมามากโดยใช้ระบบปฏิบัติการที่ต่างกันเช่น ค่ายโนเกียจะใช้ระบบ Symbian ค่าย Apple จะใช้ iPhone และค่าย Google จะใช้ Andriod จึงทำให้แพลตฟอร์มในการพัฒนาโปรแกรมจาวาบนโทรศัพท์มือถือมีความหลากหลายมากขึ้น กล่าวคือนอกจากแพลตฟอร์ Java ME ที่สามารถทำงานบนโทรศัพท์ส่วนใหญ่รวมทั้งระบบปฏิบัติการ Symbian แล้ว ยังมีแพลตฟอร์ม Android ที่พัฒนาโปรแกรมจาวาโดยใช้คำสั่งมาตรฐานที่อยู่ใน Java SE และแพลตฟอร์ม Java FX ที่พัฒนาล่าสุดโดยบริษัทซันไมโครซิสเต็มส์

<u>เกร็ดความรู้</u>

Android เป็นแพลตฟอร์มสำหรับโทรศัพท์มือถือที่พัฒนาโดยกลุ่ม Open Handset Alliance ที่ประกอบด้วยบริษัทต่างๆ อาทิเช่น Google, HTC, Intel และ Samsung เป็นต้น โดยกำหนดแพลตฟอร์มที่ใช้ระบบปฏิบัติการ Linux บนเครื่องโทรศัพท์มือถือและคำสั่งในการพัฒนาโปรแกรมโดยใช้ภาษาจาวา โทรศัพท์มือถือที่ใช้มาตรฐาน Android รุ่นแรกคือ T1-Mobile G1 โดยบริษัท HTC เมื่อปลายปีค.ศ. 2008 และได้มีการนำเข้ามาจำหน่ายในประเทศไทยแล้ว

ภายหลังจากที่ทางบริษัทออราเคิลได้เข้ามาซื้อกิจการของบริษัทซันไมโครซิสเต็มส์ ทางบริษัทก็ได้ประกาศสนับสนุนให้เทคโนโลยีจาวาเป็นแบบระบบเปิด ที่สามารถรันได้หลากหลายอุปกรณ์เช่นเดิม (Complete. Open. Integrated) โดยกำหนดให้มี API และ JVM ที่หลากหลายเช่นเดิมดังแสดงในรูปที่ 1.9



รูปที่ 1.9 แพลตฟอร์มล่าสุดของ Java Technology

1.9 โปรแกรมภาษาจาวา

โปรแกรมจาวาที่อยู่บนแพลตฟอร์ม Java SE สามารถพัฒนาได้สองรูปแบบคือ

1. โปรแกรมจาวาประยุกต์ (Java Application) คือโปรแกรมประยุกต์ใช้งานทั่วไป โดยโปรแกรมแบบนี้จะทำงานภายใต้โปรแกรมอินเตอร์พรีตเตอร์โดยตรง ซึ่งโปรแกรมลักษณะนี้เป็น stand-alone
2. โปรแกรมจาวาแอปเพล็ต (Java Applet) คือโปรแกรมภาษาจาวาที่จะทำงานภายใต้โปรแกรมเว็บเบราว์เซอร์ที่มี JVM อยู่

1.9.1 การสร้างโปรแกรมจาวาประยุกต์

การสร้างโปรแกรมจาวาประยุกต์จะมีขั้นตอนดังแสดงในรูปที่ 1.10 ซึ่งสามารถอธิบายพอสังเขปได้ดังนี้

1. ขั้นตอนแรกจะเป็นการเขียนซอร์ดโค้ดโดยใช้โปรแกรมอิดีเตอร์ใดๆ โปรแกรมที่ 1.1 เป็นตัวอย่างโปรแกรมเพื่อพิมพ์ข้อความ Hello World ออกทางจอภาพ โปรแกรมนี้ใช้ชื่อคลาสว่า `HelloWorld` ซึ่งผู้เขียนโปรแกรมจะต้องเขียนซอร์ดโค้ดแล้วเก็บลงในไฟล์ที่มีชื่อเดียวกับชื่อคลาส โดยมี extension เป็น `.java` กล่าวคือโปรแกรมนี้จะเก็บไว้ในชื่อ `HelloWorld.java` โปรแกรมนี้จะมีเมธอด `main()` เป็นจุดเริ่มต้น

2. ขั้นตอนที่สองจะเป็นการคอมไพล์โปรแกรม เพื่อแปลโปรแกรมซอร์ดโค้ดให้อยู่ในรูปของโปรแกรมไบท์โค้ด ชุดพัฒนาโปรแกรม JDK จะมีโปรแกรมชื่อ `javac.exe` ที่ทำหน้าที่เป็นคอมไพล์เลอร์ ซึ่งในตัวอย่างนี้สามารถทำการคอมไพล์โปรแกรมโดยใช้คำสั่งต่อไปนี้

```
javac HelloWorld.java
```

ผลลัพธ์ที่ได้จากการคอมไพล์โปรแกรมนี้คือ โปรแกรมไบท์โค้ดที่ชื่อ `HelloWorld.class` ในกรณีที่โปรแกรมซอร์ดโค้ดมีข้อผิดพลาด คอมไพเลอร์จะระบุข้อผิดพลาดที่เกิดขึ้น เพื่อให้ผู้เขียนโปรแกรมทำการแก้ไขโปรแกรมซอร์ดโค้ดก่อนที่จะทำการคอมไพล์โปรแกรมใหม่

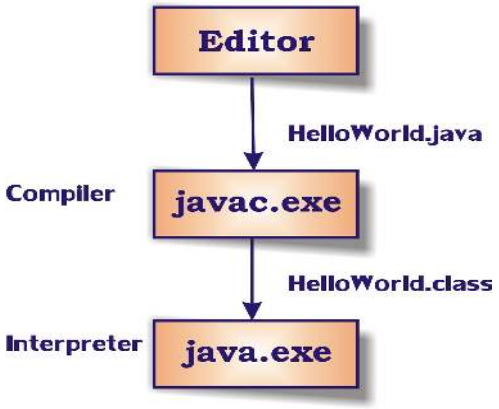
3. ขั้นตอนการประมวลผลหรือรันโปรแกรม เป็นการเรียกใช้อินเตอร์พรีตเตอร์เพื่อแปลโปรแกรมไบท์โค้ด ชุดพัฒนาโปรแกรม Java 2 SDK จะมีโปรแกรมชื่อ `java.exe` ที่ทำหน้าที่เป็นโปรแกรมอินเตอร์-พรีตเตอร์ ซึ่งในตัวอย่างนี้จะสามารถรันโปรแกรมไบท์โค้ดได้โดยใช้คำสั่งต่อไปนี้

```
java HelloWorld
```

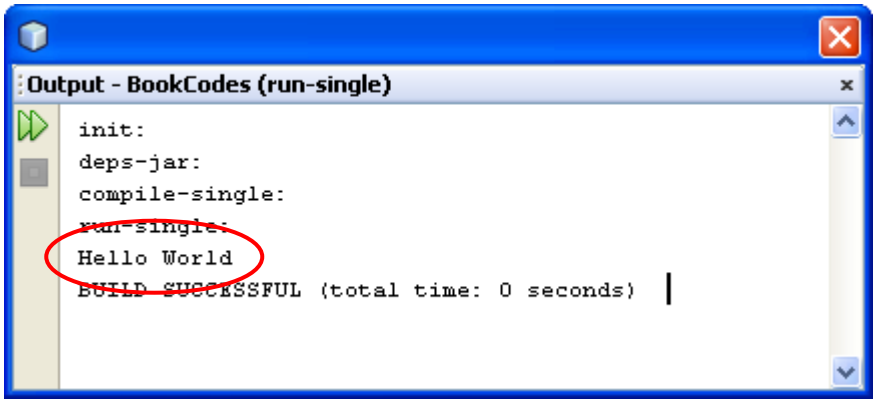
โดยโปรแกรมจะให้ผลลัพธ์ดังแสดงในรูปที่ 1.11

<u>โปรแกรมที่ 1.1</u> <u>เป็นโปรแกรมเพื่อพิมพ์ข้อความ Hello World</u>

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



รูปที่ 1.10 ขั้นตอนการสร้างโปรแกรมจาวาประยุกต์



รูปที่ 1.11 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 1.1

1.9.2 การสร้างโปรแกรมจาวาแอปเพล็ต

การสร้างโปรแกรมจาวาแอปเพล็ต จะมีขั้นตอนคล้ายกับการสร้างโปรแกรมจาวาประยุกต์ แต่โปรแกรมจาวาแอปเพล็ตจะประกอบด้วยไฟล์ที่เขียนขึ้นสองไฟล์คือ

- โปรแกรมซอร์ดโค้ด (`.java`)
- โปรแกรมเว็บเพจ (`.html`)

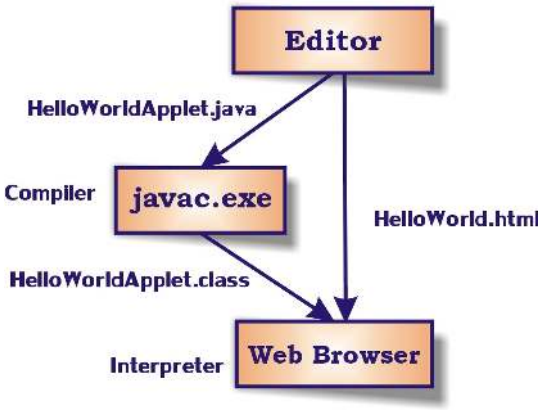
โปรแกรมที่ 1.2 เป็นตัวอย่างของโปรแกรมจาวาแอปเพล็ตเพื่อแสดงข้อความ Hello World โปรแกรมนี้จะแตกต่างจากโปรแกรมจาวาประยุกต์ตรงที่ไม่มีเมธอด `main()` และจะไม่สามารถทำงานตามลำพังได้ แต่จะต้องเรียกใช้โดยผ่านโปรแกรมภาษา HTML ตัวอย่างเช่น โปรแกรมที่ 1.3 เป็นโปรแกรมภาษา HTML ที่เรียกใช้โปรแกรม `HelloWorldApplet.class` เพื่อประมวลผลภายใต้โปรแกรมเว็บเบราว์เซอร์ที่มี JVM อยู่ ขั้นตอนการสร้างโปรแกรมนี้มีดังนี้

<u>โปรแกรมที่ 1.2</u> <u>โปรแกรมจาวาแอปเพล็ตเพื่อพิมพ์ข้อความ Hello World</u>

```
import java.awt.*;
import javax.swing.*;

public class HelloWorldApplet extends JApplet {
      public void paint(Graphics g) {
            g.drawString("Hello World",20,20);
      }
}
```

โปรแกรมที่ 1.3 โปรแกรมภาษา HTML ที่เรียกใช้ `HelloWorldApplet.class`

```
<HTML>
      <HEAD>
            <TITLE>HelloWorld Example</TITLE>
      </HEAD>
      <BODY>
            <APPLET CODE="HelloWorldApplet.class"
                        WIDTH="300"  HEIGHT="300">
            </APPLET>
      </BODY>
</HTML>
```

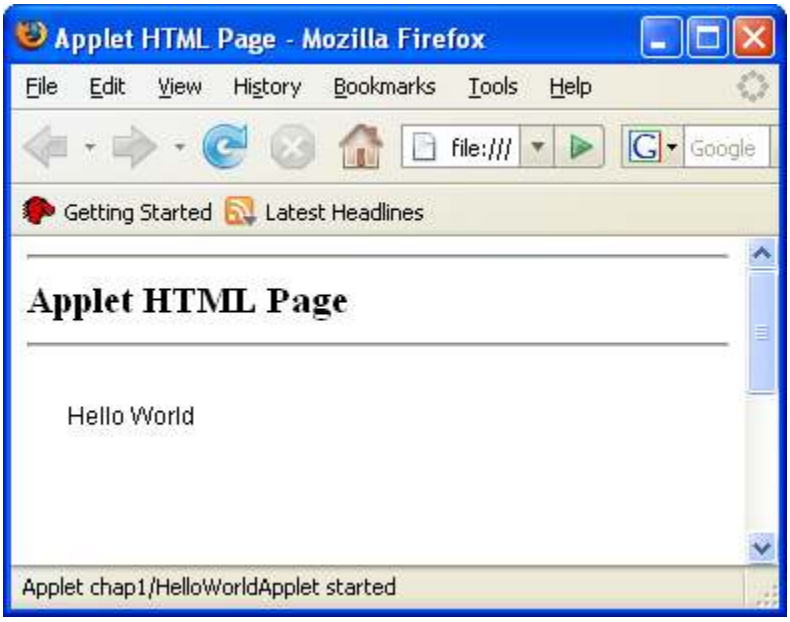


รูปที่ 1.12 ขั้นตอนการสร้างโปรแกรมจาวาแอปเพล็ต

1. เขียนโปรแกรมซอร์ดโค้ด (`HelloWorldApplet.java`) และโปรแกรม เว็บเพจ (`HelloWord.html`) โดยใช้โปรแกรมอิดีเตอร์ใดๆ
2. คอมไพล์โปรแกรม `HelloWorldApplet.java` โดยใช้คอมไพเลอร์ดังนี้

   `javac HelloWorldApplet.java`

   ตัวอย่างนี้จะได้โปรแกรมไบท์โค้ดที่ชื่อ `HelloWorldApplet.class`
3. ใช้โปรแกรมเว็บเบราว์เซอร์ใดๆ เช่น Internet Explorer หรือ Netscape เปิดโปรแกรมเว็บเพจที่ชื่อ `HelloWorld.html` ซึ่งจะเรียกโปรแกรมไบท์โค้ดที่ชื่อ `HelloWorldApplet.class` โดยอัตโนมัติ และจะมีผลรันดังรูปที่ 1.13



รูปที่ 1.13 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 1.3 บนเว็บเบราว์เซอร์

1.10 คู่มือ Java API

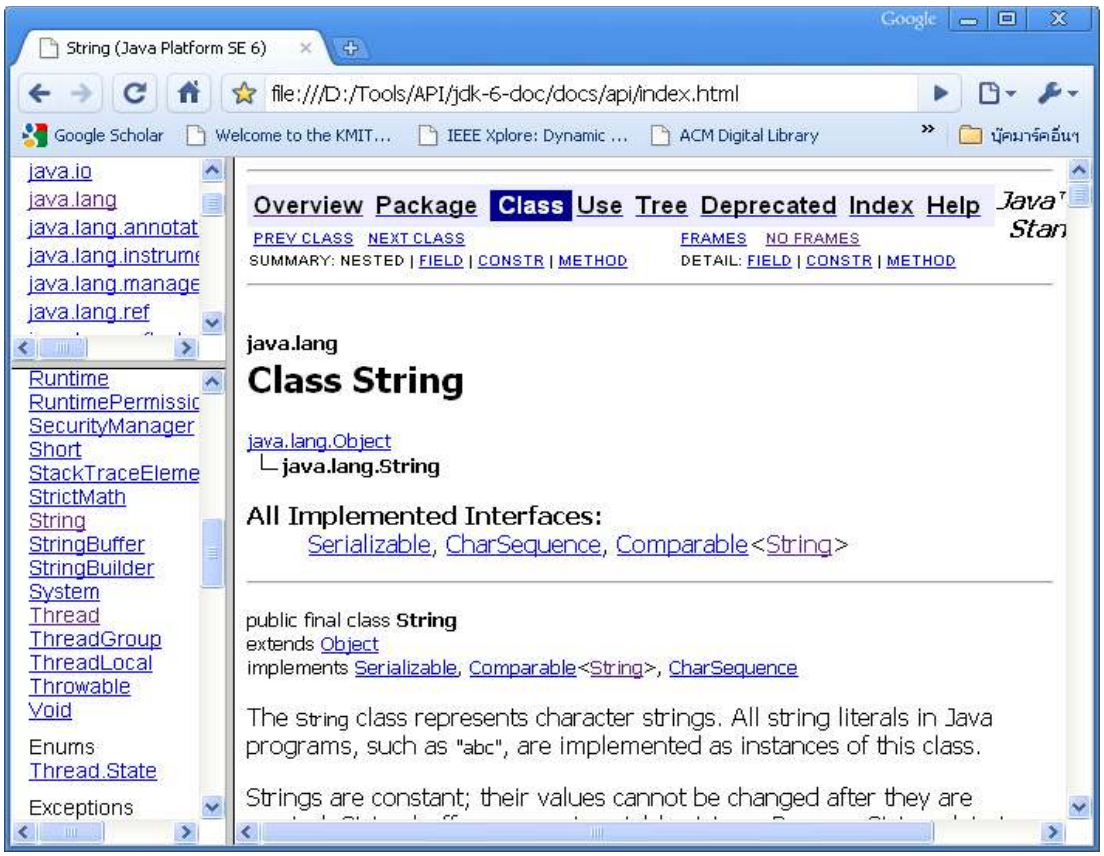
Java API เป็นข้อกำหนดที่ว่าด้วยคลาสและอินเตอร์เฟสต่างๆ ที่กำหนดไว้ในแพคเก็จมาตรฐานของภาษาจาวา แพคเก็จจะเป็นที่รวบรวมคลาสและอินเตอร์เฟสต่างๆ ที่มีหน้าที่การทำงานคล้ายกันไว้ในที่เดียวกัน Java API มีแพค-เก็จที่สำคัญอยู่หลายแพคเก็จ อาทิเช่น `java.lang`, `java.util`, `java.awt` และ `java.io` เป็นต้น

เนื่องจาก Java API มีแพคเก็จและคลาสต่างๆ อยู่เป็นจำนวนมาก จึงเป็นเรื่องยากในการที่จะจดจำคลาสและเมธอดต่างๆ ทั้งหมดที่มีอยู่ได้ ดังนั้นชุดพัฒนาโปรแกรม JDK จะมีไฟล์คู่มือ Java API ที่อยู่ในรูปของไฟล์ HTML ดังแสดงในรูปที่ 1.14 ซึ่งคู่มือ Java API จะประกอบไปด้วยเฟรมสามเฟรมคือ ส่วนที่เป็นชื่อแพคเก็จ (มุมบนซ้าย) ส่วนที่เป็นชื่อคลาสและอินเตอร์เฟสต่างๆ ของแพคเก็จ (มุมล่างซ้าย) และส่วนที่เป็นรายละเอียดของคลาสหรืออินเตอร์เฟส (ตรงกลาง) ซึ่งรูปที่ 1.14 แสดงคู่มือ Java API ที่ระบุคลาสและอินเตอร์เฟสต่างๆ ในแพคเก็จ `java.lang` และเฟรมหลักตรงกลางแสดงรายละเอียดของคลาส `String`

คู่มือ Java API จะแสดงรายละเอียดต่างๆ ของคลาสหรืออินเตอร์เฟสดังนี้

- ลำดับการสืบทอดของคลาส
- คำอธิบายเกี่ยวกับคลาสและจุดประสงค์ทั่วไป
- รายชื่อคุณลักษณะต่างๆ ของคลาส

- รายชื่อเมธอดต่างๆ ของคลาส
- รายชื่อ Constructor ต่างๆ ของคลาส
- คำอธิบายรายละเอียดของคุณลักษณะแต่ละตัวของคลาส
- คำอธิบายรายละเอียดของเมธอดแต่ละตัวของคลาส
- คำอธิบายรายละเอียดของ Constructor แต่ละตัวของคลาส



รูปที่ 1.14 คู่มือ Java API

สรุปเนื้อหาของบท

- ระบบคอมพิวเตอร์ประกอบไปด้วยส่วนหลักสามส่วนคือ ฮาร์ดแวร์ ระบบปฏิบัติการ และโปรแกรมประยุกต์
- ภาษาคอมพิวเตอร์แบ่งออกเป็นสามประเภทคือ ภาษาเครื่อง ภาษาแอสเซมบลี และภาษาระดับสูง
- ภาษาระดับสูงแบ่งออกเป็นสองประเภทคือ ภาษาเชิงกระบวนการ และภาษาเชิงอ็อบเจกต์

- ตัวแปลภาษาแบ่งออกเป็นสองแบบคือ คอมไพเลอร์ และอินเตอร์พรีตเตอร์
- ภาษาจาวาเป็นภาษาเชิงอ็อบเจกต์ที่ใช้ทั้งตัวแปลภาษาแบบคอมไพเลอร์และอินเตอร์พรีตเตอร์ในการคอมไพล์และรันโปรแกรม
- คอมไพเลอร์ของภาษาจาวาจะทำหน้าที่แปลโปรแกรมภาษาจาวาให้เป็นโปรแกรมไบท์โค้ด และจะใช้จาวาอินเตอร์พรีตเตอร์ (JVM) ในการแปลโปรแกรมไบท์โค้ดให้เป็นภาษาเครื่อง
- โปรแกรมภาษาจาวาสามารถทำงานข้ามแพลตฟอร์มได้ ถ้าระบบคอมพิวเตอร์นั้นมี JVM อยู่
- แพลตฟอร์มของ Java ประกอบไปด้วย JVM และ Java API ซึ่ง Java มีแพลตฟอร์ม 3 แบบคือ Java SE, Java EE และ Java ME
- ชุดพัฒนาโปรแกรมภาษาจาวา JDK ประกอบไปด้วยโปรแกรมต่างๆ ที่สำคัญคือ โปรแกรมคอมไพเลอร์ (javac.exe) และโปรแกรมอินเตอร์พรีตเตอร์ (java.exe)
- โปรแกรมจาวาที่อยู่บนแพลตฟอร์ม Java SE สามารถพัฒนาได้สองรูปแบบคือโปรแกรมจาวาประยุกต์ซึ่งจะทำงานภายใต้ JVM โดยตรง และโปรแกรมจาวาแอปเพล็ตซึ่งจะทำงานภายใต้เว็บเบราว์เซอร์ที่มี JVM
- คู่มือ Java API จะช่วยในการค้นหารายละเอียดของแพคเก็จและคลาสต่างๆ ที่มีอยู่ในชุดพัฒนาโปรแกรม JDK

บทที่ 2 พื้นฐานโปรแกรมภาษาจาวา

เนื้อหาในบทนี้เป็นการแนะนำไวยากรณ์ของภาษาจาวา ซึ่งประกอบไปด้วยสัญลักษณ์หรือคำต่างๆ ที่ใช้ในภาษาจาวา ชนิดข้อมูลแบบพื้นฐาน และข้อมูลค่าคงที่ การประกาศและเรียกใช้คำสั่งกำหนดค่าตัวแปร นิพจน์และตัวดำเนินการประเภทต่างๆ วิธีการแปลงชนิดข้อมูล ชนิดข้อมูลแบบอ้างอิง และแนะนำคำสั่งที่ใช้ในการรับข้อมูลเข้าและแสดงผล ซึ่งเนื้อหาในบทนี้จะเป็นการแนะนำการเขียนโปรแกรมเบื้องต้นเพื่อกำหนดค่าข้อมูลต่างๆ และแสดงผลลัพธ์ของการประมวลผล

2.1 การเขียนโปรแกรมภาษาจาวาเชิงอ็อบเจกต์

ภาษาจาวาเป็นภาษาคอมพิวเตอร์ ที่ใช้หลักการการพัฒนาโปรแกรมเชิงอ็อบเจกต์ที่เรียกย่อว่าเป็นภาษาคอมพิวเตอร์แบบ OOP การพัฒนาโปรแกรมเชิงอ็อบเจกต์จะเป็นขบวนการการวิเคราะห์ปัญหาโดยการจำลองปัญหาว่าประกอบไปด้วยอ็อบเจกต์ใดบ้าง แล้วจึงเขียนให้อยู่ในรูปแบบของโปรแกรมคอมพิวเตอร์ โปรแกรมเชิงอ็อบเจกต์จะมีคำนิยามที่สำคัญสองคำคือ อ็อบเจกต์ (Object) และคลาส (Class)

คลาสเปรียบเสมือนพิมพ์เขียวของอ็อบเจกต์ อ็อบเจกต์ที่ถูกสร้างมาจากคลาสบางครั้งจะเรียกว่าเป็น instance ของคลาส ซึ่งอ็อบเจกต์ใดๆ จะต้องเป็น instance ของคลาสใดคลาสหนึ่ง การเขียนโปรแกรมเชิงอ็อบเจกต์ ต้องมีการกำหนดนิยามของคลาสก่อนที่จะสามารถสร้างอ็อบเจกต์ (หรือ instance) ของคลาสได้ ซึ่งคลาสหนึ่งคลาสสามารถที่จะสร้างอ็อบเจกต์ได้หลายอ็อบเจกต์

ภายในคลาสจะประกอบด้วยคุณลักษณะ (Attribute) ซึ่งก็คือข้อมูลที่เก็บอยู่ในอ็อบเจกต์ โดยจะแบ่งออกเป็นสองประเภท คือตัวแปร (Variable) และค่าคงที่ (Constant) คุณลักษณะที่เป็นตัวแปรจะสามารถเปลี่ยนค่าได้ ส่วนคุณลักษณะที่เป็นค่าคงที่จะไม่สามารถเปลี่ยนค่าได้ และเมธอด (Method) ซึ่งก็คือวิธีการหรือการกระทำที่นิยามอยู่ในคลาส อ็อบเจกต์ที่สร้างมาจากคลาสนั้น อาจมีค่าคุณลักษณะต่างกันแต่จะมีการเรียกใช้เมธอดที่เหมือนกัน ตัวอย่างของการนิยามคลาสที่ชื่อ `Student` ซึ่งมีคุณลักษณชื่อ `name` และ เมธอดที่ชื่อ `sayHi` มีดังนี้

```
public class Student {
      public String name;
      public void sayHi() {
            System.out.println("Hello");
      }
}
```

จากคำสั่งการประกาศจะเห็นว่าคำสั่งในภาษาจาวาจะถูกกำหนดให้อยู่บล็อกของ {} เช่นคุณลักษณะและเมธอด

ของคลาสก็จะอยู่ในบล็อก {} ทำนองเดียวกันคำสั่งต่างๆที่อยู่ในเมธอดก็จะอยู่ภายในบล็อก {} โดยทั่วไปคลาสแต่ละคลาสจะมีคุณลักษณะและเมธอดได้ไม่จำกัดจำนวน

โดยทั่วไปภาษาจาวาจะกำหนดให้ชื่อไฟล์และชื่อคลาสต้องเป็นชื่อเดียวกัน และเป็นแบบ Case-sensitive กล่าวคือจะต้องใช้อักษรตัวเล็กตัวใหญ่อย่างถูกต้อง กรณีนี้คลาสที่ชื่อ `Student` จะเขียนโปรแกรมในไฟล์ที่ชื่อ Student.java

คลาสจะถูกเรียกใช้งานได้เมื่อมีการสร้างอ็อบเจกต์ของคลาส โดยเรียกใช้คำสั่ง `new` จะต้องมีการกำหนดชื่อของอ็อบเจกต์นั้น ตัวอย่างเช่นคำสั่งข้างล่างนี้เป็นการสร้างอ็อบเจกต์ของคลาส `Student` ที่ชื่อ `s1`

```
Student s1 = new Student();
```

โปรแกรมที่รันภาษาจาวาอาจมีหลายคลาส ซึ่งคลาสบางคลาสเป็นคลาสที่ถูกเรียกใช้มากจากคลาสอื่นเช่น คลาสที่ชื่อ `String` หรือ `System` แต่สำหรับโปรแกรมประเภทจาวาประยุกต์ โปรแกรม JVM จะเรียกคลาสที่มีเมธอดที่ชื่อ `main` ซึ่งมีรูปแบบดังนี้ มารันเป็นเมธอดแรก

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

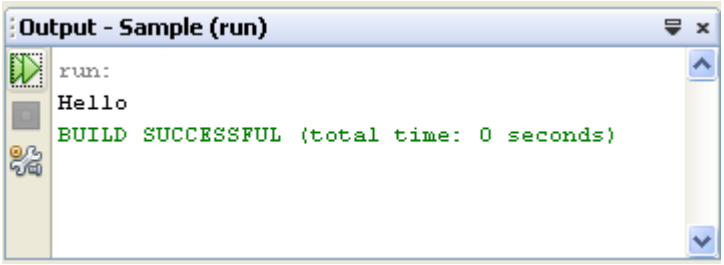
โปรแกรมที่ 2.2 แสดงตัวอย่างของคลาสที่ชื่อ `Main` ที่มีเมธอดที่ชื่อ `main` ซึ่งมีคำสั่งในการสร้างอ็อบเจกต์ชนิด `Student` แล้วเรียกใช้เมธอด `sayHi` ผลลัพธ์ของโปรแกรมนี้จะเป็นการพิมพ์ข้อความว่า `Hello` ดังแสดงในรูปที่ 2.1

<u>โปรแกรมที่ 2.1</u> <u>ตัวอย่างการเขียนโปรแกรมที่เรียกใช้คลาส `Student`</u>

```
public class Main {

      public static void main(String args[]) {
          Student s1 = new Student();
          s1.sayHi();
      }

}
```

การเขียนโปรแกรมในบทนี้ จะเน้นการเขียนแบบเชิงอ็อบเจกต์โดยโปรแกรมส่วนใหญ่จะเป็นการสร้างคลาส

ขึ้นมาหนึ่งคลาสเช่น Student.java แล้วจะมีอีกหนึ่งคลาสเช่น Main.java เพื่อมาสร้างอ็อบเจกต์ของคลาสแรกแล้วเรียกใช้เมธอดที่อยู่ภายใน



รูปที่ 2.1 ผลลัพธ์จากการรันโปรแกรมที่ 2.2

ภาษาจาวาตั้งแต่เวอร์ชัน Java SE 5 ได้กำหนดรูปแบบเมธอดที่ชื่อ `main` ให้สามารถที่จะเขียนได้อีกรูปแบบหนึ่งดังนี้

```
public static void main(String... args) {

}
```

2.2 ไวยากรณ์ระดับของคำ

คำหรือข้อความที่สามารถเขียนในโปรแกรมภาษาจาวา จะต้องเป็นคำหรือข้อความในรูปแบบใดรูปแบบหนึ่งของประเภทต่างๆ เหล่านี้

- คอมเม็นต์ (Comment)
- Identifier
- คีย์เวิร์ด (Keyword)
- สัญลักษณ์แยกคำ (Separator)
- ช่องว่าง (Whitespace)
- ข้อมูลค่าคงที่ (Literals)

2.2.1 คอมเม็นต์

คอมเม็นต์คือข้อความที่แทรกอยู่ภายในโปรแกรม ซึ่งคอมไพเลอร์จะไม่แปลข้อความนี้ให้เป็นส่วนหนึ่งของโปรแกรม กล่าวคือข้อความนี้จะไม่มีผลต่อการทำงานของโปรแกรม คอมเม็นต์เขียนไว้เพื่ออธิบายโปรแกรม เพื่อให้ผู้อ่านเข้าใจโปรแกรมง่ายยิ่งขึ้น และช่วยทำให้การแก้ไขและปรับปรุงโปรแกรมเป็นไปได้ง่ายขึ้น

ภาษาจาวากำหนดรูปแบบของการเขียนคอมเม็นต์ไว้สามรูปแบบดังนี้

1. คอมเม็นต์สำหรับข้อความบรรทัดเดียว จะใช้เครื่องหมาย // นำหน้าข้อความที่ต้องการเขียน ซึ่งจะอยู่ส่วนไหนของบรรทัดก็ได้
2. คอมเม็นต์สำหรับข้อความตั้งแต่หนึ่งบรรทัดขึ้นไป จะเริ่มต้นด้วยเครื่องหมาย /* และสิ้นสุดด้วยเครื่องหมาย */
3. คอมเม็นต์สำหรับข้อความที่ต้องการสร้างเป็นไฟล์เอกสาร ที่เป็นไฟล์ประเภท HTML จะเริ่มต้นด้วยเครื่องหมาย /** และสิ้นสุดด้วยเครื่องหมาย */ คอมเม็นต์ รูปแบบนี้สามารถสร้างเป็นไฟล์เอกสารได้โดยใช้โปรแกรม `javadoc.exe`

<u>โปรแกรมที่ 2.2</u> <u>แสดงการเขียนคอมเม็นต์ในรูปแบบต่างๆ</u>

```
/* This program is to show
   how to write comments */
public class CommentDemo {
      // Main method
      public static void main(String args[]) {
      /** This is a comment for documentation */
          System.out.println("Document");
      }
}
```

2.2.2 Identifier

identifier คือชื่อที่ตั้งขึ้นในภาษาจาวา ซึ่งอาจเป็นชื่อของคลาส ชื่อของตัวแปร ชื่อของเมธอด หรือชื่อของค่าคงที่ ซึ่งจะต้องเป็นไปตามกฎการตั้งชื่อ ดังนี้

- identifier จะต้องขึ้นต้นด้วยอักขระ A-Z, a-z, _ หรือ $ เท่านั้น
- identifier ที่ประกอบไปด้วยตัวอักขระมากกว่าหนึ่งตัว ตัวอักขระหลังจากตัวแรกนั้นจะต้องเป็นตัวอักขระข้างต้น หรือเป็นตัวเลข 0 ถึง 9 เท่านั้น
- identifier จะต้องไม่ตรงกับคีย์เวิร์ด

identifier ในภาษาจาวาจะถือว่าตัวอักษรพิมพ์ใหญ่และตัวอักษรพิมพ์เล็กต่างกัน (Case Sensitive) ดังนั้น identifier ที่ชื่อ `myVariable` จะแตกต่างจาก `MyVariable`

ตัวอย่างของ identifier ที่ถูกต้อง

- `MyVariable`
- `_MyVariable`
- `$x`

- `This_is_also_a_variable`

ตัวอย่างของ identifier ที่ไม่ถูกต้อง

- `My Variable`
- `9pns`
- `a+c`
- `Hello'World`
- `public`

หลักการตั้งชื่อที่นิยมปฏิบัติกันโดยทั่วไป

แนวทางปฏิบัติที่ใช้ในการตั้งชื่อจะมีข้อกำหนดดังนี้

- การตั้งชื่อคลาส
  - จะขึ้นต้นด้วยตัวอักษรพิมพ์ใหญ่แล้วตามด้วยตัวอักษรพิมพ์เล็กหรือตัวเลข โดยจะใช้ตัวอักษรพิมพ์ใหญ่สำหรับอักษรนำของแต่ละคำที่ตามมาในชื่อ
  - ควรเป็นคำนาม
  - ตัวอย่างเช่น Main, `HelloWorld`, `Student` หรือ `GraduateStudent` เป็นต้น
- การตั้งชื่อตัวแปร ชื่อคุณลักษณะ หรืออ็อบเจกต์
  - จะขึ้นต้นด้วยตัวอักษรพิมพ์เล็ก โดยจะใช้ตัวอักษรพิมพ์ใหญ่สำหรับอักษรนำของแต่ละคำที่ตามมาในชื่อ
  - ควรเป็นคำนามหรือเป็นชื่อสั้นๆ
  - ตัวอย่างเช่น `x`, `s1`, `id`, `name` หรือ `thesisTitle` เป็นต้น
- การตั้งชื่อเมธอด
  - จะใช้หลักการเดียวกับการตั้งชื่อตัวแปร แต่ควรเป็นคำกริยา
  - ตัวอย่างเช่น `getName`, `sayHi`, `setName` หรือ `showDetails` เป็นต้น
- การตั้งชื่อค่าคงที่
  - จะใช้ตัวอักษรพิมพ์ใหญ่ทั้งหมด และจะแยกคำโดยใช้เครื่องหมาย_ (underscore)
  - ควรเป็นคำนาม
  - ตัวอย่างเช่น `MINIMUM`, `MIN_GPA` เป็นต้น

2.2.3 คีย์เวิร์ด

คีย์เวิร์ดคือชื่อที่มีความหมายพิเศษในภาษาจาวา คอมไพเลอร์ของภาษาจาวาจะเข้าใจความหมายและคำสั่งที่จะต้องดำเนินการสำหรับคีย์เวิร์ดแต่ละตัว ภาษาจาวาได้กำหนดคีย์เวิร์ดต่างๆ ไว้ดังแสดงในตารางที่ 2.1

ตารางที่ 2.1 คีย์เวิร์ดที่ใช้ในภาษาจาวา

abstract	continue	for	new	switch
assert	default	goto	package	synchronied
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

คีย์เวิร์ดเหล่านี้ไม่สามารถจะนำมาตั้งเป็น identifier ได้ ซึ่งจากคีย์เวิร์ดข้างต้นจะเห็นว่าคีย์เวิร์ดทุกตัวจะเป็นตัวอักษรพิมพ์เล็ก และจะมีคีย์เวิร์ด `goto` และ `const` เป็นคีย์เวิร์ดที่ไม่ได้ตรงกับคำสั่งใดในภาษาจาวา ส่วนคำว่า `true` และ `false` ไม่ได้เป็นคีย์เวิร์ดในภาษาจาวาแต่จะเป็นข้อมูลค่าคงที่ชนิดตรรกะ เช่นเดียวกับคำว่า `null` ซึ่งเป็นข้อมูลค่าคงที่ของตัวแปรที่มีชนิดข้อมูลแบบอ้างอิงที่จะกล่าวถึงต่อไป

2.2.4 สัญลักษณ์แยกคำ

ภาษาจาวามีสัญลักษณ์แยกคำต่างๆ ที่สามารถนำไปใช้เขียนในโปรแกรมได้ดังตารางที่ 2.2

ตารางที่ 2.2 หน้าที่ของเครื่องหมายต่างๆ ที่ใช้ในภาษาจาวา

สัญลักษณ์แยกคำ	หน้าที่
;	เพื่อระบุการสิ้นสุดของคำสั่งต่างๆ ภายในภาษาจาวา
( )	สำหรับต่อท้ายเมธอดหรือคำสั่งอื่นๆ ในภาษาจาวา เช่น `if`,`for` เป็นต้น
,	สำหรับแยกตัวแปรหรือคำสั่งในภาษาจาวา
.	เพื่อใช้ในการระบุคุณลักษณะหรือเมธอดของอ็อบเจกต์ หรือใช้ในการระบุแพคเก็จของภาษาจาวา

{ }	**เพื่อระบุบล็อกคำสั่งของภาษาจาวา คำสั่งต่างๆ ของภาษาจาวาจะอยู่ภายในบล็อกอาทิเช่น คำสั่งที่อยู่ภายในคลาส ภายในเมธอด หรืออยู่ภายในชุดคำสั่งโครงสร้างควบคุมต่างๆ เช่น `if`,`while` หรือ `for` เป็นต้น โดยปกติบล็อกเหล่านี้สามารถซ้อนกันได้**

2.2.5 ช่องว่าง

โปรแกรมภาษาจาวาสามารถที่จะมีช่องว่างเพื่อแยกคำ ประโยค หรือคำสั่งต่างๆ ภายในโปรแกรมได้ โดยช่องว่างจะมีขนาดเท่าไรก็ได้ ทั้งนี้คอมไพเลอร์ของภาษาจาวาจะไม่นำส่วนที่เป็นช่องว่างมาเกี่ยวข้องกับขนาดของโปรแกรมไบท์โค้ด ช่องว่างจะช่วยทำให้รูปแบบของโปรแกรมซอร์ดโค้ดดูง่ายขึ้น ซึ่งรูปแบบของช่องว่างประกอบด้วย

- ช่องว่าง (กดคีย์ Space บนคีย์บอร์ด)
- แท็ป (กดคีย์ Tab บนคีย์บอร์ด)
- การขึ้นบรรทัดใหม่ (กดคีย์ Enter บนคีย์บอร์ด)

2.2.6 ข้อมูลค่าคงที่

ข้อมูลค่าคงที่คือคำที่ใช้แสดงข้อมูลที่เป็นตัวเลข ตัวอักขระ ข้อความ หรือค่าทางตรรกะ ซึ่งในภาษาจาวาได้กำหนดข้อมูลค่าคงที่ไว้ 5 ประเภทดังนี้

1. ตรรกะ (Boolean)
2. ตัวอักขระ (Character)
3. ตัวเลขจำนวนเต็ม (Integral)
4. ตัวเลขทศนิยม (Floating Point)
5. ข้อความ (String)

ซึ่งรูปแบบของการเขียนข้อมูลค่าคงที่และประเภทของชนิดข้อมูล จะกล่าวถึงในหัวข้อต่อไป

2.3 ชนิดข้อมูลแบบพื้นฐาน

ภาษาจาวาเป็นภาษาที่ต้องระบุชนิดข้อมูลอย่างชัดเจน (Strongly Typed Language) กล่าวคือข้อมูลที่เป็นตัวแปรหรือค่าคงที่ทุกตัวที่ปรากฏอยู่ในโปรแกรม จะต้องมีการประกาศ และจะต้องระบุชนิดข้อมูลด้วยเสมอ โดยชนิดข้อมูลในภาษาจาวาแบ่งเป็นสองประเภทใหญ่ๆ คือ

1. ชนิดข้อมูลแบบพื้นฐาน (Primitive Data Type)
2. ชนิดข้อมูลแบบอ้างอิง (Reference Data Type)

ชนิดข้อมูลแบบพื้นฐานคือชนิดข้อมูลที่กำหนดไว้ในภาษาจาวา ซึ่งภาษาจาวากำหนดไว้ 8 ชนิดคือ `boolean`, `char`, `byte`, `short`, `int`, `long`, `float` และ `double` ดังแสดงในตารางที่ 2.3 ซึ่งทั้ง 8 ชนิดสามารถแบ่งออกได้เป็น 4 ประเภทคือ

1. ชนิดข้อมูลตรรกะ (Logical) คือชนิด `boolean`
2. ชนิดข้อมูลอักขระ (Textual) คือชนิด `char`
3. ชนิดข้อมูลตัวเลขจำนวนเต็ม (Integral) คือชนิด `byte`,`short`,`int` และ `long`
4. ชนิดข้อมูลตัวเลขทศนิยม (Floating Point) คือชนิด `float` และ `double`

ตารางที่ 2.3 ขนาดและช่วงค่าของชนิดข้อมูลแบบพื้นฐานของภาษาจาวา

ชนิดข้อมูล	ขนาด (บิต)	ช่วงค่า	หมายเหตุ
`boolean`	1	`true` หรือ `false`	
`char`	16	'\u0000' ถึง '\uFFFF'	ข้อมูลอักขระแบบ Unicode
`byte`	8	-128 ถึง +127	
`short`	16	-32,768 ถึง +32,767	
`int`	32	-2^{31} ถึง $+2^{31}-1$	
`long`	64	-2^{63} ถึง $+2^{63}-1$	
`float`	32	-3.40E+38 ถึง +3.40E+38	IEEE 754 single precision floating point.
`double`	64	-1.80E+308 ถึง +1.80E+308	IEEE 754 double precision floating point.

ชนิดข้อมูลแบบอ้างอิงคือชนิดข้อมูลอื่นๆ ที่ไม่ใช่ชนิดข้อมูลแบบพื้นฐาน ชนิดข้อมูลทั้งสองประเภทจะมีความแตกต่างกันในเรื่องของการเก็บข้อมูลในหน่วยความจำ และวิธีการเรียกใช้งาน ซึ่งจะกล่าวถึงต่อไป

2.3.1 ชนิดข้อมูลตรรกะ

ในภาษาจาวาชนิดข้อมูล `boolean` คือชนิดข้อมูลตรรกะ โดยข้อมูลชนิดตรรกะเป็นข้อมูลที่ประกอบด้วยค่าสองค่าคือจริงและเท็จ ซึ่งตรงกับ `true` และ `false` ค่าคงที่หรือตัวแปรที่มีชนิดข้อมูลเป็น `boolean` จะมีค่าเป็นใดค่าหนึ่งภายในสองค่านี้เท่านั้น คำว่า `true` และ `false` แม้จะไม่ใช่คีย์เวิร์ดในภาษาจาวา แต่เนื่องจากเป็นคำที่สงวนไว้ (Reserved Word) จึงไม่อนุญาตให้ตั้งชื่อ identifier ตรงกับคำทั้งสอง ภาษาจาวาเป็นภาษาที่ต้องระบุชนิดข้อมูลอย่างชัดเจน ดังนั้นจึงสามารถใช้ชนิดข้อมูล `boolean` ในกรณีที่ต้องการค่าข้อมูลชนิดตรรกะเท่านั้นโดยไม่อนุญาตให้ใช้ชนิดข้อมูลอื่นๆ ซึ่งกรณีนี้จะแตกต่างจากภาษาซีหรือ C++ ที่อนุญาตให้แปลงข้อมูลชนิดตัวเลขจำนวนเต็มให้เป็นข้อมูลค่าคงที่ชนิดตรรกะได้

ตัวอย่างเช่น คำสั่ง

```
boolean flag = true;
```

เป็นการประกาศตัวแปร `flag` ให้มีชนิดข้อมูลเป็น `boolean` โดยกำหนดให้มีค่าเป็น `true`

2.3.2 ชนิดข้อมูลตัวอักขระ

ในภาษาจาวาชนิดข้อมูล `char` คือชนิดข้อมูลตัวอักขระ โดยข้อมูลชนิดตัวอักขระใช้เพื่อแสดงตัวอักขระหนึ่งตัว ซึ่งในภาษาจาวาจะถูกเก็บอยู่ในรูปของมาตรฐาน Unicode ซึ่งจะมีขนาด 16 บิต ข้อมูลค่าคงที่ซึ่งเป็นตัวอักขระแบบ Unicode จะอยู่ในเครื่องหมาย ‘ ‘ โดยจะขึ้นต้นด้วยสัญลักษณ์ \u และตามด้วยเลขฐานสิบหก (Hexadecimal number) โดยจะมีค่าตั้งแต่ ‘\u0000’ ถึง ‘\uFFFF’ เพื่อเก็บตัวอักขระของภาษาต่างๆ ได้ทั้งหมด 65,536 ตัวตัวอย่างเช่น ตัวอักขระแบบ Unicode สำหรับภาษาไทยจะมีค่าตั้งแต่ช่วง ‘\u0E00’ ถึง ‘\u0E7F’ เป็นต้น

ตัวอย่างเช่น คำสั่ง

```
char letter = '\u0041';
```

จะเป็นการประกาศตัวแปร `letter` ให้มีชนิดข้อมูลเป็น `char` โดยมีค่าเป็น `\u0041` ซึ่งมีค่าเท่ากับตัวอักษร `A`

สำหรับรหัส ASCII ที่ใช้ในระบบคอมพิวเตอร์ทั่วไป ตัวอักขระแบบ Unicode จะกำหนดค่าในช่วง ‘\u0000’ ถึง ‘\u00FF’ ให้สอดคล้องกับรหัส ASCII 128 ตัวแรก นอกจากนี้เราสามารถที่จะกำหนดข้อมูลค่าคงที่ชนิดตัวอักขระโดยกำหนดตัวอักขระภายในเครื่องหมาย ‘ ’ อาทิเช่น ‘x’, ‘1’ และ ‘$’

ตัวอย่างเช่น คำสั่ง

```
char letter = 'A';
```

จะเป็นการประกาศตัวแปร `letter` ให้มีชนิดข้อมูลเป็น `char` โดยมีค่าเป็น ตัวอักษร `A` เช่นเดียวกับคำสั่งก่อนหน้านี้

ภาษาจาวาสามารถที่จะเขียนข้อมูลค่าคงที่ที่เป็นอักขระพิเศษต่างๆ ได้ โดยใช้สัญลักษณ์ \ นำหน้าตัวอักษรภาษาอังกฤษต่างๆ อาทิเช่น '\n' คืออักขระสำหรับการขึ้นบรรทัดใหม่ เป็นต้น ตัวอักขระพิเศษที่นิยมใช้ทั่วไปจะเป็นไปตามตารางที่ 2.4

ตารางที่ 2.4 ตัวอักขระพิเศษที่นิยมใช้ทั่วไป

อักขระ	Unicode	ความหมาย
'\b'	'\u000B'	Backspace
'\t'	'\u0009'	Tab
'\n'	'\u000A'	New line
'\r'	'\u000D'	Return
'\\'	'\u005C'	Backslash
'\''	'\u0027'	Single quote
'\"'	'\u0022'	Double quote

2.3.3 ชนิดข้อมูลตัวเลขจำนวนเต็ม

ในภาษาจาวาชนิดข้อมูล `byte`, `short`, `int` และ `long` คือชนิดข้อมูลตัวเลขจำนวนเต็มโดยข้อมูลชนิดนี้คือข้อมูลที่เป็นจำนวนเต็มใดๆ ในทางคณิตศาสตร์ โดยที่แต่ละชนิดจะมีขนาดในการเก็บข้อมูลไม่เท่ากันทำให้มีช่วงของข้อมูลต่างกัน ดังที่กล่าวมาแล้วในตารางที่ 2.3 ซึ่งโดยทั่วไปภาษาจาวาจะกำหนดให้เลขจำนวนเต็มมีชนิดข้อมูลเป็น `int` การเขียนข้อมูลค่าคงที่ที่เป็นข้อมูลชนิดตัวเลขจำนวนเต็มใดๆ จะไม่มีการใช้เครื่องหมาย , (comma) เช่น 10,000 จะต้องเขียนเป็น 10000 โดยภาษาจาวาจะสามารถเขียนค่าคงที่ข้อมูลได้สามแบบดังนี้

1. เลขฐานสิบคือการเขียนเลขจำนวนเต็มทั่วไป อาทิเช่น -121 และ 75362 เป็นต้น
2. เลขฐานแปดคือการเขียนเลขจำนวนเต็มที่ขึ้นต้นด้วยเลข 0 แล้วตามด้วยตัวเลขตั้งแต่ 0 ถึง 7 อาทิเช่น 016 (มีค่าเท่ากับ 14 ในเลขฐานสิบ)
3. เลขฐานสิบหกคือการเขียนเลขจำนวนเต็มที่ขึ้นต้นด้วย 0x หรือ 0X แล้วตามด้วยตัวเลขตั้งแต่ 0 ถึง 9 หรือตัวอักษร A ถึง F อาทิเช่น 0xA2 (มีค่าเท่ากับ 162 ในเลขฐานสิบ)

การประกาศตัวแปรใดๆ ให้มีชนิดข้อมูลเป็นตัวเลขจำนวนเต็มชนิดใดชนิดหนึ่ง จะมีผลให้ภาษาจาวากำหนดขนาดของเนื้อที่สำหรับหน่วยความจำ และช่วงในการเก็บข้อมูลให้สอดคล้องกับชนิดข้อมูลนั้นโดยอัตโนมัติอาทิเช่น คำสั่ง

```
int  x = 4;
byte b = 4;
```

เป็นการประกาศตัวแปร `x` ให้มีชนิดข้อมูลเป็น `int` มีขนาดของเนื้อที่ในหน่วยความจำ 32 บิต และตัวแปร `b` ให้มีชนิดข้อมูลเป็น `byte` มีขนาดของเนื้อที่ในหน่วยความจำ 8 บิต ถึงแม้ว่าตัวแปรทั้งสองจะเก็บค่าเริ่มต้นเป็น 4

เหมือนกัน แต่ตัวแปร `x` จะสามารถเก็บข้อมูลตัวเลขจำนวนเต็มในช่วงที่กว้างกว่า (-2^{31} ถึง $2^{31}-1$)

ข้อมูลค่าคงที่ของตัวเลขจำนวนเต็มโดยทั่วไปจะถูกกำหนดให้มีชนิดข้อมูลเป็น `int` แต่ภาษาจาวาสามารถกำหนดข้อมูลค่าคงที่ของตัวเลขจำนวนเต็มให้มีชนิดข้อมูลเป็น `long` ได้โดยใส่ตัวอักษร l หรือ L ต่อท้าย อาทิเช่น

- 2l หมายถึงเลขฐานสิบที่มีค่าเท่ากับ 2 และมีชนิดข้อมูลเป็น long
- 077L หมายถึงเลขฐานแปดที่มีค่าเท่ากับ 63 และมีชนิดข้อมูลเป็น long
- 0xBAACL หมายถึงเลขฐานสิบหกที่มีชนิดข้อมูลเป็น long

2.3.4 ชนิดข้อมูลตัวเลขทศนิยม

ข้อมูลชนิดตัวเลขทศนิยมคือเลขจำนวนจริง (real number) ซึ่งในระบบคอมพิวเตอร์จะเก็บข้อมูลประเภทนี้โดยแบ่งจำนวนบิตที่เก็บข้อมูลเป็นสองส่วนคือส่วนที่เป็นความละเอียดของตัวเลข (mantissa) และส่วนที่เป็นจำนวนเลขยกกำลัง (exponent) โดยส่วนที่เป็นความละเอียดของตัวเลขจะเก็บค่าในช่วงตั้งแต่ -1.0 **ถึง** 1.0 **และ**ส่วนที่เป็นเลขยกกำลังจะเก็บค่าที่เป็นเลขยกกำลังสอง ภาษาจาวาเก็บเลขจำนวนจริงตามมาตรฐาน IEEE 754 ซึ่งจะแบ่งเลขจำนวนจริงออกเป็น single precision และ double precision โดยที่ตัวเลขแบบ single precision จะใช้เนื้อที่หน่วยความจำจำนวน 32 บิต แบ่งเป็นส่วนที่เป็นความละเอียดของตัวเลขจำนวน 24 บิต และส่วนที่เป็นจำนวนเลขยกกำลังจำนวน 8 บิต ดังแสดงในรูปที่ 2.2 ส่วนตัวเลขแบบ double precision จะใช้เนื้อที่หน่วยความจำจำนวน 64 บิต แบ่งเป็นส่วนที่เป็นความละเอียดของตัวเลขจำนวน 53 บิต และส่วนที่เป็นจำนวนเลขยกกำลังจำนวน 11 บิต

$$\text{value} = \text{mantissa} \times 2^{\text{exponent}}$$

รูปที่ 2.2 จำนวนบิตของ mantissa และ exponent สำหรับตัวเลขแบบ single precision

ภาษาจาวากำหนดชนิดข้อมูลตัวเลขทศนิยมไว้สองชนิดคือ `float` และ `double` โดยที่ชนิดข้อมูล `float` จะเก็บข้อมูลขนาด 32 บิตตามมาตรฐานแบบ single precision ส่วนข้อมูลชนิด `double` จะเก็บข้อมูลขนาด 64 บิตตามมาตรฐานแบบ double precision ภาษาจาวากำหนดข้อมูลชนิดตัวเลขทศนิยมให้เป็นเลขที่มีเครื่องหมายจุดทศนิยม อาทิเช่น 3.14 หรือ 3.0 นอกจากนี้ยังสามารถเขียนอยู่ในรูปแบบของเลขยกกำลังสิบ (exponential form) ได้โดยใช้ตัวอักษร E หรือ e ระบุจำนวนที่เป็นเลขยกกำลังสิบอาทิเช่น 6.02E23 หรือ 2e-7

ข้อมูลค่าคงที่ชนิดตัวเลขทศนิยม โดยทั่วไปจะถูกกำหนดให้มีชนิดข้อมูลเป็น `double` แต่ภาษาจาวาสามารถกำหนดข้อมูลค่าคงที่ให้มีชนิดข้อมูลเป็น `float` ได้ โดยใส่ตัวอักษร f หรือ F ต่อท้าย อาทิเช่น 2.718F หรือ 3.14f เป็นต้น

นอกจากนี้ภาษาจาวาจะกำหนดข้อมูลค่าคงที่ชนิดตัวเลขทศนิยมที่มีอักษร D หรือ d ต่อท้ายว่าเป็นข้อมูลค่าคงที่ที่มีชนิดข้อมูลเป็น `double` อาทิเช่น 3.14D แต่เนื่องจากโดยทั่วไปข้อมูลค่าคงที่ชนิดตัวเลขทศนิยมจะถูกกำหนดให้เป็นชนิด `double` อยู่แล้ว จึงไม่มีความจำเป็นต้องใส่ตัวอักษร D หรือ d ต่อท้าย

ตัวอย่างข้อมูลค่าคงที่ของตัวเลขทศนิยมที่ถูกต้อง

12. 12E2 12.0e2 12.0e2F -3.14F

ตัวอย่างข้อมูลค่าคงที่ของตัวเลขทศนิยมที่ไม่ถูกต้อง

1,234.0 1.2E108F

2.4 ตัวแปรและค่าคงที่

ข้อมูลที่เก็บอยู่ในโปรแกรมเช่นข้อมูลที่เป็นคุณลักษณะของอ็อบเจกต์ คุณลักษณะของคลาส และข้อมูลในเมธอด จะแบ่งเป็นสองประเภทคือตัวแปรและค่าคงที่ ซึ่งตัวแปรคือข้อมูลที่สามารถเปลี่ยนแปลงค่าได้ในโปรแกรมโดยใช้คำสั่งกำหนดค่า ส่วนค่าคงที่คือข้อมูลที่กำหนดค่าได้เพียงครั้งเดียวและไม่สามารถเปลี่ยนแปลงค่าได้ในโปรแกรม ทั้งตัวแปรและค่าคงที่จะต้องมีการประกาศชื่อและชนิดของข้อมูล เพื่อที่จะเตรียมเนื้อที่ในหน่วยความจำสำหรับเก็บข้อมูล

คำสั่งในการประกาศตัวแปรของภาษาจาวามีรูปแบบดังนี้

```
[modifier] dataType variableName[,variableName];
```

โดยที่

- `modifier` คือคีย์เวิร์ดระบุคุณสมบัติต่างๆ ของตัวแปรเช่น access modifier ส่วนกรณีที่ไม่ระบุจะถือว่าเป็น default
- `dataType` คือชนิดข้อมูล
- `variableName` คือชื่อของตัวแปรที่เป็นไปตามกฎการตั้งชื่อ

ตัวอย่างเช่น คำสั่ง

```
int amount;
```

เป็นการประกาศตัวแปร `amount` ให้มีชนิดข้อมูลเป็น `int`

```
double x;
```

เป็นการประกาศตัวแปร `x` ให้มีชนิดข้อมูลเป็น `double`

นอกจากนี้ภาษาจาวายังอนุญาต ให้สามารถประกาศชื่อตัวแปรที่เป็นชนิดข้อมูลเดียวกันได้หลายๆ ตัวแปรภายในคำสั่งเดียวกัน ตัวอย่างเช่น

```
float  price, wholeSalePrice;
```

เป็นคำสั่งประกาศตัวแปร `price` และ `wholeSalePrice` ให้มีชนิดข้อมูลเป็น `float`

2.4.1 คำสั่งกำหนดค่า

ตัวแปรที่มีการประกาศชนิดข้อมูลแล้วสามารถที่จะกำหนดหรือเปลี่ยนแปลงค่าได้โดยใช้คำสั่งกำหนดค่า (assignment statement) ซึ่งมีรูปแบบดังนี้

```
variableName = expression;
```

โดยที่

- `variableName` คือชื่อตัวแปร
- `expression` คือนิพจน์ซึ่งเป็นผลลัพธ์ที่ได้จากการคำนวณข้อความที่อาจประกอบไปด้วยค่าคงที่ข้อมูล ตัวแปร และตัวดำเนินการ (operator) ต่างๆ ซึ่งนิพจน์อาจเป็นนิพจน์ที่ให้ผลลัพธ์เป็นข้อมูลค่าคงที่ชนิดตัวเลขจำนวนเต็ม ตรรกะ ตัวเลขทศนิยม อักขระ หรือข้อความ ทั้งนี้ขึ้นอยู่กับชนิดข้อมูลและคำสั่งที่ใช้

ตัวอย่างต่อไปนี้แสดงการใช้คำสั่งกำหนดค่าสำหรับตัวแปร ที่ได้ทำการประกาศชนิดข้อมูลไว้แล้ว

```
x = 1;
radius = 3.14;
c = 'a';
y = x+4*3;
```

คำสั่งกำหนดค่าจะสั่งงานให้เครื่องคอมพิวเตอร์ทำงานสองขั้นตอนคือ คำนวณหาผลลัพธ์ของนิพจน์ แล้วเก็บผลลัพธ์ที่ได้ไว้ในตัวแปรอาทิเช่น คำสั่ง

```
amount = 121+14;
```

จะมีขั้นตอนคือคำนวณหาผลลัพธ์ซึ่งจะได้ค่าเป็น 135 แล้วเก็บผลลัพธ์ที่ได้ลงในตัวแปร `amount`

คำสั่งกำหนดค่าจะต้องมีชื่อตัวแปรอยู่ทางด้านซ้ายมือ การใช้คำสั่งเช่น

```
1 = x;
```

เป็นการใช้คำสั่งที่ไม่ถูกต้อง เพราะค่าคงที่ใดๆ ไม่สามารถที่จะเก็บผลลัพธ์ที่ได้จากนิพจน์ (ค่าของ `x`) ได้

นอกจากนี้เราสามารถที่จะนำชื่อตัวแปรที่จะกำหนดค่ามาประกอบในนิพจน์ได้ อาทิเช่น

```
x = x+1;
```

เป็นคำสั่งที่ให้นิพจน์เพิ่มค่าของตัวแปร x ที่มีอยู่แล้วไปอีก 1 แล้วนำผลลัพธ์ที่ได้เก็บไว้ในตัวแปร x เช่นเดิม อาทิเช่น ถ้า x มีค่าเป็น 5 จะกลายเป็น 6

เราสามารถที่จะประกาศและกำหนดค่าเริ่มต้นของตัวแปร ภายในคำสั่งเดียวกันได้ โดยมีรูปแบบคำสั่งดังนี้

```
[modifier] dataType variableName = expression [,variableName = expression];
```

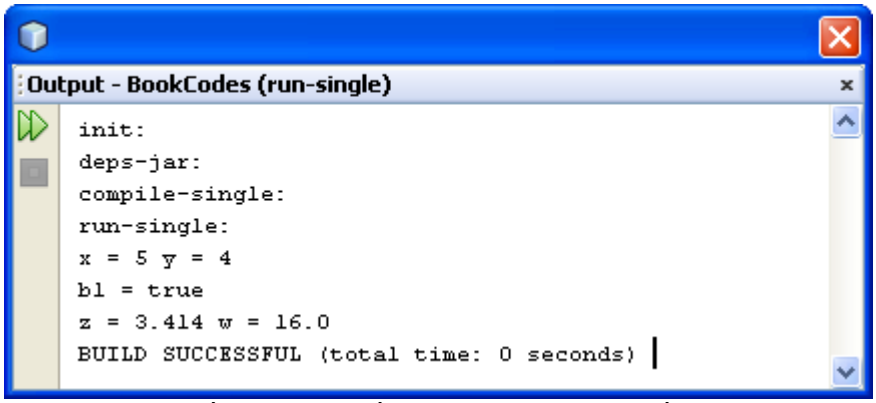
อาทิเช่น

```
int amount = 123;
float price = 12.0f;
double x = 4.0, y = 2.5;
```

โปรแกรมที่ 2.3 เป็นตัวอย่างที่แสดงการประกาศตัวแปรและการใช้คำสั่งกำหนดค่าของตัวแปรที่มีชนิดข้อมูลต่างๆ ซึ่งโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 2.3

<u>โปรแกรมที่ 2.3</u> <u>ตัวอย่างการใช้คำสั่งกำหนดค่า</u>

```
public class VariableAssignDemo {
   public void showDemo() {
       int x,y;
       boolean b1;
       float z = 3.414f;  /*  The program will not be
                              compiled successfully if
                              a character f is missing */
       double w;
       x = 5;
       y = 4;
       b1 = (x > y);
       w = x * 3.2;
       System.out.println("x = " + x + " y = " + y);
       System.out.println("b1 = " + b1);
       System.out.println("z = " + z + " w = " + w);
   }
}

-----------------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          VariableAssignDemo obj = new VariableAssignDemo();
          obj.showDemo();
      }
}
```



รูปที่ 2.3 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 2.3

2.4.2 ค่าคงที่

การประกาศค่าคงที่ในภาษาจาวาทำได้โดยการใส่คีย์เวิร์ด `final` หน้าคำสั่งประกาศชื่อ โดยมีรูปแบบดังนี้

```
[modifier] final dataType CONSTANT_NAME = expression;
```

โดยที่

- `modifier` คือคีย์เวิร์ดระบุคุณสมบัติต่างๆ ของค่าคงที่เช่น access modifier
- `dataType` คือชนิดข้อมูลของค่าคงที่
- `CONSTANT_NAME` คือชื่อของค่าคงที่ซึ่งโดยทั่วไปนิยมใช้ตัวอักษรพิมพ์ใหญ่ และแยกคำด้วยเครื่องหมาย _
- `expression` คือนิพจน์ที่ให้ผลลัพธ์เป็นชนิดข้อมูลที่สอดคล้องกับชนิดข้อมูลของค่าคงที่

ตัวอย่างเช่น คำสั่ง

```
final int MINIMUM = 4;
final double MIN_GPA = 2.00;
```

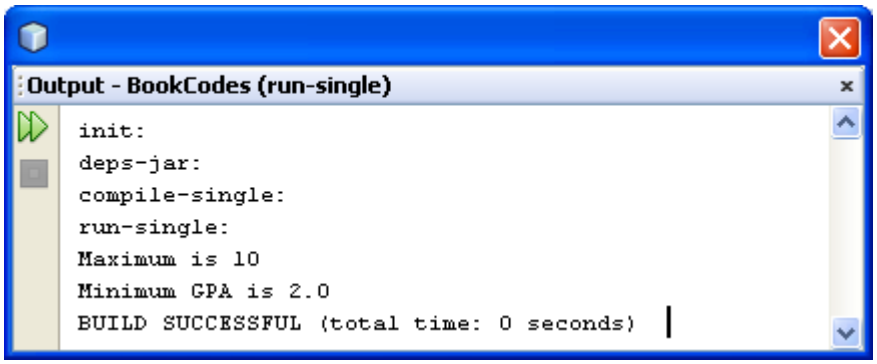
เป็นการประกาศค่าคงที่ `MINIMUM` ให้มีชนิดข้อมูลเป็น `int` โดยมีค่าเป็น 4 และค่าคงที่ `MIN_GPA` ให้มีชนิดข้อมูลเป็น `double` โดยมีค่าเป็น 2.00

ค่าคงที่จะกำหนดค่าได้เพียงครั้งเดียว โดยจะต้องมีการกำหนดค่าก่อนที่จะมีการเรียกใช้งาน ซึ่งเมื่อกำหนดค่าแล้วจะไม่สามารถเปลี่ยนแปลงค่าได้

โปรแกรมที่ 2.4 เป็นตัวอย่างที่แสดงการประกาศค่าคงที่และการใช้ค่าคงที่ ซึ่งจะให้ผลลัพธ์ดังแสดงในรูปที่ 2.4

โปรแกรมที่ 2.4 ตัวอย่างแสดงการประกาศค่าคงที่

```
public class ConstantDemo {
   public void showDemo() {
      final int MAXIMUM = 10;
      final double MIN_GPA;

      System.out.println("Maximum is " + MAXIMUM);
      MIN_GPA = 2.00;
      System.out.println("Minimum GPA is " + MIN_GPA);
//    MIN_GPA = 3.00;   //illegal
   }
}

-------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          ConstantDemo obj = new ConstantDemo();
          obj.showDemo();
      }
}
```



รูปที่ 2.4 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 2.4

2.4.3 ขอบเขตของตัวแปรและค่าคงที่

ตัวแปรและค่าคงที่ซึ่งประกาศขึ้นจะสามารถใช้งานภายในบล็อกคำสั่ง { } ที่ประกาศเท่านั้น โดยภาษาจาวาแบ่งตัวแปรและค่าคงที่เป็นสองประเภทคือ

1. ตัวแปรหรือค่าคงที่ที่เป็นคุณลักษณะของอ็อบเจกต์หรือคุณลักษณะของคลาส
2. ตัวแปรหรือค่าคงที่ที่อยู่ในบล็อกของเมธอดที่เรียกว่าค่าคงที่ภายใน (Local Constant) หรือตัวแปรภายใน (Local Variable)

ตัวแปรหรือค่าคงที่ ที่เป็นคุณลักษณะของอ็อบเจกต์หรือคุณลักษณะของคลาส คือ ตัวแปรหรือคงที่ที่ประกาศภายในบล็อกของคลาส ซึ่งอยู่นอกเมธอดของคลาส ตัวแปรหรือค่าคงที่ประเภทนี้จะมีขอบเขตใช้งานอยู่ภายในคลาส โดยที่ทุกๆ เมธอดในคลาสสามารถเรียกใช้ได้ สำหรับตัวแปรประเภทนี้จะถูกกำหนดค่าเริ่มต้นให้โดยอัตโนมัติดังตารางที่ 2.5

ตารางที่ 2.5 ค่าเริ่มต้นที่ถูกกำหนดให้อัตโนมัติ

ชนิดข้อมูล	ค่าเริ่มต้น
boolean	false
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0
char	'\u0000'
คลาส	null

สำหรับค่าคงที่หรือตัวแปรที่อยู่ภายในบล็อกของเมธอด จะมีขอบเขตการใช้งานอยู่ภายในบล็อกเท่านั้น โปรแกรมที่ 2.5 แสดงตัวอย่างตัวแปรที่เป็นคุณลักษณะของอ็อบเจกต์และตัวแปรที่เป็นตัวแปรภายใน โดยโปรแกรมนี้มีตัวแปร `i` ที่เป็นคุณลักษณะของอ็อบเจกต์ ส่วนตัวแปร `j` และ `k` ในเมธอด `method1()` เป็นตัวแปรภายใน และตัวแปร `j` ในเมธอด `method2()` ก็เป็นตัวแปรภายในอีกตัวหนึ่ง เราไม่สามารถที่จะเรียกใช้ตัวแปร `k` นอกเมธอด `method1()` ได้ นอกจากนี้ตัวแปร `j` ในเมธอดทั้งสองจะถือว่าเป็นตัวแปรคนละตัวกัน

<u>โปรแกรมที่ 2.5</u> <u>ตัวอย่างแสดงขอบเขตของตัวแปร</u>

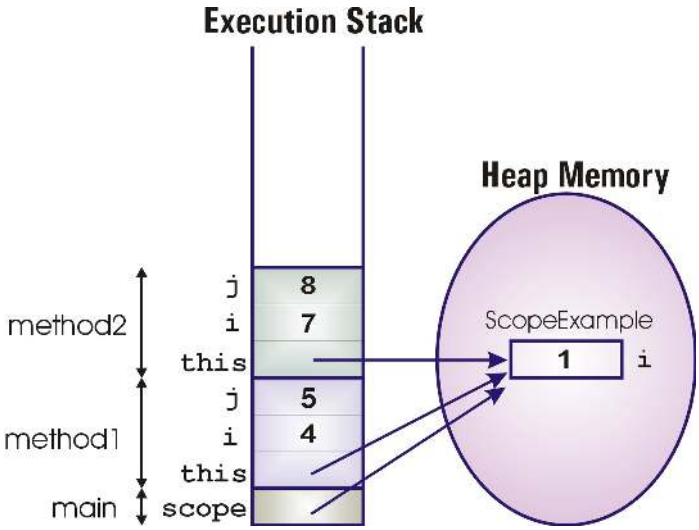
```
public class VariableScopeDemo {
   public int i;                // object variable

   public void method1() {
      int j = 4;                // local variable
      int k = 2;                // another local variable
   }

   public void method2() {
      int j = 0;                // local variable
      System.out.println(i);    // calling an object variable i
//    System.out.println(k);    // illegal
   }
}
```

ตัวแปรที่เป็นคุณลักษณะของอ็อบเจกต์จะสามารถเก็บ และเปลี่ยนแปลงค่าที่เก็บไว้ในหน่วยความจำได้ตราบใดที่อ็อบเจกต์ยังถูกอ้างอิงในโปรแกรม ตัวแปรภายในจะเป็นตัวแปรที่กำหนดในบล็อกของเมธอด ซึ่งจะมีขอบเขตการใช้งานอยู่ภายในบล็อกที่กำหนดขึ้นเท่านั้น ตัวแปรประเภทนี้จะถูกสร้างขึ้นเมื่อมีการเรียกใช้งานเมธอดที่ตัวแปรประกาศอยู่ และจะถูกลบทิ้งเมื่อสิ้นสุดการทำงานของเมธอด ดังนั้นในบางครั้งจะเรียกตัวแปรประเภทนี้ว่าตัวแปรชั่วคราว (temporary variable) ตัวแปรที่ประกาศอยู่ใน argument ของเมธอด ก็จัดอยู่ในตัวแปรประเภทนี้เช่นกัน

โปรแกรมที่ 2.6 แสดงตัวอย่างของการประกาศและใช้งานตัวแปรภายในและตัวแปรของอ็อบเจกต์ คลาส `ScopeExample` จะมีตัวแปรของอ็อบเจกต์ `i` ที่ถูกสร้างขึ้นเมื่อมีการสร้างอ็อบเจกต์ของคลาสและจะเก็บอยู่ใน Heap Memory ดังแสดงในรูปที่ 2.5 และเมื่อมีการเรียกใช้เมธอด `method1()` และ `method2()` ตัวแปรภายใน `i` และ `j` จะถูกสร้างขึ้นและถูกลบทิ้งไปเมื่อออกจากเมธอดดังแสดงในรูปที่ 2.5

<u>โปรแกรมที่ 2.6</u> <u>ตัวอย่างแสดงตัวแปรภายในและตัวแปรของอ็อบเจกต์</u>

```
public class ScopeExample {
   private int i=1;

   public void method1() {
      int i=4, j=5;

      this.i = i+j;
      method2(7);
   }
   public void method2(int i) {
      int j=8;
      this.i = i+j;
   }
}
-------------------------------------------------------------
public class TestScoping {

   public static void main(String args[]) {
      ScopeExample scope = new ScopeExample();
      scope.method1();
   }
}
```



รูปที่ 2.5 การเก็บค่าของตัวแปรในหน่วยความจำ

ภาษาจาวากำหนดว่าต้องมีการกำหนดค่าเริ่มต้นให้กับตัวแปรที่ประกาศไว้ ก่อนที่จะสามารถเรียกใช้งานได้ ในกรณีของตัวแปรของอ็อบเจกต์หรือตัวแปรของคลาส ภาษาจาวาจะกำหนดค่าเริ่มต้นให้โดยอัตโนมัติดังตารางที่ 2.5 แต่ในกรณีของตัวแปรภายใน จะต้องมีคำสั่งในการกำหนดค่าเริ่มต้นเอง มิฉะนั้นจะเกิดข้อผิดพลาดในขั้นตอนคอมไพล์ (Compile Error) ตัวอย่างเช่น คำสั่ง

```
public void demoMethod(){
      int x = (int)(Math.random()*10);
      int y,z;
      if (x > 5) {
          y = 6;
      }
      z = x+y;
}
```

จะทำให้โปรแกรมไม่สามารถคอมไพล์ผ่านได้ เนื่องจากค่า `y` มีโอกาสที่จะไม่ถูกกำหนดค่าเริ่มต้นก่อนใช้งาน

2.5 ตัวดำเนินการ

นิพจน์ในภาษาจาวาอาจจะประกอบด้วยข้อมูลค่าคงที่ ตัวแปร หรือค่าคงที่ต่างๆ โดยจะมีตัวดำเนินการต่างๆ ไว้เพื่อคำนวณหาผลลัพธ์ที่เป็นชนิดข้อมูลต่างๆ ตัวดำเนินการในภาษาจาวาแบ่งออกเป็น 4 ประเภทคือ

1. ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operator)
2. ตัวดำเนินการแบบสัมพันธ์ (Relational Operator)
3. ตัวดำเนินการทางตรรกศาสตร์ (Logical Operator)
4. ตัวดำเนินการแบบบิต (Bitwise Operator)

ทั้งนี้ตัวดำเนินการทางคณิตศาสตร์และตัวดำเนินการแบบบิต จะให้ผลลัพธ์เป็นข้อมูลชนิดจำนวนเต็มหรือจำนวนทศนิยม ส่วนตัวดำเนินการแบบสัมพันธ์และตัวดำเนินการทางตรรกศาสตร์จะให้ผลลัพธ์เป็นข้อมูลชนิดตรรกะ

2.5.1 ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการทางคณิตศาสตร์สำหรับภาษาจาวาจะประกอบไปด้วยเครื่องหมาย +, -, *, / และ % ดังแสดงในตารางที่ 2.6

ตารางที่ 2.6 ตัวดำเนินการทางคณิตศาสตร์

เครื่องหมาย	ความหมาย	ตัวอย่างนิพจน์
+	บวก	a+b
-	ลบ	a-b
*	คูณ	a*b
/	หาร	a/b
%	เศษจากการหาร	a%b

ตัวดำเนินการ / จะให้ผลลัพธ์เป็นเลขจำนวนทศนิยม ถ้าตัวถูกดำเนินการ (operand) ตัวใดตัวหนึ่งเป็นข้อมูลชนิดจำนวนทศนิยม ส่วนกรณีที่ตัวกระทำทั้งสองตัวเป็นข้อมูลชนิดจำนวนเต็ม ตัวดำเนินการจะให้ผลลัพธ์เป็นค่าจำนวนเต็ม กล่าวคือ 1/2.0 จะได้ผลลัพธ์เป็น 0.5 ส่วน 1/2 จะได้ผลลัพธ์เป็น 0

ตัวดำเนินการ % ใช้กับตัวถูกดำเนินการที่เป็นตัวเลขจำนวนเต็ม โดยจะให้ผลลัพธ์เป็นตัวเลขจำนวนเต็มที่เป็นเศษของการหาร อาทิเช่น 7%3 จะได้ผลลัพธ์เป็น 1 ตัวดำเนินการส่วนใหญ่จะต้องมีตัวถูกดำเนินการสองตัว ยกเว้นตัวดำเนินการ + และ - ที่อาจมีตัวถูกดำเนินการตัวเดียวได้เช่น –3 หรือ +4.0 เป็นต้น

ตัวอย่างต่อไปนี้เป็นการกำหนดค่า โดยมีตัวดำเนินการอยู่ในนิพจน์คำสั่ง

```
int i = 34+2;
double d1 = 34.0-0.2;
long l = 300*30;
int j = 1/2;
double d2 = 1.0/2.0;
byte b1 = 20%3;
```

ตัวดำเนินการแบบย่อ

ภาษาจาวาได้กำหนดตัวดำเนินการแบบย่อ (shortcut operator) เพื่อใช้แทนตัวดำเนินการทางคณิตศาสตร์ ที่ต้องการเปลี่ยนแปลงค่าของตัวแปรทางด้านซ้ายของคำสั่งกำหนดค่า อาทิเช่น คำสั่ง `x = x+1;` ซึ่งเป็นคำสั่งที่ต้องการเพิ่มค่าของ `x` ขึ้นอีก 1 สามารถเขียนใหม่โดยใช้ตัวดำเนินการแบบย่อได้ดังนี้

```
x += 1;
```

ข้อดีของการเขียนคำสั่งโดยใช้ตัวดำเนินการแบบย่อคือจะช่วย ทำให้โปรแกรมทำงานได้เร็วขึ้น ภาษาจาวามีตัวดำเนินการแบบย่อ 5 ตัว คือ +=, -=, *=, /= และ %= ดังแสดงในตารางที่ 2.7

ตารางที่ 2.7 ตัวดำเนินการแบบย่อ

เครื่องหมาย	ตัวอย่าง	ความหมาย
+=	x += 3;	x = x + 3;
-=	x -= 3;	x = x - 3;
*=	x *= 3;	x = x * 3;
/=	x /= 3;	x = x / 2;
%=	x %= 3;	x = x % 3;

ตัวดำเนินการเพิ่มค่าและลดค่า

ภาษาจาวายังมีตัวดำเนินการแบบย่ออีกสองตัวคือตัวดำเนินการเพิ่มค่า (increment operator) ที่ใช้เครื่องหมาย ++ และตัวดำเนินการลดค่า (decrement operator) ที่ใช้เครื่องหมาย -- ตัวดำเนินการทั้งสองตัวใช้ในการเพิ่มค่าทีละ 1 หรือลดค่าทีละ 1

ตัวดำเนินการทั้งสองสามารถใส่ไว้ข้างหน้าหรือข้างหลังตัวแปรก็ได้ ตัวอย่างเช่น

`x++` คือ `x = x+1`

`++x` คือ `x = x+1`

`x--` คือ `x = x-1`

`--x` คือ `x = x-1`

ตัวดำเนินการเพิ่มค่าและตัวดำเนินการลดค่าสามารถใช้กับตัวแปร ทั้งที่มีชนิดข้อมูลเป็นตัวเลขจำนวนเต็มและตัวเลขจำนวนทศนิยม การวางเครื่องหมายไว้ด้านหน้าตัวแปรจะมีผลให้โปรแกรมทำการเพิ่มหรือลดค่าก่อนแล้วจึงทำคำสั่งของนิพจน์นั้น ส่วนการวางเครื่องหมายไว้ด้านหลังตัวแปร จะมีผลให้โปรแกรมทำการเพิ่มหรือลดค่า หลังจากทำคำสั่งของนิพจน์นั้น

โปรแกรมที่ 2.7 แสดงตัวอย่างของการใช้ตัวดำเนินการเพิ่มค่า โดยการเปรียบเทียบผลลัพธ์ที่ได้จากการวางตำแหน่งของเครื่องหมาย ++ ไว้ด้านหน้าและด้านหลังตัวแปร ซึ่งผลลัพธ์ที่ได้จะเป็นดังแสดงในรูปที่ 2.6

<u>โปรแกรมที่ 2.7</u> <u>ตัวอย่างการใช้ตัวดำเนินการเพิ่มค่า</u>

```
public class IncrementDemo {
      public void showDemo() {
            int x;
            int y;
            x = 5;
            y = x++;
            System.out.println("x = "+x+"  y = "+y);
            y = ++x;
            System.out.println("x = "+x+"  y = "+y);
      }
}

-------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          IncrementDemo obj = new IncrementDemo();
          obj.showDemo();
      }
}
```

```
Output - Sample (run)
run:
x = 6  y = 5
x = 7  y = 7
BUILD SUCCESSFUL (total time: 1 second)
```

รูปที่ 2.6 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 2.7

2.5.2 ตัวดำเนินการแบบสัมพันธ์

ตัวดำเนินการแบบสัมพันธ์ใช้ในการเปรียบเทียบค่าของข้อมูลชนิดใดๆ สองค่า โดยจะให้ผลลัพธ์ของการเปรียบเทียบเป็นข้อมูลค่าคงที่ชนิดตรรกะ ภาษาจาวากำหนดตัวดำเนินการแบบสัมพันธ์ไว้ 6 ตัว คือ <, <=, >, >=, == และ != ดังแสดงในตารางที่ 2.8

ตารางที่ 2.8 แสดงตัวดำเนินการแบบสัมพันธ์

เครื่องหมาย	ความหมาย	ตัวอย่าง	ผลลัพธ์
<	น้อยกว่า	3<4	true
<=	น้อยกว่าหรือเท่ากับ	3<=4	true
>	มากกว่า	3>4	false
>=	มากกว่าหรือเท่ากับ	3>=4	false
==	เท่ากับ	3==4	false
!=	ไม่เท่ากับ	3!=4	true

ชนิดข้อมูลที่จะนำมาเปรียบเทียบจะต้องเป็นชนิดข้อมูลที่สอดคล้องกันอาทิเช่น การเปรียบเทียบตัวเลขกับตัวเลข ตัวอักขระกับตัวอักขระ หรืออ็อบเจกต์กับอ็อบเจกต์ เป็นต้น ตัวอย่างเช่น

```
'x' > 'y'
342 <= 431.50
"Test" == "\test"
```

ตัวถูกดำเนินการที่จะนำเปรียบเทียบอาจเป็นตัวแปรหรือนิพจน์ก็ได้ แต่จะต้องมีชนิดข้อมูลที่สอดคล้องกันด้วย ตัวอย่างเช่นถ้านิพจน์เป็น

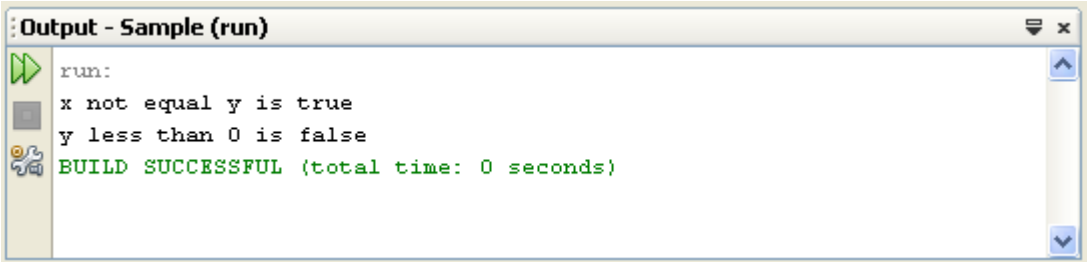
```
x < 4.23
```

ตัวแปร `x` จะต้องเป็นตัวแปรชนิดตัวเลขทศนิยมหรือจำนวนเต็ม

โปรแกรมที่ 2.8 แสดงตัวอย่างของการใช้ตัวดำเนินการแบบสัมพันธ์เพื่อเปรียบเทียบตัวแปร `x` และ `y` ที่มีชนิดข้อมูลเป็น `int` และเก็บผลลัพธ์ลงในตัวแปร `b1` ที่มีชนิดข้อมูลเป็น `boolean` ซึ่งโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 2.7

โปรแกรมที่ 2.8 การใช้ตัวดำเนินการแบบสัมพันธ์

```
public class BooleanDemo {
    public void showDemo() {
        int x = 5;
        int y = 4;
        boolean b1;
        b1 = (x != y);
        System.out.println("x not equal y is " + b1);
        System.out.println("y less than 0 is " + (y < 0));
    }
}

-----------------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          BooleanDemo obj = new BooleanDemo();
          obj.showDemo();
      }
}
```



รูปที่ 2.7 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 2.8

2.5.3 ตัวดำเนินการทางตรรกศาสตร์

ตัวดำเนินการทางตรรกศาสตร์ จะใช้กับตัวถูกดำเนินการที่เป็นนิพจน์ตรรกศาสตร์ หรือชนิดข้อมูล `boolean` ตัวดำเนินการประเภทนี้จะให้ผลลัพธ์เป็นข้อมูลค่าคงที่ชนิดตรรกะ โดยภาษาจาวากำหนดตัวดำเนินการทางตรรกศาสตร์ไว้หกตัวดังแสดงในตารางที่ 2.9

ตัวดำเนินการทุกตัวจะต้องมีตัวถูกดำเนินการสองตัว ยกเว้นตัวดำเนินการที่เป็นตัวดำเนินการที่ใช้ในการกลับค่า ซึ่งต้องการตัวถูกดำเนินการเพียงหนึ่งตัวตารางที่ 2.10 ถึง ตารางที่ 2.13 แสดงผลลัพธ์ที่เป็นข้อมูลค่าคงที่ชนิดตรรกะซึ่งได้จากการดำเนินการที่เป็นการกลับค่า, AND, OR และ Exclusive-OR ค่าทางตรรกะ
ตัวอย่างของนิพจน์ที่ใช้ตัวดำเนินการทางตรรกศาสตร์มีดังนี้

`(7>6) & (2<1)` จะได้ผลลัพธ์มีค่าเป็น `false`

`(7>6) | (2<1)` จะได้ผลลัพธ์มีค่าเป็น `true`

`!(7>6)` จะได้ผลลัพธ์มีค่าเป็น `false`

ตารางที่ 2.9 ตัวดำเนินการทางตรรกศาสตร์

เครื่องหมาย	ความหมาย
!	**กลับค่าทางตรรกะ**
&& หรือ &	**AND ค่าทางตรรกะ**
|| หรือ |	**OR ค่าทางตรรกะ**
^	**Exclusive-OR ค่าทางตรรกะ**

ตารางที่ 2.10 ผลลัพธ์ของการกลับค่าทางตรรกะ

op	!op
true	false
false	true

ตารางที่ 2.11 ผลลัพธ์ของการ AND ค่าทางตรรกะ

op1	op2	op1 & op2
true	true	true
true	false	false
false	true	false
false	false	false

ตารางที่ 2.12 ผลลัพธ์ของการ OR ค่าทางตรรกะ

op1	op2	op1 | op2
true	true	true
true	false	true
false	true	true
false	false	false

ตารางที่ 2.13 ผลลัพธ์ของการ Exclusive-OR ค่าทางตรรกะ

op1	op2	op1 ^ op2
true	true	false
true	false	true
false	true	true
false	false	false

ตัวดำเนินการที่มีเครื่องหมาย && และ || เรียกว่า ตัวดำเนินการทางตรรกศาสตร์แบบ short circuit โดยที่ && เป็นการ AND ค่าทางตรรกะ ซึ่งจะแตกต่างจาก & ตรงที่ && จะหยุดการเปรียบเทียบถ้านิพจน์ตัวแรกเป็นเท็จ เช่นเดียวกับ || ที่เป็นการ OR ค่าทางตรรกะ ซึ่งจะแตกต่างจาก | ตรงที่ || จะหยุดการเปรียบเทียบถ้านิพจน์ตัวแรกเป็นจริง ตัวอย่างเช่น

```
int x = 10;
if ( (x > 0) || (x++ < -5) ) {
      System.out.println(x);
}
```

ผลลัพธ์ของ x จะมีค่าเป็น 10 เนื่องจาก || จะไม่เรียกคำสั่งในนิพจน์ที่สอง

แต่คำสั่ง

```
int  x = 10;
if ( (x > 0) | (x++ < -5) ) {
      System.out.println(x);
}
```

ผลลัพธ์ของ x จะมีค่าเป็น 11

2.5.4 ตัวดำเนินการแบบบิต

ตัวดำเนินการแบบบิตเป็นตัวดำเนินการที่ใช้กับข้อมูลชนิดตัวเลขจำนวนเต็ม เพื่อจัดการกับข้อมูลเชิงบิตหรือเพื่อเลื่อนบิต โดยมีเครื่องหมายต่างๆ ดังแสดงในตารางที่ 2.14 และตารางที่ 2.15

ตารางที่ 2.14 ตัวดำเนินการเพื่อจัดการกับข้อมูลเชิงบิต

เครื่องหมาย	ความหมาย
~	Complement
&	AND
|	OR
^	XOR

ตารางที่ 2.15 ตัวดำเนินการเพื่อเลื่อนบิต

เครื่องหมาย	ความหมาย
>>	signed right shift
>>>	unsigned right shift
<<	left shift

เครื่องหมาย ~, &, | และ ^ ใช้ในการจัดการข้อมูลเชิงบิต เช่น 4 ^ 3 คือ 0100 ที่ทำการ XOR กับ 0011 จะมี

ค่าเป็น 0111 ส่วนเครื่องหมาย >>, >>> และ << เป็นตัวดำเนินการเพื่อเลื่อนบิต โดยจะใช้กับชนิดข้อมูลจำนวนเต็มที่เป็น int หรือ long โดยตัวถูกดำเนินการที่เป็นจำนวนบิตที่จะเลื่อนไปจะเป็นเศษของการหารด้วย 32 และ 64 สำหรับชนิดข้อมูล int และ long ตามลำดับ ดังนั้นคำสั่ง

```
int  x  =  8;
System.out.println(x >> 32);
```

หมายถึงการเลื่อนบิตของตัวแปร x ไป 0 บิต ไม่ใช่ 32 บิต

เครื่องหมาย >> เป็นการเลื่อนบิตโดยพิจารณาจากเครื่องหมาย ซึ่งถ้าบิตทางซ้ายเป็นค่า 1 ก็จะใส่ค่า 1 แทน แต่ถ้าเป็นค่า 0 ก็จะใส่ค่า 0 แทน ส่วนเครื่องหมาย >>> และ << จะเป็นการเลื่อนบิตโดยไม่พิจารณาเครื่องหมาย กล่าวคือจะใส่ค่า 0 เสมอ การเลื่อนบิตไปทางขวา (>>) คือการหารของจำนวนเต็มด้วยค่า 2 ยกกำลังจำนวนบิตที่จะเลื่อน ส่วนการเลื่อนบิตไปทางซ้าย (<<) จะเป็นการคูณเลขจำนวนเต็มด้วยค่า 2 ยกกำลังจำนวนบิตที่จะเลื่อน ตัวอย่างเช่น

128 >> 1 คือ $128/2^1 = 64$

-128 >> 4 คือ $-128/2^4 = -8$

128 << 2 คือ $128 * 2^2 = 512$

2.5.5 ลำดับความสำคัญของตัวดำเนินการ

กรณีที่นิพจน์ใดๆ มีตัวดำเนินการมากกว่าหนึ่งตัว ภาษาจาวาจะจัดลำดับความสำคัญของตัวดำเนินการเพื่อคำนวณหาผลลัพธ์ตามลำดับความสำคัญของตัวดำเนินการ โดยมีลำดับความสำคัญดังแสดงในตารางที่ 2.16

ตารางที่ 2.16 ลำดับความสำคัญของตัวดำเนินการ

ลำดับที่	เรียงจาก	ตัวดำเนินการ
1	ขวาไปซ้าย (R to L)	++, --, +, -, ~, ! (data type)
2	ซ้ายไปขวา (L to R)	*, /, %
3	ซ้ายไปขวา (L to R)	+, -
4	ซ้ายไปขวา (L to R)	<<, >>, >>>
5	ซ้ายไปขวา (L to R)	<, >, <=, >=, instanceof
6	ซ้ายไปขวา (L to R)	==, !=
7	ซ้ายไปขวา (L to R)	&
8	ซ้ายไปขวา (L to R)	^

9	ซ้ายไปขวา (L to R)	|
10	ซ้ายไปขวา (L to R)	&&
11	ซ้ายไปขวา (L to R)	||
12	ขวาไปซ้าย (R to L)	?:
13	ขวาไปซ้าย (R to L)	=, +=, -=, *=, /=, %=, <<=, >>=, >>>=, &=, ^=, |=

ตัวอย่างเช่น คำสั่ง

x = 2+3*4-(7+2);

จะคำนวณหาผลลัพธ์ตามลำดับดังนี้

1. คำนวณหาผลลัพธ์ค่า 7+2 ทำให้ได้

   x = 2+3*4-9

2. คำนวณหาผลลัพธ์ค่า 3*4 ทำให้ได้

   x = 2+12-9

3. คำนวณหาผลลัพธ์ค่า 2+12 ทำให้ได้

   x = 14-9

4. คำนวณหาผลลัพธ์ค่า 14-5 ทำให้ได้

   x = 5

2.6 การแปลงชนิดข้อมูล

นิพจน์ทางคณิตศาสตร์ในภาษาจาวาอาจมีตัวถูกดำเนินการทางคณิตศาสตร์ ที่มีชนิดข้อมูลเป็นประเภทต่างๆ ภาษาจาวาได้กำหนดให้ผลลัพธ์ของนิพจน์เป็นดังนี้

- ในกรณีที่ตัวถูกดำเนินการทั้งสองตัวมีชนิดข้อมูลเป็น `double` เหมือนกัน จะทำให้ได้ผลลัพธ์ที่มีชนิดข้อมูลเป็น `double`
- ในกรณีที่ตัวถูกดำเนินการทั้งสองตัวมีชนิดข้อมูลเป็น `float` เหมือนกัน จะทำให้ได้ผลลัพธ์ที่มีชนิดข้อมูลเป็น `float`
- ในกรณีที่ตัวถูกดำเนินการทั้งสองมีชนิดข้อมูลที่ต่างกัน ภาษาจาวาจะมีหลักการแปลงชนิดข้อมูล (type conversion) ดังนี้
  - ถ้าตัวถูกดำเนินการตัวหนึ่งมีชนิดข้อมูลเป็น `double` ตัวถูกดำเนินการอีกตัวหนึ่งจะถูกแปลงให้มีชนิดข้อมูลเป็น `double` โดยอัตโนมัติ
  - ถ้าตัวถูกดำเนินการทั้งสองไม่ได้มีชนิดข้อมูลเป็น `double` แต่มีตัวถูกดำเนินการตัวหนึ่งที่มีชนิดข้อมูล

เป็น float ตัวถูกดำเนินการอีกตัวหนึ่งจะถูกแปลงให้มีชนิดข้อมูลเป็น float โดยอัตโนมัติ

- ถ้าตัวถูกดำเนินการทั้งสองไม่ได้มีชนิดข้อมูลเป็น double หรือ float แต่มีตัวถูกดำเนินการตัวหนึ่งที่มีชนิดข้อมูลเป็น long ตัวถูกดำเนินการอีกตัวหนึ่งจะถูกแปลงให้มีชนิดข้อมูลเป็น long โดยอัตโนมัติ
- กรณีอื่นๆ ตัวถูกดำเนินการทั้งสองจะแปลงให้มีชนิดข้อมูลเป็น int

จากหลักการข้างต้นจะเห็นได้ว่าผลลัพธ์ที่ได้จากการคำนวณนิพจน์คณิตศาสตร์ จะมีชนิดข้อมูลเป็น int เป็นอย่างน้อย ดังนั้นคำสั่งต่อไปนี้

```
byte  b1, b2, b3;
b1 = 2;
b2 = 4;
b3 = b1+b2;      // illegal
```

จึงเป็นคำสั่งที่ไม่ถูกต้อง เนื่องจาก b1+b2 จะให้ค่าผลลัพธ์ที่มีชนิดข้อมูลเป็น int ซึ่งไม่สามารถกำหนดค่าให้กับตัวแปรที่มีชนิดข้อมูลเป็น byte ได้

2.6.1 การแปลงข้อมูลในคำสั่งกำหนดค่า

ภาษาจาวากำหนดให้คำสั่งกำหนดค่าจะต้องมีชนิดข้อมูลของตัวแปรทางด้านซ้ายและชนิดข้อมูลของนิพจน์ทางด้านขวาสอดคล้องกัน อาทิเช่น

```
int i = 4;
double x = 3.0;
```

ในกรณีที่ตัวแปรและนิพจน์มีชนิดข้อมูลที่แตกต่างกัน คอมไพเลอร์ของภาษาจาวาจะทำการแปลงชนิดข้อมูลทั้งสองชนิดให้สอดคล้องกัน โดยการแปลงชนิดข้อมูลมีสองรูปแบบคือ

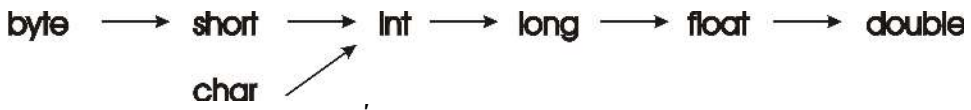
1. การแปลงข้อมูลที่กว้างขึ้น (widening conversion) คือการแปลงจากชนิดข้อมูลที่มีขนาดเล็กกว่าไปเป็นชนิดข้อมูลที่มีขนาดใหญ่กว่า
2. การแปลงข้อมูลที่แคบลง (narrowing conversion) คือการแปลงจากชนิดข้อมูลที่มีขนาดใหญ่กว่าไปเป็นชนิดข้อมูลที่มีขนาดเล็กลง ซึ่งอาจมีผลให้เสียความละเอียดของข้อมูลบางส่วนไป

ภาษาจาวากำหนดขนาดของชนิดข้อมูลต่างๆ ที่สามารถแปลงข้อมูลให้กว้างขึ้นได้ ดังแสดงในรูปที่ 2.8 ซึ่งมีหลักการดังนี้

- ชนิดข้อมูลตัวเลขจำนวนเต็มสามารถแปลงให้เป็นชนิดข้อมูลตัวเลขทศนิยมได้
- ชนิดข้อมูล float สามารถแปลงให้เป็นชนิดข้อมูล double ได้
- ชนิดข้อมูลตัวเลขจำนวนเต็มมีขนาดเรียงกันจากน้อยไปมากดังนี้

```
byte → short → int → long
```

- ชนิดข้อมูล `char` สามารถแปลงให้เป็นชนิดข้อมูล `int` ได้
- ชนิดข้อมูล `boolean` จะไม่มีความสัมพันธ์กับชนิดข้อมูลแบบพื้นฐานอื่นๆ



รูปที่ 2.8 การแปลงชนิดข้อมูล

ในคำสั่งกำหนดค่า ถ้าผลลัพธ์ของนิพจน์เป็นชนิดข้อมูลที่มีขนาดเล็กกว่าชนิดข้อมูลของตัวแปร ภาษาจาวาจะทำการแปลงข้อมูลให้เป็นชนิดข้อมูลของตัวแปรดังกล่าวโดยอัตโนมัติอาทิเช่น คำสั่ง

```
int i = 4;
long l = i;
```

นิพจน์ `i` จะถูกปรับชนิดข้อมูลจาก `int` ให้เป็น `long` โดยอัตโนมัติ

หรือคำสั่ง

```
double  x = 3;
```

นิพจน์ที่มีค่าเป็น 3 จะถูกปรับชนิดข้อมูลจาก `int` ให้เป็น `double` โดยอัตโนมัติ

ในกรณีที่คำสั่งกำหนดค่ามีชนิดข้อมูลของตัวแปรที่มีขนาดเล็กกว่า ชนิดข้อมูลของนิพจน์ ภาษาจาวาจะไม่สามารถแปลงชนิดข้อมูลของนิพจน์ให้เป็นขนาดที่เล็กลงเท่ากับชนิดข้อมูลของตัวแปรโดยอัตโนมัติ แต่คอมไพเลอร์จะแจ้งข้อผิดพลาดในขั้นตอนการคอมไพล์ (compile error) ตัวอย่างเช่น คำสั่ง

```
int amount = 123L;
```

หรือ

```
float f = 4.0;
```

จะไม่สามารถคอมไพล์ผ่านได้ เนื่องจากชนิดข้อมูลของนิพจน์มีขนาดใหญ่กว่าชนิดข้อมูลของตัวแปร

โปรแกรมที่ 2.9 แสดงตัวอย่างของข้อผิดพลาดในการแปลงชนิดข้อมูล ซึ่งจะทำให้โปรแกรมนี้ไม่สามารถคอมไพล์ผ่านได้

โปรแกรมที่ 2.9 โปรแกรมที่มีข้อผิดพลาดในการแปลงชนิดข้อมูล

```
public class PromotionDemo {
      public void showDemo() {
            int i;
            long l;
            float fl = 4.2f;
            i = 4;
            l = i;
            fl = i;
            double x = fl;
            fl = 4.2;                //illegal
      }
}
```

2.6.2 Typecasting

ภาษาจาวาจะสามารถทำการแปลงชนิดข้อมูล ให้เป็นชนิดข้อมูลที่มีขนาดเล็กลงได้ โดยใช้วิธีการที่เรียกว่า typecasting ซึ่งมีรูปแบบดังนี้

```
(targetType) expression
```

โดยที่

- `targetType` คือชนิดข้อมูลที่ต้องการ

การใช้ typecasting จะช่วยทำให้โปรแกรมที่มีคำสั่งซึ่งจำเป็นต้องแปลงชนิดข้อมูลให้มีขนาดเล็กลงสามารถคอมไพล์ผ่านได้ แต่จะทำให้ข้อมูลบางส่วนสูญหายไปในบางกรณี

ตัวอย่างเช่น คำสั่ง

```
int amount = (int)3.0;
```

จะทำการแปลงนิพจน์ 3.0 ที่มีชนิดข้อมูลเป็น `double` ให้เป็น 3 ที่มีชนิดข้อมูลเป็น `int`

หรือตัวอย่างคำสั่ง

```
int x;
double y = 1.25;
x = (int)y;
```

จะทำการแปลงนิพจน์ `y` ที่มีค่า 1.25 ให้มีชนิดข้อมูลเป็น `int` ที่มีค่า 1 แต่จะทำให้ตัวเลขที่เป็นส่วนทศนิยมหายไป

โปรแกรมที่2.10 แสดงตัวอย่างการใช้ typecasting ในการแปลงชนิดข้อมูล โดยนิพจน์ `b1+b2` จะให้ผลลัพธ์ที่มีชนิดข้อมูลเป็น `int` จึงต้องแปลงชนิดข้อมูลให้เป็น `byte` เพื่อที่จะสามารถกำหนดค่าให้กับตัวแปร `b3` ซึ่งมีชนิดข้อมูลเป็น `byte` ได้ เช่นเดียวกันกับค่า 3.2 ซึ่งมีชนิดข้อมูลเป็น `double` จะถูกแปลงชนิดข้อมูลให้เป็น `float`

โปรแกรมที่ 2.10 การใช้ typecasting ในการแปลงชนิดข้อมูล

```
public class TypecastingDemo {
      public void showDemo() {
            byte b1 = 4;
            byte b2 = 3;
            byte b3;
            b3 = (byte) (b1 + b2);
            float f1;
            f1 = (float) 3.2;
      }
}

-----------------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          TypecastingDemo obj = new TypecastingDemo();
          obj.showDemo();
      }
}
```

2.7 ชนิดข้อมูลแบบอ้างอิง

ตัวแปรหรือค่าคงที่ที่ประกาศเป็นชนิดข้อมูลอื่นๆ ซึ่งไม่ใช่ชนิดข้อมูลแบบพื้นฐาน จะเป็นชนิดข้อมูลแบบอ้างอิงซึ่งก็คืออ็อบเจกต์ในภาษาจาวา โดยแบ่งออกเป็นสองแบบคือ

1. ชนิดข้อมูลที่เป็นคลาส
2. ชนิดข้อมูลที่เป็นอะเรย์

ตัวอย่างของชนิดข้อมูลที่เป็นคลาสคือ ชนิดข้อมูล `String` โดย `String` ไม่ใช่ชนิดข้อมูลแบบพื้นฐาน แต่เป็นคลาสที่นิยามไว้ใน Java API อาทิเช่น คำสั่ง

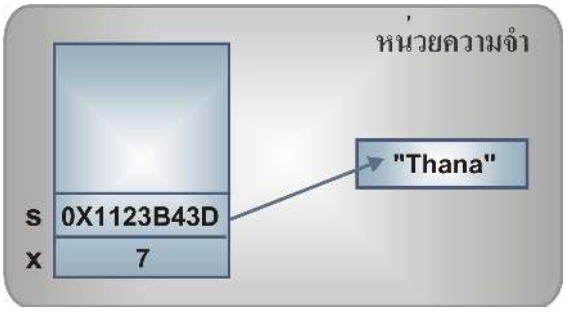
```
String id;
```

เป็นคำสั่งประกาศตัวแปร `id` ให้เป็นอ็อบเจกต์ของคลาส `String` สำหรับตัวอย่างของชนิดข้อมูลที่เป็นอะเรย์จะถูกกล่าวถึงในบทที่ 8

ชนิดข้อมูลแบบอ้างอิงจะมีวิธีการเก็บข้อมูลในหน่วยความจำที่แตกต่างจาก การเก็บข้อมูลของชนิดข้อมูลแบบพื้นฐาน ทั้งนี้ข้อมูลที่เก็บในหน่วยความจำของชนิดข้อมูลแบบอ้างอิงจะเป็นตำแหน่งอ้างอิงที่เก็บข้อมูลในหน่วยความจำ แต่ในกรณีของชนิดข้อมูลแบบพื้นฐาน ข้อมูลที่เก็บในหน่วยความจำจะเป็นค่าของข้อมูลจริงๆ อาทิเช่น คำสั่ง

```
int x = 7;
String s = new String("Thana");
```

เป็นการประกาศตัวแปรที่มีชนิดข้อมูลแบบพื้นฐาน `int` ที่ชื่อ `x` และให้เก็บค่าเป็น 7 และเป็นการประกาศตัวแปรชนิดข้อมูลแบบอ้างอิง (ซึ่งก็คืออ็อบเจกต์) `s` ของคลาส `String` และให้เก็บค่าเป็นข้อความว่า `Thana` ซึ่งตัวแปรทั้งสองตัวอาจจะมีการเก็บข้อมูลในหน่วยความจำดังแสดงในรูปที่ 2.9 ซึ่งค่าที่เก็บในตัวแปร `x` จะเป็นค่าข้อมูลที่เป็นค่า 7 ส่วนค่าของตัวแปร `s` จะเป็นตำแหน่งอ้างอิงในหน่วยความจำที่ใช้เก็บข้อความว่า `Thana`



รูปที่ 2.9 ตัวอย่างของการเก็บข้อมูลในหน่วยความจำ

การประกาศตัวแปร (หรืออ็อบเจกต์) ของชนิดข้อมูลแบบอ้างอิงจะเป็นเพียงการประกาศชื่อตัวแปร (หรืออ็อบเจกต์) แต่จะไม่มีการจองเนื้อที่ในหน่วยความจำเพื่อเก็บข้อมูล ในกรณีที่ตัวแปรดังกล่าวเป็นคุณลักษณะของอ็อบเจกต์ หรือคุณลักษณะของคลาส ภาษาจาวาจะกำหนดตำแหน่งอ้างอิงเริ่มต้นให้มีค่าเป็น `null` โดยอัตโนมัติ ซึ่งเนื้อที่ในหน่วยความจำเพื่อเก็บข้อมูลต่างๆ ของตัวแปรชนิดข้อมูลแบบอ้างอิงจะถูกสร้างขึ้น เมื่อมีการเรียกใช้คำสั่ง `new` อาทิเช่น คำสั่ง

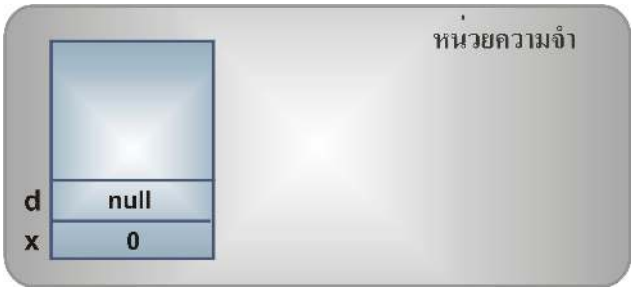
```
Date d;
```

เป็นคำสั่งในการประกาศตัวแปร (หรืออ็อบเจกต์) `d` ให้เป็นคลาสชนิด `Date` ที่กำหนดไว้ใน Java API และจะได้ค่าในหน่วยความจำดังแสดงในรูปที่ 2.10 (ก)

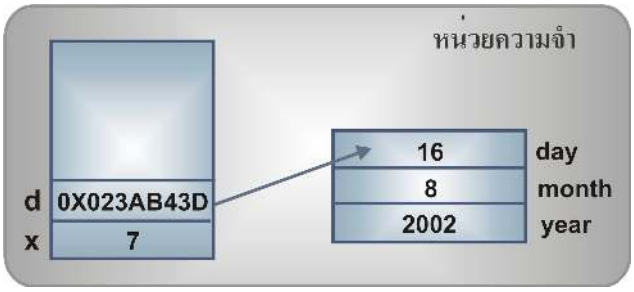
และคำสั่ง

```
d = new Date(16, 8, 2002);
```

จะเป็นคำสั่งในการจองเนื้อที่ในหน่วยความจำเพื่อเก็บคุณลักษณะของอ็อบเจกต์ที่ชื่อ `d` ที่มีอยู่ 3 ตัว คือ `day`, `month` และ `year` ซึ่งมีชนิดข้อมูลเป็น `int` ดังแสดงในรูปที่ 2.10 (ข)



(ก)



(ข)

รูปที่ 2.10 ตัวอย่างของการเก็บข้อมูลในหน่วยความจำ

2.7.1 คลาส `String`

- `String` เป็นคลาสที่กำหนดไว้ใน Java API ตัวแปรที่มีชนิดข้อมูลเป็น `String` ก็คืออ็อบเจกต์ชนิดหนึ่ง ซึ่ง `String` มีข้อแตกต่างจากอ็อบเจกต์ทั่วๆ ไปดังนี้
- `String` เป็นอ็อบเจกต์ที่มีค่าคงที่ข้อมูลซึ่งก็คือข้อความใดๆ ที่อยู่ภายในเครื่องหมาย double quote (" ") ตัวอย่างเช่น

```
"This is a java course"
```

- `String` เป็นอ็อบเจกต์ที่สามารถถูกสร้างขึ้นและกำหนดค่าได้โดยไม่จำเป็นต้องใช้คำสั่ง `new` ตัวอย่างเช่น เราสามารถใช้คำสั่ง

```
String s = "Thana";
```

โดยไม่จำเป็นที่จะต้องใช้คำสั่ง

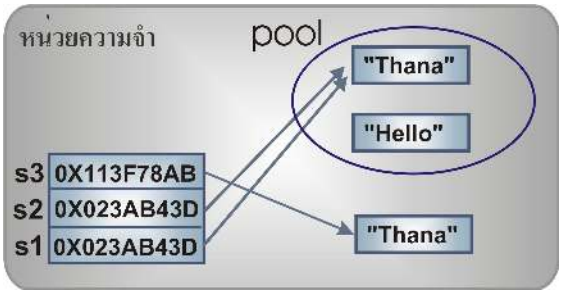
```
String s = new String("Thana");
```

- ในกรณีที่ไม่ใช้คำสั่ง `new` ภาษาจาวาจะกำหนดตำแหน่งอ้างอิงในหน่วยความจำของข้อความที่ระบุในเครื่อง (" ") โดยพิจารณาจาก String Pool ว่ามีข้อความเดิมอยู่หรือไม่ หากมีก็จะใช้ตำแหน่งอ้างอิงที่ซ้ำกัน แต่ถ้ายังไม่มีก็จะสร้างข้อความขึ้นมาใหม่และกำหนดตำแหน่งอ้างอิงของข้อความนั้น ส่วนกรณีที่ใช้คำสั่ง `new` ภาษาจาวาจะสร้างข้อความใหม่และจองเนื้อที่ในหน่วยความจำเสมอ โปรแกรมที่ 2.11 แสดง

ตัวอย่างการสร้างและกำหนดค่าอ็อบเจกต์ชนิด `String` ทั้งในกรณีที่ใช้และไม่ใช้คำสั่ง `new` อ็อบเจกต์ `s1` และ `s2` จะมีข้อมูลในหน่วยความจำเป็นตำแหน่งอ้างอิงที่เดียวกัน ส่วน `s3` จะมีข้อมูลในหน่วยความจำเป็นตำแหน่งอ้างอิงที่ต่างกันเนื่องจากมีการสร้างข้อความขึ้นมาใหม่ดังแสดงในรูปที่ 2.11

<u>โปรแกรมที่ 2.11</u> <u>ตัวอย่างการประกาศและสร้างอ็อบเจกต์ชนิด `String`</u>

```
public class StringDemo {
      public void showDemo() {
            String s1 = "Thana";
            String s2 = "Thana";
            String s3 = new String("Thana");
      }
}

-------------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          StringDemo obj = new StringDemo();
          obj.showDemo();
      }
}
```



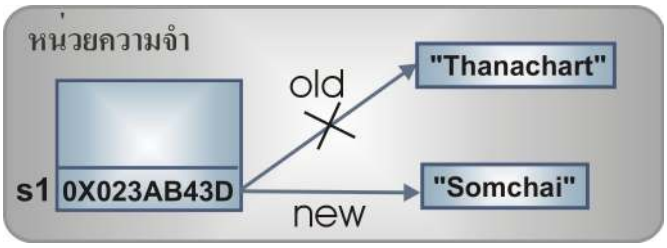
รูปที่ 2.11 ตัวอย่างการเก็บข้อมูลชนิด `String` ในหน่วยความจำ

- `String` เป็นอ็อบเจกต์ที่เปลี่ยนค่าไม่ได้ (Immutable Object) การกำหนดค่าให้กับอ็อบเจกต์ชนิด `String` ใหม่ เป็นการเปลี่ยนตำแหน่งอ้างอิงในหน่วยความจำของอ็อบเจกต์ดังกล่าว แต่จะไม่ได้มีการเปลี่ยนค่าภายในตำแหน่งอ้างอิงเดิม ตัวอย่างเช่น คำสั่ง

```
String  s1;
s1 = "Thanachart";
s1 = "Somchai";
```

จะมีผลทำให้ตำแหน่งอ้างอิงในหน่วยความจำของอ็อบเจกต์ `s1` เปลี่ยนไป ดังแสดงในรูปที่ 2.12 ซึ่งเป็น

ตัวอย่างของการเปลี่ยนตำแหน่งอ้างอิงในการเก็บข้อมูลชนิด String ในหน่วยความจำ



รูปที่ 2.12 ตัวอย่างการเปลี่ยนตำแหน่งอ้างอิงในการเก็บข้อมูล

- String เป็นอ็อบเจกต์ที่มีตัวดำเนินการที่ใช้ในการเชื่อมข้อความสองข้อความเข้าด้วยกัน โดยใช้เครื่องหมาย + อาทิเช่น

```
String s1 = "Hello"+" World";
```

ตัวดำเนินการในการเชื่อมข้อความสามารถใช้เชื่อมข้อมูลชนิด String กับตัวถูกดำเนินการที่เป็นชนิดข้อมูลอื่นๆ ที่ไม่ใช่ชนิด String ได้ ซึ่งภาษาจาวาจะแปลงชนิดข้อมูลดังกล่าวให้เป็นชนิด String โดยอัตโนมัติอาทิเช่น คำสั่ง

```
String s1 = "This";
String s2 = s1+ " is a test ";
String s3 = s1+4;
```

จะทำให้ได้ข้อความของตัวแปร s2 เป็น "This is a test" และ s3 เป็น "This4"

ตัวดำเนินการเพื่อเชื่อมข้อความจะมีตัวดำเนินการแบบย่อที่ใช้เครื่องหมาย += เพื่อเชื่อมข้อความ แล้วกำหนดค่าในอ็อบเจกต์ของคลาส String โดยใช้ชื่อเดิมอาทิเช่น คำสั่ง

```
String s1 = "This";
s1 += " is a test";
```

เป็นคำสั่งเชื่อมข้อความของอ็อบเจกต์ s1 เดิมกับข้อความที่ว่า " is a test" แล้วเก็บลงในอ็อบเจกต์ s1 เช่นเดิม โดยทำให้ s1 มีข้อความเป็น "This is a test"

2.7.2 คลาส Math

Java API ได้กำหนดให้มีคลาส Math ที่อยู่ในแพคเก็จ java.lang ซึ่งจะมีเมธอดต่างๆ ในการจัดการกับฟังก์ชันหรือคำสั่งทางคณิตศาสตร์ต่างๆ คลาส Math เป็นคลาสแบบ final และเมธอดทุกเมธอด จะเป็นเมธอดของคลาส (มีคีย์เวิร์ด static อยู่ ซึ่งจะกล่าวถึงในบทที่ 6) การเรียกใช้เมธอดเหล่านี้ทำได้โดยไม่จำเป็นต้องสร้างอ็อบเจกต์ แต่สามารถเรียกผ่านชื่อคลาสได้โดยตรงอาทิเช่น Math.exp(4.0); เป็นการคำนวณหาค่า exponential

ของ 4.0

คลาส `Math` ได้ประกาศค่าคงที่สองตัวดังนี้

- `final static double E = 2.7182818284590452354;`
- `final static double PI = 3.14158265358979323846;`

ซึ่งการเรียกใช้ `Math.E` จะมีค่าเป็น 2.718281828... และ `Math.PI` จะมีค่าเป็น 3.141582653...

เมธอดอื่นๆ ในคลาส `Math` ที่สำคัญมีดังนี้

- `static int abs(int x);`
- `static long abs(long x);`
- `static float abs(float x);`
- `static double abs(double x);`
- `static double acos(double x);`
- `static double asin(double x);`
- `static double atan(double x);`
- `static double atan2(double x, double y);`
- `static double ceil(double x);`
- `static double cos(double x);`
- `static double exp(double x);`
- `static double floor(double x);`
- `static double log(double x);`
- `static int max(int x, int y);`
- `static long max(long x, long y);`
- `static float max(float x, float y);`
- `static double max(double x, double y);`
- `static int min(int x, int y);`
- `static long min(long x, long y);`
- `static float min(float x, float y);`
- `static double min(double x, double y);`
- `static double pow(double x, double y);`
- `static double random();`
- `static double rint(double x);`
- `static int round(float x);`
- `static long round(double x);`
- `static double sin(double x);`
- `static double sqrt(double x);`
- `static double tan(double x);`

2.8 คำสั่งอินพุตและเอาต์พุต

โปรแกรมคอมพิวเตอร์โดยทั่วไป จะต้องมีการอ่านข้อมูลเข้าเพื่อใช้ในการประมวลผล และจะมีการนำผลลัพธ์ที่ได้จากการประมวลผลออกมาแสดง ขบวนการในการอ่านข้อมูลเข้าและแสดงผลลัพธ์เรียกว่า อินพุต/เอาต์พุต (Input/Output) ภาษาจาวามีวิธีการหลายวิธีการในการจัดการกับอินพุต/เอาต์พุต และมีคลาสหลายคลาสที่เกี่ยวข้องกับขบวนการอินพุต/เอาต์พุต แต่วิธีการง่ายๆ วิธีการหนึ่งคือการใช้อินพุตมาตรฐาน (Standard Input) และการใช้เอาต์พุตมาตรฐาน (standard output) ภาษาจาวามีอ็อบเจกต์ที่เป็นอินพุต/เอาต์พุตมาตรฐานสามอ็อบเจกต์คือ `System.in`, `System.out` และ `System.err`

- อ็อบเจกต์ `System.in` เป็นอ็อบเจกต์ที่มีเมธอดสำหรับการอ่านข้อมูลทางอุปกรณ์อินพุตมาตรฐาน ซึ่งโดยทั่วไปก็คือคีย์บอร์ด
- อ็อบเจกต์ `System.out` เป็นอ็อบเจกต์ที่มีเมธอดสำหรับการแสดงข้อมูลออกทางอุปกรณ์เอาต์พุตมาตรฐานซึ่งโดยทั่วไปคือจอภาพ
- อ็อบเจกต์ `System.err` เป็นอ็อบเจกต์ที่มีเมธอดสำหรับการแสดงข้อผิดพลาด (Error) ออกทางอุปกรณ์ที่ใช้ในการแสดงข้อผิดพลาด ซึ่งโดยทั่วไปจะกำหนดเป็นจอภาพ

2.8.1 System.out.println()

อ็อบเจกต์ `System.out` หมายถึงอ็อบเจกต์ที่ชื่อ `out` ซึ่งเป็นคุณลักษณะของคลาส `System` อ็อบเจกต์ที่ชื่อ `out` เป็นอ็อบเจกต์ของคลาส `PrintStream` ที่มีเมธอดที่เกี่ยวข้องกับการแสดงผลอยู่หลายเมธอด แต่เมธอดที่นิยมใช้ทั่วไปคือ

```
print(String s)
```

และ

```
println(String s)
```

ซึ่งเมธอดทั้งสองใช้ในการแสดงผลข้อมูลที่มี argument เป็นชนิดข้อมูล `String`

เมธอด `println()` จะมีผลให้โปรแกรมขึ้นบรรทัดใหม่หลังจากพิมพ์ข้อมูลที่ต้องการแสดง เมธอด `print()` และ `println()` สามารถรับ argument ที่เป็นชนิดข้อมูลอื่น ซึ่งโปรแกรมจะแปลงเป็นชนิดข้อมูล `String` ให้โดยอัตโนมัติตัวอย่างเช่น คำสั่ง

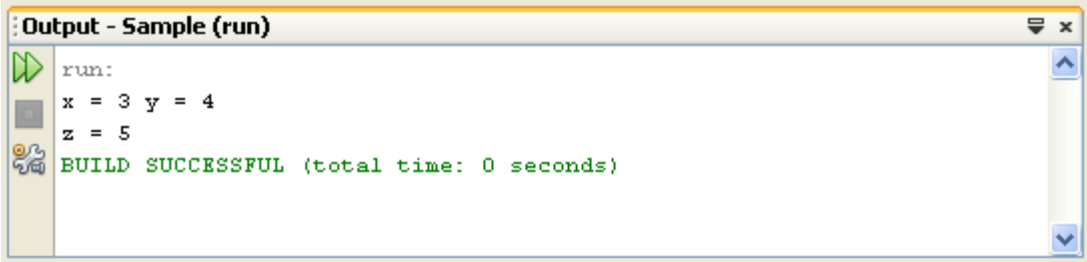
```
int x = 3;
System.out.println(x);
```

เป็นชุดคำสั่งในการพิมพ์ค่าของตัวแปร `x` ที่มีชนิดข้อมูลเป็น `int` ออกทางจอภาพ โดยโปรแกรมจะแปลงชนิดข้อมูล `int` ให้เป็น `String` แล้วจึงพิมพ์ข้อความออกมา โปรแกรมที่ 2.12 แสดงตัวอย่างของการใช้คำสั่ง

System.out.print() และ System.out.println() เพื่อแสดงข้อมูลชนิดต่างๆ ออกทางจอภาพดังแสดงในรูปที่ 2.13

<u>โปรแกรมที่ 2.12</u> <u>การใช้คำสั่งเพื่อพิมพ์ข้อความต่างๆ ออกทางจอภาพ</u>

```
public class PrintDemo {
      public void showDemo() {
            int x = 3, y = 4, z = 5;
            System.out.print("x = "+x);
            System.out.println(" y = "+y);
            System.out.println("z = "+z);
      }
}

-----------------------------------------------------------------
public class Main {

      public static void main(String args[]) {
          PrintDemo obj = new PrintDemo();
          obj.showDemo();
      }
}
```



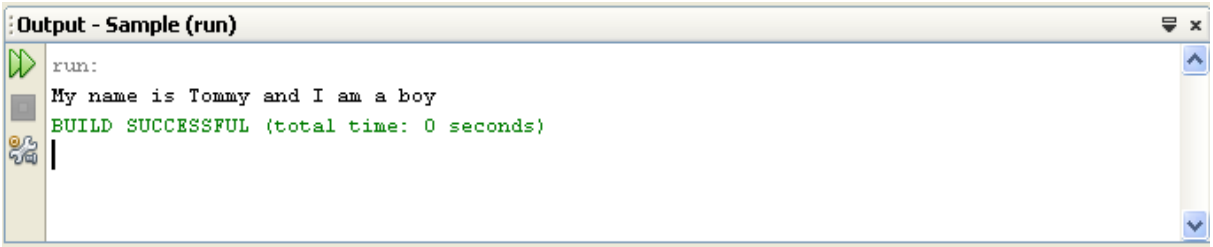
รูปที่ 2.13 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 2.12

2.8.2 การรับข้อมูลเข้ามาทาง Command Line

เมธอด main() ที่เป็นตำแหน่งเริ่มต้นการทำงานของโปรแกรมจาวาประยุกต์ จะมี argument เป็น String args[] ซึ่งหมายถึงตัวแปรอะเรย์ (จะกล่าวถึงในบทที่ 8) args ที่มีชนิดข้อมูลเป็น String ซึ่งสามารถรับ argument ที่ส่งผ่านมาจาก command line ได้โดยมีตัวอย่างของการทำงานดังรูปที่ 2.14 ซึ่งเป็นการกำหนดให้ args[0] มีค่าเป็น Tommy และตัวแปร args[1] มีค่าเป็น boy โปรแกรมที่ 2.13 เป็นตัวอย่างการป้อนอินพุตทาง command line แล้วแสดงผลออกทางจอภาพ

<u>โปรแกรมที่ 2.13</u> <u>โปรแกรมแสดงการป้อนอินพุตทาง command line</u>

```
public class InputDemo {
      public static void main(String args[]) {
            System.out.println("My name is "+ args[0] +
                                " and I am a " + args[1]);
      }
}
```



รูปที่ 2.14 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 2.12

สรุปเนื้อหาของบท

- ภาษาจาวาเป็นภาษาเชิงอ็อบเจกต์ ที่มีคำนิยามที่สำคัญสองคำคือคลาสและอ็อบเจกต์
- คลาสเปรียบเสมือนพิมพ์เขียวของอ็อบเจกต์ อ็อบเจกต์ที่ถูกสร้างมาจากคลาส
- คลาสจะประกอบด้วยคุณลักษณะและเมธอด
- คลาสจะถูกเรียกใช้งานได้เมื่อมีการสร้างอ็อบเจกต์ของคลาสโดยเรียกใช้คำสั่ง `new` แล้วเรียกใช้เมธอด
- คอมเม็นต์ คือข้อความที่แทรกอยู่ภายในโปรแกรม ซึ่งคอมไพเลอร์จะไม่แปลข้อความนี้ให้เป็นส่วนหนึ่งของโปรแกรม
- identifier คือชื่อที่ตั้งขึ้นในภาษาจาวา ซึ่งอาจเป็นชื่อของคลาส ชื่อของตัวแปร ชื่อของเมธอด หรือชื่อของค่าคงที่ ซึ่งจะต้องเป็นไปตามกฎการตั้งชื่อ
- การตั้งชื่อในภาษาจาวา
  - สำหรับคลาส นิยมขึ้นต้นด้วยตัวอักษรพิมพ์ใหญ่
  - สำหรับเมธอดและตัวแปร นิยมขึ้นต้นด้วยตัวอักษรพิมพ์เล็ก
  - ถ้าชื่อที่ตั้งขึ้นมีมากกว่า 1 คำ นิยมขึ้นต้นคำใหม่ด้วยตัวอักษรพิมพ์ใหญ่
  - ต้องไม่ตรงกับคีย์เวิร์ด
- คีย์เวิร์ด คือชื่อที่มีความหมายพิเศษในภาษาจาวา คอมไพเลอร์ของภาษาจาวาจะเข้าใจความหมายและคำสั่งที่จะต้องดำเนินการสำหรับคีย์เวิร์ดแต่ละตัว

- ข้อมูลค่าคงที่ คือคำที่ใช้แสดงข้อมูลที่เป็นตัวเลข ตัวอักขระ ข้อความ หรือค่าทางตรรกะ
- ชนิดข้อมูลในภาษาจาวาแบ่งเป็นสองประเภท คือชนิดข้อมูลแบบพื้นฐาน และชนิดข้อมูลแบบอ้างอิง
- ชนิดข้อมูลแบบพื้นฐานที่ใช้ในภาษาจาวามีทั้งหมด 8 ชนิดคือ `char, byte, short, int, long, float, double` และ `boolean`
- ข้อมูลที่เก็บอยู่ในโปรแกรมแบ่งเป็นสองประเภท คือตัวแปรและค่าคงที่ ซึ่งตัวแปรคือข้อมูลที่สามารถเปลี่ยนแปลงค่าได้ในโปรแกรมโดยใช้คำสั่งกำหนดค่า ส่วนค่าคงที่คือข้อมูลที่กำหนดค่าได้เพียงครั้งเดียว และไม่สามารถเปลี่ยนแปลงค่าได้ในโปรแกรม
- ตัวแปรที่มีการประกาศชนิดข้อมูลแล้วสามารถที่จะกำหนดหรือเปลี่ยนแปลงค่าได้โดยใช้คำสั่งกำหนดค่า
- การประกาศค่าคงที่ในภาษาจาวาทำได้โดยการใส่คีย์เวิร์ด `final` หน้าคำสั่งประกาศชื่อ
- ค่าคงที่หรือตัวแปรที่อยู่ภายในบล็อกของเมธอด จะมีขอบเขตการใช้งานอยู่ภายในบล็อกเท่านั้น
- ตัวดำเนินการที่ใช้ในภาษาจาวามีทั้งหมด 4 แบบคือ
  - ตัวดำเนินการทางคณิตศาสตร์ : +, -, *, /, %, +=, -=, *=, /=, %=, ++ และ --
  - ตัวดำเนินการทางตรรกศาสตร์ : !, &&, &, || และ |
  - ตัวดำเนินการแบบสัมพันธ์ : <, <=, >, >=, == และ !=
  - ตัวดำเนินการแบบบิต : ~, &, |, ^, >>, >>>, <<
- กรณีที่นิพจน์ใดๆ มีตัวดำเนินการมากกว่าหนึ่งตัว ภาษาจาวาจะจัดลำดับความสำคัญของตัวดำเนินการ เพื่อคำนวณหาผลลัพธ์ตามลำดับความสำคัญของตัวดำเนินการ
- ความแตกต่างระหว่างการแปลงข้อมูลชนิดอัตโนมัติและ Typecasting
  - การแปลงข้อมูลโดยอัตโนมัติคือ การที่ชนิดข้อมูลที่มีขนาดเล็กกว่าถูกแปลงให้เป็นชนิดข้อมูลที่มีขนาดใหญ่กว่าโดยอัตโนมัติ
  - Typecasting คือการที่จะต้องระบุชนิดข้อมูลที่มีขนาดเล็กกว่าให้กับนิพจน์ที่มีชนิดข้อมูลขนาดใหญ่กว่า
- ชนิดข้อมูลแบบอ้างอิงซึ่งก็คืออ็อบเจกต์ในภาษาจาวา โดยแบ่งออกเป็นสองแบบคือชนิดข้อมูลที่เป็นคลาส และชนิดข้อมูลที่เป็นอะเรย์
- ข้อมูลที่เก็บในหน่วยความจำของชนิดข้อมูลแบบอ้างอิง จะเป็นตำแหน่งอ้างอิงที่เก็บข้อมูลในหน่วยความจำ แต่ในกรณีของชนิดข้อมูลแบบพื้นฐาน ข้อมูลที่เก็บในหน่วยความจำจะเป็นค่าของข้อมูลจริงๆ
- `String` เป็นชนิดข้อมูลแบบคลาส ไม่ใช่ชนิดข้อมูลแบบพื้นฐาน
- คลาส `Math` จะมีเมธอดต่างๆ ในการจัดการกับฟังก์ชันหรือคำสั่งทางคณิตศาสตร์ต่างๆ
- คำสั่งที่ใช้ในการแสดงผลลัพธ์ของข้อมูลในภาษาจาวาคือคำสั่ง `System.out.println()`

บทที่ 3 โครงสร้างควบคุม

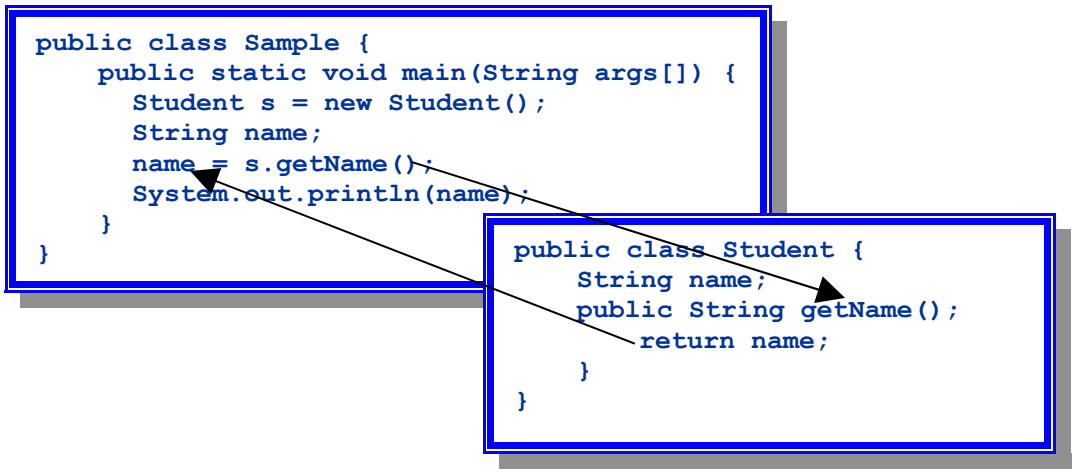
เนื้อหาในบทนี้เป็นการแนะนำคำสั่งที่เกี่ยวข้องกับโครงสร้างควบคุมในภาษาจาวา ซึ่งเป็นการควบคุมลำดับการทำงานของคำสั่งต่างๆ ในโปรแกรมภาษาจาวา โดยจะกล่าวถึงคำสั่งโครงสร้างควบคุมสองประเภทคือ คำสั่งโครงสร้างแบบเลือกทำซึ่งได้แก่คำสั่ง `if`,`if..else` และ `switch` และคำสั่งโครงสร้างแบบทำซ้ำซึ่งได้แก่คำสั่ง `while`, `do..while` และ `for` และในตอนท้ายของบทนี้จะกล่าวถึงคำสั่งโครงสร้างควบคุมแบบซ้อน

3.1 คำสั่งโครงสร้างควบคุม

คำสั่งโครงสร้างควบคุม (Control Structure) เป็นคำสั่งที่ใช้ในการกำหนดลำดับการทำงานของคำสั่งต่างๆ ภาษาจาวามีโครงสร้างควบคุมสามรูปแบบคือ

1. โครงสร้างแบบตามลำดับ (Sequential Structure)
2. โครงสร้างแบบเลือกทำ (Selection Structure)
3. โครงสร้างแบบทำซ้ำ (Repetition Structure)

โดยทั่วไปคำสั่งในภาษาจาวาจะมีโครงสร้างควบคุมแบบตามลำดับ โดยจะทำงานตามลำดับของคำสั่งที่มีอยู่ในโปรแกรม ซึ่งจะเริ่มทำงานจากเมธอด `main()` โดยจะทำงานจากคำสั่งแรกของเมธอด `main()` และทำงานเรียงตามลำดับคำสั่งต่อไปเรื่อยๆ กรณีที่มีการเรียกใช้เมธอดของอ็อบเจกต์ โปรแกรมจะเข้าไปทำคำสั่งภายในเมธอดนั้น และเมื่อสิ้นสุดคำสั่งสุดท้ายของเมธอดก็จะกลับมาทำคำสั่งที่เรียกใช้เมธอด ดังแสดงในรูปที่ 3.1



รูปที่ 3.1 ขั้นตอนการทำงานตามลำดับของคำสั่งที่มีอยู่ในโปรแกรม

3.2 โครงสร้างแบบเลือกทำ

โครงสร้างแบบเลือกทำเป็นการให้เลือกทำชุดคำสั่งในกรณีที่นิพจน์ตรรกศาสตร์มีค่าเป็นจริงตามเงื่อนไข ซึ่งชุดคำสั่งโครงสร้างแบบเลือกทำจะประกอบไปด้วยคำสั่งดังต่อไปนี้

- คำสั่ง `if`
- คำสั่ง `if..else`
- คำสั่ง `if` แบบซ้อน (nested if)
- คำสั่ง `switch`

3.2.1 คำสั่ง `if`

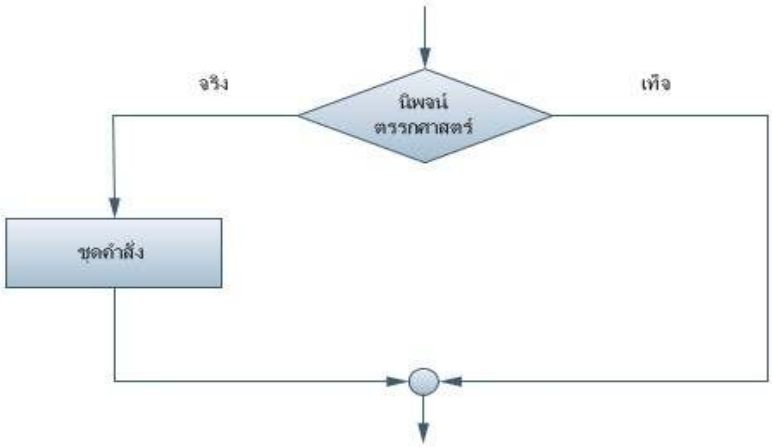
คำสั่ง `if` จะมีรูปแบบดังนี้

```
if (logical expression) {
    statements
}
```

โดยที่

- `logical expression` คือนิพจน์ตรรกศาสตร์ที่ให้ผลลัพธ์เป็นข้อมูลค่าคงที่ชนิด `boolean`
- `statements` คือชุดคำสั่งใดๆ

ชุดคำสั่งที่อยู่ในบล็อก { } จะทำงานในกรณีที่นิพจน์ตรรกศาสตร์ให้ค่าเป็นจริง ซึ่งคำสั่ง `if` สามารถแสดงลำดับการทำงานเป็นโฟว์ชาร์ต (flowchart) ได้ดังแสดงในรูปที่ 3.2



รูปที่ 3.2 โฟว์ชาร์ตของคำสั่ง `if`

ตัวอย่างของการใช้คำสั่ง `if` มีดังนี้

```
if (radius >= 0) {
      area = radius*radius*Math.PI;
      System.out.println(area);
}
```

จากตัวอย่างนี้ชุดคำสั่งที่อยู่ในบล็อก (คำสั่งกำหนดค่าตัวแปร `area` และคำสั่งแสดงผล) จะทำงานในกรณีที่นิพจน์ `radius >=0` เป็นจริง กล่าวคือตัวแปร `radius` มีค่ามากกว่าหรือเท่ากับ 0

กรณีที่ชุดคำสั่งในบล็อก { } มีเพียงคำสั่งเดียวเราสามารถที่จะตัดเครื่องหมาย { } ออกไปได้อาทิเช่น คำสั่ง

```
if((x > 0) && (x < 10)) {
     System.out.println(x);
}
```

สามารถเขียนใหม่ได้เป็น

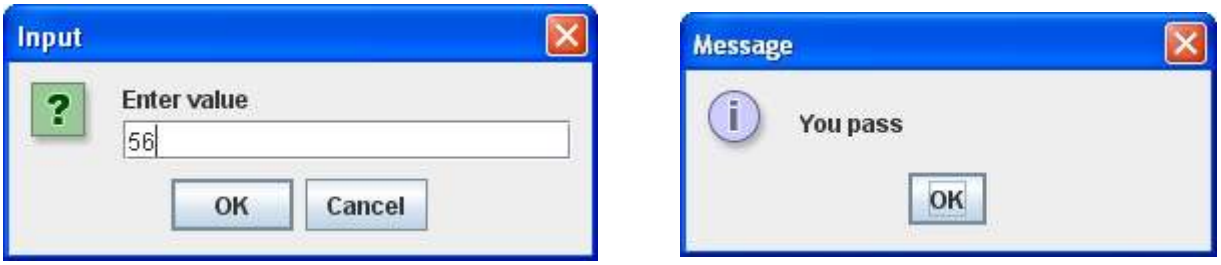
```
if((x > 0) && (x < 10))
     System.out.println(x);
```

แต่อย่างไรก็ตามการเขียนโปรแกรมที่ดีควรใช้บล็อก { } เพื่อให้เกิดความเข้าใจง่าย และง่ายต่อการเพิ่มคำสั่ง (กล่าวคือถ้ามีมากกว่าหนึ่งคำสั่งต้องใช้บล็อก { } เสมอ)

<u>โปรแกรมที่ 3.1</u> <u>ตัวอย่างการใช้คำสั่ง `if`</u>

```
import javax.swing.JOptionPane;

public class SampleIf {
    public void showDemo() {
        String inputStr = JOptionPane.showInputDialog("Enter value");
        int score = Integer.parseInt(inputStr);
        if (score >= 50) {
            JOptionPane.showMessageDialog(null, "You pass");
            // null to tell that there is no parent frame
        }
    }
}
-----------------------------------------------------------------------
public class Main {

    public static void main(String[] args) {
        SampleIf obj = new SampleIf();
        obj.showDemo();
    }
}
```

โปรแกรมที่ 3.1 เป็นตัวอย่างของโปรแกรมที่ใช้คำสั่ง `if` โดยโปรแกรมจะรับตัวเลขเข้ามาทางไดอะล็อก ซึ่งจะเป็นข้อมูลชนิด `String` ที่จะเก็บอยู่ในตัวแปรที่ชื่อ `inputStr` โดยการเรียกใช้เมธอดที่ชื่อ `showInputDialog` ของคลาส `JOptionPane` แล้วจึงส่งค่าเป็น argument ของเมธอด `Integer.parseInt()` เพื่อทำการแปลงชนิดข้อมูล `String` ให้เป็นชนิดข้อมูล `int` แล้วกำหนดไว้ในตัวแปร `score` ซึ่งคำสั่ง `if` จะตรวจสอบว่าถ้าตัวแปร `score` มีค่ามากกว่าหรือเท่ากับ 50 จะพิมพ์ข้อความ `You pass` ออกมาทางไดอะล็อกที่เรียกใช้เมธอด `showMessageDialog` ของคลาส `JOptionPane` ตัวอย่างของผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.1 แสดงดังรูปที่ 3.3



รูปที่ 3.3 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.1

<u>คลาส JOptionPane</u>

คลาส `JOptionPane` เป็นคลาสที่กำหนดมาในภาษาจาวาเพื่อใช้ในการสร้าง DialogBox แบบมาตรฐาน คลาสนี้เป็นคลาสขนาดใหญ่ที่มีเมธอดหลายๆ เมธอดแต่เมธอดที่สำคัญคือ

- `showConfirmDialog` เป็นไดอะล็อกเพื่อยืนยันคำถามเช่น yes/no/cancel
- `showInputDialog` เป็นไดอะล็อกสำหรับรับค่าข้อมูลชนิด String
- `showMessageDialog` เป็นไดอะล็อกเพื่อแสดงข้อความออกมา

3.2.2 คำสั่ง `if..else`

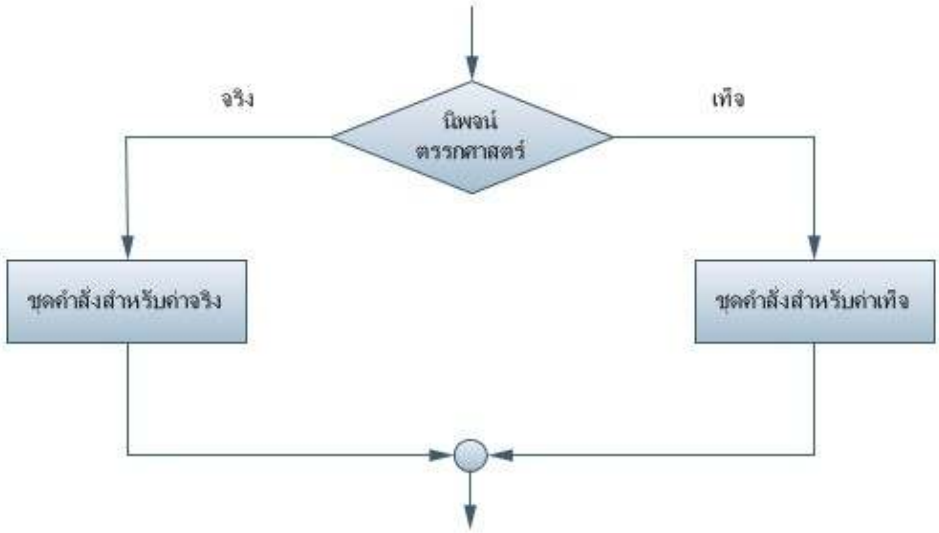
คำสั่ง `if` จะไม่มีการทำคำสั่งใด ถ้านิพจน์ตรรกศาสตร์มีค่าเป็นเท็จ ดังนั้นถ้าต้องการทำคำสั่งบางคำสั่งเมื่อนิพจน์ตรรกศาสตร์มีค่าเป็นเท็จ จะต้องใช้คำสั่ง `if..else` แทน ซึ่งมีรูปแบบของคำสั่งดังนี้

```
if (logical expression) {
    true statements
} else {
    false statements
}
```

โดยที่

- `true statements` คือชุดคำสั่งสำหรับค่าจริง
- `false statements` คือชุดคำสั่งสำหรับค่าเท็จ

โดยมีโฟว์ชาร์ตของการทำงานของคำสั่งนี้ดังแสดงในรูปที่ 3.4



รูปที่ 3.4 โฟว์ชาร์ตของคำสั่ง if..else

ตัวอย่างของการใช้คำสั่ง `if..else` มีดังนี้

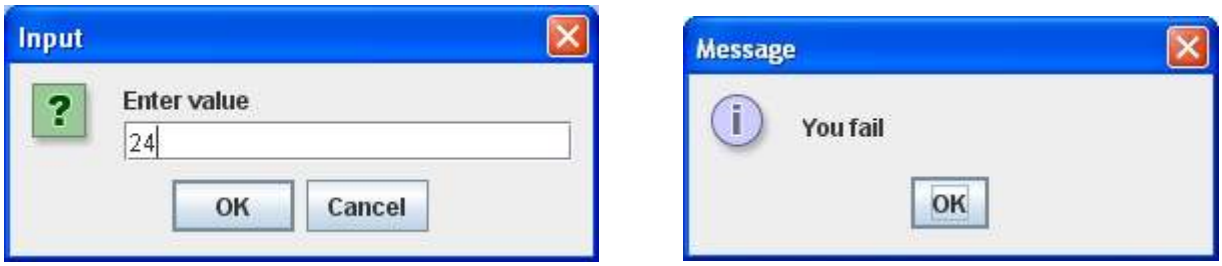
```
if (radius >= 0) {
      area = radius*radius*Math.PI;
      System.out.println(area);
} else {
      System.out.println("Negative radius");
}
```

จากตัวอย่างนี้ถ้าตัวแปร `radius` มีค่ามากกว่าหรือเท่ากับ 0 (นิพจน์มีค่าเป็นจริง) โปรแกรมจะทำการคำนวณค่า กำหนดค่าและพิมพ์ค่าของตัวแปร `area` ออกมา แต่ถ้าตัวแปร `radius` มีค่าน้อยกว่า 0 (นิพจน์เป็นเท็จ) โปรแกรมจะแสดงข้อความ `Negative radius` ออกมาแทน

โปรแกรมที่ 3.2 ตัวอย่างการใช้คำสั่ง `if..else`

```
import javax.swing.JOptionPane;

public class SampleIfElse {
    public void showDemo() {
        String inputStr = JOptionPane.showInputDialog("Enter value");
        int score = Integer.parseInt(inputStr);
        if (score >= 50) {
            JOptionPane.showMessageDialog(null, "You pass");
        } else {
            JOptionPane.showMessageDialog(null, "You fail");
        }
    }
}
```

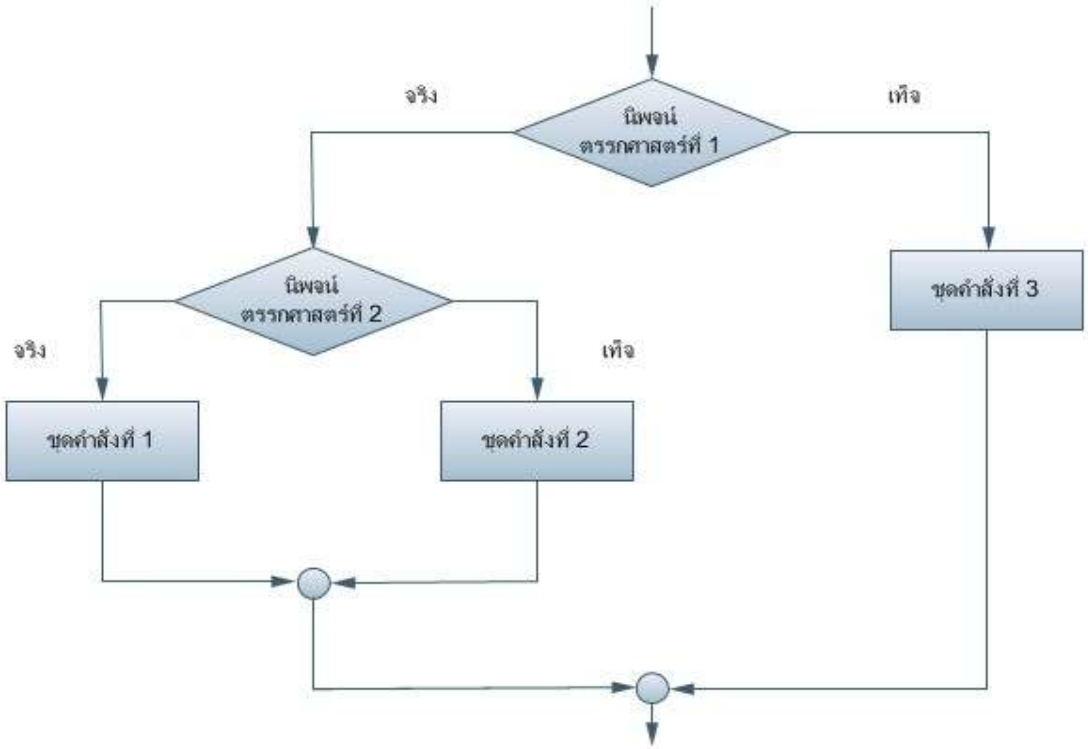
โปรแกรมที่ 3.2 เป็นตัวอย่างของการใช้คำสั่ง `if..else` โดยโปรแกรมจะพิมพ์ข้อความ `You pass` ถ้าค่าของตัวแปร `score` ที่รับเข้ามาทางไดอะล็อก มีค่ามากกว่าหรือเท่ากับ 50 และจะพิมพ์ข้อความ `You fail` ถ้าค่าของ `score` มีค่าน้อยกว่า 50 โดยมีตัวอย่างของผลลัพธ์ดังแสดงในรูปที่ 3.5



รูปที่ 3.5 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.2

3.2.3 คำสั่ง `if` แบบซ้อน

คำสั่ง `if` หรือ `if..else` สามารถที่จะซ้อนอยู่ข้างในคำสั่ง `if` หรือ `if..else` อื่นได้อาทิเช่น โฟว์ชาร์ตที่แสดงในรูปที่ 3.6 ซึ่งมีขั้นตอนการทำงานคือ โปรแกรมจะทำชุดคำสั่งที่ 3 ถ้านิพจน์ตรรกศาสตร์ที่ 1 เป็นเท็จ แต่ถ้าเป็นจริงจะตรวจสอบผลลัพธ์ของนิพจน์ตรรกศาสตร์ที่ 2 ซึ่งถ้ามีค่าเป็นจริง จะทำชุดคำสั่งที่ 1 แต่ถ้ามีค่าเป็นเท็จจะทำชุดคำสั่งที่ 2



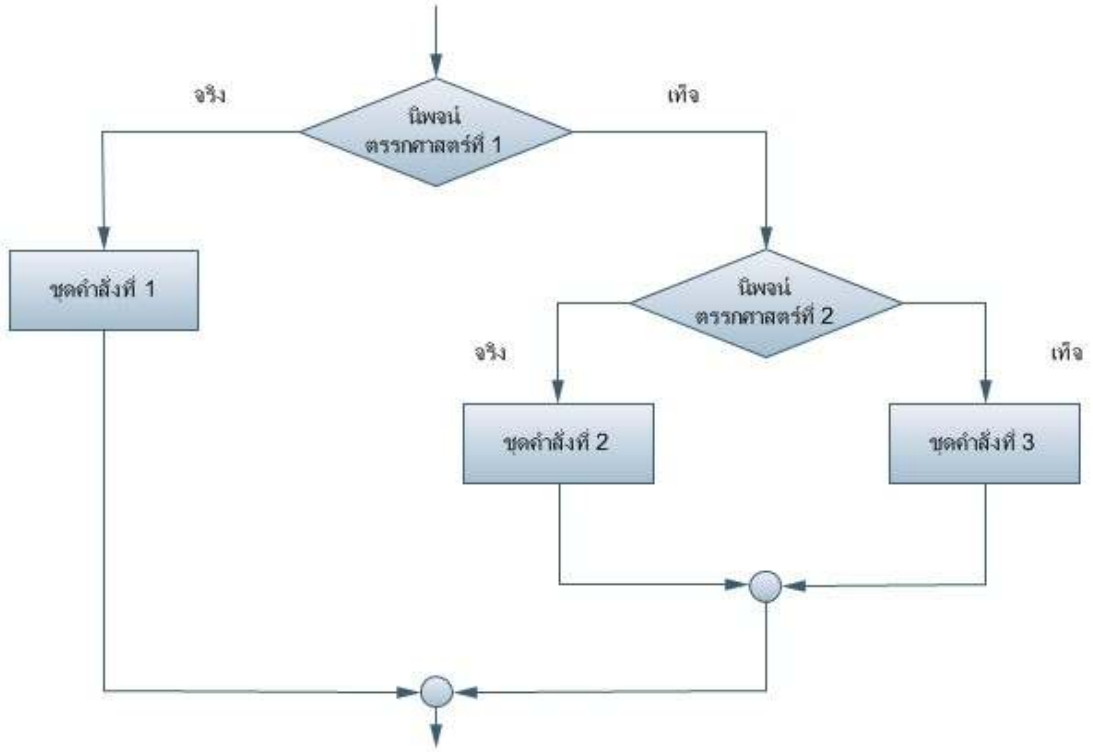
รูปที่ 3.6 โฟว์ชาร์ตของคำสั่ง `if` แบบซ้อน

จากโฟว์ชาร์ตในรูปที่ 3.6 สามารถที่จะเขียนเป็นคำสั่ง `if` แบบซ้อนได้ดังนี้

```
if (logical expression 1) {
     if (logical expression 2) {
         statements 1
     } else {
         statements 2
     }
} else {
     statements 3
}
```

ตัวอย่างเช่น

```
if (radius >= 0) {
     area = radius*radius*Math.PI;
     if(area >= 500.0) {
        System.out.println("Big Circle");
     } else {
        System.out.println("Small Circle");
     }
} else {
    System.out.println("Negative radius");
}
```



รูปที่ 3.7 โฟว์ชาร์ตของคำสั่ง `if` แบบซ้อน

โครงสร้างเลือกทำแบบซ้อนยังมีกรณีที่เป็นดังโฟว์ชาร์ตในรูปที่ 3.7 ซึ่งจะมีขั้นตอนการทำงานคือ โปรแกรมจะทำชุดคำสั่งที่ 1 ถ้านิพจน์ตรรกศาสตร์ที่ 1 เป็นจริง แต่ถ้าเป็นเท็จจะตรวจสอบผลลัพธ์ของนิพจน์ตรรกศาสตร์ที่ 2 ซึ่งถ้ามีค่าเป็นจริง จะทำชุดคำสั่งที่ 2 แต่ถ้ามีค่าเป็นเท็จจะทำชุดคำสั่งที่ 3 ซึ่งจะมีรูปแบบของคำสั่งดังนี้

```
if (logical expression 1) {
        statements 1
} else {
        if (logical expression 2) {
                statements 2
        } else {
                statements 3
        }
}
```

ในกรณีนี้สามารถที่เขียนคำสั่ง `if` แบบซ้อนได้ใหม่ ในรูปของ `if..else if..else` โดยมีรูปแบบของคำสั่งดังนี้

```
if (logical expression 1) {
        statements 1
} else if (logical expression 2) {
        statements 2
} else {
        statements 3
}
```

ตัวอย่างของการใช้คำสั่ง `if..else if..else` มีดังนี้

```
if (score >= 80) {
     System.out.println('A');
} else if(score >= 70) {
     System.out.println('B');
} else if(score >= 60) {
     System.out.println('C');
} else if(score >= 50) {
            System.out.println('D');
} else {
     System.out.println('F');
}
```

ตัวอย่างนี้จะพิมพ์ตัวอักษร `A` ถ้าตัวแปร `score` มีค่ามากกว่าหรือเท่ากับ 80 พิมพ์ตัวอักษร `B` ถ้าตัวแปร `score` มีค่าตั้งแต่ 70 จนถึง 79 พิมพ์ตัวอักษร `C` ถ้าตัวแปร `score` มีค่าตั้งแต่ 60 จนถึง 69 พิมพ์ตัวอักษร `D` ถ้าตัวแปร `score` มีค่าตั้งแต่ 50 จนถึง 59 และพิมพ์ตัวอักษร `F` ถ้าตัวแปร `score` มีค่าต่ำกว่า 50

<u>โปรแกรมที่ 3.3</u> <u>การใช้คำสั่ง `if..else if..else`</u>

```
import javax.swing.JOptionPane;

public class SampleIfElseIf {
    public void showDemo() {
        String inputStr = JOptionPane.showInputDialog("Enter value");
        int x = Integer.parseInt(inputStr);
        if (x == 1) {
            JOptionPane.showMessageDialog(null, "Value is one");
        } else if (x == 2) {
            JOptionPane.showMessageDialog(null, "Value is two");
        } else {
            JOptionPane.showMessageDialog(null, "Other than 1 and 2 ");
        }
    }
}
```

โปรแกรมที่ 3.3 แสดงตัวอย่างการใช้คำสั่ง `if..else if..else` โปรแกรมนี้จะอ่านค่าที่ป้อนเข้ามาทางไดอะล็อกบล็อก โดยโปรแกรมจะพิมพ์ข้อความว่า "`Value is one`" ถ้าค่าที่ป้อนเข้ามาเป็น 1 พิมพ์ข้อความว่า

"Value is two" ถ้าค่าที่ป้อนเข้ามาเป็น 2 และพิมพ์ข้อความว่า "Other than 1 or 2" ถ้าค่าที่ป้อนเข้ามาเป็นค่าอื่นๆ

3.2.4 คำสั่ง switch

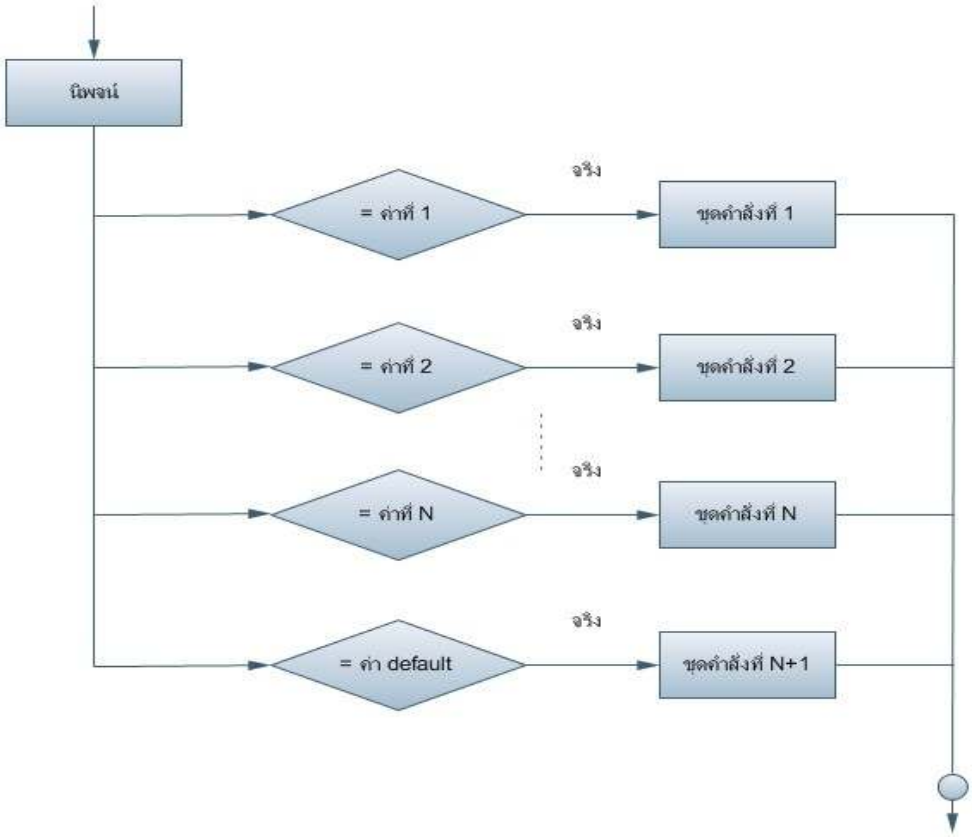
คำสั่ง switch เป็นคำสั่งโครงสร้างแบบเลือกทำ โดยมีรูปแบบดังนี้

```
switch (expression) {
    case value 1 : statements 1
                   break;
    case value 2 : statements 2
                   break;
                     :
    case value N : statements N
                   break;
    default : statements N+1
              break;
 }
```

โดยที่

- expression คือนิพจน์ที่ต้องมีชนิดข้อมูลเป็น char, byte, short หรือ int เท่านั้น
- value 1 .. value N คือข้อมูลค่าคงที่ซึ่งมีชนิดข้อมูลที่สอดคล้องกับชนิดข้อมูลของ expression

คำสั่ง switch จะตรวจสอบค่าของนิพจน์ที่อาจเป็นชนิดข้อมูล char, byte, short หรือ int กรณีที่ค่าของนิพจน์มีค่าตรงกับค่าที่ 1 โปรแกรมจะทำชุดคำสั่งที่ 1 ถ้ามีค่าตรงกับค่าที่ 2 ก็จะทำชุดคำสั่งที่ 2 ถ้ามีค่าตรงกับค่าที่ N ก็จะทำชุดคำสั่งที่ N ส่วนกรณีที่มีค่าไม่ตรงกับค่าใดๆ เลยในคำสั่ง case ก็จะทำชุดคำสั่งใน default คือชุดคำสั่งที่ N+1 ซึ่งแสดงขั้นตอนการทำงานได้ดังโฟว์ชาร์ตในรูปที่ 3.8



รูปที่ 3.8 โฟว์ชาร์ตของคำสั่ง switch

ตัวอย่างของการใช้คำสั่ง `switch` มีดังนี้

```
switch (x) {
     case 1 : System.out.println("One");
              break;
     case 2 : System.out.println("Two");
              break;
     case 3 : System.out.println("Three");
              break;
   default : System.out.println("Other");
  }
```

ตัวอย่างนี้จะพิมพ์ข้อความตามค่าของ `x` ตั้งแต่ 1 ถึง 3 และจะพิมพ์ข้อความว่า `other` ถ้าค่าของ `x` เป็นค่าอื่นๆ คำสั่ง `switch` มีข้อกำหนดต่างๆ ดังนี้

- นิพจน์ต้องมีชนิดข้อมูลเป็น `char,byte,short` หรือ `int` เท่านั้น
- ชนิดข้อมูลของนิพจน์และค่าที่ 1 ถึง N ต้องเป็นชนิดข้อมูลเดียวกัน
- คำสั่ง `break` จะหยุดการทำงานภายในบล็อก { } ถ้าไม่มีคำสั่ง `break` โปรแกรมจะทำคำสั่งถัดไป ตัวอย่าง

เช่น

```
switch (x) {
    case 1 : System.out.println("One");
    case 2 : System.out.println("Two");
             break;
    default : System.out.println("Other");
}
```

จะพิมพ์ข้อความ `One` ตามด้วย `Two` ในกรณีที่ `x` มีค่าเป็น 1

- เงื่อนไข `default` จะมีหรือไม่มีก็ได้

โปรแกรมที่ 3.4 แสดงตัวอย่างของการใช้คำสั่ง `switch` ซึ่งจะได้ผลลัพธ์เป็นข้อความว่า `Value is one` ถ้าตัวเลขที่ป้อนเข้ามามีค่าเป็น 1 เป็นข้อความว่า `Value is two` ถ้าตัวเลขที่ป้อนเข้ามามีค่าเป็น 2 และเป็นข้อความว่า `Other than 1 or 2` ถ้าตัวเลขที่ป้อนเข้ามามีค่าอื่นๆ

<u>โปรแกรมที่ 3.4 ตัวอย่างการใช้คำสั่ง `switch`</u>

```
import javax.swing.JOptionPane;

public class SampleSwitch {
    public void showDemo() {
        String inputStr = JOptionPane.showInputDialog("Enter value");
        int x = Integer.parseInt(inputStr);
        switch (x) {
           case 1: JOptionPane.showMessageDialog(null, "Value is one");
                   break;
           case 2: JOptionPane.showMessageDialog(null, "Value is two");
                   break;
           default: JOptionPane.showMessageDialog(null, "Other than 1 and 2");
        }
    }
}
```

3.3 โครงสร้างแบบทำซ้ำ

โครงสร้างแบบทำซ้ำ เป็นคำสั่งที่ใช้ในการสั่งให้ชุดคำสั่งใดๆ ทำซ้ำหลายครั้งตามเงื่อนไขที่ระบุ ซึ่งชุดคำสั่งโครงสร้างแบบทำซ้ำประกอบด้วยคำสั่ง

- คำสั่ง `while`
- คำสั่ง `do..while`
- คำสั่ง `for`

3.3.1 คำสั่ง while

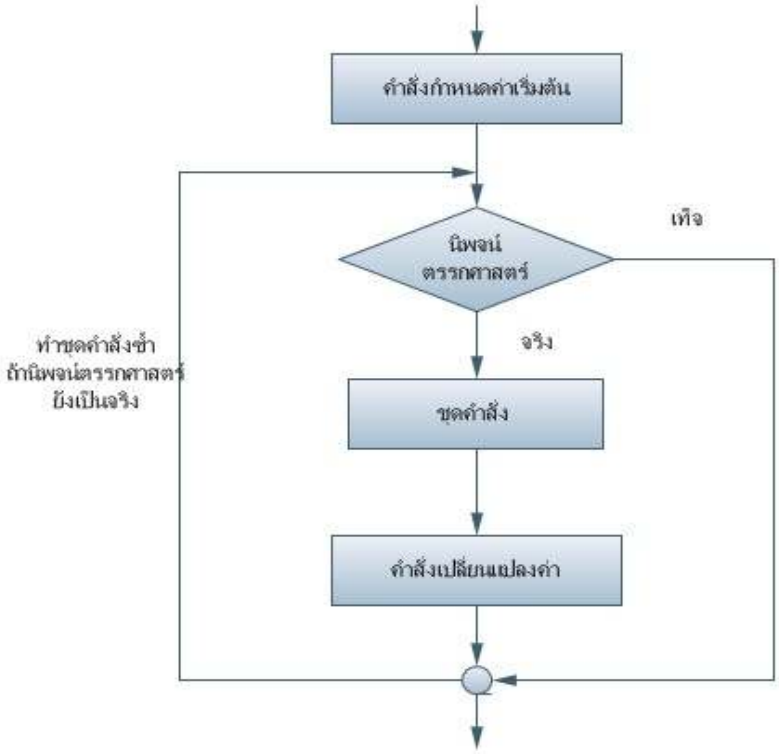
คำสั่ง `while` จะมีรูปแบบดังนี้

```
initial statements
while (logical expression) {
          statements
        update statements
}
```

โดยที่

- `initial statements` คือคำสั่งที่ใช้ในการกำหนดค่าเริ่มต้น
- `logical expression` คือนิพจน์ตรรกศาสตร์
- `update statements` คือคำสั่งที่ใช้ในการเปลี่ยนแปลงค่า

คำสั่ง `while` จะทำชุดคำสั่งและคำสั่งเปลี่ยนแปลงค่าที่อยู่ในบล็อก { } ตราบใดที่นิพจน์ตรรกศาสตร์ยังมีค่าเป็นจริง คำสั่งที่อยู่ในบล็อก { } จะต้องมีคำสั่งในการเปลี่ยนแปลงค่า เพื่อเปลี่ยนค่านิพจน์ตรรกศาสตร์ให้มีค่าเป็นเท็จ มิฉะนั้นแล้วโปรแกรมจะทำคำสั่งที่อยู่ในบล็อกแบบไม่มีที่สิ้นสุด คำสั่ง `while` สามารถเขียนเป็นโฟว์ชาร์ตได้ ดังแสดงในรูปที่ 3.9



รูปที่ 3.9 โฟว์ชาร์ตของคำสั่ง while

ตัวอย่างของการใช้คำสั่ง `while` เพื่อพิมพ์ข้อความว่า Hello World สิบครั้ง มีดังนี้

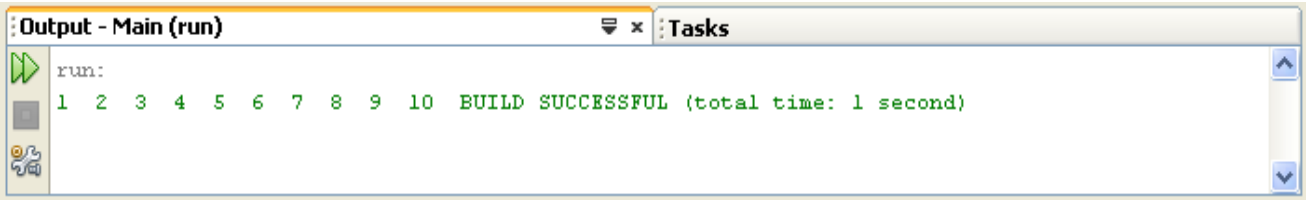
```
int count = 1;
while(count <= 10) {
      System.out.println("Hello World");
      count++;
}
```

ตัวอย่างนี้มีคำสั่ง `int count = 1;` เป็นคำสั่งกำหนดค่าเริ่มต้น โดยมีนิพจน์ตรรกศาสตร์คือ `count <= 10` ซึ่งโปรแกรมจะทำคำสั่งในบล็อก { } ตราบเท่าที่นิพจน์ตรรกศาสตร์มีค่าเป็นจริง (กล่าวคือค่าของตัวแปร `count` น้อยกว่าหรือเท่ากับ `10`) ส่วนคำสั่ง `count++;` เป็นคำสั่งเปลี่ยนแปลงค่าของตัวแปร `count` โดยจะเพิ่มค่าทีละหนึ่งในแต่ละรอบจนกระทั่งมีค่าเป็น `11` ซึ่งจะมีผลทำให้นิพจน์ `count <= 10` มีค่าเป็นเท็จ หากไม่มีคำสั่ง `count++;` ในบล็อก { } จะทำให้ค่า `count` มีค่าเป็น `1` ตลอด และจะทำให้นิพจน์ `count <= 10` มีค่าเป็นจริงเสมอ ซึ่งก็จะทำให้โปรแกรมพิมพ์ข้อความ Hello World โดยไม่มีที่สิ้นสุด

<u>โปรแกรมที่ 3.5</u> <u>ตัวอย่างการใช้คำสั่ง `while`</u>

```
public class SampleWhile {
    public void showDemo() {
        int i = 1;
        while(i <= 10) {
           System.out.print(i+"  ");
           i++;
        }
    }
}
```

โปรแกรมที่ 3.5 แสดงตัวอย่างการใช้คำสั่ง while เพื่อพิมพ์ค่าตัวเลข 1 ถึง 10 ซึ่งผลลัพธ์ของโปรแกรมเป็นดังแสดงในรูปที่ 3.10



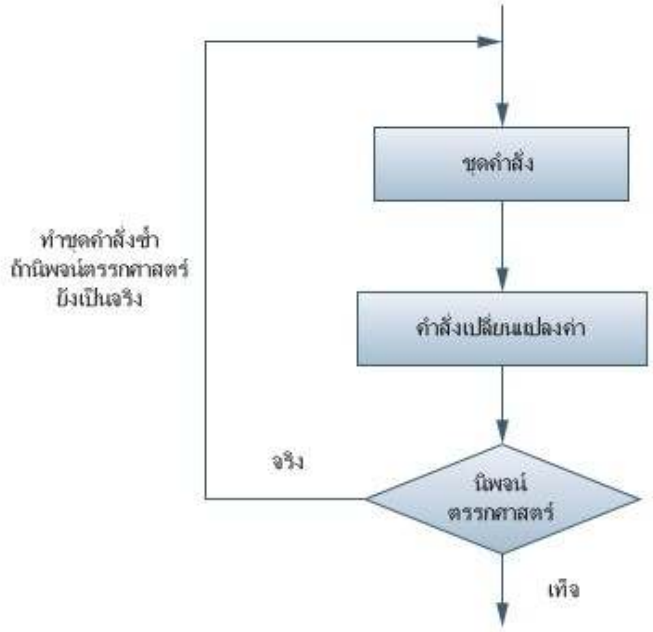
รูปที่ 3.10 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.5

3.3.2 คำสั่ง do..while

คำสั่ง do..while เป็นคำสั่งที่เป็นโครงสร้างแบบทำซ้ำที่มีการทำงานคล้ายกับคำสั่ง while โดยมีรูปแบบ คำสั่งดังนี้

```
initial statements
do {
        statements
       update statements
} while(logical expression);
```

โดยมีโฟว์ชาร์ตของการทำงานของคำสั่งนี้ดังแสดงในรูปที่ 3.11 ข้อแตกต่างของคำสั่ง while กับคำสั่ง do..while คือคำสั่ง do..while จะทำคำสั่งในบล็อก { } อย่างน้อยหนึ่งครั้งแล้วจึงทำการตรวจสอบเงื่อนไขในนิพจน์ตรรกศาสตร์ ขณะที่คำสั่ง while จะทำการตรวจสอบเงื่อนไขในนิพจน์ตรรกศาสตร์ก่อนซึ่งหากมีค่าเป็นจริงถึงจะทำคำสั่งในบล็อก { }



รูปที่ 3.11 โฟว์ชาร์ตของคำสั่ง do..while

ตัวอย่างของการใช้คำสั่ง do..while เพื่อพิมพ์ข้อความว่า Hello World สิบครั้งมีดังนี้

```
int count = 1;
do {
      System.out.println("Hello World");
      count++;
} while(count <= 10);
```

โปรแกรมที่ 3.6 ตัวอย่างการใช้คำสั่ง `do..while`

```
public class SampleDoWhile {
      public void showDemo() {
         int i = 1;
         do {
            System.out.print(i+"  ");
            i++;
         } while (i <= 10);
      }
}
```

โปรแกรมที่ 3.6 แสดงตัวอย่างการใช้คำสั่ง `do..while` เพื่อพิมพ์ค่าตัวเลข 1 ถึง 10 ซึ่งจะได้ผลลัพธ์เช่นเดียวกับโปรแกรมที่ 3.5

3.3.3 คำสั่ง `for`

คำสั่ง `for` เป็นคำสั่งที่เป็นโครงสร้างแบบทำซ้ำ ที่มีรูปแบบของคำสั่งดังนี้

```
for(initial statements; expression; update statements) {
    statements
}
```

โดยมีโฟว์ชาร์ตของการทำงานของคำสั่งนี้เช่นเดียวกับโฟว์ชาร์ตของคำสั่ง while ในรูปที่ 3.9 คำสั่ง `for` จะใช้ในกรณีที่ทราบจำนวนครั้งในการทำซ้ำที่แน่นอน ส่วนคำสั่ง `while` หรือ `do..while` นิยมใช้ในกรณีที่ไม่ทราบจำนวนครั้งในการทำซ้ำล่วงหน้า

ตัวอย่างเช่นคำสั่งพิมพ์ข้อความ Hello World สิบครั้ง ควรใช้คำสั่ง `for` มากกว่าใช้คำสั่ง `while` หรือ `do..while` โดยสามารถเขียนคำสั่งได้ดังนี้

```
for (int count = 1; count <= 10; count++) {
      System.out.println("Hello World");
}
```

ในกรณีนี้จะเห็นได้ว่าชุดคำสั่งที่ใช้คำสั่ง `for` จะมีรูปแบบที่กระชับกว่าเมื่อเทียบกับกรณีของคำสั่ง `while` หรือ `do..while`

คำสั่งกำหนดค่าเริ่มต้นหรือคำสั่งเปลี่ยนแปลงค่าที่อยู่ในคำสั่ง `for` สามารถที่จะมีมากกว่าอย่างละหนึ่งคำสั่ง โดยจะใช้เครื่องหมาย , ในการแยกคำสั่ง อาทิเช่น

```
for (int i = 0, j = 0; i < 4; i++, j += 2) {
     System.out.println(i + "  " + j);
}
```

เป็นคำสั่งที่จะพิมพ์ค่าของตัวแปร i และ j ที่กำหนดขึ้นภายในคำสั่ง for สี่ครั้ง โดยตัวแปร i จะเพิ่มค่าขึ้นทีละหนึ่ง และตัวแปร j จะเพิ่มค่าขึ้นทีละสอง

<u>โปรแกรมที่ 3.7</u> <u>ตัวอย่างการใช้คำสั่ง for</u>

```
public class SampleFor {
    public void showDemo() {
         for (int i=1; i<=10; i++) {
            System.out.print(i+"  ");
         }
    }
}
```

โปรแกรมที่ 3.7 แสดงตัวอย่างการใช้คำสั่ง for เพื่อพิมพ์ตัวเลข 1 ถึง 10 ซึ่งจะได้ผลลัพธ์เช่นเดียวกับโปรแกรมที่ 3.5 และโปรแกรมที่ 3.6

ตัวแปรที่ประกาศในคำสั่งกำหนดค่าของคำสั่ง for จะมีขอบเขตการใช้งานได้เฉพาะภายในบล็อก { } ของคำสั่ง for เท่านั้น การใช้งานตัวแปรนอกบล็อก { } จะเกิด compile error ดังแสดงในตัวอย่างของโปรแกรมที่ 3.8

<u>โปรแกรมที่ 3.8</u> <u>ขอบเขตการใช้งานของตัวแปร</u>

```
public class VariableScope {
      public void showDemo() {
         for (int i=1; i<10; i++) {
             System.out.print(i+"  ");
         }
         System.out.println("i = "+i);    //illegal
      }
}
```

เราไม่สามารถใช้คำสั่ง for ในกรณีที่ไม่ทราบจำนวนครั้งในการทำซ้ำที่แน่นอนตัวอย่างเช่นโปรแกรมที่ 3.9 เป็นการสุ่มตัวเลขสองตัวที่มีค่าระหว่าง 1 ถึง 10 โดยให้ทำการสุ่มตัวเลขทั้งสองจนกว่าผลการสุ่มของเลขทั้งสองมีค่าเท่ากัน เมธอด Math.random() เป็นเมธอดที่ใช้ในการสุ่มเลขจำนวนทศนิยมระหว่าง 0 ถึง 1 และคำสั่ง

```
i = (int)(Math.random()*10) + 1;
```

เป็นคำสั่งที่ใช้ในการสุ่มตัวเลขระหว่าง 1 ถึง 10 แล้วกำหนดค่าให้กับตัวแปร `i` โปรแกรมนี้สามารถเขียนใหม่โดยใช้คำสั่ง `while` ได้ แต่ไม่สามารถที่จะเขียนโดยใช้คำสั่ง `for` เนื่องจากไม่ทราบจำนวนครั้งที่ต้องการสุ่มตัวเลขที่แน่นอนได้

<u>โปรแกรมที่ 3.9</u> <u>การสุ่มตัวเลขสองตัวที่มีค่าระหว่าง 1 ถึง 10</u>

```
public class ShowNotFor {
    public void showDemo() {
        int i,j;
        do {
            i = (int)(Math.random()*10)+1;
            j = (int)(Math.random()*10)+1;
        } while (i != j);
    }
}
```

3.4 โครงสร้างแบบซ้อน (Nested Structure)

คำสั่งโครงสร้างควบคุมสามารถเขียนซ้อนกันได้อาทิเช่น ใช้คำสั่ง `if` แบบซ้อน คำสั่ง `for` แบบซ้อน โครงสร้างแบบซ้อนสามารถที่จะมีคำสั่งโครงสร้างควบคุมที่อยู่ภายในและภายนอกแตกต่างกันได้ อาทิเช่น คำสั่ง

```
for (int i = 1; i <= 40; i++) {
  if ((i%5) == 0) {
      System.out.println(i);
  }
}
```

เป็นคำสั่ง `if` ที่อยู่ภายในคำสั่ง `for` เพื่อที่จะพิมพ์ตัวเลขระหว่าง 1 ถึง 40 ที่หาร 5 ลงตัวออกมา

ภาษาจาวาอนุญาตให้เขียนคำสั่งโครงสร้างควบคุมซ้อนกันหลายๆ ชั้นได้ คำสั่งโครงสร้างแบบซ้อนที่ใช้กันทั่วไปแบบหนึ่งคือคำสั่ง `for` แบบซ้อน ตัวอย่างเช่น คำสั่ง

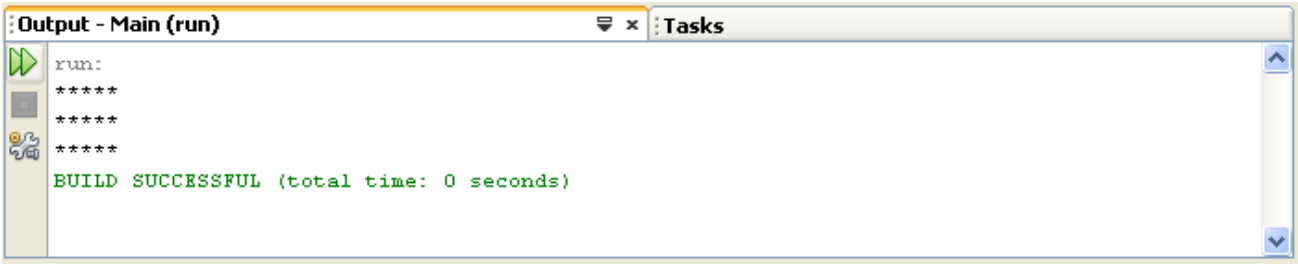
```
for (int i = 1; i <= 3; i++) {
  for (int j = 1; j <= 4; j++) {
      System.out.println("i =”+i+" j =” +j);
  }
}
```

มีคำสั่ง `for` ที่อยู่ภายนอกที่จะทำซ้ำสามครั้ง โดยกำหนดให้ตัวแปร `i` มีค่าเริ่มต้นเป็น 1 และเพิ่มทีละหนึ่ง ส่วนคำสั่ง `for` ที่อยู่ภายในจะทำซ้ำสี่ครั้ง ในแต่ละรอบของคำสั่ง `for` ภายนอก โดยกำหนดให้ตัวแปร `j` มีค่าเริ่มต้นเป็น 1 และเพิ่มค่าทีละหนึ่ง ดังนั้นคำสั่งแสดงผล `(System.out.println())` จะทำซ้ำทั้งหมด 12 ครั้ง

โปรแกรมที่ 3.10 เป็นตัวอย่างการใช้คำสั่ง `for` แบบซ้อน โดยคำสั่ง `for` ที่อยู่ภายในจะพิมพ์เครื่องหมาย '*' เท่ากับจำนวนคอลัมน์ (5 ครั้ง) และคำสั่ง `for` ที่อยู่ภายนอกจะทำคำสั่ง `for` ที่อยู่ภายในซ้ำเท่ากับจำนวนแถว (3 ครั้ง) โปรแกรมจะได้ผลลัพธ์ดังรูปที่ 3.12

<u>โปรแกรมที่ 3.10</u> <u>ตัวอย่างการใช้คำสั่ง `for` แบบซ้อน</u>

```
public class NestedFor {
     public void showDemo() {
          for (int i=1; i<=3; i++) {
            for (int j=1; j<=5; j++) {
               System.out.print('*');
            }
            System.out.println();
          }
     }
}
```

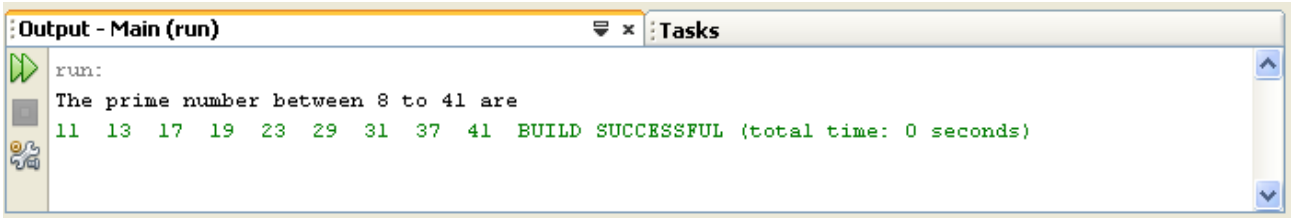


รูปที่ 3.12 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.10

โปรแกรมที่ 3.11 แสดงตัวอย่างโปรแกรมที่มีคำสั่งโปรแกรมแบบซ้อนหลายๆ รูปแบบอยู่ภายใน โปรแกรมนี้จะสุ่มเลขจำนวนเต็มมาสองค่า แล้วจะพิมพ์ตัวเลขเฉพาะ (Prime Number) ที่อยู่ระหว่างเลขทั้งสองนั้นออกมา ซึ่งรูปที่ 3.13 แสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.11

โปรแกรมที่ 3.11 ตัวอย่างการใช้คำสั่งโครงสร้างแบบซ้อนหลายรูป

```
public class SampleNested {
    public void showDemo() {
        int n1 = (int) (Math.random()*10)+2;
        int n2 = (int) (Math.random()*50);
        boolean flag = false;
        int j = 0;
        System.out.println("The prime number between n1+" to "+n2+" are ");
        for (int i=n1; i<=n2; i++) {
            flag = false;
            j = 2;
            do {
                if ((i % j++) == 0)
                    flag = true;
            } while ((j < i) & (!flag));
            if (!flag)
                System.out.print(i+"  ");
        }
    }
}
```



รูปที่ 3.13 ตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.11

3.4.1 คำสั่ง `break`, `continue` และ label

คำสั่ง `break` และ `continue` เป็นคำสั่งที่ใช้ในโครงสร้างแบบทำซ้ำ เพื่อที่จะควบคุมการทำงานของคำสั่งในบล็อก { } โดยคำสั่ง `break` จะทำให้หยุดสิ้นสุดการทำงานของโครงสร้างแบบทำซ้ำ ส่วนคำสั่ง `continue` จะข้ามการทำงานคำสั่งที่เหลือภายในบล็อก { } โดยไปเริ่มการทำซ้ำในรอบต่อไปใหม่

ตัวอย่างเช่น คำสั่ง

```
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        break;
    System.out.println(i);
}
```

จะพิมพ์ค่า `i` ตั้งแต่ 1 ถึง 2 ออกมา แล้วจะหยุดการทำงานของคำสั่ง แต่ถ้าเปลี่ยนจากคำสั่ง `break` ไปเป็น

คำสั่ง continue ดังนี้

```
for (int i = 1; i <= 5; i++) {
    if (i == 3)
         continue;
    System.out.println(i);
}
```

โปรแกรมจะพิมพ์ค่า i ตั้งแต่ 1 ถึง 5 โดยข้ามค่าที่ 3 ทั้งนี้เนื่องจากเจอคำสั่ง continue

ในกรณีของคำสั่งโครงสร้างควบคุมแบบซ้อน ถ้าคำสั่ง break หรือ continue ปรากฏอยู่ในคำสั่งโครงสร้างแบบทำซ้ำที่อยู่ข้างใน คำสั่งโครงสร้างแบบทำซ้ำที่อยู่ข้างนอกจะยังทำซ้ำต่อ โปรแกรมที่ 3.12 แสดงตัวอย่างของการใช้คำสั่งโครงสร้างควบคุมแบบซ้อนที่มีคำสั่ง break อยู่ในคำสั่งโครงสร้างแบบทำซ้ำที่อยู่ข้างใน โปรแกรมนี้จะรันคำสั่ง break เมื่อ j มีค่าเป็น 3 โดยคำสั่งในคำสั่ง for ภายในจะหยุดทำงาน แต่คำสั่ง for ภายนอกจะยังทำงานต่อ โดยจะได้ผลลัพธ์ดังแสดงในรูปที่ 3.14

<u>โปรแกรมที่ 3.12</u> <u>ตัวอย่างการใช้คำสั่ง break</u>

```
public class SampleBreak1 {
   public void showDemo() {
      int i, j, product;
      for (i=1; i<=3; i++) {
         for (j=1; j<=3; j++) {
            product = i*j;
            if (j==3) break;
            System.out.println(i+" * "+j+" = "+product);
         }
      }
   }
}
```

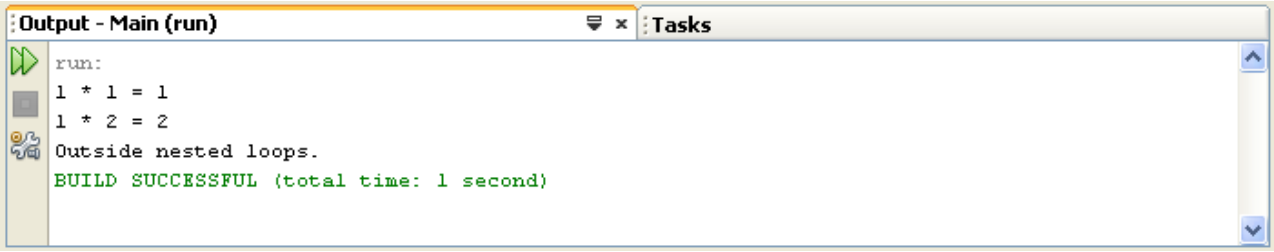
```
Output - Main (run)        Tasks
run:
1 * 1 = 1
1 * 2 = 2
2 * 1 = 2
2 * 2 = 4
3 * 1 = 3
3 * 2 = 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

รูปที่ 3.14 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.12

ในกรณีที่ต้องการจะหยุดการทำงานของคำสั่งโครงสร้างควบคุมที่อยู่ภายนอก เราจะต้องกำหนด label ขึ้นมาเติม ดังตัวอย่างในโปรแกรมที่ 3.13 ซึ่งกำหนด label ที่ชื่อ `outer` และมีคำสั่ง `break outer;` เพื่อที่จะทำให้โปรแกรมหยุดการทำงานของคำสั่ง `for` ภายนอก โดยจะได้ผลลัพธ์ดังแสดงในรูปที่ 3.15

<u>โปรแกรมที่ 3.13</u> <u>ตัวอย่างการใช้คำสั่ง `break` และ label</u>

```
public class SampleBreak2 {
   public void showDemo() {
      int i, j, product;
outer: for (i=1; i<=3; i++) {
          for (j=1; j<=3; j++) {
              product = i*j;
              if (j==3) break outer;
              System.out.println(i+" * "+j+" = "+product);
          }
       }
       System.out.println("Outside nested loops.");
   }
}
```



รูปที่ 3.15 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 3.13

สรุปเนื้อหาของบท

- คำสั่งโครงสร้างควบคุม เป็นคำสั่งที่ใช้ในการกำหนดลำดับการทำงานของคำสั่งต่างๆ โดยมีสามรูปแบบคือ โครงสร้างแบบตามลำดับ โครงสร้างแบบเลือกทำ และ โครงสร้างแบบทำซ้ำ
- คำสั่งที่เป็นคำสั่งของโครงสร้างแบบเลือกทำคือ คำสั่ง `if, if..else` หรือ `switch`
- คำสั่ง `if..else` แตกต่างจากคำสั่ง `if` ตรงที่ คำสั่ง `if..else` จะมีการทำคำสั่งสำหรับค่าเท็จถ้านิพจน์ตรรกศาสตร์เป็นเท็จ ส่วนคำสั่ง `if` จะไม่มีการทำคำสั่งใดถ้านิพจน์ตรรกศาสตร์เป็นเท็จ

- คำสั่ง `if` หรือ `if..else` สามารถที่จะซ้อนอยู่ข้างในคำสั่ง `if` หรือ `if..else` อื่นได้
- คำสั่ง `switch` จะมีลักษณะโครงสร้างการทำงานคล้ายคลึงกับคำสั่ง `if..else if..else ..` แต่ชนิดข้อมูลของตัวแปรที่จะนำมาใช้กับคำสั่ง `switch` จะต้องเป็นชนิด `char, byte, short` หรือ `int` เท่านั้น
- คำสั่งที่เป็นคำสั่งของโครงสร้างแบบทำซ้ำคือ คำสั่ง `while, do..while` หรือ `for`
- คำสั่ง `while` แตกต่างจากคำสั่ง `do..while` ตรงที่ คำสั่ง `while` จะไม่มีการทำชุดคำสั่งเลยถ้านิพจน์ตรรกศาสตร์เป็นเท็จ ส่วนคำสั่ง `do..while` จะมีการทำชุดคำสั่งหนึ่งครั้งถ้านิพจน์ตรรกศาสตร์เป็นเท็จ
- คำสั่ง `for` มีลักษณะการทำงานที่เหมือนกับคำสั่ง `while` แต่จะมีการรวมคำสั่งกำหนดค่าเริ่มต้น นิพจน์ตรรกศาสตร์และคำสั่งเปลี่ยนแปลงค่าไว้หลังคำสั่ง `for`
- คำสั่ง `for` จะใช้ในกรณีที่ทราบจำนวนครั้งในการทำซ้ำที่แน่นอน ส่วนคำสั่ง `while` หรือ `do..while` นิยมใช้ในกรณีที่ไม่ทราบจำนวนครั้งในการทำซ้ำล่วงหน้า
- คำสั่งโครงสร้างทั้งหมดที่กล่าวมาแล้วข้างต้น สามารถนำมาใช้ร่วมกันเป็นลักษณะแบบซ้อนได้ เช่น คำสั่ง `switch` อยู่ภายในคำสั่ง `while` หรือคำสั่ง `for` อยู่ภายในคำสั่ง `for` ซึ่งเรียกโครงสร้างในลักษณะนี้ว่า โครงสร้างแบบซ้อน
- คำสั่ง `break` จะทำให้หยุดสิ้นสุดการทำงานของโครงสร้างแบบทำซ้ำ ส่วนคำสั่ง continue จะข้ามการทำงานคำสั่งที่เหลือภายในบล็อก { } โดยไปเริ่มการทำซ้ำในรอบต่อไปใหม่

บทที่ 4 หลักการเชิงอ็อบเจกต์

เนื้อหาในบทนี้เป็นการแนะนำหลักการการพัฒนาโปรแกรมเชิงอ็อบเจกต์ อธิบายความ หมายของคำนิยามต่างๆ ที่ใช้ในการพัฒนาโปรแกรมเชิงอ็อบเจกต์เช่น คลาส อ็อบเจกต์ คุณลักษณะ และเมธอด แนะนำการประกาศคำนิยามดังกล่าวโดยใช้ภาษาจาวา อธิบายคุณลักษณะเด่นของโปรแกรมเชิงอ็อบเจกต์ แนะนำคลาสที่ใช้ในการจำลองข้อกำหนดของโปรแกรมเชิงอ็อบเจกต์ และในส่วนท้ายของบทจะเป็นการแนะนำขั้นตอนการพัฒนาโปรแกรมโดยใช้หลักการเชิงอ็อบเจกต์

4.1 องค์ประกอบของโปรแกรมเชิงอ็อบเจกต์

ภาษาจาวาเป็นภาษาคอมพิวเตอร์ ที่ใช้หลักการการพัฒนาโปรแกรมเชิงอ็อบเจกต์ที่เรียกย่อว่าเป็นภาษาคอมพิวเตอร์แบบ OOP การพัฒนาโปรแกรมเชิงอ็อบเจกต์จะเป็นขบวนการการวิเคราะห์ปัญหาโดยการจำลองปัญหาว่าประกอบไปด้วยอ็อบเจกต์ใดบ้าง แล้วจึงเขียนให้อยู่ในรูปแบบของโปรแกรมคอมพิวเตอร์ โปรแกรมเชิงอ็อบเจกต์จะมีคำนิยามที่สำคัญสองคำคือ อ็อบเจกต์ (Object) และคลาส (Class)

4.1.1 อ็อบเจกต์

อ็อบเจกต์คือสิ่งต่างๆ ที่มีอยู่ในชีวิตประจำวันแบ่งได้เป็นสองประเภทคือ

1. สิ่งที่เป็นรูปธรรม (Tangible) คือสิ่งที่เป็นวัตถุและจับต้องได้อาทิเช่น นักศึกษา ใบลงทะเบียน ปากกา และรถ เป็นต้น
2. สิ่งที่เป็นนามธรรม (Intangible) คือสิ่งที่ไม่สามารถจับต้องได้อาทิเช่น คะแนนรายชื่อวิชา บัญชีเงินฝาก และตารางเที่ยวบิน เป็นต้น

อ็อบเจกต์ต่างๆ จะประกอบไปด้วยคุณลักษณะ (Attribute) และพฤติกรรม (Behavior) คุณลักษณะก็คือข้อมูลของอ็อบเจกต์ ส่วนพฤติกรรมหรือเรียกว่าเมธอด (Method) คือสิ่งที่อ็อบเจกต์สามารถกระทำได้ ซึ่งในโปรแกรมเชิงอ็อบเจกต์ก็คือคำสั่งในการทำงาน โปรแกรมเชิงอ็อบเจกต์จะประกอบด้วยอ็อบเจกต์ต่างๆ หลายอ็อบเจกต์ ซึ่งแต่ละอ็อบเจกต์จะมีคุณลักษณะต่างๆ ที่เป็นข้อมูลของอ็อบเจกต์ และโปรแกรมสามารถจัดการกับข้อมูลเหล่านี้ได้โดยการเรียกใช้เมธอดต่างๆ

ตัวอย่างของอ็อบเจกต์

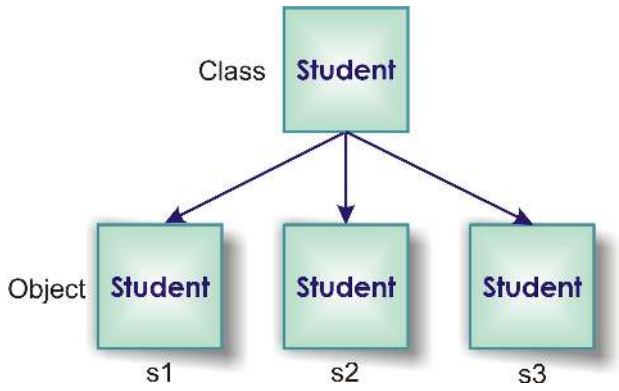
- นักศึกษา อาจจะประกอบไปด้วยคุณลักษณะเช่น รหัส ชื่อ และเกรดเฉลี่ย และอาจจะมีเมธอดเช่น การลงทะเบียน สอบ และเดิน
- รถยนต์ อาจจะประกอบไปด้วยคุณลักษณะเช่น ยี่ห้อ รุ่น และสี และอาจจะมีเมธอดเช่น เคลื่อนที่ หยุดและเลี้ยว
- สุนัข อาจจะประกอบไปด้วยคุณลักษณะเช่น ชื่อ พันธุ์ และสี และอาจจะมีเมธอดเช่น เห่า คลาน และกระดิกหาง

ตัวอย่างของโปรแกรมเชิงอ็อบเจกต์อาจยกตัวอย่างของโปรแกรมระบบจัดการบัญชีเงินฝากของธนาคาร ซึ่งอาจประกอบไปด้วยอ็อบเจกต์ต่างๆ อาทิเช่น บัญชี (Account) ลูกค้า (Customer) เครื่องเอทีเอ็ม (ATM) และรายการ (Transaction) อ็อบเจกต์ชนิดบัญชีอาจจะมีข้อมูลภายในต่างๆ อาทิเช่น เลขที่บัญชี ชื่อเจ้าของบัญชี วันที่เปิดบัญชี และยอดเงินคงเหลือ เป็นต้น อ็อบเจกต์ชนิดบัญชีอาจจะมีเมธอดต่างๆ อาทิเช่น ฝาก ถอน และการโอนเงิน เป็นต้น

4.1.2 คลาส

โดยทั่วไปโปรแกรมเชิงอ็อบเจกต์แต่ละโปรแกรมจะประกอบไปด้วยอ็อบเจกต์ ที่เป็นชนิดเดียวกันหลายๆ อ็อบเจกต์ แต่อาจมีข้อมูลหรือคุณลักษณะที่ต่างกันอาทิเช่น โปรแกรมระบบจัดการบัญชีเงินฝากของธนาคารอาจมีอ็อบเจกต์ ชนิดบัญชีหลายๆ อ็อบเจกต์ โดยที่แต่ละอ็อบเจกต์อาจจะมีข้อมูลที่เป็นเลขที่บัญชีหรือชื่อเจ้าของบัญชีที่ต่างกัน โปรแกรมเชิงอ็อบเจกต์จะมีคำสั่งในการสร้างอ็อบเจกต์โดยสร้างมาจากคลาส ซึ่งเป็นตัวกำหนดนิยามของอ็อบเจกต์ว่าจะต้องประกอบด้วยคุณลักษณะและเมธอดใดบ้าง

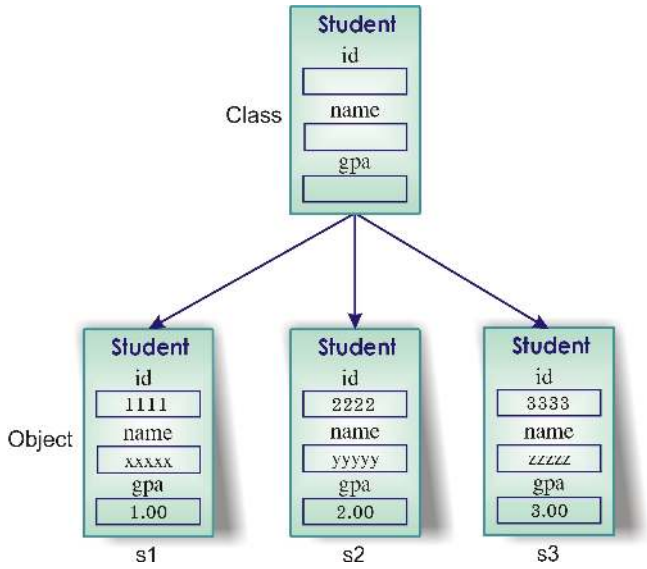
คลาสเปรียบเสมือนพิมพ์เขียวของอ็อบเจกต์ อ็อบเจกต์ที่ถูกสร้างมาจากคลาสบางครั้งจะเรียกว่าเป็น instance ของคลาส ซึ่งอ็อบเจกต์ใดๆ จะต้องเป็น instance ของคลาสใดคลาสหนึ่ง การเขียนโปรแกรมเชิงอ็อบเจกต์ ต้องมีการกำหนดนิยามของคลาสก่อนที่จะสามารถสร้างอ็อบเจกต์ (หรือ instance) ของคลาสได้ ซึ่งคลาสหนึ่งคลาสสามารถที่จะสร้างอ็อบเจกต์ได้หลายอ็อบเจกต์ รูปที่ 4.1 แสดงตัวอย่างของคลาส `Student` ซึ่งสร้างอ็อบเจกต์ขึ้นมาสามอ็อบเจกต์ที่ชื่อ `s1, s2` และ `s3` เป็นต้น



รูปที่ 4.1 ตัวอย่างของอ็อบเจกต์และคลาส

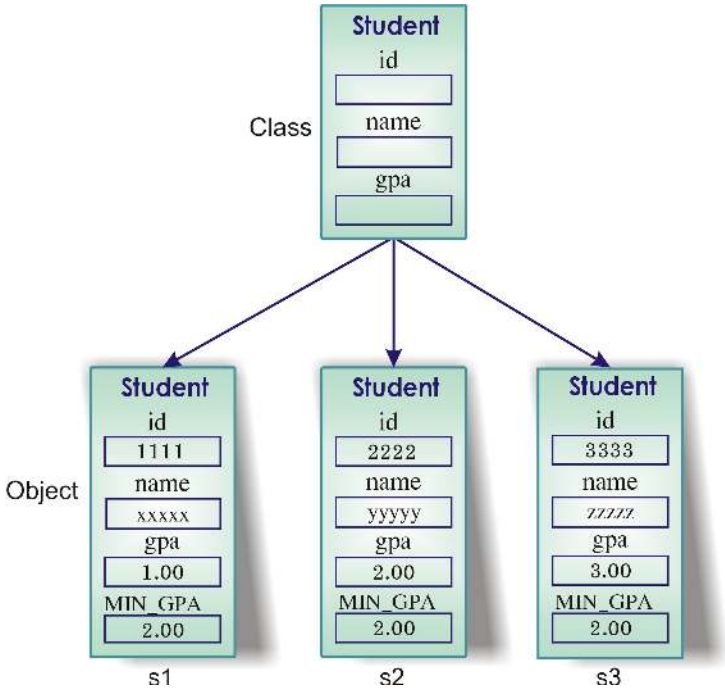
4.1.3 คุณลักษณะ

คุณลักษณะของอ็อบเจกต์ คือข้อมูลที่เก็บอยู่ในอ็อบเจกต์ ซึ่งจะแบ่งออกเป็นสองประเภท คือตัวแปร (Variable) และค่าคงที่ (Constant) โดยที่คุณลักษณะที่เป็นตัวแปรจะสามารถเปลี่ยนค่าได้ ส่วนคุณลักษณะที่เป็นค่าคงที่จะไม่สามารถเปลี่ยนค่าได้ รูปที่ 4.2 แสดงตัวอย่างของอ็อบเจกต์ `s1`, `s2` และ `s3` ที่เป็นอ็อบเจกต์ของคลาส `Student` ซึ่งมีคุณลักษณะของอ็อบเจกต์ที่เป็นรหัส ชื่อ และคะแนนเฉลี่ยสะสมที่ต่างกัน โดยกำหนดในค่าของตัวแปร `id, name` และ `gpa` ตามลำดับ

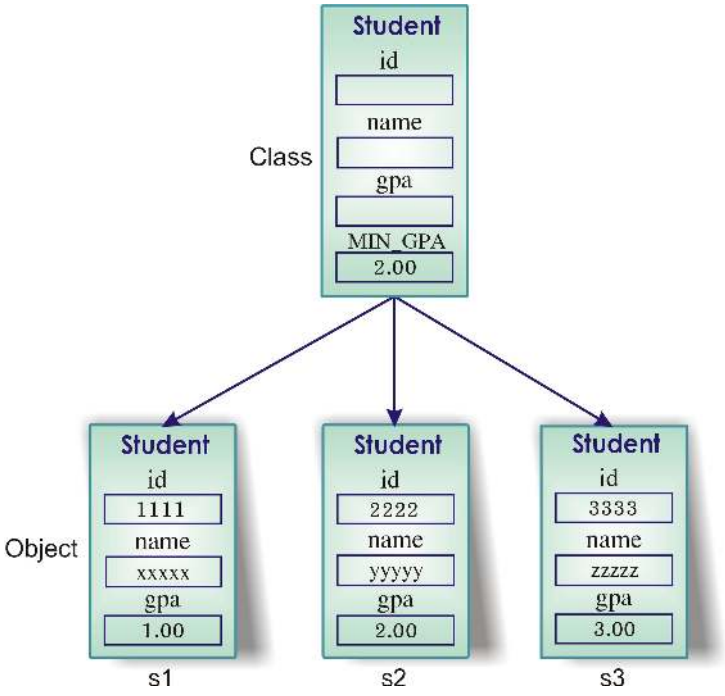


รูปที่ 4.2 ตัวอย่างคุณลักษณะของอ็อบเจกต์

โปรแกรมเชิงอ็อบเจกต์ได้กำหนดนิยามคุณลักษณะอีกประเภทหนึ่ง ที่เรียกว่า คุณลักษณะของคลาส (Class Data Value) ซึ่งจะเป็นคุณลักษณะที่ทุกอ็อบเจกต์ใช้ร่วมกัน อาทิเช่น คลาส `Student` อาจกำหนดให้มีคุณลักษณะของคลาสที่เป็นค่าคงที่ที่ชื่อ `MIN_GPA` เพื่อเก็บค่าคะแนนเฉลี่ยสะสมขั้นต่ำของนักศึกษาทุกคน ทั้งนี้นักศึกษาทุกคนจะต้องมีค่าคะแนนเฉลี่ยสะสมสูงกว่าคะแนนเฉลี่ยสะสมขั้นต่ำ การเก็บคุณลักษณะที่เหมือนกันเช่นนี้ถ้ากำหนดให้เป็นคุณลักษณะของอ็อบเจกต์ จะทำให้สิ้นเปลืองเนื้อที่ในหน่วยความจำ ดังแสดงให้เห็นในรูปที่ 4.3 แต่ถ้ากำหนดให้เป็นคุณลักษณะของคลาส จะทำให้สามารถประหยัดพื้นที่ในหน่วยความจำได้ดังแสดงให้เห็นในรูปที่ 4.4



รูปที่ 4.3 คุณลักษณะ `MIN_GPA` ซึ่งเป็นคุณลักษณะของอ็อบเจกต์

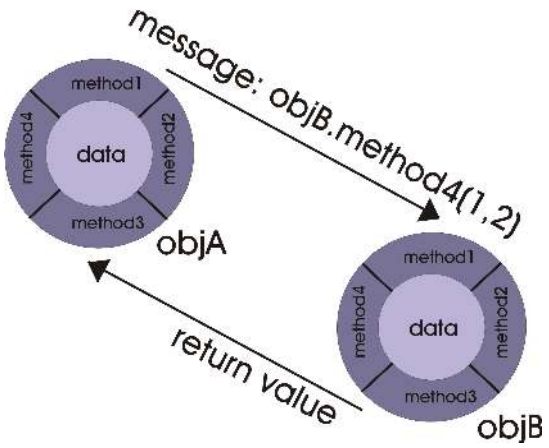


รูปที่ 4.4 คุณลักษณะ `MIN_GPA` ซึ่งเป็นคุณลักษณะของคลาส

4.1.4 เมธอด

เมธอดเป็นวิธีการหรือการกระทำที่นิยามอยู่ในคลาสหรืออ็อบเจกต์เพื่อใช้ในการจัดการกับคุณลักษณะของอ็อบเจกต์หรือคุณลักษณะของคลาสอาทิเช่น อ็อบเจกต์ชนิดบัญชีเงินฝากอาจมีคุณลักษณะ `balance` เพื่อเก็บยอดเงินคงเหลือที่อยู่ในบัญชี และอาจมีเมธอด `deposit()` เพื่อเป็นวิธีการในการฝากเงินให้กับอ็อบเจกต์ ในเมธอดจะมีคำสั่งเพื่อจัดการกับวิธีการฝากเงินและจะทำการปรับเปลี่ยนค่าของคุณลักษณะ `balance` เราสามารถเปรียบเทียบเมธอดได้กับฟังก์ชัน Procedure หรือ Subroutine ของโปรแกรมเชิงกระบวนการ

การเขียนโปรแกรมเชิงอ็อบเจกต์จะกำหนดให้อ็อบเจกต์ต่างๆ สื่อสารกัน โดยการผ่านข่าวสาร (Message) ระหว่างอ็อบเจกต์ที่เป็นผู้ส่ง (Sender) กับอ็อบเจกต์ที่เป็นผู้รับ (Receiver) โดยการเรียกใช้เมธอดอาทิเช่น รูปที่ 4.5 แสดงตัวอย่างของการส่งข่าวสารจากอ็อบเจกต์ `objA` ที่เป็นผู้ส่ง เพื่อเรียกการทำงานของเมธอด `method4()` ของอ็อบเจกต์ `objB` ที่เป็นผู้รับ การส่งข่าวสารระหว่างกันอาจมีการส่งข้อมูลจาก `objA` ผ่านไปยัง `objB` โดยผ่านทาง argument ของเมธอด (ตัวอย่างเช่น ค่า 1 และ 2 ในรูปที่ 4.5) และผู้รับก็อาจส่งค่ากลับ (Return Value) มายังผู้ส่ง



รูปที่ 4.5 ตัวอย่างของการส่งข่าวสารระหว่างอ็อบเจกต์

4.2 การเขียนโปรแกรมเชิงอ็อบเจกต์โดยใช้ภาษาจาวา

4.2.1 การประกาศคลาส

โปรแกรมภาษาจาวาแต่ละโปรแกรมจะประกอบไปด้วย คลาสอย่างน้อยหนึ่งคลาส โดยมีรูปแบบการประกาศดังนี้

```
[modifier] class ClassName {
    [class member]
}
```

โดยที่

- `modifier` คือคีย์เวิร์ด (Keyword) ของภาษาจาวาที่ใช้เป็น access modifier เช่น `public` หรืออธิบายคุณสมบัติอื่นๆ ของคลาส เช่น `abstract` และ `final` ซึ่งเป็นเงื่อนไขเพิ่มเติม (Option)
- `class` คือคีย์เวิร์ดของภาษาจาวาเพื่อระบุว่าเป็นการประกาศคลาส
- `ClassName` คือชื่อของคลาสที่เป็นชื่อ identifier ใดๆ ที่สอดคล้องกับกฎการตั้งชื่อ
- `class member` คือเมธอดหรือคุณลักษณะของคลาส

ตัวอย่างการประกาศคลาส `Student` สามารถทำได้ดังนี้

```
public class Student {

}
```

4.2.2 การประกาศคุณลักษณะ

คุณลักษณะของอ็อบเจกต์ คือตัวแปรหรือค่าคงที่ซึ่งประกาศภายในอ็อบเจกต์ โดยมีรูปแบบการประกาศดังนี้

```
[modifier] dataType attributeName;
```

โดยที่

- `modifier` คือคีย์เวิร์ดของภาษาจาวาที่อธิบายคุณสมบัติต่างๆ ของตัวแปรหรือค่าคงที่อาทิเช่น `public`, `private`, `static`, `final` และ `transient` เป็นต้น
- `dataType` คือชนิดข้อมูลซึ่งอาจเป็นชนิดข้อมูลแบบพื้นฐานหรือชนิดข้อมูลแบบอ้างอิงทีจะกล่าวถึงในบทต่อไป
- `attributeName` คือชื่อของคุณลักษณะที่เป็นชื่อ identifier ใดๆ ซึ่งสอดคล้องกับกฎการตั้งชื่อ

ตัวอย่างการประกาศคุณลักษณะ `id`, `name` และ `gpa` ซึ่งมีชนิดข้อมูลเป็น `String` และ `double` ในคลาส `Student` สามารถทำได้ดังนี้

```
public class Student {
    public String id;
    public String name;
    public double gpa;
}
```

4.2.3 การประกาศเมธอด

ภาษาจาวากำหนดรูปแบบของการประกาศเมธอดที่อยู่ในคลาสไว้ดังนี้

```
[modifier] return_type methodName([arguments]) {
    [method_body]
}
```

โดยที่

- `modifier` คือคีย์เวิร์ดของภาษาจาวาที่เป็น access modifier เช่น `public` หรือ `private` หรืออธิบายคุณสมบัติอื่นๆ ของเมธอดเช่น `abstract` หรือ `static` เป็นต้น
- `return_type` คือชนิดข้อมูลของค่าที่ส่งกลับหลังจากเสร็จสิ้นการทำงานของคำสั่งในเมธอดนี้ โดยชนิดข้อมูลของค่าที่ส่งกลับอาจเป็นชนิดข้อมูลแบบพื้นฐานหรือชนิดข้อมูลแบบอ้างอิง ในกรณีที่ไม่มีการส่งค่าใดๆ กลับจะต้องระบุชนิดข้อมูลเป็น `void`

- `methodName` คือชื่อของเมธอดที่เป็นชื่อ identifier ใดๆ ซึ่งสอดคล้องกับกฎการตั้งชื่อ
- `arguments` คือตัวแปรที่ใช้ในการรับข้อมูลที่อ็อบเจกต์ส่งมาให้ โดยอาจมีมากกว่าหนึ่งตัวแปร หรือไม่มีเลยก็ได้ขึ้นอยู่กับการกำหนดเมธอด
- `method_body` คือคำสั่งต่างๆ ของภาษาจาวาที่อยู่ในเมธอด

โปรแกรมที่ 4.1 แสดงตัวอย่างโปรแกรมภาษาจาวาที่ประกาศคลาส `Student` โดยมีคุณลักษณะคือ `id`, `name`, `gpa` และ `MIN_GPA` และมีเมธอดคือ `setId()`, `setName()`, `setGpa()` และ `showDetails()`

โปรแกรมที่ 4.1 คลาส `Student`

```
public class Student {
    public String id;
    public String name;
    public double gpa;
    public static final double MIN_GPA = 2.00;

    public void setId(String ID) {
        id = ID;
    }
    public void setName(String n) {
        name = n;
    }
    public void setGpa(double GPA) {
        gpa = GPA;
    }
    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}
```

เมธอด `main()`

โปรแกรมภาษาจาวาที่เป็นโปรแกรมจาวาประยุกต์ (Java Application) จะเริ่มต้นการทำงานในคลาสที่มีเมธอด `main()` โดยมีรูปแบบของเมธอดดังนี้

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

4.2.4 การประกาศและสร้างอ็อบเจกต์

อ็อบเจกต์ทุกอ็อบเจกต์ในโปรแกรมภาษาจาวาจะต้องมีคำสั่งประกาศเพื่อระบุว่าอ็อบเจกต์นั้นเป็นอ็อบเจกต์ของคลาสใด โดยมีรูปแบบดังนี้

```
[modifier] ClassName objectName;
```

โดยที่

- `modifier` คือคีย์เวิร์ดที่อธิบายคุณสมบัติต่างๆ ของอ็อบเจกต์
- `ClassName` คือชื่อของคลาส
- `objectName` คือชื่อของอ็อบเจกต์ที่เป็นชื่อ identifier ใดๆ ซึ่งสอดคล้องกับกฎการตั้งชื่อ

ตัวอย่างเช่น คำสั่ง

```
Student  s1;
```

เป็นคำสั่งประกาศอ็อบเจกต์ `s1` ให้เป็นอ็อบเจกต์ของคลาส `Student` ทั้งนี้คลาสที่ระบุในคำสั่งประกาศอ็อบเจกต์จะต้องเป็นคลาสที่มีการนิยามไว้แล้ว กล่าวคือจะต้องมีคลาส `Student` ที่นิยามไว้แล้ว

คำสั่งในการประกาศอ็อบเจกต์ไม่ได้เป็นคำสั่งที่ใช้ในการสร้างอ็อบเจกต์ คำสั่งที่ใช้ในการสร้างอ็อบเจกต์จะมีรูปแบบดังนี้

```
objectName = new ClassName([arguments]);
```

โดยที่

- `objectName` คือชื่อของอ็อบเจกต์
- `new` คือคีย์เวิร์ดของภาษาจาวาเพื่อใช้ในการสร้างอ็อบเจกต์
- `ClassName` คือชื่อของคลาส
- `arguments` คือค่าที่ต้องการส่งผ่านในการเรียกเมธอดในการสร้างอ็อบเจกต์ซึ่งอาจมีหรือไม่มีก็ได้

การสร้างอ็อบเจกต์โดยเรียกใช้คำสั่ง `new` นี้จะเป็นการเรียกใช้เมธอดในการสร้างอ็อบเจกต์ที่เรียกว่า constructor ของคลาสนั้นๆ ซึ่งจะมีการอธิบายการทำงานของ constructor ในบทที่ 6

ตัวอย่างคำสั่งในการสร้างอ็อบเจกต์ `s1` ทำได้ดังนี้

```
s1 = new Student();
```

นอกจากนี้คำสั่งในการประกาศและสร้างอ็อบเจกต์สามารถที่จะรวมเป็นคำสั่งเดียวกันได้โดยมีรูปแบบคำสั่งดังนี้

```
[modifier] ClassName objectName = new className([arguments]);
```

อาทิเช่น

```
Student s1 = new Student();
```

4.2.5 การเรียกใช้สมาชิกของอ็อบเจกต์

สมาชิกของอ็อบเจกต์ประกอบด้วยคุณลักษณะและเมธอด การเรียกใช้สมาชิกของอ็อบเจกต์ทำได้โดยการใช้เครื่องหมายจุด (.) กล่าวคือคุณลักษณะของอ็อบเจกต์สามารถเรียกใช้ โดยมีรูปแบบดังนี้

```
objectName.attributeName;
```

โดยที่

- `objectName` คือชื่อของอ็อบเจกต์ที่สร้างขึ้น
- `attributeName` คือชื่อของคุณลักษณะของอ็อบเจกต์นั้น

นอกจากนี้เราสามารถที่จะส่งข่าวสารไปยังอ็อบเจกต์ภายหลังจากที่มีการสร้างอ็อบเจกต์ขึ้นมาได้โดยการเรียกใช้เมธอดของอ็อบเจกต์นั้น รูปแบบคำสั่งในการเรียกใช้เมธอดมีดังนี้

```
objectName.methodName([arguments]);
```

โดยที่

- `objectName` คือชื่อของอ็อบเจกต์ที่สร้างขึ้น
- `methodName` คือชื่อของเมธอดของอ็อบเจกต์นั้น
- `arguments` คือค่าที่ต้องการส่งผ่านไปให้กับเมธอดของอ็อบเจกต์นั้น โดยที่จะต้องมีชนิดข้อมูลและจำนวน argument ให้สอดคล้องกับที่ประกาศในเมธอดของอ็อบเจกต์นั้น

ตัวอย่างเช่น คำสั่ง

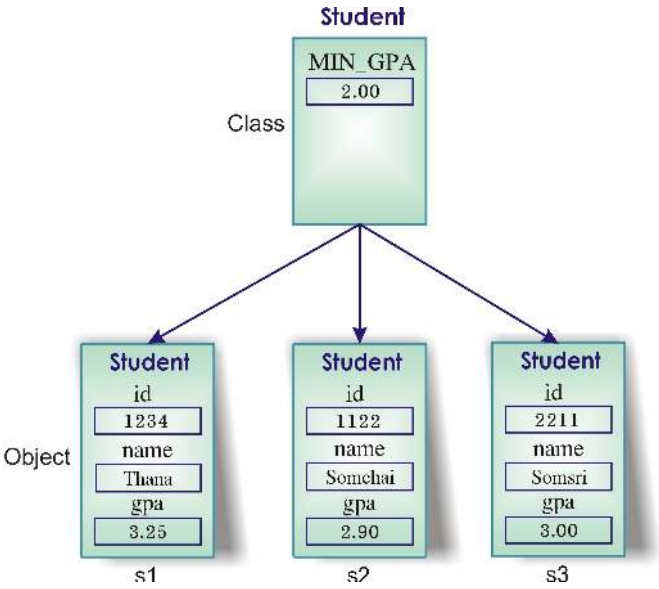
```
s1.setName("Thana");
```

เป็นการเรียกใช้เมธอด `setName()` ของอ็อบเจกต์ `s1` ซึ่งเป็นอ็อบเจกต์ของคลาส `Student` โดยจะส่งผ่าน argument ชนิด `String` ที่มีค่าเป็นข้อความ `Thana` ให้กับเมธอดดังกล่าว โปรแกรมที่ 4.2 แสดงตัวอย่างโปรแกรม

ของคลาส `Sample` ที่มีเมธอด `main()` อยู่ โดยโปรแกรมนี้จะสร้างอ็อบเจกต์ของคลาส `Student` ขึ้นมาสามอ็อบเจกต์ คือ อ็อบเจกต์ `s1`, `s2` และ `s3` จากนั้นจะทำการเรียกเมธอด `setName()` เพื่อกำหนดค่าให้กำหนดค่าให้กับคุณลักษณะ `name` ของอ็อบเจกต์แต่ละตัว ซึ่งผลลัพธ์ของโปรแกรมนี้จะได้อ็อบเจกต์ดังแสดงใน รูปที่ 4.6

<u>โปรแกรมที่ 4.2</u> <u>คลาส `Sample` ที่เรียกใช้เมธอดของคลาส `Student`</u>

```
public class Sample {
    public static void main(String args[]) {
        Student s1 = new Student();
        Student s2 = new Student();
        Student s3 = new Student();
        s1.setId("1234");
        s1.setName("Thana");
        s1.setGpa(3.25);
        s2.setId("1122");
        s2.setName("Somchai");
        s2.setGpa(2.90);
        s3.setId("2211");
        s3.setName("Somsri");
        s3.setGpa(3.00);
    }
}
```



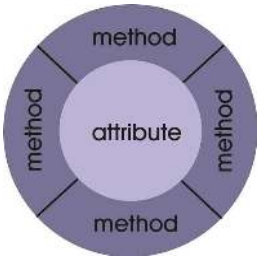
รูปที่ 4.6 ผลลัพธ์ของโปรแกรมที่ 4.2

4.3 คุณลักษณะเด่นของโปรแกรมเชิงอ็อบเจกต์

โปรแกรมเชิงอ็อบเจกต์ จะมีคุณลักษณะเด่นอยู่สามประการคือ การห่อหุ้ม (Encapsulation) การสืบทอด (Inheritance) และการมีได้หลายรูปแบบ (Polymorphism)

4.3.1 การห่อหุ้ม

หลักการที่สำคัญประการหนึ่งของการเขียนโปรแกรมเชิงอ็อบเจกต์คือการห่อหุ้ม ซึ่งคุณลักษณะของอ็อบเจกต์จะถูกห่อหุ้มอยู่ภายใน เพื่อไม่ให้อ็อบเจกต์อื่นๆ สามารถเข้าถึงข้อมูลที่เป็นคุณลักษณะได้โดยตรง การจะเรียกใช้คุณ-ลักษณะของอ็อบเจกต์จะทำได้โดยการเรียกผ่านเมธอดเท่านั้นดังแสดงในรูปที่ 4.7



รูปที่ 4.7 หลักการของการห่อหุ้ม

อ็อบเจกต์แต่ละอ็อบเจกต์จะประกอบไปด้วยส่วนที่เป็น interface และส่วนที่เป็น implementation ส่วนที่เป็น interface คือส่วนของอ็อบเจกต์ที่อนุญาตให้อ็อบเจกต์อื่นสามารถเรียกใช้งานได้ ซึ่งในภาษาจาวาก็คือคุณลักษณะหรือเมธอดที่ถูกประกาศให้มี access modifier เป็น `public` สำหรับส่วนที่เป็น implementation คือส่วนของอ็อบเจกต์ที่ไม่สามารถเข้าถึงได้จากภายนอก อ็อบเจกต์อื่นๆ จะไม่สามารถเห็นส่วนที่เป็น implementation ของอ็อบเจกต์ได้ ในภาษาจาวาก็คือคุณลักษณะหรือเมธอดที่ถูกประกาศให้มี access modifier เป็น `private`

การเขียนโปรแกรมเชิงอ็อบเจกต์โดยใช้หลักการของการห่อหุ้ม สามารถทำได้โดยกำหนดให้คุณลักษณะของอ็อบเจกต์มี access modifier เป็น `private` และกำหนดให้เมธอดที่สามารถเรียกจากอ็อบเจกต์ภายนอกได้มี access modifier เป็น `public` ดังแสดงตัวอย่างในโปรแกรมที่ 4.3

โปรแกรมที่ 4.3 การใช้หลักการของการห่อหุ้ม

```
public class Student {
    private String id;
    private String name;
    private double gpa;
    private static final double MIN_GPA = 2.00;

    public void setId(String ID) {
        id = ID;
    }
    public void setName(String n) {
        name = n;
    }
    public void setGpa(double GPA) {
        gpa = GPA;
    }
    public void showDetails() {
        System.out.println("ID: "+id);
        System.out.println("Name: "+name);
        System.out.println("GPA: "+gpa);
    }
}
```

ข้อดีของการห่อหุ้มคือ

- การซ่อนเร้นข้อมูล (Information Hiding) ทำให้อ็อบเจกต์สามารถติดต่อกับอ็อบเจกต์ภายนอกผ่านเมธอดที่เป็นส่วนของ interface เท่านั้น ซึ่งถ้ามีการเปลี่ยนแปลงคุณลักษณะหรือเมธอดที่อยู่ภายในอ็อบเจกต์ ก็จะไม่มีผลกระทบใดๆ ต่ออ็อบเจกต์ภายนอกที่เรียกใช้
- ความเป็นโมดูล (Modularity) การพัฒนาโปรแกรมเชิงอ็อบเจกต์จะสามารถกำหนดให้อ็อบเจกต์แต่ละอ็อบเจกต์มีความเป็นอิสระต่อกัน ถ้ามีการเปลี่ยนแปลงเกิดขึ้นภายในอ็อบเจกต์หนึ่งก็จะไม่มีผลกระทบต่ออ็อบเจกต์อื่น

เครื่องโทรศัพท์เป็นตัวอย่างหนึ่ง ของการแสดงคุณลักษณะเด่นด้านการห่อหุ้มของอ็อบเจกต์ ส่วนที่เป็น interface ก็คือปุ่มของโทรศัพท์ที่ผู้ใช้สามารถติดต่อกับเครื่องได้ สำหรับระบบอิเล็กทรอนิกส์ที่อยู่ในเครื่องโทรศัพท์คือส่วนที่เป็น implementation ซึ่งผู้ใช้ไม่สามารถมองเห็นและไม่จำเป็นต้องเข้าใจการทำงานของระบบดังกล่าว และหากระบบอิเล็กทรอนิกส์ภายในเปลี่ยนแปลงไป ผู้ใช้ยังสามารถที่จะติดต่อกับเครื่องได้โดยใช้ปุ่มที่เป็นส่วน interface เช่นเดิม

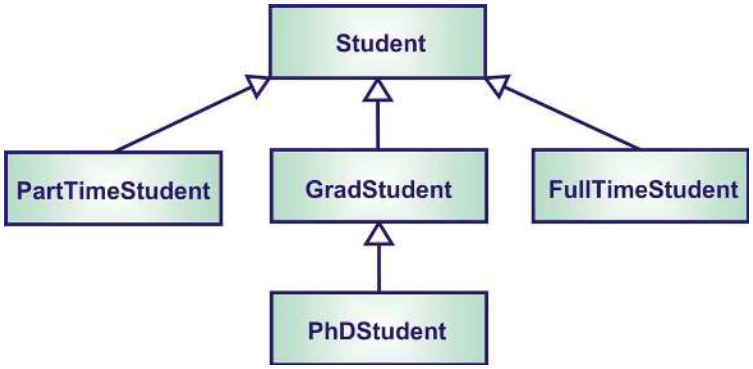
4.3.2 การสืบทอด

คุณลักษณะเด่นประการหนึ่งของการโปรแกรมเชิงอ็อบเจกต์คือ ความสามารถที่จะนำเอาโปรแกรมที่ออกแบบไว้แล้วมาใช้ใหม่ได้ ซึ่งสำหรับโปรแกรมเชิงกระบวนการจะทำได้โดยการกำหนดคำสั่งที่ต้องใช้บ่อยไว้ในโปรแกรมย่อย (ฟังก์ชัน หรือ Procedure) ซึ่งจะมีความซับซ้อนในการออกแบบเพื่อให้ได้โปรแกรมย่อยที่เหมาะสม ทำให้การนำโปรแกรมมาใช้ใหม่ของโปรแกรมเชิงกระบวนการค่อนข้างจะเป็นไปได้ยากกว่า และเมื่อโปรแกรมมีความซับซ้อนขึ้นการนำโปรแกรมมาใช้ใหม่ก็เป็นไปได้ยากขึ้น เนื่องจากเราไม่สามารถออกแบบโปรแกรมครอบคลุมปัญหาทั้งหมด ไว้ล่วงหน้าได้

วิธีการนำโปรแกรมมาใช้ใหม่ของโปรแกรมเชิงอ็อบเจกต์ จะใช้หลักการของการสืบทอด ซึ่งเป็นการนิยามคลาสใหม่จากรูปแบบของคลาสที่มีอยู่แล้ว โดยคลาสใหม่จะนำคุณลักษณะและเมธอดของคลาสเดิมมาใช้ได้ โดยทั่วไปคลาสต่างๆ ที่มีอยู่จะมีโครงสร้างที่มีความสัมพันธ์กันตามลำดับชั้น เราสามารถที่จะออกแบบโปรแกรมเชิงอ็อบเจกต์เพื่อสร้างคลาสที่เป็นคลาสแบบทั่วไป (Generalized Class) ซึ่งจะมีคุณลักษณะและเมธอดเพื่อให้คลาสอื่นๆ ที่เป็นคลาสเฉพาะ (Specific Class) สืบทอดได้ คลาสที่เป็นคลาสแบบทั่วไปจะเรียกว่าเป็น superclass หรือ parent class ส่วนคลาสที่เป็นคลาสเฉพาะที่สืบทอดมาจะเรียกว่าเป็น subclass หรือ child class

รูปที่ 4.8 แสดงตัวอย่างของคลาสที่ใช้หลักการของการสืบทอด ตัวอย่างนี้จะมีคลาส `Student` เป็นคลาสแบบทั่วไป และมีคลาส `GradStudent, PartTimeStudent` และ `FullTimeStudent` สืบทอดมาจากคลาส `Student` และคลาส `PhDStudent` จะสืบทอดมาจากคลาส `GradStudent` อีกชั้นหนึ่ง การสืบทอดจะมีผลให้คุณลักษณะและเมธอดของ Superclass สืบทอดไปยัง Subclass ตัวอย่างเช่น ถ้าคลาส `Student` ในรูปที่ 4.8 มีคุณลักษณะ `id, name, gpa` และ `MIN_GPA` และมีเมธอด `setId(), setName(), setGpa()` และ `showDetails()` คลาส `GradStudent, PartTimeStudent` และ `FullTimeStudent` ก็จะสืบทอดคุณลักษณะและเมธอดเหล่านั้นมาด้วย นอกจากนี้คลาส `PhDStudent` ที่สืบทอดมาจากคลาส `GradStudent` ก็จะได้รับคุณลักษณะและเมธอดที่สืบทอดมาจากคลาส `Student` ที่เป็น superclass ของคลาส `GradStudent` ด้วย

ข้อดีของการสืบทอดคือ ความสามารถในการที่จะนำโปรแกรมเดิมมาพัฒนาเพิ่มเติมใหม่ได้ง่ายขึ้น และยังช่วยทำให้โปรแกรมแต่ละโปรแกรมมีขนาดเล็ก ซึ่งทำให้ง่ายต่อการเข้าใจและการปรับปรุงแก้ไขโปรแกรมทำได้ง่ายขึ้น โดยในภาษาจาวาจะใช้คีย์เวิร์ด `extends` เพื่อระบุการสืบทอด โปรแกรมที่ 4.4 แสดงตัวอย่างของคลาสที่ใช้หลักการในการสืบทอดตามไดอะแกรมของคลาสในรูปที่ 4.8



รูปที่ 4.8 ตัวอย่างของคลาสที่ใช้หลักการของการสืบทอด

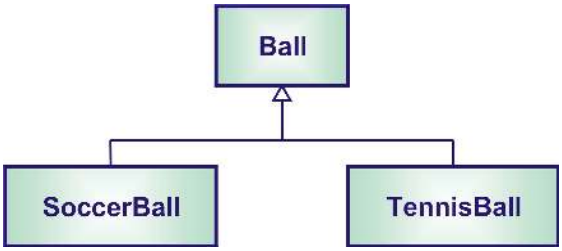
<u>โปรแกรมที่ 4.4</u> <u>ตัวอย่างการใช้หลักการในการสืบทอด</u>

```
public class PartTimeStudent extends Student { }

-------------------------------------------------------
public class FullTimeStudent extends Student { }

-------------------------------------------------------
public class GradStudent extends Student {
    private String thesisTitle;
    private String supervisor;
    public void setThesisTitle(String t) {
        thesisTitle = t;
    }
    public void setSupervisor(String s) {
        supervisor = s;
    }
}

-------------------------------------------------------
public class PhDStudent extends GradStudent {
    public boolean passQualify;
    public boolean isPassQualify() {
        return passQualify;
    }
}
```

4.3.3 การมีได้หลายรูปแบบ

หลักการของการมีได้หลายรูปแบบคือ คุณสมบัติของโปรแกรมเชิงอ็อบเจกต์ที่สามารถตอบสนองต่อข่าวสาร (เมธอด) เดียวกันด้วยวิธีการที่ต่างกัน และสามารถกำหนดอ็อบเจกต์ได้หลายรูปแบบ หลักการของการมีได้หลายรูป

แบบเป็นหลักการที่สืบเนื่องมาจากหลักการของการสืบทอด ตัวอย่างเช่น รูปที่ 4.9 มีคลาส `Ball` ซึ่งเป็น superclass ของคลาส `SoccerBall` และ `TennisBall`



รูปที่ 4.9 ตัวอย่างของคลาส `Ball`

คลาส `Ball` อาจมีเมธอด `throwBall()` เพื่อกำหนดพฤติกรรมการโยนลูกบอล ซึ่ง subclass ที่ชื่อ `SoccerBall` และ `TennisBall` อาจกำหนดวิธีการการโยนลูกบอลแต่ละชนิดที่ต่างกันกล่าวคือเขียนคำสั่งในเมธอด `throwBall()` ที่สืบทอดมาด้วยชุดคำสั่งที่ต่างกันดังแสดงในโปรแกรมที่ 4.5 หลักการของการมีได้หลายรูปแบบยังรวมไปถึงความ สามารถที่จะอ้างอิงอ็อบเจกต์ที่สืบทอดมาได้หลายรูปแบบ ตัวอย่างเช่นคลาส `SoccerBall` ที่สืบทอดมาจากคลาส `Ball` สามารถที่จะอ้างอิงอ็อบเจกต์ของคลาส `SoccerBall` ได้ทั้งในรูปแบบของ `SoccerBall` หรือ `Ball` ได้ ตัวอย่างเช่นคำสั่ง

```
SoccerBall b2 = new SoccerBall();
Ball b3 = new SoccerBall();
```

ซึ่งข้อดีของการใช้หลักการของการมีได้หลายรูปแบบคือ ทำให้โปรแกรมสามารถปรับเปลี่ยนหรือเพิ่มเติมได้ง่ายขึ้น

โปรแกรมที่ 4.5 ตัวอย่างการใช้หลักการของการมีได้หลายรูปแบบ

```
public class Ball {
    public void throwBall() { }
}

-----------------------------------------------------------
public class SoccerBall extends Ball {
    public void throwBall() {
        System.out.println("Throwing soccerball");
    }
}

-----------------------------------------------------------
public class TennisBall extends Ball {
    public void throwBall() {
        System.out.println("Throwing tennisball");
    }
}

-----------------------------------------------------------
public class TestBall {
    public static void main(String args[]) {
        Ball b1 = new Ball();
        SoccerBall b2 = new SoccerBall();
        Ball b3 = new SoccerBall();
    }
}
```

4.4 การเขียนโปรแกรมเชิงอ็อบเจกต์เพื่อสร้างคลาสแบบ abstract และอินเตอร์เฟส

โปรแกรมเชิงอ็อบเจกต์นอกเหนือจากการนิยามคลาสแล้ว ในบางกรณีเราอาจจำเป็นที่จะต้องนิยามเมธอดต่างๆ ภายในคลาส ขึ้นมาก่อนได้ โดยยังไม่ต้องระบุคำสั่งหรือวิธีการของคลาสนั้น อาทิเช่นเราอาจต้องการกำหนดคลาสชื่อ `Vehicle` (ยานพาหนะ) ซึ่งอาจมีเมธอดที่ชื่อ `move()` อยู่ ทั้งนี้อาจมีคลาสอื่นๆที่จะสืบทอดจาก `Vehicle` เช่น `Car` หรือ `Motorbike` ซึ่งจะมีวิธีการหรือคำสั่งในการ `move()` ที่แตกต่างกันภาษาจาวาได้กำหนดคลาสลักษณะนี้ไว้สองแบบคือ คลาสแบบ `abstract` และอินเตอร์เฟส

4.4.1 คลาสแบบ `abstract`

คลาสที่มี modifier เป็น `abstract` จะหมายความว่าคลาสนั้นยังเป็นคลาสที่ไม่สมบูรณ์ โดยมีเมธอดแบบ `abstract` ซึ่งเป็นเมธอดที่ยังไม่สมบูรณ์อย่างน้อยหนึ่งเมธอดอยู่ในคลาส เมธอดแบบ `abstract` จะมีรูปแบบการ

ประกาศดังนี้

```
[modifier] abstract return_type methodName([arguments]);
```

ทั้งนี้เมธอดแบบ abstract เป็นเมธอดที่ยังไม่สมบูรณ์ เนื่องจากไม่มีบล็อกคำสั่ง ({ }) อยู่ภายในเมธอด โปรแกรมที่ 4.6 แสดงตัวอย่างของคลาส Student ที่เป็นคลาสแบบ abstract คลาสนี้จะมีเมธอด showDetails() ซึ่งเป็นเมธอดแบบ abstract อยู่ภายในคลาส

<u>โปรแกรมที่ 4.6 ตัวอย่างคลาสแบบ abstract</u>

```
public abstract class Student {
      protected String id;
      protected String name;
      protected double gpa;
      public void setId(String ID) {
         id = ID;
      }
      public void setName(String n) {
         name = n;
      }
      public void setGpa(double GPA) {
         gpa = GPA;
      }
      public abstract void showDetails();
}
```

คลาสแบบ abstract กำหนดขึ้นมาเพื่อให้คลาสอื่นสืบทอด โดยคลาสที่มาสืบทอดจะต้องกำหนดบล็อกคำสั่งในเมธอดที่ยังไม่สมบูรณ์ ทั้งนี้เราจะไม่สามารถจะสร้างอ็อบเจกต์ของคลาสแบบ abstract ได้ โปรแกรมที่ 4.7 แสดงตัวอย่างของคลาส FullTimeStudent ที่สืบทอดมาจากคลาส Student คลาสนี้จะต้องมีการกำหนดบล็อกคำสั่งในเมธอด showDetails() เพื่อทำให้เป็นเมธอดที่สมบูรณ์ ซึ่งจะทำให้คลาส FullTimeStudent เป็นคลาสที่สามารถใช้ในการสร้างอ็อบเจกต์ได้ อนึ่งถึงแม้ว่าเราจะไม่สามารถสร้างอ็อบเจกต์ของคลาส Student ได้แต่เราสามารถประกาศอ็อบเจกต์ของคลาส Student และสร้างอ็อบเจกต์ของคลาส FullTimeStudent ได้ตามหลักการของการมีได้หลายรูปแบบดังนี้

```
Student s1 = new FullTimeStudent();
```

โปรแกรมที่ 4.7 ตัวอย่างคลาสที่สืบทอดมาจากคลาสแบบ abstract

```
public class FullTimeStudent extends Student {
      private int credit;
      private final int MAX_YEAR = 4;
      public FullTimeStudent(int c) {
            credit = c;
      }
      public void showDetails() {
            System.out.println("ID: "+id);
            System.out.println("Name: "+name);
            System.out.println("GPA: "+gpa);
            System.out.println("Credit: "+credit);
      }
}
```

4.4.2 อินเตอร์เฟส

อินเตอร์เฟส (interface) มีลักษณะคล้ายกับคลาสแบบ abstract แต่จะประกอบด้วยเมธอดที่ยังไม่สมบูรณ์เท่านั้น โดยมีรูปแบบดังนี้

```
[modifier] interface InterfaceName {
    [methods();]
}
```

โดยที่

- InterfaceName คือชื่อของอินเตอร์เฟส
- methods() เป็นเมธอดที่ยังไม่สมบูรณ์ เช่นเดียวกับเมธอดแบบ abstract แต่ไม่ต้องใช้คีย์เวิร์ด abstract

โปรแกรมที่ 4.8 แสดงตัวอย่างของกรณีกำหนดอินเตอร์เฟส Student อินเตอร์เฟสนี้จะมีเมธอด 4 เมธอดที่ยังไม่มีบล็อกคำสั่งคือเมธอด setId(), setName(), setGpa() และ showDetails()

โปรแกรมที่ 4.8 ตัวอย่างอินเตอร์เฟส

```
public interface Student {
      public void setId(String ID);
      public void setName(String n);
      public void setGpa(double GPA);
      public void showDetails();
}
```

อินเตอร์เฟสกำหนดขึ้นมาเพื่อให้คลาสอื่นนำไปใช้งานโดยใช้คีย์เวิร์ด `implements` โดยมีรูปแบบดังนี้

```
[modifier] class ClassName implements InterfaceName {
    [methods();]
}
```

โปรแกรมที่ 4.9 ได้แสดงตัวอย่างของคลาส `PartTimeStudent` ที่ `implements` อินเตอร์เฟส `Student` คลาสที่ `implements` อินเตอร์เฟสจะต้องเขียนบล็อกคำสั่งในเมธอดทุกเมธอดที่กำหนดไว้ในอินเตอร์เฟส เพื่อทำให้เมธอดเหล่านี้สมบูรณ์ และสามารถสร้างอ็อบเจกต์ของคลาสนั้นได้

อินเตอร์เฟสจะเหมือนกับคลาสแบบ `abstract` ตรงที่เราจะไม่สามารถสร้าง อ็อบเจกต์ของอินเตอร์เฟสได้ ประโยชน์ของอินเตอร์เฟสก็คือการกำหนดรูปแบบของเมธอดต่างๆ ที่คลาสอื่นๆ จะต้อง `implements` ไว้ล่วงหน้า ซึ่งสามารถอาศัยหลักการของการมีได้หลายรูปแบบมาเรียกใช้เมธอดเหล่านั้นได้จากคลาสที่ `implements` อินเตอร์เฟส อาทิเช่น คำสั่ง

```
Student  s1 = new PartTimeStudent(6);
s1.setId("1111");
s1.setName("Thana");
s1.setGpa(3.00);
s1.showDetails();
```

อินเตอร์เฟสยังช่วยในการสืบทอดแบบหลายคลาสได้ (Multiple Inheritance) เนื่องจากภาษาจาวากำหนดให้คลาสใดๆ สามารถสืบทอดคลาสอื่นได้เพียงคลาสเดียวเท่านั้น แต่จะสามารถ `implements` อินเตอร์เฟสได้หลายอินเตอร์เฟส อาทิเช่น

```
public class MyApplet extends Applet implements
                  MouseListener,ActionListener {
      ...
}
```

เป็นโปรแกรมจาวาแอปเพล็ตที่สืบทอดมาจากคลาส `Applet` แต่ขณะเดียวกันเป็นคลาสที่ `implements` อินเตอร์เฟส `MouseListener` และ `ActionListener` เป็นต้น

โปรแกรมที่ 4.9 ตัวอย่างคลาสที่ implements อินเตอร์เฟส

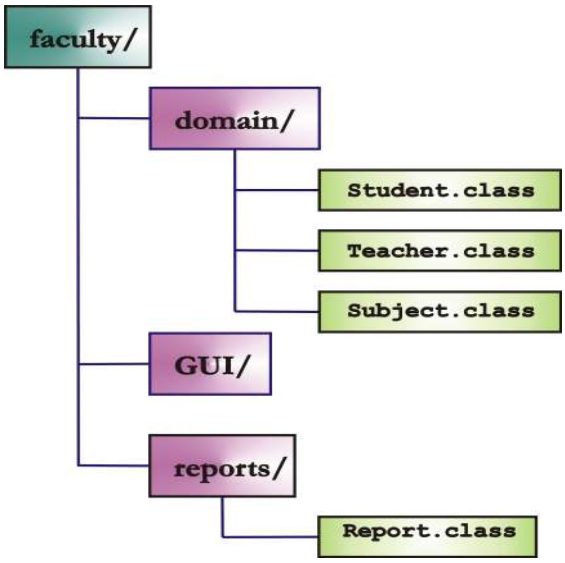
```
public class PartTimeStudent implements Student {
      private String id;
      private String name;
      private double gpa;
      private int credit;
      private final int MAX_YEAR = 8;
      public PartTimeStudent(int c) {
            credit = c;
      }
      public void setId(String ID) {
            id = ID;
      }
      public void setName(String n) {
            name = n;
      }
      public void setGpa(double GPA) {
            gpa = GPA;
      }
      public void showDetails() {
            System.out.println("ID: "+id);
            System.out.println("Name: "+name);
            System.out.println("GPA: "+gpa);
            System.out.println("Credit: "+credit);
      }
}
```

4.5 แพคเก็จ

แพคเก็จจะเป็นที่รวบรวมคลาสและอินเตอร์เฟสต่างๆ ที่มีหน้าที่การทำงานคล้ายกันไว้ในที่เดียวกัน โดยทั่วไปโปรแกรมภาษาจาวาจะประกอบไปด้วยคลาสหลายคลาสที่ประกาศอยู่ในโปรแกรมไฟล์ต่างๆ ไฟล์โปรแกรมภาษาจาวาแต่ละไฟล์อาจมีคลาสที่ประกาศไว้หลายคลาสได้ แต่จะมีคลาสที่มี access modifier เป็น `public` ได้เพียงคลาสเดียว ซึ่งวิธีการปฏิบัติที่นิยมใช้คือการกำหนดให้ในแต่ละโปรแกรมไฟล์มีคลาสเพียงคลาสเดียว โดยมีชื่อไฟล์เป็นชื่อเดียวกับชื่อคลาส

การจัดระบบไฟล์และคลาสจึงจำเป็นอย่างยิ่งสำหรับการพัฒนาโปรแกรมขนาดใหญ่ เราสามารถที่จะแบ่งโปรแกรมให้เป็นแพคเก็จต่างๆ โดยจัดกลุ่มของคลาสที่ทำงานคล้ายกันไว้ในแพคเก็จเดียวกัน แพคเก็จแต่ละแพคเก็จจะใช้ในการเก็บคลาสหรือแพคเก็จย่อยได้

ซึ่งโปรแกรมจาวาจะสร้างไดเร็กทอรี่ (Directory) ที่มีชื่อเดียวกับแพคเก็จ ในการเก็บคลาสหรือแพคเก็จย่อยเหล่านั้น รูปที่ 4.10 แสดงตัวอย่างโครงสร้างไดเร็กทอรี่ของโปรแกรมที่มีคลาสและแพคเก็จหลายๆ ตัว โปรแกรมจะอยู่ในแพคเก็จ `faculty` ซึ่งจะประกอบด้วย แพคเก็จย่อย 3 แพคเก็จคือ `domain`, `GUI` และ `reports` โดยโปรแกรม `Student.class`, `Teacher.class` และ `Subject.class` จะอยู่ในแพคเก็จย่อย `domain` ส่วนโปรแกรม `Report.class` จะอยู่ในแพคเก็จย่อย `reports`



รูปที่ 4.10 ตัวอย่างแพคเก็จ

4.5.1 การประกาศและใช้แพคเก็จ

โครงสร้างโปรแกรมไฟล์ของภาษาจาวาจะประกอบด้วยคำสั่งดังนี้

```
[<package_declaration>]
[<import_declaration>]
[<class_declaration>]
```

โดยที่

- `package_declaration` เป็นการระบุว่าโปรแกรมไฟล์อยู่ในแพคเก็จใด
- `import_declaration` เป็นคำสั่งในการเรียกใช้แพคเก็จอื่นๆ
- `class_declaration` เป็นคำสั่งในการประกาศคลาส

คำสั่งในการระบุแพคเก็จจะมีรูปแบบดังนี้

```
package <package_name>[<sub_package_name>]
```

ตัวอย่างเช่น

```
package faculty.domain;
```

หรือ

```
package faculty;
```

โปรแกรมไฟล์หนึ่งไฟล์จะมีคำสั่ง package ได้เพียงคำสั่งเดียว โดยจะต้องเป็นคำสั่งแรกของโปรแกรม ในกรณีที่ไม่มีคำสั่ง package โปรแกรมไฟล์นั้นจะถูกกำหนดไว้ในแพคเก็จ default

การเรียกใช้แพคเก็จอื่น ๆ จะทำได้โดยการใช้คำสั่ง import ซึ่งมีรูปแบบของคำสั่งดังนี้

```
import <package_name>[.<sub_package_name>].<Class_name>
หรือ
import <package_name>[.<sub_package_name>].*;
```

ตัวอย่างเช่น

```
import faculty.reports.Report;
```

หรือ

```
import java.awt.*;
```

คำสั่ง import จะต้องกำหนดไว้ก่อนคำสั่งการประกาศคลาส ซึ่งโปรแกรมไฟล์ภาษาจาวาแต่ละโปรแกรมสามารถมีคำสั่ง import ได้หลายคำสั่ง

4.6 Unified Modeling Language

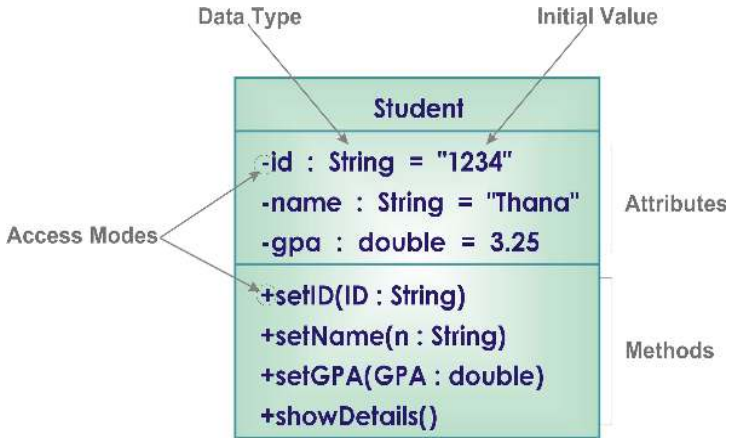
Unified Modeling Language (UML) เป็นภาษาที่ใช้รูปกราฟิกเพื่อจำลองระบบซอฟต์แวร์ UML ถูกพัฒนาขึ้นเพื่อจำลองโปรแกรมเชิงอ็อบเจกต์ในต้นทศวรรษที่ 1990 ซึ่งในปัจจุบัน UML ได้กลายเป็นมาตรฐานที่ใช้ในการจำลองระบบซอฟต์แวร์ตามข้อกำหนดของ OMG (Object Management Group) หนังสือเล่มนี้จะใช้สัญลักษณ์ของ UML ในการจำลองระบบโปรแกรมต่างๆ แต่เนื่องจาก UML เป็นภาษาที่มีข้อกำหนดต่างๆ มาก ดังนั้นเนื้อหาในบทนี้จะกล่าวถึงเฉพาะส่วนต่างๆ ที่จำเป็นต่อความเข้าใจที่จะใช้ในหนังสือเล่มนี้เท่านั้น

4.6.1 ไดอะแกรมของคลาส

ไดอะแกรมของคลาส (Class Diagram) เป็นสัญลักษณ์ที่ใช้แสดงคลาสใน UML โดยไดอะแกรมของคลาสจะประกอบด้วยส่วนต่างๆ สามส่วนคือ ชื่อของคลาส คุณลักษณะภายในคลาส และเมธอดภายในคลาส ดังแสดงในรูปที่ 4.11 ไดอะแกรมของคลาสจะระบุข้อมูลต่างๆ สำหรับคุณลักษณะภายในคลาสคือ ชื่อของคุณลักษณะ ชนิดข้อมูล สถานะการเข้าถึง (access modes) และค่าเริ่มต้น ส่วนเมธอดภายในคลาสจะระบุถึงสถานะการเข้าถึง ชื่อของเมธอด ชนิดข้อมูลของ argument และชนิดข้อมูลของค่าที่ส่งกลับ สำหรับสถานะการเข้าถึง UML ได้กำหนดสัญลักษณ์ต่างๆ ไว้ดังนี้

+ สำหรับ `public`

- สำหรับ `private`



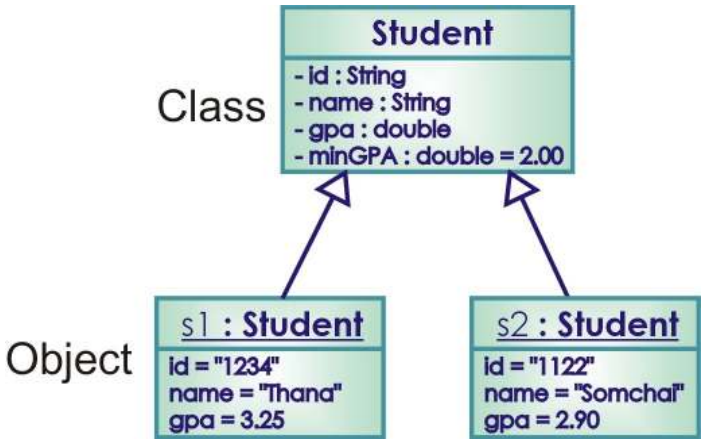
รูปที่ 4.11 ตัวอย่าง UML ของคลาส `Student`

ไดอะแกรมของคลาสสามารถที่จะแสดงความสัมพันธ์ของคลาส ที่มีการสืบทอดกัน โดยการใช้ลูกศรเป็นสัญลักษณ์ดังแสดงตัวอย่างในรูปที่ 4.8 นอกจากนี้ไดอะแกรมของคลาสยังสามารถที่จะแสดงความสัมพันธ์หรือแสดงคุณสมบัติอื่นๆ เช่น แสดงว่าเป็นคลาสแบบ `interface` ได้อีกด้วย

4.6.2 ไดอะแกรมของอ็อบเจกต์

UML สามารถที่จะแสดงให้เห็นได้ว่าอ็อบเจกต์ที่สร้างขึ้นมาเป็นอ็อบเจกต์ของคลาสใด และมีค่าของคุณลักษณะต่างๆ อย่างไร โดยใช้ไดอะแกรมของอ็อบเจกต์ (Object Diagram) ที่ประกอบไปด้วยส่วนต่างๆ สองส่วน คือ ส่วนที่ระบุชื่อของอ็อบเจกต์ และส่วนที่ระบุค่าของคุณลักษณะภายในอ็อบเจกต์ ดังตัวอย่างในรูปที่ 4.12 ซึ่งเป็นการ

แสดงอ็อบเจกต์ s1 และ s2 ของคลาส Student จากรูปจะเห็นได้ว่าสัญลักษณ์ลูกศรที่เป็นเส้นปะจะเป็นการระบุว่าอ็อบเจกต์ s1 เป็นอ็อบเจกต์ของคลาส Student



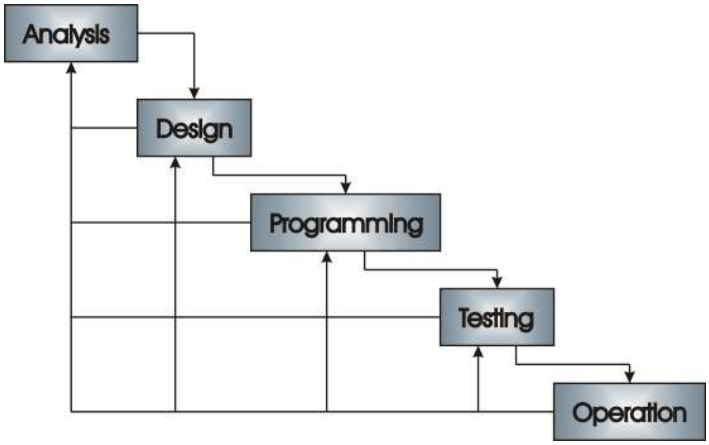
รูปที่ 4.12 ไดอะแกรมของอ็อบเจกต์ s1 และ s2 ของคลาส Student

4.7 ขั้นตอนการพัฒนาโปรแกรม

การพัฒนาโปรแกรมคอมพิวเตอร์ ไม่ได้หมายถึงเพียงแค่การเขียนโปรแกรมคำสั่งภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง แต่จะรวมถึงขั้นตอนทั้งหมดในการพัฒนาโปรแกรม ซึ่งขั้นตอนการพัฒนาโปรแกรมนี้เรียกว่า วัฏจักรการพัฒนาโปรแกรม (Software Development Life Cycle เรียกย่อว่า SDLC) ซึ่งจะประกอบไปด้วยขั้นตอนต่างๆ 5 ขั้นตอนคือ

1. ขั้นตอนการวิเคราะห์ (Analysis)
2. ขั้นตอนการออกแบบ (Design)
3. ขั้นตอนการเขียนโปรแกรม (Programming)
4. ขั้นตอนการทดสอบ (Testing)
5. ขั้นตอนการทำงาน (Operation)

ซึ่งแต่ละขั้นตอนมักไม่ได้ผลลัพธ์ที่สมบูรณ์ที่จะสามารถทำขั้นตอนต่อไปได้ โดยไม่ต้องกลับมาทำขั้นตอนเดิมอีก แต่ระหว่างทำงานในแต่ละขั้นตอนอาจต้องย้อนกลับไปทำงานในขั้นตอนอื่นๆ ก่อนหน้านั้นดังแสดงรูปที่ 4.13



รูปที่ 4.13 ขั้นตอนการพัฒนาโปรแกรม

ขั้นตอนการวิเคราะห์ เป็นการศึกษาความเป็นไปได้ (feasibility study) ของปัญหาที่ต้องการพัฒนาโปรแกรม ทีมพัฒนาโปรแกรมจะทำการวิเคราะห์ปัญหาและพิจารณาว่าสามารถที่จะพัฒนาโปรแกรมคอมพิวเตอร์เพื่อแก้ไขปัญหาได้หรือไม่ ผลลัพธ์ที่ได้จากขั้นตอนนี้คือข้อกำหนดตามความต้องการ (requirement specification) ซึ่งจะอธิบายคุณลักษณะของโปรแกรม โดยข้อกำหนดนี้จะเขียนอยู่ในรูปแบบต่างๆ เพื่อให้ผู้ใช้ ผู้ปฏิบัติงาน และนักพัฒนาโปรแกรมสามารถเข้าใจได้ และสามารถทดสอบโปรแกรมที่พัฒนาขึ้นได้ว่าถูกต้องและตรงตามความต้องการของผู้ใช้หรือไม่

ขั้นตอนการออกแบบ จะเป็นการทำให้ข้อกำหนดตามความต้องการเปลี่ยนไปอยู่ในรูปแบบของรายละเอียดของโปรแกรมที่จะต้องพัฒนา ซึ่งในการออกแบบโดยใช้หลักการเชิงอ็อบเจกต์ก็อาจจะได้ไดอะแกรมต่างๆ ในรูปของ UML เพื่อที่จะอธิบายคลาสและอ็อบเจกต์ต่างๆ ที่สอดคล้องกับข้อกำหนดตามความต้องการ

ขั้นตอนการเขียนโปรแกรม จะเป็นการเขียนโปรแกรมโดยใช้ภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่งให้เป็นไปตามโปรแกรมที่ได้ออกแบบไว้ ซึ่งการเลือกใช้ภาษาคอมพิวเตอร์ควรที่จะต้องสอดคล้องกับวิธีการที่ได้ออกแบบไว้ ในกรณีที่ขั้นตอนการออกแบบเป็นการใช้หลักการเชิงอ็อบเจกต์ก็ควรที่จะต้องใช้ภาษาคอมพิวเตอร์เชิงอ็อบเจกต์เช่น ภาษาจาวาในการเขียนโปรแกรม ขั้นตอนนี้จะเป็นขั้นตอนที่ไม่ยากนักเนื่องจากเป็นการเขียนโปรแกรมตามข้อกำหนดที่ได้จากการออกแบบ ดังนั้นการพัฒนาโปรแกรมที่ดีจะต้องมีขั้นตอนการออกแบบที่ดีเพื่อให้ได้โปรแกรมที่ถูกต้อง

ขั้นตอนการทดสอบ จะเป็นขั้นตอนที่จะทดสอบโปรแกรมที่ได้พัฒนาขึ้น โดยอาจเป็นการทดสอบการทำงานของโปรแกรมจากข้อมูลที่นำมาทดสอบต่างๆ และตรวจสอบดูว่าโปรแกรมสามารถทำงานได้ถูกต้องตามข้อกำหนดตามความต้องการหรือไม่ การทดสอบโปรแกรมเชิงอ็อบเจกต์อาจเป็นการทดสอบทีละยูนิต (Unit Testing) หรือการ

ทดสอบโดยรวม (Integration Testing) การทดสอบทีละยูนิตเป็นการทดสอบการทำงานของแต่ละคลาส ส่วนการทดสอบโดยรวมเป็นการทดสอบการทำงานเมื่อรวมคลาสต่างๆ เข้าด้วยกัน

ขั้นตอนการทำงาน เป็นขั้นตอนสุดท้ายที่ได้ส่งมอบโปรแกรมให้กับผู้ใช้งานแล้ว ซึ่งโปรแกรมที่ส่งมอบก็เปรียบเสมือนสินค้าอื่นๆ เช่น บ้านหรือรถยนต์ ที่อาจมีข้อบกพร่องที่ต้องแก้ไขและต้องมีการบำรุงรักษา ผู้ใช้โปรแกรมอาจพบข้อผิดพลาดของโปรแกรมในระหว่างการใช้งาน ซึ่งข้อผิดพลาดเหล่านี้อาจเกิดขึ้นจากขั้นตอนการออกแบบ ขั้นตอนการเขียนโปรแกรม หรือแม้กระทั่งเกิดขึ้นในขั้นตอนการวิเคราะห์โปรแกรม ที่อาจวิเคราะห์ความต้องการผิดพลาด ซึ่งจากข้อผิดพลาดเหล่านี้ทำให้นักพัฒนาโปรแกรมอาจจะต้องนำโปรแกรมมาแก้ไข ขั้นตอนนี้เปรียบเสมือนขั้นตอนการบำรุงรักษาโปรแกรม (software maintenance) ซึ่งจะมีระยะเวลายาวนานตามระยะเวลาการใช้งานของโปรแกรม ซึ่งมักจะพบว่าขั้นตอนนี้จะมีค่าใช้จ่ายในการพัฒนาโปรแกรมมากที่สุดเมื่อเทียบกับขั้นตอนอื่นๆ

จากที่กล่าวมาข้างต้นแล้วว่า วัฏจักรของการพัฒนาโปรแกรมไม่สามารถที่จะทำแต่ละขั้นตอนให้สมบูรณ์ 100% ได้ แต่จะต้องมีการย้อนกลับมาทำขั้นตอนอื่นๆ เสมอ โดยทั่วไปนักพัฒนาโปรแกรมที่ดีจะต้องให้ความสำคัญกับขั้นตอนการวิเคราะห์ และขั้นตอนการออกแบบให้มากที่สุด เพราะโปรแกรมที่มีการออกแบบที่ดีจะช่วยลดความยุ่งยากในขั้นตอนการทำงาน (บำรุงรักษา) ซึ่งจะมีผลทำให้ต้นทุนในการพัฒนาโปรแกรมถูกลง

สรุปเนื้อหาของบท

- โปรแกรมเชิงอ็อบเจกต์จะมีคำนิยามที่สำคัญสองคำคือ อ็อบเจกต์และคลาส
- อ็อบเจกต์คือสิ่งต่างๆ ที่มีอยู่ในชีวิตประจำวันจะประกอบไปด้วยคุณลักษณะและ เมธอด
- คลาสเปรียบเสมือนพิมพ์เขียวของอ็อบเจกต์ อ็อบเจกต์จะถูกสร้างมาจากคลาส อ็อบเจกต์หลายอ็อบเจกต์ สามารถถูกสร้างจากคลาสหนึ่งคลาสได้
- คุณลักษณะของอ็อบเจกต์คือข้อมูลที่เก็บอยู่ในอ็อบเจกต์ ซึ่งจะแบ่งออกเป็นตัวแปรและค่าคงที่
- คุณลักษณะของคลาสเป็นคุณลักษณะที่ใช้ร่วมกันของทุกอ็อบเจกต์
- เมธอดคือวิธีการเพื่อใช้ในการจัดการกับคุณลักษณะของอ็อบเจกต์หรือคุณลักษณะของคลาส
- ภาษาจาวามีนิยามในการเขียนโปรแกรมเชิงอ็อบเจกต์เพื่อประกาศคลาส คุณลักษณะ เมธอด และอ็อบเจกต์
- โปรแกรมเชิงอ็อบเจกต์จะมีคุณลักษณะเด่นอยู่สามประการคือ การห่อหุ้ม การสืบทอด และการมีได้หลายรูปแบบ
- การห่อหุ้มคือการที่ให้คุณลักษณะถูกห่อหุ้มอยู่ภายในเมธอด โดยกำหนดให้คุณลักษณะมี access modifier เป็น `private` และกำหนดให้เมธอดมี ccess modifier เป็น `public`

- ข้อดีของการห่อหุ้มคือการซ่อนเร้นข้อมูลและความเป็นโมดูล
- การสืบทอดคือการที่คลาสใหม่สามารถนำเอาคุณลักษณะและเมธอดของคลาสที่ออกแบบไว้แล้วมาใช้ได้
- การมีได้หลายรูปแบบคือการที่กำหนดให้มีการตอบสนองต่อเมธอดเดียวกันด้วยวิธีการที่ต่างกัน และสามารถกำหนดอ็อบเจกต์ได้หลายรูปแบบ
- การสืบทอดหมายถึง การที่คลาสที่เป็น subclass สืบทอดคุณลักษณะและเมธอดมาจากคลาสที่เป็น superclass ซึ่งภาษาจาวาจะใช้คีย์เวิร์ด `extends`
- คำสั่ง `package` เป็นการระบุว่าคลาสอยู่ในแพคเก็จใด
- คำสั่ง `import` เป็นการเรียกใช้คลาสในแพคเก็จต่างๆ
- Unified Modeling Language (UML) เป็นภาษาที่ใช้รูปกราฟิกเพื่อจำลองระบบซอฟต์แวร์ ในที่นี้ได้แนะนำสัญลักษณ์ของ UML ที่สำคัญสองอย่างคือ ไดอะแกรมของคลาสและไดอะแกรมของอ็อบเจกต์
- วัฏจักรการพัฒนาโปรแกรมจะประกอบไปด้วยขั้นตอนต่างๆ 5 ขั้นตอนคือ ขั้นตอนการวิเคราะห์ ขั้นตอนการออกแบบ ขั้นตอนการเขียนโปรแกรม ขั้นตอนการทดสอบ และขั้นตอนการบำรุงรักษาโปรแกรม

บทที่ 5 การสร้างส่วนต่อประสานกราฟิกกับผู้ใช้

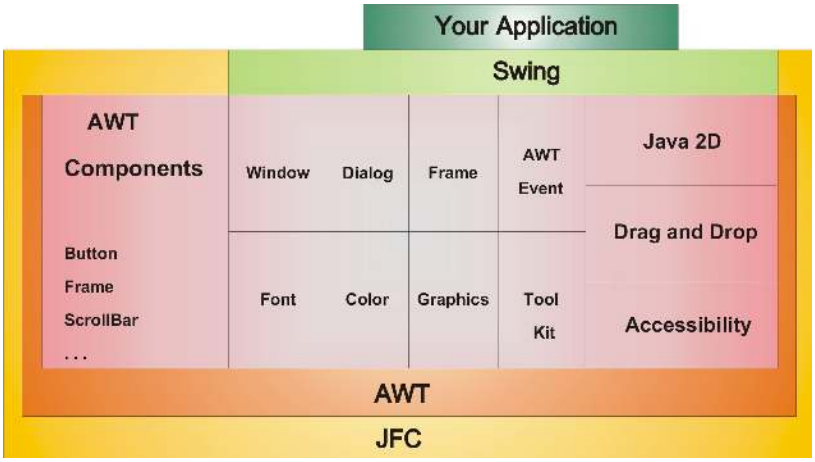
เนื้อหาในบทนี้เป็นการแนะนำการเขียนโปรแกรมจาวาประยุกต์ที่มีส่วนต่อประสานกราฟิกกับผู้ใช้ โดยจะเป็นการแนะนำคลาสและอินเตอร์เฟสที่สำคัญในแพคเก็จ `javax.swing` อธิบายคลาสประเภท `Container` และคลาสที่เป็นส่วนประกอบกราฟิกอื่นๆ แนะนำการจัดวางส่วนประกอบกราฟิกโดยใช้อ็อบเจกต์ประเภท `LayoutManager` และอธิบายการสร้างเมนู

5.1 Java Foundation Class

เนื้อหาที่ผ่านมาเน้นการพัฒนาโปรแกรมที่มีส่วนต่อประสานข้อความกับผู้ใช้ (text-mode) แต่ระบบปฏิบัติการส่วนใหญ่จะมีส่วนต่อประสานกราฟิกกับผู้ใช้ (Graphical User Interface เรียกย่อว่า GUI) ทั้งนี้เนื่องจากใช้งานง่ายกว่า ดังนั้นการพัฒนาโปรแกรมในปัจจุบันควรที่จะมีส่วนต่อประสานกราฟิกกับผู้ใช้ ซึ่งโปรแกรมลักษณะนี้เรียกว่าโปรแกรม GUI

การพัฒนาโปรแกรม GUI ผู้พัฒนาโปรแกรมต้องมีความเข้าใจชุดคำสั่งและองค์ประกอบทางด้านกราฟิกของระบบปฏิบัติการที่ต้องการพัฒนา ภาษาคอมพิวเตอร์ส่วนใหญ่จะใช้ชุดคำสั่งเฉพาะในแต่ละระบบปฏิบัติการ ดังนั้นผู้พัฒนาโปรแกรมจะต้องเรียนรู้ชุดคำสั่งในแต่ละระบบปฏิบัติการและโปรแกรม GUI ที่พัฒนาขึ้นมาจะขึ้นอยู่กับแพลตฟอร์ม (Platform Dependent) แต่ภาษาจาวาจะสนับสนุนการพัฒนาโปรแกรม GUI ที่สามารถใช้งานได้หลายแพลตฟอร์ม โดยจะใช้ชุดคำสั่งเดียวกัน อาทิเช่น เราสามารถที่จะนำโปรแกรม GUI ที่พัฒนาบนระบบปฏิบัติการ Linux มาใช้บนระบบปฏิบัติการอื่นๆ ที่มีส่วนต่อประสานกราฟิกกับผู้ใช้ได้

ชุดคำสั่งของโปรแกรมภาษาจาวาที่ใช้ในการพัฒนาโปรแกรม GUI จะอยู่ในชุดของแพคเก็จ Java Foundation Classes (JFC) ดังแสดงในรูปที่ 5.1 ซึ่งประกอบด้วยแพคเก็จต่างๆ ดังนี้



รูปที่ 5.1 ส่วนประกอบที่สำคัญของ JFC

- Abstract Window Toolkit (AWT) เป็นแพคเก็จที่ใช้ในการพัฒนาโปรแกรม GUI ขั้นพื้นฐาน ซึ่งจะให้โปรแกรม GUI ที่เป็น look and feel ที่ขึ้นอยู่กับแพลตฟอร์ม ภาษาจาวาได้กำหนดแพคเก็จ AWT ขึ้นมาตั้งแต่เวอร์ชันแรก (JDK 1.0) โดยกำหนดไว้ในแพคเก็จ `java.awt`
- Swing เป็นแพคเก็จที่มีส่วนประกอบกราฟิกที่มีคุณลักษณะและรูปแบบที่ดีกว่าส่วนประกอบกราฟิกของแพคเก็จ AWT ซึ่งจะเหมาะกับการพัฒนาโปรแกรม GUI ที่นำไปใช้งานจริง ภาษาจาวาได้กำหนดแพคเก็จ Swing ขึ้นในจาวาเวอร์ชัน 2 (Java 2) โดยกำหนดในแพคเก็จชื่อ `javax.swing` โดยทั่วไปโปรแกรมที่ใช้ชุดคำสั่งในแพคเก็จ Swing จะทำงานช้ากว่าโปรแกรม GUI ที่ใช้แพคเก็จ AWT แต่จะมีรูปแบบที่สวยงามกว่า
- Java 2D เป็นชุดคำสั่งกราฟิกที่มีคลาสที่ช่วยในการพัฒนาโปรแกรมกราฟิกสองมิติและคลาสที่ใช้ในการจัดการกับรูปภาพ
- Accessibility เป็นชุดคำสั่งที่มีคลาสในการพัฒนาโปรแกรมที่มีอินพุตหรือเอาต์พุตในลักษณะพิเศษ อาทิเช่น screen reader, screen magnifier และ audio text reader
- Drag and Drop เป็นชุดคำสั่งของเทคโนโลยีที่ช่วยในการแลกเปลี่ยนข้อมูลระหว่างโปรแกรมที่พัฒนาโดยภาษาจาวากับโปรแกรมภาษาอื่นๆ

5.1.1 แพคเก็จ AWT

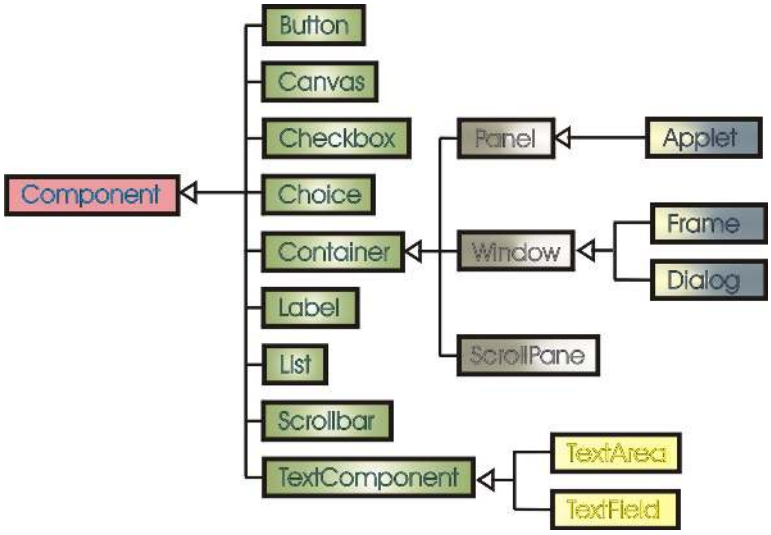
ภาษาจาวาในเวอร์ชันแรกๆ จะใช้แพคเก็จ AWT ในการพัฒนาโปรแกรม ก่อนที่จะมาพัฒนาโดยใช้แพคเก็จ Swing ทั้งนี้คลาสและอินเตอร์เฟสของ Swing หลายๆ ตัวสืบทอดมาจากคลาสของแพคเก็จ AWT เนื้อหาในที่นี้จะ

แนะนำให้รู้จักคลาสและอินเตอร์เฟสต่างๆ ที่สำคัญในแพคเก็จ AWT ซึ่งมีดังนี้

- `Component` เป็นคลาสที่เป็น superclass ของคลาสประเภทส่วนประกอบกราฟิกทุกคลาสในแพคเก็จ AWT
- `Container` เป็นคลาสที่ใช้ในการใส่ส่วนประกอบกราฟิก
- `LayoutManager` เป็นอินเตอร์เฟสที่ใช้ในการจัดตำแหน่งและขนาดของส่วนประกอบกราฟิก
- `Graphics` เป็นคลาสแบบ `abstract` ที่ใช้ในการวาดรูปกราฟิก อาทิเช่น วาดเส้น วาดสี่เหลี่ยม หรือเขียนข้อความ เป็นต้น
- `Color` เป็นคลาสที่ใช้ในการจัดการสีของส่วนประกอบกราฟิก
- `Font` เป็นคลาสที่ใช้ในการจัดการฟอนต์ของส่วนประกอบกราฟิก
- `AWTEvent` เป็นคลาสที่เกี่ยวข้องกับเหตุการณ์ (Event) ทางกราฟิก

การเขียนโปรแกรม GUI นั้นจะเป็นการสร้างอ็อบเจกต์ต่างๆ ที่เป็นอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิก ซึ่งคลาสเหล่านี้จะสืบทอดมาจากคลาสที่ชื่อ `Component` ดังแสดงในรูปที่ 5.2 ซึ่งคลาสที่เป็น subclass ของคลาส `Component` จะแบ่งเป็นสองกลุ่มคือ

1. คลาสที่เป็นคลาสประเภท `Container` เป็นคลาสที่ใช้ในการใส่ส่วนประกอบกราฟิกต่างๆ
2. คลาสที่เป็นส่วนประกอบกราฟิกอื่นๆ อาทิเช่น `Button`, `Choice` และ `List` เป็นต้น

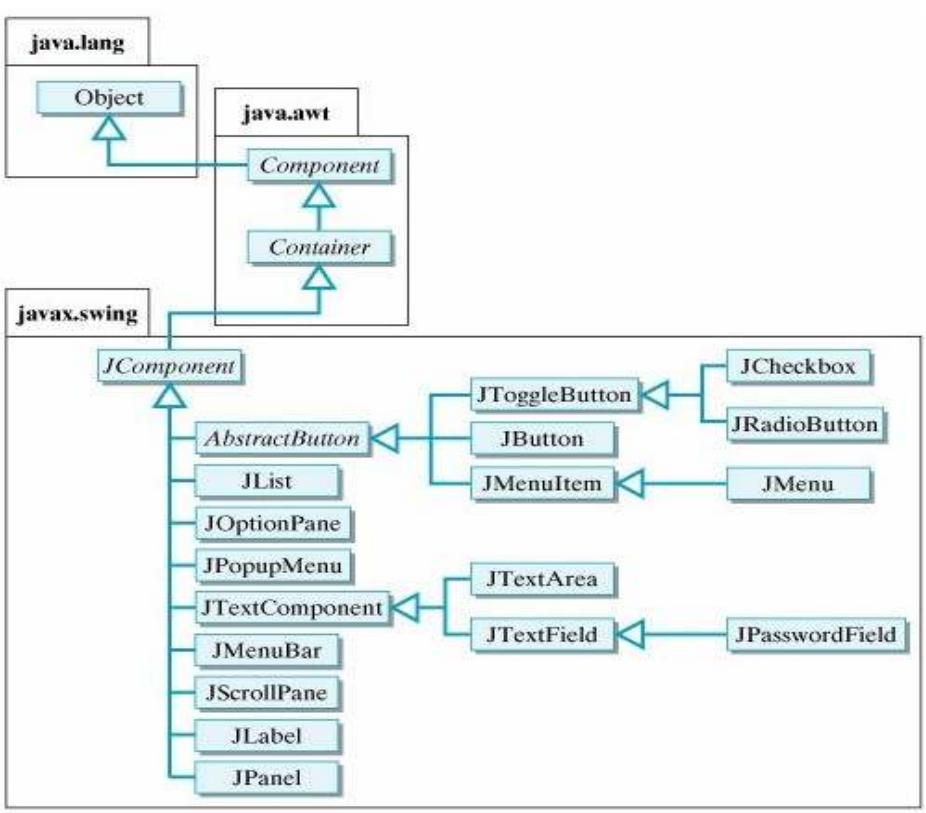


รูปที่ 5.2 คลาสต่างๆ ที่สืบทอดมาจากคลาสที่ชื่อ `Component`

5.1.2 แพคเก็จ Swing

Swing เป็นแพคเก็จสำหรับพัฒนาโปรแกรม GUI ซึ่งมีส่วนประกอบกราฟิกที่มากชนิดกว่าที่มีอยู่ในแพคเก็จ AWT นอกจากนี้ส่วนประกอบกราฟิกของแพคเก็จ Swing จะมีลักษณะที่ดีกว่าส่วนประกอบกราฟิกของแพคเก็จ AWT ส่วนประกอบกราฟิกของแพคเก็จ Swing สามารถกำหนดรูปแบบของ look and feel ที่ทำให้ได้โปรแกรม GUI ที่มีรูปแบบของกราฟิกเหมือนกันในทุกแพลตฟอร์ม ซึ่งจะแตกต่างกับแพคเก็จ AWT ที่จะมีรูปแบบของกราฟิกซึ่งขึ้นอยู่กับ look and feel ของแต่ละแพลตฟอร์ม ซึ่งการกำหนด look and feel ได้เองนี้ทำให้โปรแกรม GUI ที่ใช้อ็อบเจกต์ของแพคเก็จ Swing แสดงผลได้ช้ากว่าโปรแกรม GUI ที่ใช้อ็อบเจกต์ของแพคเก็จ AWT

แพคเก็จ Swing ประกอบด้วยคลาสต่างๆ ที่เป็นประเภทส่วนประกอบกราฟิก (Graphical Component) เพื่อนำไปใช้ในการพัฒนาโปรแกรม GUI แพคเก็จ Swing จะช่วยในการสร้างโปรแกรม GUI ประเภท look and feel โดยจะไม่ขึ้นอยู่กับแพลตฟอร์มที่ใช้งาน ภาษาจาวาได้กำหนดแพคเก็จ Swing อยู่ในแพคเก็จ `javax.swing` แพคเก็จ Swing จะประกอบไปด้วยคลาสและอินเตอร์เฟสต่างๆ เพื่อใช้ในการพัฒนาโปรแกรม GUI โดยคลาสและอินเตอร์เฟสต่างๆ เหล่านี้จะมีความสัมพันธ์ดังแสดงในรูปที่ 5.3



รูปที่ 5.3 คลาสและอินเตอร์เฟสต่างๆ ในแพคเก็จ Swing

5.2 คลาสประเภท Container

อ็อบเจกต์ของคลาส Container จะใช้ในการใส่อ็อบเจกต์ของส่วนประกอบกราฟิกหลายๆ อ็อบเจกต์ไว้เพื่อแสดงผล โปรแกรม GUI จะต้องมีการสร้างอ็อบเจกต์ของคลาสประเภท Container อย่างน้อยหนึ่งอ็อบเจกต์ ขึ้นมาก่อน เพื่อใช้ในการใส่อ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกอื่นๆ คลาสประเภท Container ที่อยู่ในแพคเก็จ AWT มีดังนี้

- Frame
- Panel
- Dialog
- Applet

โดยจะมีลำดับชั้นของการสืบทอดดังแสดงในรูปที่ 5.2

คลาสประเภท Container เป็นคลาสที่สืบทอดมาจากคลาสที่ชื่อ Component ซึ่งจัดเป็นส่วนประกอบกราฟิกชนิดหนึ่ง ดังนั้นเราจึงสามารถที่จะใส่อ็อบเจกต์ประเภท Container บางชนิดลงในอ็อบเจกต์ของคลาสประเภท Container ตัวอื่นๆ ได้อาทิเช่น ใส่อ็อบเจกต์ของคลาส Panel ลงในอ็อบเจกต์ของคลาส Frame คลาสที่ชื่อ Container เป็นคลาสแบบ abstract ซึ่งเราไม่สามารถที่จะสร้างอ็อบเจกต์ของคลาสดังกล่าวได้ แต่จะต้องสร้างอ็อบเจกต์ของคลาสอื่นๆ ที่สืบทอดมาจากคลาสที่ชื่อ Container แทน คลาสที่ชื่อ Container จะมีเมธอด add() ที่ใช้ในการใส่ส่วนประกอบกราฟิกอื่นๆ เมธอดนี้จะมีรูปแบบที่สำคัญดังนี้

- public void add(Component c)
- public void add(Component c, int position)

โดยที่

- c คืออ็อบเจกต์ของส่วนประกอบกราฟิกที่ต้องการใส่
- position คือตำแหน่งที่ต้องการวางส่วนประกอบกราฟิก

โดยปกติคลาสประเภท Container จะมีการจัดวางส่วนประกอบกราฟิกโดยกำหนดตำแหน่งและขนาดตามรูปแบบการจัดวางผัง ที่กำหนดโดยอ็อบเจกต์ประเภท LayoutManager โดยอัตโนมัติซึ่งจะกล่าวถึงในหัวข้อต่อไป

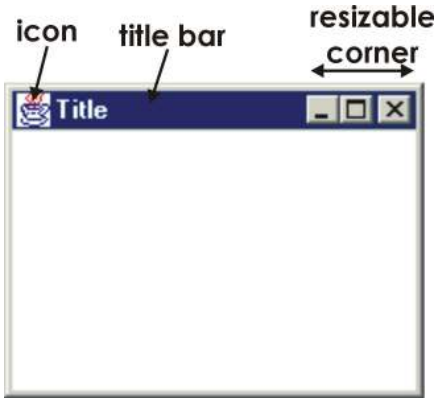
5.2.1 คลาส Frame

คลาส Frame เป็นคลาสที่สืบทอดมาจากคลาสที่ชื่อ Window ซึ่งจัดว่าเป็นคลาสประเภท Container แบบหนึ่ง โดยอ็อบเจกต์ของคลาส Frame จะประกอบด้วย title bar, resizable corner, icon และ menu bar ดังแสดงใน

รูปที่ 5.4 โปรแกรม GUI สำหรับโปรแกรมจาวาประยุกต์จะเริ่มต้นด้วยการสร้างอ็อบเจกต์ของคลาส `Frame` อย่างน้อยหนึ่งอ็อบเจกต์ ซึ่งคลาส `Frame` จะมีรูปแบบของ constructor ที่สำคัญดังนี้

- `public Frame()`
- `public Frame(String title)`

โดยที่ `title` คือข้อความที่ต้องการจะแสดงตรง title bar ของ `Frame`



รูปที่ 5.4 รูปแบบของ `Frame`

โดยทั่วไปคลาส `Frame` จะมีขนาดเริ่มต้นเป็น (0, 0) ดังนั้นจึงจำเป็นจะต้องกำหนดขนาดของอ็อบเจกต์ของ `Frame` โดยใช้เมธอด `setSize()` นอกจากนี้คลาส `Frame` จะถูกกำหนดให้ไม่สามารถมองเห็นได้ในตอนเริ่มต้น ดังนั้นจึงต้องมีการเรียกใช้เมธอด `setVisible(true)` เพื่อกำหนดให้อ็อบเจกต์ของคลาส `Frame` สามารถมองเห็นได้

อ็อบเจกต์ของคลาส `Frame` สามารถที่จะปรับขนาดได้โดยการลากเมาส์ปรับขนาดของเฟรมซึ่งคลาส `Frame` จะมีเมธอดที่เกี่ยวข้องกับการปรับขนาดของเฟรมดังนี้

- `public boolean isResizable()`

เป็นเมธอดเพื่อตรวจสอบว่าอ็อบเจกต์ของคลาส `Frame` นี้สามารถปรับขนาดได้หรือไม่

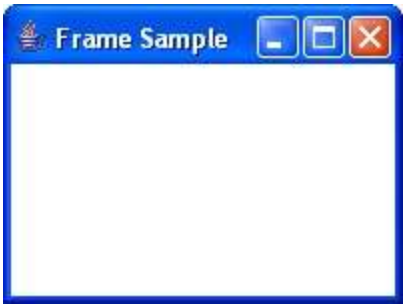
- `public void setResizable(boolean canResize)`

เป็นเมธอดเพื่อกำหนดให้อ็อบเจกต์ของคลาส `Frame` นี้ สามารถปรับขนาดได้หรือไม่

โปรแกรมที่ 5.1 แสดงตัวอย่างการสร้างอ็อบเจกต์ของคลาส `Frame` ที่ชื่อ `fr` และกำหนดให้ `fr` มีขนาดเริ่มต้นเป็น 200x150 พิกเซล โดยกำหนดไว้ในคำสั่ง `fr.setSize(200,150);` ส่วนคำสั่ง `fr.setVisible(true);` เป็นคำสั่งเพื่อที่จะทำให้สามารถมองเห็นอ็อบเจกต์ `fr` ได้ โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 5.5

โปรแกรมที่ 5.1 คลาส `FrameSample`

```
import java.awt.*;

public class FrameSample {
      private Frame fr;
      public void init() {
            fr = new Frame("Frame Sample");
            fr.setSize(200,150);
            fr.setVisible(true);
      }
      public static void main(String args[]) {
            FrameSample obj = new FrameSample();
            obj.init();
      }
}
```



รูปที่ 5.5 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.1

5.2.2 คลาส Panel

คลาส `Panel` จะเป็นคลาสประเภท `Container` เช่นเดียวกับคลาส `Frame` แต่ `Panel` จะแตกต่างจาก `Frame` ตรงที่ `Panel` จะไม่มี tile bar และไม่มี resizable corner คลาส `Panel` จะมี subclass ที่ชื่อ `Applet` เพื่อใช้ในการเขียนโปรแกรมจาวาแอปเพล็ต ซึ่งทำให้เราสามารถใส่ส่วนประกอบกราฟิกลงในโปรแกรมจาวาแอปเพล็ตได้ โปรแกรมจาวาประยุกต์ไม่สามารถที่จะสร้างโปรแกรม GUI โดยเริ่มต้นจากการสร้างอ็อบเจกต์ของคลาส `Panel` ได้โดยตรง ทั้งนี้เนื่องจากโปรแกรม GUI ต้องมีการสร้าง `Frame` ขึ้นมาก่อน

คลาส `Panel` สามารถที่จะใส่ลงไปใน `Frame` ได้จึงทำให้เราสามารถที่จะสร้างโปรแกรม GUI ที่มีการวางผังที่ซับซ้อนขึ้นได้ โดยการใส่อ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกต่างๆ ลงในอ็อบเจกต์ของคลาส `Panel` ก่อน แล้วจึงนำอ็อบเจกต์ของคลาส `Panel` ใส่ลงในอ็อบเจกต์ของคลาสประเภท `Container` อีกชั้นหนึ่ง ทั้งนี้เรา

สามารถที่จะกำหนดให้ อ็อบเจกต์ของคลาส Panel แต่ละตัวสามารถมีตัวจัดวางผังที่ต่างกันได้ และเรายังสามารถที่จะใส่ อ็อบเจกต์ของคลาส Panel ลงไปในอ็อบเจกต์ของคลาส Panel ตัวอื่นอีกทีหนึ่งก็ได้

5.2.3 คลาส Dialog

คลาส Dialog เป็นคลาสที่สืบทอดมาจากคลาส Window เพื่อใช้เป็นอินพุตและเอาต์พุตกับผู้ใช้ คลาส Dialog จะทำงานอยู่ภายใต้อ็อบเจกต์ของคลาส Frame ทั้งนี้การสร้างอ็อบเจกต์ของคลาส Dialog จะต้องระบุอ็อบเจกต์ของคลาส Frame ที่คู่กันซึ่งเรียกว่าเฟรมแม่ (parent frame) อ็อบเจกต์ของคลาส Dialog อาจมีโหมดการทำงานแบบ modal ซึ่งผู้ใช้ต้องปิดอ็อบเจกต์ของคลาส Dialog ก่อนจึงจะกลับมาทำงานในเฟรมแม่ที่คู่กันได้ คลาส Dialog มีรูปแบบของ constructor ที่สำคัญดังนี้

- `public Dialog(Frame parent)`
- `public Dialog(Frame parent,String title)`
- `public Dialog(Frame parent, boolean isModal)`
- `public Dialog(Frame parent, String title, boolean isModal)`

โดยที่

- parent คืออ็อบเจกต์ของคลาส Frame ที่เป็นเฟรมแม่
- title คือข้อความที่ต้องการจะแสดงตรง title bar ของ Dialog
- isModal คือข้อมูลชนิด boolean เพื่อระบุว่าอ็อบเจกต์ของ Dialog มีโหมดเป็น modal หรือไม่

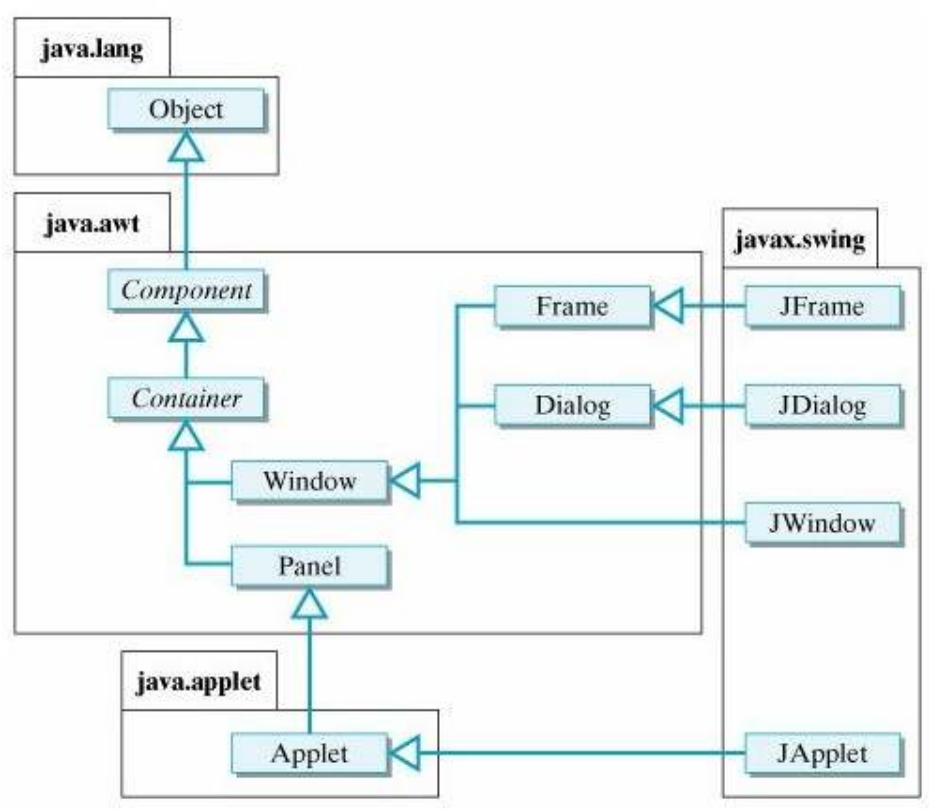
นอกจากนี้ Java API ได้กำหนดให้มีคลาสที่สืบทอดมาจากคลาส Dialog อื่นๆ อีกอาทิเช่น คลาส FileDialog ซึ่งจะใช้ในการระบุชื่อไฟล์ที่ต้องการใช้งานในโปรแกรม

5.2.4 คลาส JFrame

แพคเก็จ Swing มีคลาสประเภท Container ที่แตกต่างจากคลาสประเภท Container ของแพคเก็จ AWT คือ JFrame, JDialog, JPanel และ JApplet ดังรูปที่ 5.6 โดยทั่วไปโปรแกรม GUI ที่เป็นโปรแกรมจาวาประยุกต์จะใช้ Container ที่เป็นอ็อบเจกต์ของคลาส JFrame

คลาส JFrame จะสืบทอดมาจากคลาส Frame โดยมี constructor ที่สำคัญดังนี้

- `public JFrame()`
- `public JFrame(String title)`



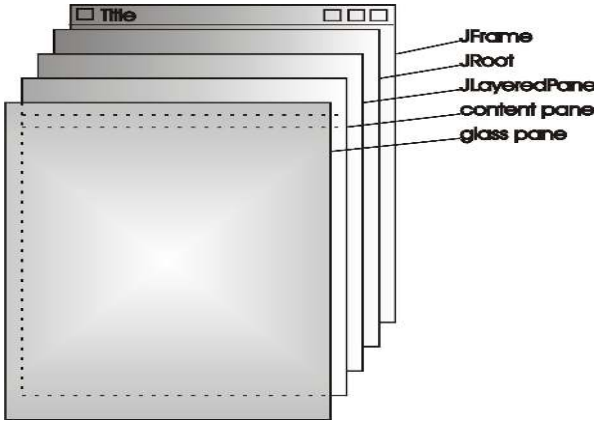
รูปที่ 5.6 คลาสชนิด Container ของแพคเก็จ Swing

อ็อบเจกต์ของคลาส `JFrame` จะแตกต่างกับ `Frame` ตรงที่จะมีหน้าต่าง (Pane) อยู่ 4 หน้าต่างดังแสดงในรูปที่ 5.7 คือ

- root pane
- layer pane
- glass pane
- content pane

เราไม่สามารถที่จะใส่ส่วนประกอบกราฟิกลงใน `JFrame` ได้โดยตรง แต่จะต้องใส่ลงในหน้าต่างที่เป็น content pane แทน ทั้งนี้หน้าต่างดังกล่าวเป็นอ็อบเจกต์ของคลาสประเภท `Container` แบบหนึ่ง เราสามารถที่จะเรียกอ็อบเจกต์ของคลาสประเภท `Container` ดังกล่าวมาได้โดยใช้เมธอดที่ชื่อ `getContentPane()` และสามารถที่จะใส่ส่วนประกอบกราฟิกลงในอ็อบเจกต์ดังกล่าวได้โดยใช้เมธอด `add()` ดังตัวอย่างเช่น

```
JFrame fr = new JFrame();
JButton bn1 = new JButton("Submit");
Container content = fr.getContentPane();
content.add(bn1);
```



รูปที่ 5.7 หน้าต่างที่อยู่ในคลาส `JFrame`

โปรแกรม Java SE 5 ได้กำหนดให้เราสามารถที่จะใส่ส่วนประกอบกราฟิกลงใน `JFrame` ได้โดยตรง โดยการกำหนดให้คำสั่ง `add()` และ `setLayout()` ใน `JFrame` จะเป็นการส่งคำสั่งไปยัง content pane โดยอัตโนมัติ ดังนั้นเราจึงสามารถที่จะเขียนคำสั่งข้างบนใหม่ได้เป็น

```
JFrame fr = new JFrame();
JButton bn1 = new JButton("Submit");
fr.add(bn1);
```

นอกจากนี้คลาส `JFrame` ยังมีเมธอดที่จะถูกเรียกใช้งาน เมื่อมีการปิด `Jframe` ดังนี้

- `public void setDefaultCloseOperation(int operation)`

โดยเราสามารถกำหนดค่าของ `operation` ได้ทั้งหมด 4 ค่าดังนี้

- `DO_NOTHING_ON_CLOSE` (ถูกกำหนดในอินเตอร์เฟส `WindowConstants`):
  จะไม่ทำอะไร โดยให้ไปเรียกใช้เมธอด `windowClosing` ของอ็อบเจกต์ `WindowListener` ที่มีการลงทะเบียนไว้แล้วแทน
- `HIDE_ON_CLOSE` (ถูกกำหนดในอินเตอร์เฟส `WindowConstants`):
  จะทำการซ่อน `JFrame` หลังจากเรียกใช้อ็อบเจกต์ของ WindowListener ที่มีการลงทะเบียนไว้แล้ว
- `DISPOSE_ON_CLOSE` (ถูกกำหนดในอินเตอร์เฟส `WindowConstants`):
  จะทำการซ่อนและทำลาย `JFrame` หลังจากเรียกใช้อ็อบเจกต์ของ `WindowListener` ที่มีการลงทะเบียนไว้แล้ว
- `EXIT_ON_CLOSE` (ถูกกำหนดในคลาส `JFrame`):
  จะมีการเรียกใช้เมธอด `exit` ของคลาส `System`

โดยค่าของ operation จะมีค่าเริ่มต้นเป็น HIDE_ON_CLOSE

โปรแกรมที่ 5.2 แสดงตัวอย่างการสร้างอ็อบเจกต์ของคลาส JFrame ที่ชื่อ fr โดยใช้คำสั่ง fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); เพื่อให้มีการเรียกใช้เมธอด exit ของคลาส System เมื่อมีการปิด Jframe และกำหนดให้ fr มีขนาดเริ่มต้นเป็น 200x150 พิกเซล โดยกำหนดไว้ในคำสั่ง fr.setSize(200,150); ส่วนคำสั่ง fr.setVisible(true); เป็นคำสั่งเพื่อที่จะทำให้สามารถมองเห็นอ็อบเจกต์ fr ได้ โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 5.8

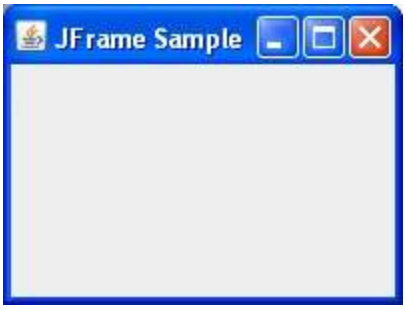
5.3 การจัดวางผังของส่วนประกอบกราฟิก

โปรแกรม GUI ของภาษาจาวาจะจัดวางส่วนประกอบกราฟิกต่างๆ ลงในอ็อบเจกต์ของคลาสประเภท Container โดยอัตโนมัติ ทั้งนี้จะกำหนดขนาดและตำแหน่งของส่วนประกอบกราฟิกได้โดยใช้อ็อบเจกต์ของคลาสประเภท LayoutManager ซึ่ง LayoutManager เป็นอินเตอร์เฟสที่ใช้ในการกำหนดวิธีการจัดวางผังส่วนประกอบกราฟิก ภาษาจาวาได้กำหนดให้มีคลาสที่ implements อินเตอร์เฟสที่ชื่อ LayoutManager เพื่อใช้เป็นตัวจัดวางผังของส่วนประกอบกราฟิกทั้งหมดของคลาสคือ

1. BorderLayout
2. FlowLayout
3. GridLayout
4. CardLayout
5. GridBagLayout

<u>โปรแกรมที่ 5.2</u> <u>คลาส JFrameSample</u>

```
import javax.swing.JFrame;

public class JFrameSample {

    private JFrame fr;

    public void init() {
        fr = new JFrame("JFrame Sample");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.setSize(200, 150);
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        JFrameSample obj = new JFrameSample();
        obj.init();
    }
}
```



รูปที่ 5.8 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.2

การกำหนดวิธีการวางผังของอ็อบเจกต์ของคลาสประเภท `Container` จะทำได้โดยการสร้างอ็อบเจกต์ประเภท `LayoutManager` ของคลาสใดคลาสหนึ่งในห้าคลาสข้างต้น จากนั้นเราสามารถที่จะกำหนดการวางผังโดยการเรียกใช้เมธอด `setLayout()` ของคลาสประเภท `Container` โดยส่งผ่าน argument ที่เป็นอ็อบเจกต์ประเภท `LayoutManager` ที่สร้างขึ้นมา ตัวอย่างเช่นคำสั่ง

```
JFrame fr = new JFrame("Demo");
FlowLayout fl = new FlowLayout();
fr.setLayout(fl);
```

เป็นการกำหนดให้อ็อบเจกต์ของคลาส `JFrame` ที่ชื่อ `fr` ซึ่งมีวิธีการจัดวางผังในรูปแบบของ `FlowLayout` เนื่องจากคำสั่งในตัวอย่างข้างต้นจะไม่มีการนำอ็อบเจกต์ `fl` ซึ่งเป็นอ็อบเจกต์ของคลาส `FlowLayout` ไปอ้างอิงในคำสั่งอื่นๆ ดังนั้นเราจึงสามารถที่จะเขียนคำสั่งในการสร้างอ็อบเจกต์และคำสั่งกำหนดรูปแบบการวางผังภายในคำสั่งเดียวกันได้ดังนี้

```
fr.setLayout(new FlowLayout());
```

โดยทั่วไปคลาสประเภท `Container` แต่ละชนิด จะมีการกำหนดวิธีการวางผังของอ็อบเจกต์ของคลาสแต่ละชนิดไว้เริ่มต้นอยู่แล้วดังนี้

- คลาส `JWindow, JFrame` และ `JDialog` จะกำหนดให้เป็น `BorderLayout`
- คลาส `JPanel` และ `JApplet` จะกำหนดให้เป็น `FlowLayout`

5.3.1 BorderLayout

`BorderLayout` เป็นการจัดวางผังที่กำหนดให้สามารถวางอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกได้ 5 อ็อบเจกต์ตามตำแหน่งทิศต่างๆ ของอ็อบเจกต์ประเภท `Container` 5 ตำแหน่งคือ ทิศเหนือ (NORTH) ทิศใต้

(SOUTH) ทิศตะวันออก (EAST) ทิศตะวันตก (WEST) และตรงกลาง (CENTER) ซึ่งอ็อบเจกต์ที่เป็นส่วนประกอบกราฟิกจะถูกใส่ในตำแหน่งทิศต่างๆ โดยจะมีการกำหนดขนาดให้โดยอัตโนมัติ

โปรแกรมที่ 5.3 แสดงตัวอย่างโปรแกรม GUI ที่ประกอบไปด้วยปุ่ม 5 ปุ่ม โดยได้สร้างอ็อบเจกต์ของคลาส `JButton` ขึ้นมา 5 อ็อบเจกต์ซึ่งมีชื่อเป็น `bn1` จนถึง `bn5` และมีข้อความบนปุ่มเป็น `B1` จนถึง `B5` ตามลำดับ จากนั้นได้ใส่อ็อบเจกต์ชนิด `JButton` แต่ละตัวลงไปในอ็อบเจกต์ชนิด `JFrame` ที่ชื่อ `fr` โดยใช้เมธอด `add()` ตามตำแหน่งทิศต่างๆ โดยที่ อ็อบเจกต์ `bn1` จะอยู่ทางทิศเหนือ อ็อบเจกต์ `bn2` จะอยู่ทางทิศใต้ อ็อบเจกต์ `bn3` จะอยู่ทางทิศตะวันออก อ็อบเจกต์ `bn4` จะอยู่ทางทิศตะวันตก และอ็อบเจกต์ `bn5` จะอยู่ตรงกลาง ซึ่งโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 5.9

การใช้เมธอด `add()` ในกรณีที่อ็อบเจกต์ประเภท `Container` มีการจัดวางผังแบบ `BorderLayout` นั้น จะต้องมี argument เพื่อระบุตำแหน่งทิศที่ต้องการวางอ็อบเจกต์ที่เป็นส่วนประกอบกราฟิกอาทิเช่น คำสั่ง

```
fr.add(bn1,BorberLayout.NORTH);
```

เป็นคำสั่งที่ใช้ในการใส่อ็อบเจกต์ `bn1` ลงใน `Jframe` ในตำแหน่งทิศเหนือ

กรณีที่เรียกใช้เมธอด `add()` โดยไม่ส่งผ่าน argument เพื่อระบุตำแหน่งทิศที่ต้องการวางอ็อบเจกต์ที่เป็นส่วนประกอบกราฟิกนั้น ภาษาจาวาจะถือว่าเป็นการกำหนดให้ใส่ส่วนประกอบกราฟิกไว้ตำแหน่งตรงกลางของอ็อบเจกต์ของคลาสประเภท `Container` อาทิเช่น คำสั่ง

```
fr.add(bn1);
```

จะถือว่าเป็นคำสั่งเช่นเดียวกับคำสั่ง

```
fr.add(bn1,BorderLayout.CENTER);
```

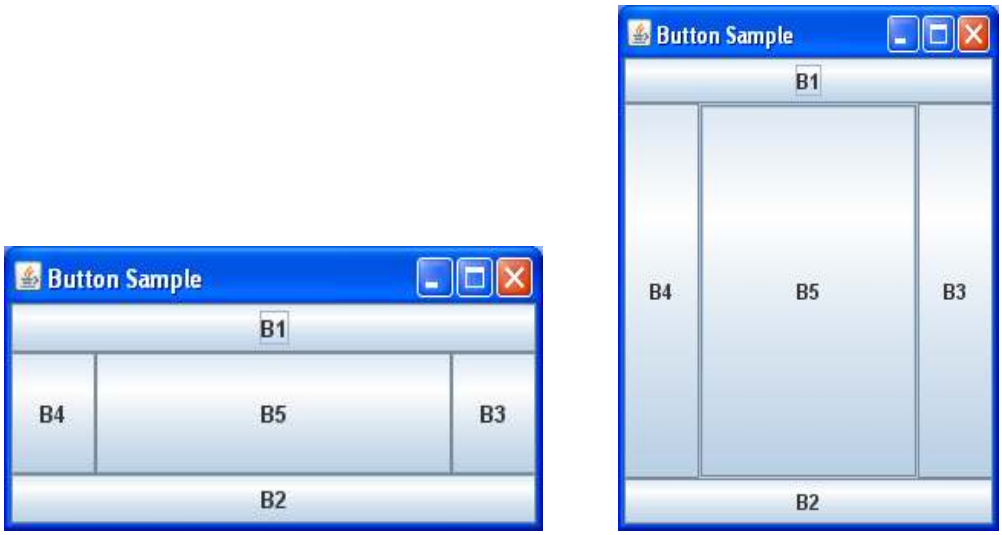
โปรแกรมที่ 5.3 ตัวอย่างการใช้ตัวจัดวางผังแบบ `BorderLayout`

```
import java.awt.BorderLayout;
import java.awt.Container;
import javax.swing.JButton;
import javax.swing.JFrame;

public class BorderLayoutSample {

    private JFrame fr;
    private JButton bn1, bn2, bn3, bn4, bn5;

    public void init() {
        fr = new JFrame("Button Sample");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        bn1 = new JButton("B1");
        bn2 = new JButton("B2");
        bn3 = new JButton("B3");
        bn4 = new JButton("B4");
        bn5 = new JButton("B5");
        fr.add(bn1, BorderLayout.NORTH);
        fr.add(bn2, BorderLayout.SOUTH);
        fr.add(bn3, BorderLayout.EAST);
        fr.add(bn4, BorderLayout.WEST);
        fr.add(bn5);
        fr.setSize(200, 150);
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        BorderLayoutSample obj = new BorderLayoutSample();
        obj.init();
    }
}
```

รูปที่ 5.9 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.3

การปรับขนาดของอ็อบเจกต์ของคลาสประเภท `Container` จะมีผลทำให้ขนาดของอ็อบเจกต์ของส่วนประกอบกราฟิก เปลี่ยนแปลงไปตามขนาดของอ็อบเจกต์ของคลาสประเภท `Container` ดังนี้

- อ็อบเจกต์ของส่วนประกอบกราฟิกที่อยู่ตำแหน่งทิศเหนือ ใต้ และตรงกลาง จะปรับขนาดตามขนาดของอ็อบเจกต์ของคลาสประเภท `Container` ตามแนวนอน ดังแสดงในรูปที่ 5.10 (ก)
- อ็อบเจกต์ของส่วนประกอบกราฟิกที่อยู่ตำแหน่งทิศตะวันออก ตะวันตก และตรงกลางจะปรับขนาดตามขนาดของอ็อบเจกต์ของคลาสประเภท `Container` ตามแนวตั้ง ดังแสดงในรูปที่ 5.10 (ข)



(ก) แนวนอน *(ข) แนวตั้ง*

รูปที่ 5.10 ขนาดของอ็อบเจกต์ที่ถูกปรับ

5.3.2 FlowLayout

`FlowLayout` เป็นการจัดวางผังส่วนประกอบกราฟิกไว้ตำแหน่งบนสุดของอ็อบเจกต์ของคลาสประเภท `Container` โดยจะเรียงอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกจากซ้ายไปขวา และถ้าความกว้างของอ็อบเจกต์ของคลาสประเภท `Container` ในแต่ละแถวไม่พอ ตัวจัดวางผังแบบ `FlowLayout` จะนำอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบที่เหลือวางในตำแหน่งถัดไปด้านล่าง การจัดวางผังแบบ `FlowLayout` จะปรับขนาดของอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกต่างๆ ตามความเหมาะสม ทั้งนี้ขึ้นอยู่กับข้อกำหนดต่างๆ อาทิเช่น ขนาดของอ็อบเจกต์ของคลาส `JButton` จะมีขนาดตามขนาดของข้อความบนปุ่ม เป็นต้น

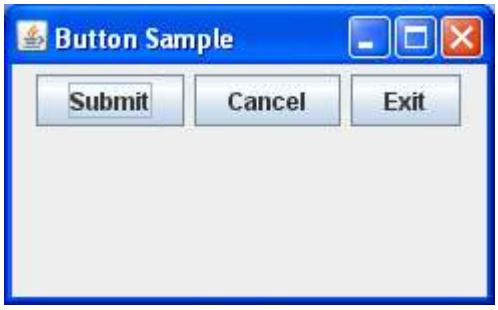
โปรแกรมที่ 5.4 แสดงตัวอย่างโปรแกรม GUI ที่สร้างอ็อบเจกต์ของคลาส `JFrame` ขึ้นมา แล้วกำหนดให้มีการจัดวางผังแบบ `FlowLayout` โดยใช้คำสั่ง

```
fr.setLayout(new FlowLayout());
```

โปรแกรมที่ 5.4 ตัวอย่างการใช้ตัวจัดวางผังแบบ `FlowLayout`

```
import java.awt.*;
import javax.swing.*;

public class FlowLayoutSample {

    private JFrame fr;
    private JButton bn1, bn2, bn3;

    public void init() {
        fr = new JFrame("Button Sample");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.setLayout(new FlowLayout());
        bn1 = new JButton("Submit");
        bn2 = new JButton("Cancel");
        bn3 = new JButton("Exit");

        fr.add(bn1);
        fr.add(bn2);
        fr.add(bn3);
        fr.setSize(200, 150);
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        FlowLayoutSample obj = new FlowLayoutSample();
        obj.init();
    }
}
```

โปรแกรม GUI นี้จะมีอ็อบเจกต์ของคลาส `JButton` อยู่สามอ็อบเจกต์ โดยกำหนดให้มีข้อความเป็น Submit, Cancel และ Exit จากนั้นโปรแกรมได้ใส่อ็อบเจกต์ของคลาส `JButton` แต่ละตัวลงในอ็อบเจกต์ของคลาส `JFrame` ซึ่งจะได้โปรแกรม GUI ที่มีปุ่มสามปุ่มเรียงจากซ้ายไปขวาดังรูปที่ 5.11 (ก) และในกรณีที่ `JFrame` ขนาดแคบลงปุ่มบางปุ่มก็จะลงมาอยู่ในตำแหน่งถัดไปดังรูปที่ 5.11 (ข)



(ก)

(ข)

รูปที่ 5.11 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่5.4

5.3.3 GridLayout

`GridLayout` เป็นการจัดวางผังที่แบ่งอ็อบเจกต์ของคลาสประเภท `Container` เป็นช่องย่อยๆ หลายช่อง โดยแต่ละช่องย่อยจะมีขนาดความกว้างและความสูงเท่ากัน `GridLayout` จะอนุญาตให้ใส่อ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกได้หนึ่งอ็อบเจกต์ ในแต่ละช่องย่อย โดยจะกำหนดให้ขนาดของอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกมีขนาดเท่ากับขนาดของช่องย่อย การสร้างตัวจัดวางผังแบบ `GridLayout` ทำได้โดยการสร้างอ็อบเจกต์ของคลาส `GridLayout` ที่มีรูปแบบของ constructor เป็นดังนี้

```
public GridLayout(int row, int col)
```

โดยที่ `row` และ `col` คือจำนวนแถวและคอลัมน์ที่ต้องการแบ่งอ็อบเจกต์ของคลาสประเภท `Container` ให้เป็นช่องย่อย

การใส่อ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกลงในช่องย่อย จะทำได้โดยการเรียกใช้เมธอด `add()` เช่นเดียวกัน แต่ในกรณีนี้โปรแกรม GUI จะใส่อ็อบเจกต์จากช่องย่อยบนสุดเรียงจากซ้ายไปขวา โดยไม่สามารถที่จะข้ามช่องย่อยใดช่องย่อยหนึ่งได้

<u>โปรแกรมที่ 5.5</u> <u>ตัวอย่างการใช้ตัวจัดวางผังแบบ `GridLayout`</u>

```
import java.awt.*;
import javax.swing.*;

public class GridLayoutSample {
    private JFrame fr;
    private JButton bn1, bn2, bn3, bn4, bn5;
    public void init() {
        fr = new JFrame("Button Sample");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        bn1 = new JButton("B1");
        bn2 = new JButton("B2");
        bn3 = new JButton("B3");
        bn4 = new JButton("B4");
        bn5 = new JButton("B5");
        fr.setLayout(new GridLayout(3, 2));
        fr.add(bn1, BorderLayout.NORTH);
        fr.add(bn2, BorderLayout.SOUTH);
        fr.add(bn3, BorderLayout.EAST);
        fr.add(bn4, BorderLayout.WEST);
        fr.add(bn5, BorderLayout.CENTER);
        fr.setSize(200, 150);
        fr.setVisible(true);
    }
    public static void main(String args[]) {
        GridLayoutSample obj = new GridLayoutSample();
        obj.init();
    }
}
```

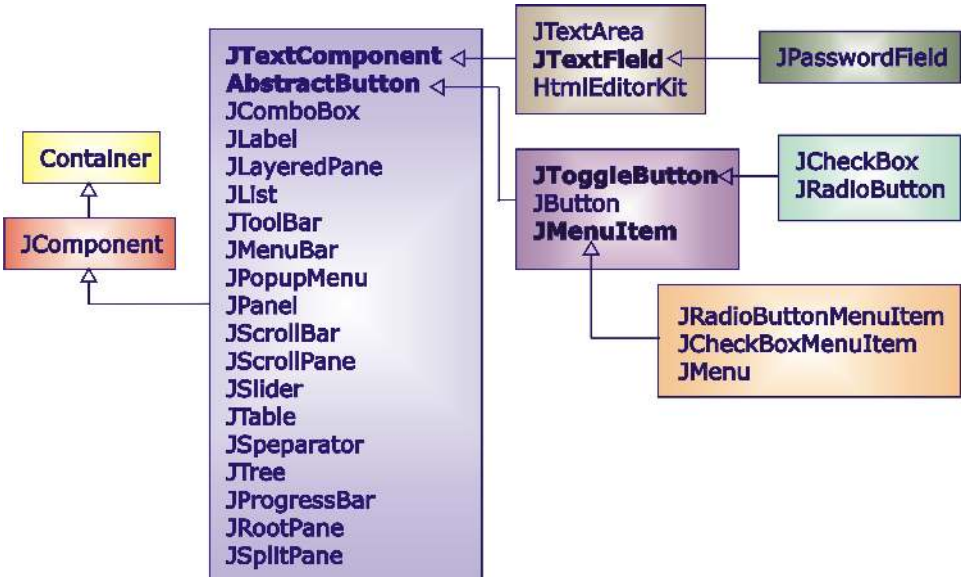
รูปที่ 5.12 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.5

5.4 ส่วนประกอบกราฟิกในแพคเก็จ Swing

แพคเก็จ AWT มีคลาสที่เป็นส่วนประกอบกราฟิกที่สำคัญดังนี้

- `Button` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็นปุ่ม
- `Label` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่มีไว้ในการแสดงข้อความ
- `TextField` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์เพื่อให้ผู้ใช้ป้อนข้อความได้สูงสุดหนึ่งบรรทัด
- `TextArea` เป็นที่ทำหน้าที่คล้ายกับ `TextField` แต่จะมีหลายบรรทัด
- `Checkbox` เป็นคลาสที่ทำหน้าที่คล้ายกับปุ่มโดยใช้สวิทซ์ “on-off”
- `Choice` เป็นคลาสที่ผู้ใช้สามารถเลือกรายการจากที่กำหนดมาให้ได้
- `List` เป็นคลาสที่จะทำหน้าที่คล้าย `Choice` แต่จะแสดงรายการได้หลายแถวพร้อมๆ กัน
- `Scollbar` เป็นคลาสที่ทำหน้าที่เป็นแถบควบคุมเพื่อให้ผู้ใช้สามารถเลื่อนไปยังตำแหน่งที่ต้องการได้
- `Canvas` เป็นคลาสที่เป็นพื้นที่ว่างเปล่าเพื่อใช้ในการวาดรูปกราฟิกลงไป

สำหรับคลาสที่เป็นส่วนประกอบกราฟิกของแพคเก็จ Swing ทุกคลาสจะสืบทอดมาจากคลาส `JComponent` ดังแสดงในรูปที่ 5.13 โดยคลาส `JComponent` จะสืบทอดมาจากคลาส `Container` ในแพคเก็จ AWT อีกต่อหนึ่ง

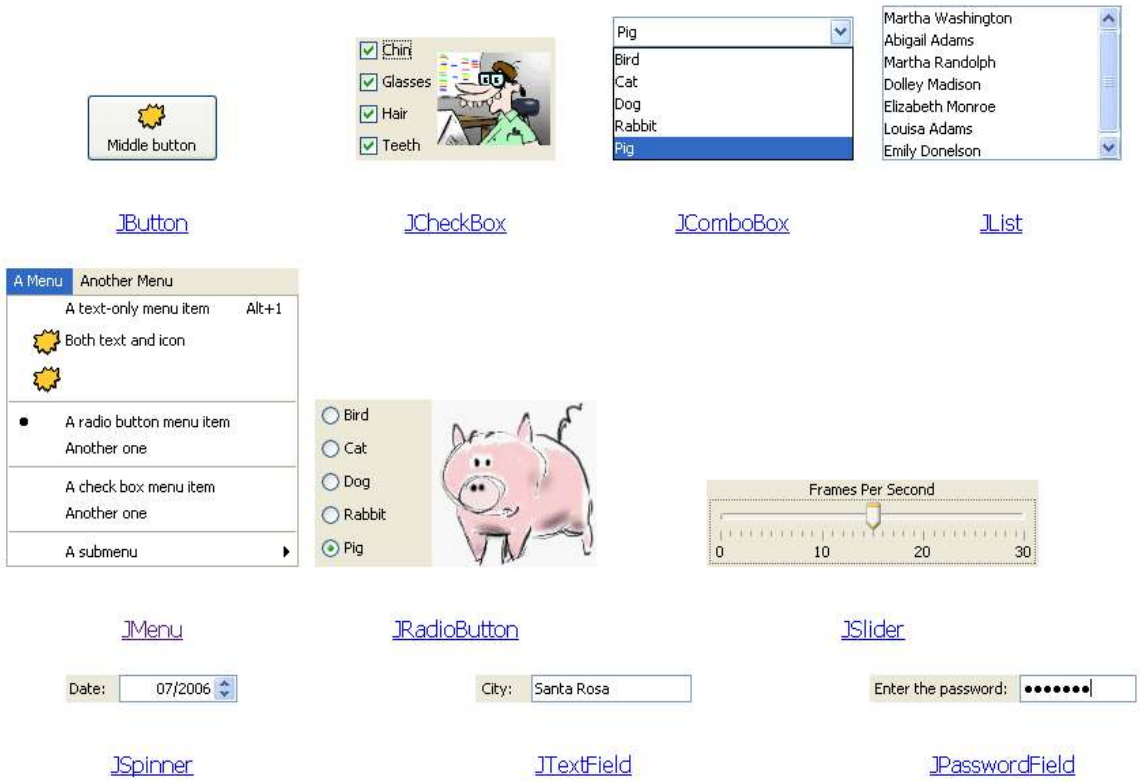


รูปที่ 5.13 คลาสต่างๆ ที่สืบทอดมาจากคลาส `Jcomponent`

แพคเก็จ Swing จะมีคลาสที่เป็นส่วนประกอบกราฟิกที่สอดคล้องกับคลาสในแพคเก็จ AWT โดยคลาสเหล่านี้จะมีชื่อขึ้นต้นด้วยตัวอักษร 'J' โดยมีคลาสที่สำคัญดังนี้

- `JButton` เป็นคลาสที่ทำหน้าที่เป็นปุ่มในแพคเก็จ Swing
- `JLabel` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่ไว้ในการแสดงข้อความในแพคเก็จ Swing
- `JTextField` เป็นคลาสที่ใช้ในการป้อนข้อความหนึ่งบรรทัดในแพคเก็จ Swing
- `JTextArea` เป็นคลาสที่ใช้ในการป้อนข้อความหลายบรรทัดในแพคเก็จ Swing
- `JScrollBar` เป็นคลาสที่ทำหน้าที่เป็นแถบควบคุมเพื่อให้ผู้ใช้เลื่อนไปยังตำแหน่งที่ต้องการได้ในแพคเก็จ Swing
- `JCheckBox` เป็นคลาสที่ทำหน้าที่คล้ายปุ่มในแพคเก็จ Swing
- `JChoice` เป็นคลาสที่ผู้ใช้สามารถเลือกรายการได้ในแพคเก็จ Swing

การเขียนโปรแกรม GUI โดยมากจะใช้ส่วนประกอบกราฟฟิคของแพคเก็จ Swing ที่มี Look and Feel ที่ดีกว่าของแพคเก็จ AWT รูปที่ 5.14 แสดงตัวอย่างส่วนประกอบกราฟฟิคของแพคเก็จ Swing บางคลาส ซึ่งเนื้อหาในหัวข้อย่อยต่อไปจะอธิบายการใช้งานของคลาสที่สำคัญบางคลาส



รูปที่ 5.14 ตัวอย่าง Look and Feel คลาสที่เป็นส่วนประกอบกราฟิกในแพคเกจ Swing

5.4.1 คลาส `JButton`

`JButton` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่แสดงเป็นปุ่ม โดยจะมีข้อความ (text) ปรากฏอยู่บนปุ่ม ผู้ใช้สามารถใช้อุปกรณ์อินพุตเช่นเมาส์หรือคีย์บอร์ดกดเลือกปุ่มได้ `JButton` เป็นคลาสที่สืบทอดมาจากคลาส `JComponent` เราสามารถที่จะสร้างอ็อบเจกต์ของคลาส `JButton` โดยเรียกใช้ constructor ของคลาส `JButton` ที่มีรูปแบบดังนี้

- `public JButton()`
- `public JButton(String text)`
- `public Jbutton(Icon icon)`
- `public JButton(String text, Icon icon)`

โดยที่

- `text` คือข้อความที่จะปรากฏอยู่บนปุ่ม
- `Icon` คือไอคอนที่ต้องการแสดง

คลาส `Button` ยังมีเมธอดอื่นๆ ที่สำคัญในการจัดการกับข้อความดังนี้

- `public void setText(String text)` เป็นเมธอดที่ใช้ในการกำหนดหรือเปลี่ยนข้อความของปุ่ม
- `public String getText()` เป็นเมธอดที่ใช้ในการเรียกดูข้อความของปุ่ม
- `public void setMnemonic(char c)` หรือ `void setMnemonic(int i)`
  เป็นเมธอดในการกำหนดคีย์ที่เป็น shortcut ให้กับอ็อบเจกต์ของคลาส `JButton`
- `public void setIcon(Icon c)`
  เป็นเมธอดในการใส่ไอคอน (icon) ลงในอ็อบเจกต์ของคลาส `JButton`

นอกจากนี้เรายังสามารถเรียกใช้เมธอดของคลาส `JComponent` ซึ่งเป็น superclass ของ JButton ที่มีเมธอดที่เพิ่มลักษณะต่างๆ ให้กับอ็อบเจกต์ที่เป็นส่วนประกอบกราฟิก อาทิเช่น

- `public void setBorder(Border bd)` เป็นเมธอดสำหรับกำหนดขอบให้กับส่วนประกอบกราฟิก
- `public void setToolTipText(String text)` เป็นเมธอดสำหรับกำหนดข้อความที่จะแสดงเป็น tooltip ให้กับส่วนประกอบกราฟิก

ตัวอย่างการสร้างอ็อบเจกต์ของคลาส `JButton` ที่แสดงข้อความ คีย์ที่เป็น shortcut และ tooltip สามารถกำหนดได้ตามคำสั่งดังนี้

```
JButton b1 = new JButton("Demo button");
b1.setMnemonic(KeyEvent.VK_D);
b1.setToolTipText("Click this button ");
```

5.4.2 คลาส `JLabel`

`JLabel` เป็นคลาสที่ใช้สร้างอ็อบเจกต์ที่เป็นส่วนประกอบกราฟิกที่ใช้ในการแสดงข้อความ โดยที่คลาส `JLabel` มี constructor ที่สำคัญดังนี้

- `public JLabel(String text)`
- `public JLabel(String text, int align)`
- `public Jlabel(Icon icon)`
- `public Jlabel(Icon icon, int align)`
- `public JLabel(String text, Icon icon)`

โดยที่

- `text` คือข้อความที่ต้องการแสดง
- `align` คือการกำหนดการวางแนว(ชิดซ้าย ขวา หรือตรงกลาง) ของข้อความ

- `Icon` คือไอคอนที่ต้องการแสดง

คลาส `JLabel` ยังมีเมธอดอื่นๆในการจัดการและกำหนด Look and Feel ของอ็อบเจกต์เช่นเดียวกับ JButton ซึ่งคำสั่งเหล่านี้สามารถหาได้จาก Java API หรือเรียกดูจากคำสั่งในโปรแกรม IDE เช่น NetBeans

โปรแกรมที่ 5.6 แสดงตัวอย่างการสร้างอ็อบเจกต์ชนิด JLabel ขึ้นมาสามอ็อบเจกต์โดยแต่ละอ็อบเจกต์จะใช้คำสั่ง constructor ที่แตกต่างกัน และจะมีไอคอนประกอบข้อความในอ็อบเจกต์ชื่อ label1 และ label3 โดยไอคอนจะสร้างมาจากไฟล์ที่ชื่อ testImg.jpg ที่เก็บอยู่ในไดเร็กทอรี่ที่ชื่อ images ซึ่งอยู่ภายในไดเร็กทอรี่เดียวกับไฟล์ที่ชื่อ JLabelDemo.class โปรแกรมนี้จะทำการโหลดไฟล์ดังกล่าวมาใส่ในอ็อบเจกต์ที่ชื่อ icon โดยมีคำสั่งดังนี้

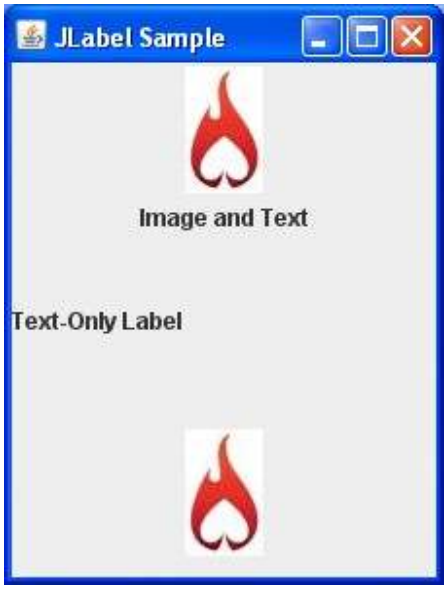
```
URL imageURL = JLabelDemo.class.getResource("images/testImg.jpg");

if (imageURL != null) {
    icon = new ImageIcon(imageURL);
}
```

โปรแกรมนี้จะให้ผลลัพธ์ดังรูปที่ 5.15

<u>โปรแกรมที่ 5.6</u> <u>ตัวอย่างการสร้างอ็อบเจกต์ชนิด JLabel</u>

```
import java.awt.*;
import java.net.URL;
import javax.swing.*;

public class JLabelDemo {

    private JFrame fr;
    private JLabel label1, label2, label3;

    public void init() {
        ImageIcon icon = null;
        fr = new JFrame("JLabel Sample");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        URL imageURL = JLabelDemo.class.getResource("images/testImg.jpg");

        if (imageURL != null) {
            icon = new ImageIcon(imageURL);
        }

        fr.setLayout(new GridLayout(3, 1));
        //Create the first label.
        label1 = new JLabel("Image and Text", icon, JLabel.CENTER);
        //Set the position of its text, relative to its icon:
        label1.setVerticalTextPosition(JLabel.BOTTOM);
        label1.setHorizontalTextPosition(JLabel.CENTER);
```

```
        //Create the other labels.
        label2 = new JLabel("Text-Only Label");
        label3 = new JLabel(icon);

        //Create tool tips, for the heck of it.
        label1.setToolTipText("A label containing both image and text");
        label2.setToolTipText("A label containing only text");
        label3.setToolTipText("A label containing only an image");

        //Add the labels.
        fr.add(label1);
        fr.add(label2);
        fr.add(label3)

        fr.pack();
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        JLabelDemo obj = new JLabelDemo();
        obj.init();
    }
}
```



รูปที่ 5.15 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.6

5.4.3 คลาส JTextField

JTextField คือคลาสที่ใช้ในการสร้างอ็อบเจกต์เพื่อให้ผู้ใช้ป้อนข้อความหนึ่งบรรทัด โดยที่คลาส JTextField มี constructor ที่สำคัญดังนี้

- public JTextField()
- public JTextField(int col)
- public JTextField(String text)
- public JTextField(String text, int col)

โดยที่

- text คือข้อความเริ่มต้นที่ต้องการแสดง
- col คือจำนวนคอลัมน์ที่ต้องการแสดง

คลาส JTextField มีเมธอดที่สำคัญดังนี้

- public int getColumns()
- public String getText()
- public boolean isEditable()
- public void select(int selectionStart, int selectionEnd)
- public void selectAll()
- public void setEditable(boolean b)
- public void setText(String t)
- public void setColumn(int column)

เมธอด setText() ใช้ในการกำหนดหรือเปลี่ยนข้อความของอ็อบเจกต์ชนิด JTextFiled ในกรณีที่เราต้องการกำหนดให้ JTextField สามารถอ่านข้อความได้อย่างเดียว เราสามารถทำได้โดยเรียกใช้เมธอด setEditable(boolean b) โดยกำหนดให้ argument มีค่าเป็น false ส่วนเมธอด select() ใช้ในการเลือกข้อความใน JTextField

โปรแกรมที่ 5.7 แสดงตัวอย่างการสร้างเฟรมที่มีอ็อบเจกต์ของคลาส Label และ TextField แสดงอยู่ โปแกรมนี้จะกำหนดให้อ็อบเจกต์ fr ของคลาส JFrame มีการจัดวางผังแบบ FlowLayout ผลลัพธ์ของโปรแกรมนี้เป็นดังแสดงในรูปที่ 5.12

โปรแกรมที่ 5.7 ตัวอย่างการสร้างอ็อบเจกต์ `JTextField`

```
import java.awt.*;
import javax.swing.*;

public class JTextFieldDemo {

    private JFrame fr;
    private JLabel l;
    private JTextField tf;

    public void init() {
        fr = new JFrame("JTextField Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        l = new JLabel("Name: ");
        tf = new JTextField("Numnonda", 15);
        fr.setLayout(new FlowLayout());
        fr.add(l);
        fr.add(tf);
        fr.pack();
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        JTextFieldDemo obj = new JTextFieldDemo();
        obj.init();
    }
}
```

รูปที่ 5.16 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.7

5.4.4 คลาส `JTextArea`

`JTextArea` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่สามารถป้อนและแก้ไขข้อความได้ `JTextArea` จะแตกต่างจาก `JTextField` ตรงที่จะสามารถกำหนดจำนวนบรรทัดได้หลายบรรทัด โดยที่คลาส `JTextArea` มี constructor ที่สำคัญดังนี้

- `public JTextArea()`
- `public JTextArea(String Text)`
- `public JTextArea(String Text, int row, int col)`
- `public JTextArea(int row, int col)`

โดยที่

- `text` คือข้อความเริ่มต้นที่ต้องการแสดง
- `row` และ `col` คือจำนวนแถวและคอลัมน์ของ `TextArea`
- `scrollbar` เป็นการกำหนดการมี scrollbar ของ `TextArea`

คลาส `JTextArea` จะมีเมธอดที่สำคัญดังนี้

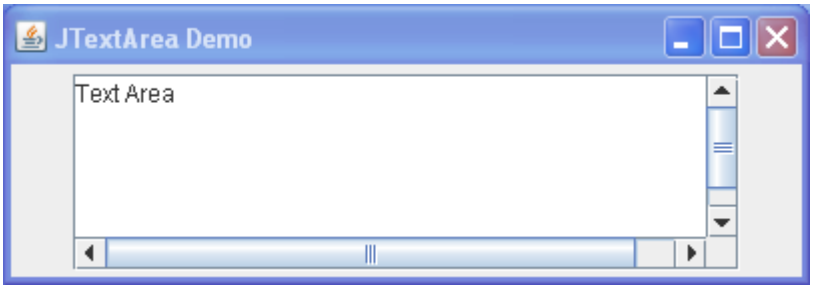
- `public int getColumns()`
- `public int getRows()`
- `public String getSelectedText()`
- `public boolean isEditable()`
- `public void select(int selectionStart, int selectionEnd)`
- `public void selectAll()`
- `public void setEditable(boolean b)`
- `public void setText(String t)`
- `public void setColumns(int column)`
- `public void setRows(int rows)`

ทั้งนี้เมธอดที่สำคัญที่อยู่ในคลาส `JTextArea` จะคล้ายกับเมธอดของคลาส `JTextArea` แต่จะเพิ่มเมธอดที่เกี่ยวข้องกับการจัดการจำนวนแถวขึ้นมา อาทิเช่น `setRow()` และ `getRow()`

โปรแกรมที่ 5.8 แสดงตัวอย่างการสร้าง `JFrame` ที่มีอ็อบเจกต์ของคลาส `JTextArea` โดยโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 5.8

<u>โปรแกรมที่ 5.8</u> <u>ตัวอย่างการสร้างอ็อบเจกต์ `JTextArea`</u>

```
import java.awt.*;
import javax.swing.*;

public class JTextAreaDemo {
    private JFrame fr;
    private JTextArea ta;

    public void init() {
        fr = new JFrame("JTextArea Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ta = new JTextArea("Text Area", 5, 30);
        JscrollPane jScrollPane = new JScrollPane(ta);
        fr.setLayout(new FlowLayout());
        fr.add(jScrollPane);
        fr.pack();
        fr.setVisible(true);
    }
    public static void main(String args[]) {
        JTextAreaDemo obj = new JTextAreaDemo();
        obj.init();
    }
}
```



รูปที่ 5.17 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.8

5.4.5 คลาส JcheckBox และ JRadioButton

JCheckBox เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่ทำหน้าที่คล้ายปุ่ม โดยให้ผู้ใช้กดช่องที่เป็นสวิทซ์เพื่อเลือกหรือไม่เลือกรายการ JCheckBox จะมีข้อความอยู่ข้างๆ เพื่ออธิบายความหมายของรายการ JCheckBox มี constructor ที่สำคัญดังนี้

- public JCheckBox(String label)
- public JCheckBox(String label, boolean state)
- public JCheckBox(Icon icon)
- public JCheckBox(Icon icon, boolean state)
- public JCheckBox(String label, Icon icon)
- public JCheckBox(String label, Icon icon, boolean state)

โดยที่

- label คือข้อความที่ต้องการแสดงใน JCheckBox
- Icon คือไอคอนที่ต้องการแสดง โดยที่
- state เป็นตัวกำหนดสถานะเริ่มต้นของ JCheckBox กรณีที่ constructor ไม่ได้กำหนดสถานะเริ่มต้นจะถือว่ามีค่าเป็น false (ไม่ถูกเลือก)

คลาส JCheckBox มีเมธอดที่สำคัญดังนี้

- public String getLabel()
- public Object[] getSelectedObjects()
- public void setLabel(String label)
- public void setSelected(boolean state)

โปรแกรมที่ 5.9 แสดงตัวอย่างการใช้ JCheckBox โดยโปรแกรมจะสร้างเฟรมที่มีอ็อบเจกต์ชนิด JCheckBox เพื่อให้ผู้ใช้เลือกสามอ็อบเจกต์ดังแสดงในในรูปที่ 5.18

เราสามารถที่จะสร้างตัวเลือกที่เป็นแบบ Radio Button โดยใช้ `JRadioButton` ในกรณีนี้จะกำหนดให้อ็อบเจกต์ `JRadioButton` หลายตัวอยู่ในกลุ่มเดียวกัน โดยอยู่ในกลุ่มของอ็อบเจกต์ของคลาส `ButtonGroup` ซึ่งจะต้องใช้เมธอด `add()`

คลาส `ButtonGroup` ไม่ใช่คลาสที่เป็นส่วนประกอบกราฟิก แต่จะใช้ในการสร้างอ็อบเจกต์เพื่อกำหนดกลุ่มของ `AbstractButton` โดยมี constructor ดังนี้

- `public ButtonGroup()`

โปรแกรมที่ 5.10 แสดงตัวอย่างของการสร้าง `JRadioButton` โดยปรับเปลี่ยน `JRadioButton` ในโปรแกรมที่ 5.9 ให้อยู่ในกลุ่มของอ็อบเจกต์ที่ชื่อ `chg` โปรแกรมนี้จะมีผลลัพธ์ที่เป็นส่วนติดต่อกับผู้ใช้ดังรูปที่ 5.19

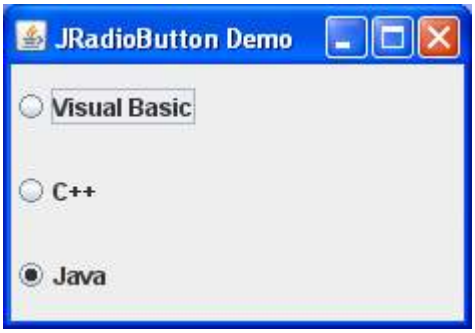
<u>โปรแกรมที่ 5.9</u> <u>ตัวอย่างการสร้างอ็อบเจกต์ `JCheckBox`</u>

```
import java.awt.FlowLayout;
import javax.swing.*;

public class JCheckBoxDemo {

    private JFrame fr;
    private JCheckBox c1, c2, c3;

    public void init() {
        fr = new JFrame("JCheckBox Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c1 = new JCheckBox("Visual Basic");
        c2 = new JCheckBox("C++", false);
        c3 = new JCheckBox("Java", true);
        fr.setLayout(new FlowLayout());
        fr.add(c1);
        fr.add(c2);
        fr.add(c3);
        fr.pack();
        fr.setVisible(true);
    }

     public static void main(String args[]) {
        JCheckBoxDemo obj = new JCheckBoxDemo();
        obj.init();
    }
}
```

รูปที่ 5.18 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.9

โปรแกรมที่ 5.10 ตัวอย่างของการสร้างอ็อบเจกต์ `JRadioButton`

```
import java.awt.GridLayout;
import javax.swing.*;

public class JRadioButtonDemo {

    private JFrame fr;
    private JRadioButton c1, c2, c3;
    private ButtonGroup chg;

    public void init() {
        fr = new JFrame("JRadioButton Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        c1 = new JRadioButton("Visual Basic");
        c2 = new JRadioButton("C++", false);
        c3 = new JRadioButton("Java", true);
        chg = new ButtonGroup();
        chg.add(c1);
        chg.add(c2);
        chg.add(c3);
        fr.setLayout(new GridLayout(3,1));
        fr.add(c1);
        fr.add(c2);
        fr.add(c3);

        fr.pack();
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        JRadioButtonDemo obj = new JRadioButtonDemo();
        obj.init();
    }
}
```



รูปที่ 5.19 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.10

5.4.6 คลาส JComboBox

JComboBox เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็นรายการให้ผู้ใช้สามารถเลือกได้โดย JComboBox จะแสดงรายการปรากฏให้เห็นเฉพาะรายการที่เลือกเพียงรายการเดียว ปกติก่อนที่จะมีการเลือกรายการ JComboBox จะแสดงรายการแรกที่มีอยู่ และหากมีการคลิกเมาส์ อ็อบเจกต์ JComboBox จะแสดงรายการทั้งหมดที่มีอยู่ โดยที่คลาส JComboBox มี constructor ที่สำคัญดังนี้

- public JComboBox()
- public JcomboBox([]Object objs)

คลาส JComboBox มีเมธอดต่างๆ ที่สำคัญดังนี้

- public void addItem(Object item)
- public void insertItemAt(Object item, int pos)
- public Object getItem(int index)
- public int getItemCount()
- public Object getSelectedIndex()
- public Object getSelectedItem()
- public void setSelectedIndex(int pos)
- public void setSelectedItem(Object item)

เราสามารถที่จะใส่รายการลงในอ็อบเจกต์ JComboBox ได้โดยใช้เมธอด addItem(Object item) คลาส JComboBox ยังมีเมธอด setSelectedIndex(int pos) และ setSelectedItem(Object item) เพื่อใช้ในการเลือกให้อ็อบเจกต์ JComboBox แสดงรายการที่ตำแหน่งหรือข้อความที่ต้องการให้แสดงได้ ส่วนเมธอด getSelectedIndex() และ getSelectedItem() ใช้ในการแสดงตำแหน่งหรือข้อความที่ถูกเลือก

โปรแกรมที่ 5.11 แสดงตัวอย่างเฟรมที่มีอ็อบเจกต์ชนิด JComboBox แสดงอยู่ คำสั่ง cb.addItem() ใช้ในการใส่รายการต่างๆ ลงในอ็อบเจกต์ cb ส่วนคำสั่ง cb.setSelectedItem ("Thailand") เป็นการกำหนดให้อ็อบเจกต์ cb เลือกแสดงรายการที่ชื่อ Thailand โปรแกรมนี้จะให้ผลลัพธ์ที่เป็นส่วนติดต่อกับผู้ใช้ดังรูปแสดงในรูปที่ 5.20

โปรแกรมที่ 5.11 ตัวอย่างการสร้างอ็อบเจกต์ JComboBox

```
import javax.swing.JComboBox;
import javax.swing.JFrame;

public class JComboBoxDemo {

    private JFrame fr;
    private JComboBox cb;

    public void init() {
        fr = new JFrame("JRadioButton Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        cb = new JComboBox();
        cb.addItem("New Zealand");
        cb.addItem("Thailand");
        cb.addItem("USA");
        cb.addItem("Japan");
        cb.setSelectedItem("Thailand");
        fr.add(cb);
        fr.pack();
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        JComboBoxDemo obj = new JComboBoxDemo();
        obj.init();
    }
}
```

รูปที่ 5.20 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.11

5.4.7 คลาส JList

JList เป็นคลาสที่ใช้สร้างอ็อบเจกต์ที่เป็นส่วนประกอบกราฟิกเพื่อให้ผู้ใช้สามารถเลือกรายการคล้ายกับ JComboBox แต่จะแตกต่างกันตรงที่ JList จะแสดงรายการหลายรายการโดยที่คลาส JList มี constructor ที่สำคัญดังนี้

- `public JList()`
- `public JList([]Object objs)`

คลาส `JList` มีเมธอดที่สำคัญดังนี้

- `public void setSelectedIndex(int index)`
- `public void setSelectedIndices(int[]indices)`
- `public void setSelectedValue(Object item, boolean state)`
- `public void setSelectionMode(int selectionMode)`
- `public int getSelectedIndex()`
- `public int[] getSelectedIndices()`
- `public Object getSelectedValue()`
- `public Object[] getSelectedValues()`
- `public int getSelectionMode()`

เราสามารถที่จะใส่รายการลงในอ็อบเจกต์ `JList` ได้ตอนสร้างอ็อบเจกต์ โดยสามารถใส่เข้าไปเป็นแบบอะเรย์ (โดยอะเรย์จะมีการกล่าวถึงอย่างละเอียดในบทที่ 8) ดังตัวอย่างโปรแกรมที่ 5.12 ซึ่งจะได้ผลลัพธ์ดังแสดงในรูปที่ 5.21

<u>โปรแกรมที่ 5.12</u> <u>ตัวอย่างการสร้างอ็อบเจกต์ `JList`</u>

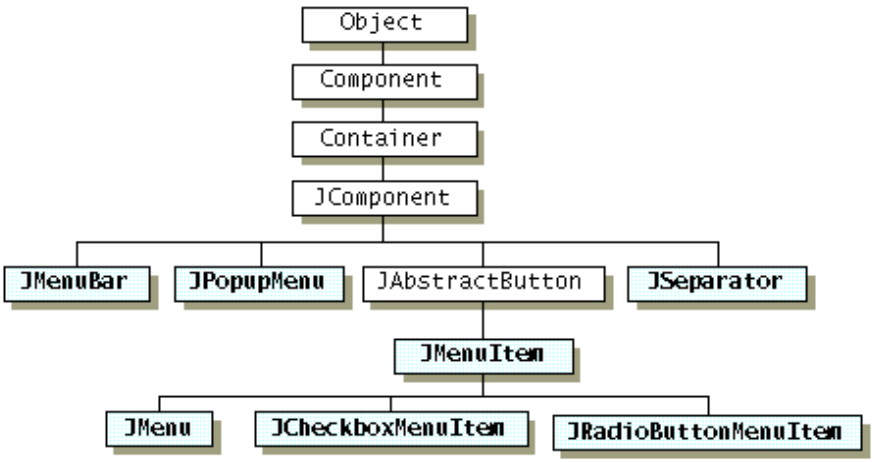
```
import java.awt.*;
import javax.swing.*;

public class JListDemo {

    private JFrame fr;
    private JList list;
    private String[] choices = {"Java SE", "Java EE", "Java ME"};

    public void init() {
        fr = new JFrame("JList Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        list = new JList(choices);
        list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        fr.setLayout(new FlowLayout());
        fr.add(list);
        fr.pack();
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        JListDemo obj = new JListDemo();
        obj.init();
    }
}
```

รูปที่ 5.21 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.12

5.5 การสร้างเมนู

โปรแกรมจาวาประยุกต์ที่มีส่วนต่อประสานกราฟิกกับผู้ใช้ สามารถที่จะมีเมนูให้ผู้ใช้เลือกรายการได้ เมนูเป็นอ็อบเจกต์ของคลาสที่สืบทอดมาจากคลาส `JComponent` โดยมีลำดับชั้นของคลาสต่างๆ ที่เกี่ยวข้องกับการสร้างเมนู ดังแสดงในรูปที่ 5.22 ซึ่งคลาสต่างๆ ที่สำคัญมีดังนี้

- `JMenuBar` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เก็บกลุ่มของอ็อบเจกต์ของคลาส `JMenu` ซึ่งจะปรากฏเป็นแถบเมนูโดยอ็อบเจกต์ของคลาสนี้จะต้องมีอ็อบเจกต์ของคลาส `JFrame` ที่คู่กัน
- `JMenu` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เก็บกลุ่มของอ็อบเจกต์ของคลาส `JMenuItem` และตัวแยกรายการ (`JSeparator`)
- `JMenuItem` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็นรายการ
- `JCheckboxMenuItem` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็นรายการ โดยจะมีเครื่องหมายถูกที่จะแสดงขึ้นเมื่อรายการนี้ถูกเลือก
- `JRadioButtonMenuItem` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็นรายการให้เลือกเพียงตัวเดียวแบบ Radio Button



รูปที่ 5.22 คลาสที่เกี่ยวข้องกับการสร้างเมนู

5.5.1 การสร้าง JMenuBar

คลาส JMenuBar เป็นคลาสที่จะแสดงเป็นแถบเมนูที่ปรากฏอยู่บน JFrame และมี constructor ดังนี้

- public JMenuBar()

เราสามารถที่จะใส่อ็อบเจกต์ชนิด JMenuBar ลงในอ็อบเจกต์ประเภท Container ได้ โดยใช้เมธอด setJMenuBar() ของคลาสประเภท Container

5.5.2 การสร้าง JMenu

JMenu เป็นคลาสที่ใช้ในการสร้างรายการที่จะแสดงอยู่ข้างในอ็อบเจกต์ของคลาส JMenuBar โดยมี constructor ที่สำคัญดังนี้

- public JMenu()
- public JMenu(String label)

โดยที่

- label คือข้อความที่ปรากฏอยู่ในรายการ

เราจะใช้เมธอด add() ในคลาส JMenuBar เพื่อที่จะใส่อ็อบเจกต์ของคลาส JMenu ลงใน JMenuBar โปรแกรมที่ 5.13 แสดงตัวอย่างการสร้างอ็อบเจกต์ของคลาส JMenu เพื่อใส่ลงใน JMenuBar โดยมีชื่อรายการต่างๆ โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 5.23

รูปที่ 5.23 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.13

โปรแกรมที่ 5.13 ตัวอย่างการสร้าง JMenu ที่ใส่ลงในอ็อบเจกต์ของคลาส JMenuBar

```
import javax.swing.*;

public class JMenuDemo {
private JFrame fr;
   private JMenuBar mb;
   private JMenu m1,m2,m3,m4;
   public void init() {
      fr = new JFrame("JMenu Demo");
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      mb = new JMenuBar();
      m1 = new JMenu("File");
      m2 = new JMenu("Edit");
      m3 = new JMenu("View");
      m4 = new JMenu("Help");
      fr.setJMenuBar(mb);
      mb.add(m1);
      mb.add(m2);
      mb.add(m3);
      mb.add(m4);
      fr.setSize(200,150);
      fr.setVisible(true);
   }
   public static void main(String args[]) {
      JMenuDemo mm = new JMenuDemo();
      mm.init();
   }
}
```

5.5.3 การสร้าง JMenuItem

JMenuItem คือคลาสที่ใช้ในการสร้างรายการย่อยที่อยู่ในอ็อบเจกต์ชนิด JMenu คลาส JMenuItem มี constructor ที่สำคัญดังนี้

- public JMenuItem()
- public JMenuItem(String label)
- public JMenuItem(String label, int mnemonic)

เราสามารถที่จะใส่อ็อบเจกต์ของคลาส JMenuItem ลงในอ็อบเจกต์ของคลาส JMenu โดยเรียกใช้เมธอด add() ในคลาส JMenu นอกจากนี้เราสามารถที่จะกำหนดคีย์ที่เป็น mnemonic ของรายการที่อยู่ในอ็อบเจกต์ของคลาส JMenuItem ได้โดย

เราสามารถที่จะสรุปขั้นตอนการสร้างเมนูได้ดังนี้

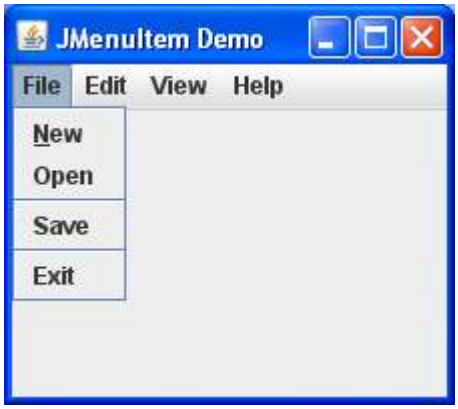
1. สร้างอ็อบเจกต์ของคลาส JMenuBar แล้วใส่ลงในอ็อบเจกต์ประเภท Container

2. สร้างอ็อบเจกต์ของคลาส JMenu หนึ่งอ็อบเจกต์หรือมากกว่า แล้วใส่ลงในอ็อบเจกต์ของคลาส JMenuBar
3. สร้างอ็อบเจกต์ของคลาส JMenuItem หนึ่งอ็อบเจกต์หรือมากกว่า แล้วใส่ลงใน อ็อบเจกต์ของคลาส JMenu

โปรแกรมที่ 5.14 แสดงตัวอย่างการสร้าง JMenuItem และ JMenu ลงใน JMenuBar โดยจะมีผลลัพธ์ดังแสดงในรูปที่ 5.24

โปรแกรมที่ 5.14 ตัวอย่างการสร้าง JMenuItem

```
import javax.swing.*;

public class JMenuItemDemo {

    private JFrame fr;
    private JMenuBar mb;
    private JMenu m1, m2, m3, m4;
    private JMenuItem mi1, mi2, mi3, mi4;

    public void init() {
        fr = new JFrame("MenuItem Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mb = new JMenuBar();
        m1 = new JMenu("File");
        m1.setMnemonic('F');
        m2 = new JMenu("Edit");
        m3 = new JMenu("View");
        m4 = new JMenu("Help");
        fr.setJMenuBar(mb);
        mb.add(m1);
        mb.add(m2);
        mb.add(m3);
        mb.add(m4);
        mi1 = new JMenuItem("New");
        mi2 = new JMenuItem("Open");
        mi3 = new JMenuItem("Save");
        mi4 = new JMenuItem("Exit");
        m1.add(mi1);
        m1.add(mi2);
        m1.addSeparator();
        m1.add(mi3);
        m1.addSeparator();
        m1.add(mi4);
        fr.setSize(200, 200);
        fr.setVisible(true);
    }
    public static void main(String args[]) {
        JMenuItemDemo mid = new JMenuItemDemo();
        mid.init();
    }
}
```



รูปที่ 5.24 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.14

5.5.4 คลาส `JCheckBoxMenuItem`

`JCheckBoxMenuItem` คือรายการเมนูที่มีเครื่องหมายระบุว่ารายการนี้ถูกเลือก คลาส `JCheckBoxMenuItem` มี constructor ที่สำคัญดังนี้

- `public JCheckBoxMenuItem()`
- `public JCheckBoxMenuItem(String label)`
- `public JCheckBoxMenuItem(String label, boolean state)`

โดยที่

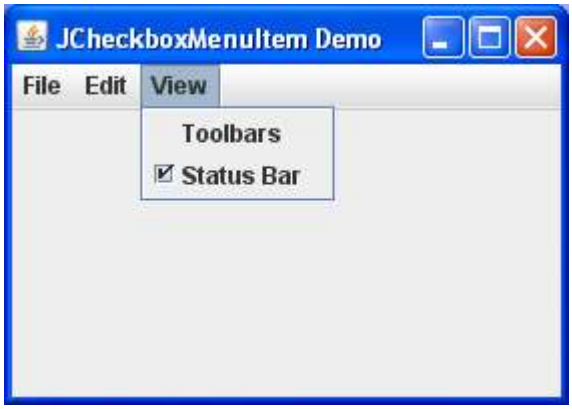
- `label` คือข้อความในรายการเมนู
- `state` คือสถานะในการเลือกโดยทั่วไป จะมีค่าเป็น `false`

นอกจากนี้เราสามารถที่จะเปลี่ยนสถานะของอ็อบเจกต์ของคลาส `JCheckBoxMenuItem` โดยใช้เมธอด `setState(boolean b)`

โปรแกรมที่ 5.15 แสดงตัวอย่างการสร้างอ็อบเจกต์ของคลาส `JCheckBoxMenuItem` โดยจะมีผลลัพธ์ดังแสดงในรูปที่ 5.25

โปรแกรมที่ 5.15 ตัวอย่างการสร้าง `JCheckBoxMenuItem`

```
import javax.swing.*;

public class JCheckBoxMenuDemo {
   private JFrame fr;
   private JMenuBar mb;
   private JMenu m1,m2,m3;
   private JMenuItem mi;
   private JCheckBoxMenuItem cbm;
   public void init() {
      fr = new JFrame("JCheckboxMenuItem Demo");
      fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      mb = new JMenuBar();
      m1 = new JMenu("File");
      m2 = new JMenu("Edit");
      m3 = new JMenu("View");
      fr.setJMenuBar(mb);
      mb.add(m1);
      mb.add(m2);
      mb.add(m3);
      mi = new JMenuItem("Toolbars");
      cbm = new JCheckBoxMenuItem("Status Bar", true);
      m3.add(mi);
      m3.add(cbm);
      fr.setSize(200,200);
      fr.setVisible(true);
   }
   public static void main(String args[]) {
      JCheckBoxMenuDemo obj= new JCheckBoxMenuDemo();
      obj.init();
   }
}
```



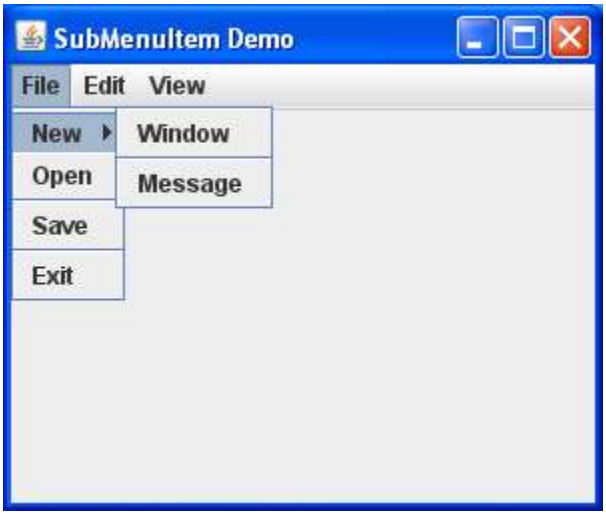
รูปที่ 5.25 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.15

5.5.5 การสร้างเมนูย่อย

เมนูสามารถที่จะมีเมนูย่อยได้โดยเมนูย่อยจะเป็นอ็อบเจกต์ของคลาส `JMenu` ซึ่งขั้นตอนในการสร้างเฟรมที่มีเมนูย่อยมีดังนี้

1. สร้างอ็อบเจกต์ของคลาส `JMenuBar` แล้วใส่ลงไปในอ็อบเจกต์ของคลาส `JFrame`
2. สร้างอ็อบเจกต์ของคลาส `JMenu` แล้วใส่ลงไปในอ็อบเจกต์ของคลาส `JMenuBar`
3. สร้างอ็อบเจกต์ของคลาส `JMenu` สำหรับเมนูย่อยแล้วใส่ลงไปในอ็อบเจกต์ของคลาส `JMenu` ที่เป็นเมนูหลัก
4. สร้างอ็อบเจกต์ของคลาส `JMeunItem` แล้วใส่ลงไปในอ็อบเจกต์ของคลาส `JMenu` ที่เป็นเมนูย่อย

โปรแกรมที่ 5.16 แสดงตัวอย่างการสร้าง `JFrame` ที่มีเมนูย่อยโดยโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 5.26



รูปที่ 5.26 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.16

โปรแกรมที่ 5.16 ตัวอย่างการสร้าง `JFrame` ที่มีเมนูย่อย

```
import javax.swing.*;

public class SubmenuDemo {

    private JFrame fr;
    private JMenuBar mb;
    private JMenu m1, m2, m3, ms1;
    private JMenuItem mi2, mi3, mi4, msi1, msi2;

    public void init() {
        fr = new JFrame("SubMenuItem Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        mb = new JMenuBar();
        m1 = new JMenu("File");
        m2 = new JMenu("Edit");
        m3 = new JMenu("View");
        fr.setJMenuBar(mb);
        mb.add(m1);
        mb.add(m2);
        mb.add(m3);
        ms1 = new JMenu("New");
        mi2 = new JMenuItem("Open");
        mi3 = new JMenuItem("Save");
        mi4 = new JMenuItem("Exit");
        m1.add(ms1);
        m1.add(mi2);
        m1.addSeparator();
        m1.add(mi3);

        m1.addSeparator();
        m1.add(mi4);
        msi1 = new JMenuItem("Window");
        msi2 = new JMenuItem("Message");
        ms1.add(msi1);
        ms1.addSeparator();
        ms1.add(msi2);
        fr.setSize(200, 200);
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        SubmenuDemo obj = new SubmenuDemo();
        obj.init();
    }
}
```

5.6 คุณลักษณะของคลาส Component

ส่วนประกอบกราฟิกต่างๆ จะมีคุณลักษณะอื่นอาทิเช่น รูปแบบของฟอนต์ สีของพื้นหลังหรือสีของพื้นหน้า (Foreground) เราสามารถที่จะกำหนดคุณลักษณะของส่วนประกอบกราฟิกได้ โดยปกติส่วนประกอบกราฟิกจะใช้คุณลักษณะแบบเดียวกับอ็อบเจกต์ประเภท `Container` ที่บรรจุอยู่เว้นแต่จะมีการกำหนดคุณลักษณะเฉพาะของส่วนประกอบกราฟิกนั้นๆ

เมธอดที่ใช้ในการกำหนดคุณลักษณะของส่วนประกอบกราฟิก จะอยู่ในคลาส `Component` โดยมีเมธอดที่สำคัญคือ

- `public void setFont(Font f)`
- `public void setForeground(Color c)`
- `public void setBackground(Color c)`

เมธอด `setFont()` ใช้ในการกำหนดรูปแบบของฟอนต์ ส่วนเมธอด `setForeground()` และ `setBackground()` ใช้ในการกำหนดสีพื้นหน้าและพื้นหลังตามลำดับ

คลาส `Font` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์เพื่อกำหนดรูปแบบของฟอนต์ โดยมี constructor ดังนี้

- `public Font(String name, int style, int size)`

โดยที่

- `name` คือชื่อของฟอนต์
- `style` เป็นรูปแบบของฟอนต์ ซึ่งคลาส `Font` ได้กำหนดค่าคงที่ไว้คือ `Font.PLAIN, Font.BOLD` และ `Font.ITALIC`
- `size` คือขนาดของฟอนต์

ตัวอย่างของการสร้างอ็อบเจกต์ของคลาส `Font` มีดังนี้

- `Font fn1 = new Font("AngsanaUPC",Font.PLAIN,16);`
- `Font fn2 = new Font("Time Romans", Font.BOLD + Font.ITALIC, 14);`

คลาส `Color` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์สำหรับกำหนดรูปแบบของสี โดยมี constructor ดังนี้

- `public Color(int r, int g, int b)`

โดยที่

- `r, g, b` คือค่าความเข้มของแสงสีแดง เขียวและน้ำเงิน ตามลำดับ

คลาส `Color` มีคุณลักษณะที่กำหนดสีที่ใช้ทั่วไปไว้แล้วหลายๆ สีอาทิเช่น

```
yellow = new Color(255,255,0);
black = new Color(0,0,0);
```

โปรแกรมที่ 5.12 แสดงตัวอย่างการกำหนดคุณลักษณะของอ็อบเจกต์ของคลาส `Frame` และคลาส `Button` ให้มีสีและฟอนต์ต่างๆ โดยจะได้ผลลัพธ์ดังแสดงในรูปที่ 5.27

<u>โปรแกรมที่ 5.17</u> <u>ตัวอย่างการกำหนดคุณลักษณะของอ็อบเจกต์ประเภทกราฟิก</u>

```
import java.awt.*;
import javax.swing.*;


public class AttributeDemo {

    private JFrame fr;
    private JButton bn1, bn2, bn3;

    public void init() {
        fr = new Jframe("ShowAttribute");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        bn1 = new JButton("OK");
        bn2 = new JButton("Cancel");
        bn3 = new JButton("Help");
        fr.setLayout(new FlowLayout());
        fr.add(bn1);
        fr.add(bn2);
        fr.add(bn3);
        bn2.setFont(new Font("TimesRoman", Font.BOLD, 16));
        fr.getContentPane().setBackground(Color.blue);
        bn2.setForeground(Color.red);
        fr.setSize(200, 150);
        fr.setVisible(true);
    }

    public static void main(String args[]) {
        AttributeDemo ad = new AttributeDemo();
        ad.init();
    }
}
```

รูปที่ 5.27 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 5.12

สรุปเนื้อหาของบท

- AWT เป็นแพคเก็จที่ใช้ในการพัฒนาโปรแกรม GUI ขั้นพื้นฐาน ซึ่งจะให้โปรแกรม GUI ที่เป็น look and feel ที่ขึ้นอยู่กับแพลตฟอร์ม โดยกำหนดในแพคเก็จชื่อ `java.awt`
- Swing เป็นแพคเก็จที่มีส่วนประกอบกราฟิกที่มีคุณลักษณะและรูปแบบที่ดีกว่าส่วนประกอบกราฟิกของแพคเก็จ AWT และสามารถกำหนดรูปแบบของ look and feel ที่ทำให้ได้โปรแกรม GUI ที่มีรูปแบบของกราฟิกเหมือนกันในทุกแพลตฟอร์ม โดยกำหนดในแพคเก็จชื่อ `javax.swing`
- คลาส `Component` อยู่ในแพคเก็จ AWT เป็น superclass ของคลาสประเภทส่วนประกอบกราฟิกทุกคลาส
- คลาส `Container` อยู่ในแพคเก็จ AWT เป็นคลาสประเภทตัวใส่ส่วนประกอบกราฟิก สืบทอดมาจากคลาสที่ชื่อ `Component`
- คลาสประเภทตัวใส่ส่วนประกอบกราฟิกที่สำคัญในแพคเก็จ AWT คือคลาส `Frame` และ `Panel`
- คลาสประเภทส่วนประกอบกราฟิกที่สำคัญในแพคเก็จ AWT คือ `Button, Label, TextField, TextArea, Checkbox, Choice` และ `List`
- รูปแบบการจัดวางผังส่วนประกอบกราฟิกในตัวใส่ส่วนประกอบกราฟิกมีทั้งหมด 5 รูปแบบคือ `BorderLayout, FlowLayout, GridLayout, CardLayout` และ `GridBagLayout`
- การจัดวางผังของส่วนประกอบกราฟิกแบบ `BorderLayout` จะเป็นการวางตามทิศต่างๆ ได้ 5 ทิศ ส่วน `FlowLayout` จะวางไว้ในตำแหน่งบนสุดโดยเรียงจากซ้ายไปขวา สำหรับ `GridLayout` จะวางเรียงจากซ้ายไปขวา และบนลงล่างในช่องย่อยที่มีขนาดเท่ากัน ตามจำนวนแถวและคอลัมน์ที่ได้ระบุไว้
- คลาสประเภทตัวใส่ส่วนประกอบกราฟิกที่สำคัญในแพคเก็จ Swing คือคลาส `JFrame` และ `JPanel`
- คลาสประเภทส่วนประกอบกราฟิกที่สำคัญในแพคเก็จ Swing คือ `JButton, JLabel, JTextField, JTextArea, JCheckBox, JRadioButton, JComboBox` และ `JList`

- `JLabel` จะแสดงข้อความ ซึ่งผู้ใช้ไม่สามารถป้อนข้อความได้ แต่สำหรับ `JTextField` ผู้ใช้สามารถป้อนข้อความได้ยาวหนึ่งบรรทัด ส่วน `JtextArea` ผู้ใช้สามารถป้อนข้อความได้หลายบรรทัด
- `JCheckBox` จะเป็นช่องให้ผู้ใช้เลือกหรือไม่เลือกโดยสามารถเลือกได้หลายช่องพร้อมกัน ซึ่งจะแตกต่างจาก `JRadioButton` ที่จะสามารถเลือกได้เพียงช่องเดียวเท่านั้น
- `JComboBox` จะแสดงรายการที่ถูกเลือกเพียงรายการเดียว โดยจะแสดงหลายรายการเมื่อผู้ใช้คลิกเมาส์เท่านั้น ซึ่งจะแตกต่างจาก `JList` ที่จะสามารถแสดงได้หลายรายการพร้อมกัน
- คลาสที่จะนำมาใช้ในการแสดงแถบเมนูและเมนูย่อยในแพคเก็จ Swing คือ `JMenuBar, JMenu` และ `JMenuItem`
- คลาส `Font` และ `Color` จะถูกนำมาใช้ในกรณีที่ต้องการระบุคุณลักษณะ ฟอนต์และสีของตัวใส่ส่วนประกอบกราฟิกและส่วนประกอบกราฟิก

บทที่ 6 การเขียนโปรแกรมภาษาจาวาเชิงอ็อบเจกต์

เนื้อหาในบทที่ผ่านมาเป็นการแนะนำการใช้คำสั่งเบื้องต้นต่างๆ ในภาษาจาวา โดยการเขียนโปรแกรมในตัวอย่างที่ผ่านมาส่วนใหญ่ เป็นการเขียนโปรแกรมภาษาจาวาโดยใช้หลักการเชิงกระบวนการ เนื้อหาในบทนี้จะเป็นการแนะนำการเขียนโปรแกรมภาษาจาวาเชิงอ็อบเจกต์ โดยแนะนำการใช้เมธอด การใช้ constructor การเขียนโปรแกรมโดยใช้คุณลักษณะเด่นของโปรแกรมเชิงอ็อบเจกต์ อาทิเช่น การห่อหุ้ม การสืบทอด และการมีได้หลายรูปแบบ เป็นต้น จากนั้นจะเป็นการแนะนำคลาสภายใน Generic Type และ Annotation

6.1 เมธอด

เมธอดเป็นสมาชิกของคลาสเพื่อระบุวิธีการหรือพฤติกรรม ที่อ็อบเจกต์ของคลาสสามารถกระทำได้ เมธอดทำหน้าที่ในการส่งข่าวสารระหว่างอ็อบเจกต์ ซึ่งเปรียบเทียบได้กับการใช้ function หรือ procedure ที่อยู่ในโปรแกรมเชิงกระบวนการ การเขียนโปรแกรมโดยใช้หลักการเชิงอ็อบเจกต์จะเป็นการแบ่งโปรแกรมออกเป็นโมดูลย่อยๆ โดยแต่ละโมดูลจะมีเมธอดอยู่ภายในที่สามารถติดต่อกับโมดูลโดยการเรียกใช้เมธอด ทำให้สามารถปรับเปลี่ยนแก้ไขโปรแกรมได้ง่ายขึ้น และสามารถที่จะใช้ลักษณะเด่นต่างๆ ของหลักการเชิงอ็อบเจกต์ ได้เช่น หลักการของการห่อหุ้ม การสืบทอด และการมีได้หลายรูปแบบ

6.1.1 การเรียกใช้เมธอด

เมธอดที่กำหนดขึ้นในคลาสใดๆ สามารถเรียกใช้งานได้สองรูปแบบคือ

1. การเรียกใช้งานจากคลาสที่ต่างกัน
2. การเรียกใช้งานภายในคลาสเดียวกัน

การเรียกใช้เมธอดจากคลาสที่ต่างกัน จะต้องมีการสร้างอ็อบเจกต์ของคลาสที่มีเมธอดที่จะถูกเรียกใช้งานก่อน จึงจะสามารถเรียกใช้เมธอดได้ ดังตัวอย่างในโปรแกรมที่ 6.1 คลาส `MyMain` จะต้องสร้างอ็อบเจกต์ของคลาส `NumericalClass` ก่อน จึงจะเรียกใช้งานเมธอด `calculate()` ได้ โดยใช้คำสั่ง

```
NumericalClass obj = new NumericalClass();
obj.calculate();
```

ทั้งนี้เมธอดที่จะถูกเรียกใช้งานจากคลาสที่ต่างกัน จะต้องไม่มี modifier เป็น `private`

การเรียกใช้เมธอดภายในคลาสเดียวกันสามารถทำได้โดยไม่จำเป็นต้องสร้างอ็อบเจกต์ของคลาสขึ้นมาก่อน และสามารถเรียกเมธอดได้ทุกเมธอด ยกเว้นเมธอดที่มี modifier เป็น `static` จะไม่สามารถเรียกใช้เมธอดที่ไม่มี modifier เป็น `static` (เรียกว่า เมธอดแบบ non-static) ได้ ซึ่งจะกล่าวถึงภายหลัง โปรแกรมที่ 6.2 แสดงตัวอย่างการเรียกใช้เมธอดภายในคลาสเดียวกัน โปรแกรมนี้กำหนดเมธอดในคลาส `NumericalClass` ขึ้นมาใหม่ที่ชื่อ `callMethod()` แล้วเรียกใช้เมธอด `calculate()` ภายในเมธอดที่กำหนดขึ้นใหม่

<u>โปรแกรมที่ 6.1</u> <u>การเรียกใช้เมธอดจากคลาสที่ต่างกัน</u>

```
public class NumericalClass {
    public void calculate() {
        double score = Math.random()*100;
        if (score >= 80) {
            System.out.println("Grade is A");
        } else if (score >= 70) {
            System.out.println("Grade is B");
        } else if (score >= 60){
            System.out.println("Grade is C");
        } else if (score >= 50){
            System.out.println("Grade is D");
        } else {
            System.out.println("Grade is F");
        }
    }
}
-----------------------------------------------------------------------------------------------------------
public class MyMain {
    public static void main(String args[]) {
        NumericalClass obj = new NumericalClass();
        obj.calculate();
    }
}
```

6.1.2 การส่งผ่าน argument

เมธอดที่กำหนดขึ้นในคลาสอาจมี argument ที่รับค่าเพื่อนำไปใช้ในเมธอดอาทิเช่น เมธอด `setGpa()` ในคลาส `Student` ของโปรแกรมที่ 6.3 จะมี argument สำหรับรับค่าข้อมูลที่มีชนิดข้อมูลเป็น `double` ในกรณีนี้การเรียกใช้เมธอดจะต้องส่ง argument ที่มีชนิดข้อมูลที่สอดคล้องกันไปพร้อมกัน ดังนั้นการเรียกใช้เมธอด `setGpa()` จะต้องส่ง argument ที่มีชนิดข้อมูลเป็น `double` ไปพร้อมกับชื่อเมธอด ดังแสดงในคลาส `MyMain2` ในโปรแกรมที่ 6.3 (คำสั่ง `s1.setGpa(3.0);`)

โปรแกรมที่ 6.2 การเรียกใช้เมธอดภายในคลาสเดียวกัน

```
public class NumericalClass {
    public void calculate() {
        double score = Math.random()*100;
        if (score >= 80) {
            System.out.println("Grade is A");
        } else if (score >= 70) {
            System.out.println("Grade is B");
        } else if (score >= 60){
            System.out.println("Grade is C");
        } else if (score >= 50){
            System.out.println("Grade is D");
        } else {
            System.out.println("Grade is F");
        }
    }
    public void callMethod() {
        calculate();
    }
}
```

โปรแกรมที่ 6.3 ตัวอย่างการส่งผ่าน argument

```
public class Student {
      String id;
      String name;
      double gpa;
      public void setGpa(double GPA) {
         gpa = GPA;
      }
      public double getGpa() {
         return gpa;
      }
}

-------------------------------------------------------
public class MyMain2 {
      public static void main(String args[]) {
            Student s1 = new Student();
            s1.setGpa(3.0);
      }
}
```

argument ของเมธอดจะมีชนิดข้อมูลเป็นสองแบบตามชนิดข้อมูลของตัวแปรดังนี้

1.argument ที่มีชนิดข้อมูลแบบพื้นฐาน

2.argument ที่มีชนิดข้อมูลแบบอ้างอิง

ในกรณีของ argument ที่มีชนิดข้อมูลแบบพื้นฐาน เราสามารถที่จะส่งค่าคงที่ข้อมูล ตัวแปร หรือนิพจน์ให้กับ argument ได้อาทิเช่น การเรียกใช้เมธอด `setGpa()` นั้น argument ที่จะส่งผ่านอาจมีรูปแบบเป็น

- ค่าคงที่ข้อมูล เช่น

```
s1.setGpa(3.0);
```

- ตัวแปร เช่น

```
double  x = 3.0;
s1.setGpa(x);
```

- นิพจน์ เช่น

```
s1.setGpa(3.0+0.05);
```

ส่วนกรณี argument ที่มีชนิดข้อมูลแบบอ้างอิง เราจะต้องส่งอ็อบเจกต์ที่มีชนิดข้อมูลที่สอดคล้องไปเท่านั้น ยกเว้นกรณีที่ argument นั้นมีชนิดข้อมูลเป็น `String` ซึ่งในกรณีนี้จะสามารถส่งข้อมูลค่าคงที่ได้ โปรแกรมที่ 6.4 เป็นตัวอย่างของคลาส `StudentV1` ที่มีคุณลักษณะ `dob` ซึ่งมีชนิดข้อมูลเป็นคลาส `MyDate` ที่กำหนดไว้ในโปรแกรมที่ 6.5 เพื่อเก็บวัน เดือน ปีเกิดของอ็อบเจกต์ชนิด `StudentV1` คลาส `StudentV1` มีเมธอด `setDOB()` เพื่อใช้ในการกำหนดค่าให้กับคุณลักษณะ `dob`

โปรแกรมที่ 6.4 ตัวอย่างการส่งผ่าน argument ที่เป็นอ็อบเจกต์

```
public class StudentV1 {
    String id;
    String name;
    MyDate dob;

    public void setDOB(MyDate d) {
      dob = d;
    }
    public MyDate getDOB() {
      return dob;
    }
}
```

โปรแกรมที่ 6.6 แสดงการเรียกใช้เมธอดดังกล่าวโดยต้องส่งผ่านอ็อบเจกต์ของคลาส `MyDate` ให้กับเมธอดดังนี้

```
MyDate d1 = new MyDate(16, 12, 1980);
s1.setDOB(d1);
```

สำหรับกรณีของโปรแกรมที่ 6.6 เนื่องจากอ็อบเจกต์ `d1` ที่สร้างขึ้นมา จะไม่มีการอ้างอิงภายในโปรแกรมนี้อีก

ดังนั้นเราสามารถที่จะสร้างอ็อบเจกต์และส่งผ่านไปยัง argument ในคำสั่งเดียวกันได้ดังนี้

```
s1.setDOB(new MyDate (16, 12, 1980));
```

โปรแกรมที่ 6.5 คลาส `MyDate` เพื่อเก็บค่าวัน เดือน และปี

```
public class MyDate {
    private int day;
    private int month;
    private int year;

    public MyDate(int d, int m, int y) {
        day = d;
        month = m;
        year = y;
    }

    public void setDay(int d) {
        day = d;
    }

    public void setMonth(int m) {
        month = m;
    }

    public void setYear(int y) {
        year = y;
    }

    public void showDetails() {
        System.out.println("Date : " + day + "/" + month + "/" + year);
    }
}
```

โปรแกรมที่ 6.6 ตัวอย่างการเรียกใช้เมธอดที่มี argument เป็นอ็อบเจกต์

```
public class TestStudentV1 {
    public static void main(String args[]) {
       StudentV1 s1 = new StudentV1();
       MyDate d1 = new MyDate(16, 12, 1980);
       s1.setDOB(d1);
    }
}
```

ชนิดข้อมูลของ argument ที่จะส่งผ่านไปยังเมธอดไม่จำเป็นที่จะต้องเป็นชนิดข้อมูลเดียวกัน แต่ต้องเป็นชนิดข้อมูลที่สามารถแปลงข้อมูลให้กว้างขึ้นได้โดยอัตโนมัติอาทิเช่น เราสามารถที่จะส่ง argument ที่มีชนิดข้อมูลเป็น `int` ให้กับเมธอด `setGpa()` ที่ต้องการ argument ที่มีชนิดข้อมูลเป็น `double` ได้ดังนี้ `s1.setGpa(3);`

เมธอดใดๆ อาจมี argument สำหรับรับค่ามากกว่าหนึ่งตัว แต่การเรียกใช้เมธอดเหล่านี้จะต้องส่ง argument ที่มีชนิดข้อมูลที่สอดคล้องกันและมีจำนวนเท่ากัน ยกเว้นกรณีที่ argument ของเมธอดเป็นแบบ varagrs (Variable Arguments) โดยมีสัญลักษณ์เป็น `...` เช่นถ้ามีการประกาศเมธอด `calMax(int... x)` จะทำให้สามารถเรียกใช้เมธอดนี้ได้โดยไม่ต้องส่งหรือจะส่ง argument กี่ตัวก็ได้ เช่น `obj.calMax();` หรือ `obj.calMax(1,2,3);` จะสามารถใช้ได้ทั้งสิ้น

โปรแกรมที่ 6.7 แสดงตัวอย่างของคลาส `NumericalSample` ซึ่งมีเมธอด `calMax()` ที่รับ argument สองตัวที่มีชนิดข้อมูลเป็น `int` และ `double` ตามลำดับ การเรียกใช้เมธอดนี้จะต้องส่งผ่าน argument จำนวนสองตัวอาทิเช่น คำสั่ง `obj.calMax(3,4.0);` ในเมธอด `main()` นอกจากนี้เรายังจะสามารถเรียกใช้เมธอด `calMax()` ในรูปแบบอื่นได้โดยส่ง argument ที่มีชนิดข้อมูลที่สอดคล้องกันอาทิเช่น

```
obj.calMax(3,4);
obj.calMax(3,4L);
```

แต่เราไม่สามารถที่จะเรียกใช้เมธอดโดยมีจำนวน argument ไม่เท่ากับที่กำหนด หรือมีชนิดข้อมูลที่ไม่สอดคล้องกันอาทิเช่น คำสั่ง

```
obj.calMax(3);
obj.calMax(4.0);
obj.calMax(3L,4.0);
obj.calMax(3.0,4);
obj.calMax(3,4.0,5);
```

จะเป็นคำสั่งในการเรียกใช้เมธอดที่ผิด

<u>โปรแกรมที่ 6.7 ตัวอย่างการส่งผ่าน argument จำนวนสองตัว</u>

```
public class NumericalSample {
   public void calMax(int i, double d) {
      if (i > d) {
         System.out.println("Max = "+i);
      } else {
         System.out.println("Max = "+d);
      }
   }
   public static void main(String args[]) {
      NumericalSample obj = new NumericalSample();
      obj.calMax(3,4.0);
   }
}
```

ในกรณีที่ argument มีชนิดข้อมูลเป็นแบบพื้นฐาน และมีการส่ง argument โดยใช้ตัวแปรให้กับเมธอดที่ถูกเรียกใช้งาน โปรแกรมภาษาจาวาจะถือว่าเป็นการส่งค่าไปให้เท่านั้น หากมีการเปลี่ยนแปลงค่าของ argument ภายในเมธอดที่ถูกเรียกใช้งาน จะไม่มีผลทำให้ค่าของ argument ที่ส่งไปเปลี่ยนค่าไปด้วย โปรแกรมที่ 6.8 แสดงตัวอย่างการเรียกใช้เมธอด `method1()` ในคลาส `ArgumentPassing` ซึ่งรับ argument ที่เป็นตัวแปร `x` เข้ามา ซึ่งเมื่อ `x` ในเมธอดเปลี่ยนค่าเป็น 3 จะไม่มีผลทำให้ค่าของ `x` ในเมธอด `main()` เปลี่ยนไปด้วย จึงได้ผลลัพธ์ออกมาเป็น 4 เช่นเดิม อนึ่งตัวแปร `x` ที่อยู่ในเมธอด `main()` และเมธอด `method1()` เป็นตัวแปรคนละตัวกัน และมีตำแหน่งอ้างอิงสำหรับเก็บข้อมูลในหน่วยความจำที่ต่างกัน เพียงแต่มีชื่อเดียวกันเท่านั้น

ส่วนกรณีที่ argument มีชนิดข้อมูลเป็นแบบอ้างอิงจะเป็นการส่งตำแหน่งอ้างอิงของอ็อบเจกต์ไปให้กับเมธอดที่ถูกเรียกใช้งาน ดังนั้นการเปลี่ยนแปลงค่าของคุณลักษณะของอ็อบเจกต์จะมีผลทำให้ค่าของคุณลักษณะของอ็อบเจกต์ที่ส่งไปเปลี่ยนไปด้วย ตัวอย่างเช่นในโปรแกรมที่ 6.8 การเรียกใช้เมธอด `method2()` โดยการส่งผ่านตัวแปรที่เป็นอ็อบเจกต์ของคลาส `Date` ไป แล้วเปลี่ยนค่าของคุณลักษณะวัน เดือน และปีของอ็อบเจกต์ในเมธอด `method2()` จะมีผลทำให้ค่าของคุณลักษณะวัน เดือน และปีของอ็อบเจกต์ `d1` ในเมธอด `main()` เปลี่ยนไปด้วย

โปรแกรมที่ 6.8 ตัวอย่างการเปลี่ยนแปลงค่าของ argument

```
public class ArgumentPassing {
   public void method1(int x) {
        x = 3;
    }

    public void method2(MyDate d) {
        d.setDay(1);
        d.setMonth(1);
        d.setYear(2002);
    }

    public int method3() {
        return 0;
    }

    public static void main(String args[]) {
        int x = 4;
        ArgumentPassing obj = new ArgumentPassing();
        MyDate d1 = new MyDate(16, 12, 1980);
        obj.method1(x);
        System.out.println("x = " + x);
        obj.method2(d1);
        d1.showDetails();
        obj.method3();
    }
}
```

6.1.3 การรับค่าที่ส่งกลับมา

เมธอดใดๆ ของคลาสสามารถที่จะมีค่าที่ส่งกลับมาได้ ทั้งนี้การประกาศเมธอดจะต้องระบุชนิดข้อมูลของค่าที่จะส่งกลับใน return_type ซึ่งชนิดข้อมูลของค่าที่จะส่งกลับอาจเป็นชนิดข้อมูลแบบพื้นฐาน หรือเป็นชนิดข้อมูลแบบอ้างอิง ในกรณีที่เมธอดไม่มีข้อมูลจะส่งกลับมาจะต้องระบุให้ return_type เป็น `void` อาทิเช่น เมธอด `setGpa()` ในคลาส `student` ของโปรแกรมที่ 6.3 นอกจากนี้เมธอดที่มีค่าที่จะส่งกลับมาจะต้องมีคำสั่ง `return` ซึ่งจะระบุค่าที่ส่งกลับโดยมีรูปแบบดังนี้

```
return value;
```

โดยที่ `value` ขึ้นอยู่กับชนิดข้อมูลที่จะส่งกลับ ถ้าเป็นชนิดข้อมูลแบบพื้นฐานอาจเป็นค่าคงที่ข้อมูล ตัวแปร หรือนิพจน์ก็ได้ แต่ถ้าเป็นชนิดข้อมูลแบบอ้างอิงต้องเป็นอ็อบเจกต์เท่านั้น ยกเว้นชนิดข้อมูล `String` ที่อาจเป็นข้อมูลค่าคงที่ได้ คำสั่ง `return` จะมีผลให้โปรแกรมสิ้นสุดการทำงานในบล็อกของเมธอดนั้นแล้วกลับไปยังคำสั่งเดิมที่เรียกใช้เมธอดนั้น

โดยทั่วไปการเรียกใช้เมธอดที่มีการส่งค่ากลับมานั้น จะต้องเรียกใช้เมธอดนั้นในนิพจน์อาทิเช่น การเรียกใช้เมธอด `getGpa()` ของคลาส `Student` ในโปรแกรมที่ 6.3 อาจทำได้ดังนี้

```
double d = s1.getGpa();
```

หรือ

```
System.out.println("GPA: "+ s1.getGpa());
```

ทั้งสองคำสั่งเป็นการเรียกใช้เมธอดในนิพจน์ แต่คำสั่งแรกจะเก็บค่าที่ส่งกลับในตัวแปรที่ `d` ส่วนคำสั่งที่สองจะไม่เก็บค่าที่ส่งกลับแต่จะพิมพ์ค่าออกมา

นอกจากนี้เราสามารถที่เรียกใช้เมธอดที่มีการส่งค่ากลับมา โดยไม่อยู่ในรูปของนิพจน์ได้อาทิเช่น เมธอด `method3()` ในโปรแกรมที่ 6.8 จะส่งค่าที่มีชนิดข้อมูลเป็น `int` กลับมา แต่เนื่องจากไม่ต้องการใช้ค่าดังกล่าว จึงสามารถใช้คำสั่ง `obj.method3();` ได้โดยไม่ต้องใช้คำสั่งกำหนดค่า

6.1.4 modifier ของเมธอด

เมธอดของคลาสจะมี modifier ที่ประกาศไว้หน้า return_type ของเมธอดดังที่ระบุไว้ในหัวข้อที่ 4.2.3 ซึ่ง modifier ของเมธอดแบ่งออกเป็น

- access modifier เพื่อระบุถึงระดับการเข้าถึง
- `static` เพื่อระบุว่าเป็นเมธอดแบบ `static`
- `abstract` เพื่อระบุว่าเมธอดยังไม่สมบูรณ์
- `synchronized` เพื่อระบุว่าเมธอดจะมีการล็อกอ็อบเจกต์ ซึ่งใช้ในกรณีของโปรแกรมที่เป็นแบบเธรด (thread)
- `final` เพื่อระบุว่าเมธอดนี้ไม่สามารถ override ได้

access modifier

modifier ประเภทนี้ใช้เพื่อระบุระดับการเข้าถึง โดยมีคีย์เวิร์ดต่างๆ ดังนี้

- `public`
- `protected`
- `private`
- default (ไม่ระบุคีย์เวิร์ดใดๆ)

เมธอดหรือคุณลักษณะที่มี modifier เป็นแบบ `public` จะทำให้เมธอดหรือคุณลักษณะนั้นสามารถที่จะถูกเรียกใช้จากเมธอดของคลาสอื่นได้ ส่วนเมธอดหรือคุณลักษณะที่มี modifier เป็นแบบ `private` จะถูกป้องกันไม่ให้เมธอดจากคลาสอื่นเรียกใช้ได้ โดยจะสามารถถูกเรียกใช้ได้จากเมธอดภายในคลาสเดียวกันเท่านั้น โปรแกรมที่ 6.9 แสดงตัวอย่างการประกาศเมธอดและคุณลักษณะที่มี modifier เป็นแบบ `public` และ `private` ตัวอย่างนี้แสดงให้เห็นว่าเมธอดและคุณลักษณะที่มี modifier เป็นแบบ `private` ไม่สามารถที่จะให้คลาสอื่นเรียกใช้งานได้ กรณีที่เมธอดหรือคุณลักษณะไม่ระบุคีย์เวิร์ดใดๆ สำหรับ access modifier ภาษาจาวาจะถือว่าเป็น default ซึ่งจะทำให้เมธอดหรือคุณลักษณะนั้นสามารถที่จะถูกเรียกใช้จากเมธอดของคลาสอื่นที่อยู่ในแพคเก็จเดียวกันได้ สำหรับความหมายของ modifier ที่เป็นแบบ `protected` จะกล่าวถึงต่อไป

6.2 การเขียนโปรแกรมโดยใช้หลักการของการห่อหุ้ม

ข้อดีของการห่อหุ้มประการหนึ่งคือการซ่อนเร้นข้อมูลโปรแกรมเชิงอ็อบเจกต์ที่ดีจะออกแบบให้อ็อบเจกต์มีเมธอดที่ประกาศให้อ็อบเจกต์อื่นๆ เรียกใช้งานได้ แต่จะซ่อนรายละเอียดหรือวิธีการที่เมธอดนั้นใช้ และซ่อนคุณลักษณะต่างๆ ของอ็อบเจกต์ไว้ หลักการห่อหุ้มของ อ็อบเจกต์ในภาษาจาวาทำได้โดยการระบุ modifier ของเมธอดหรือคุณลักษณะดังนี้

- คุณลักษณะของอ็อบเจกต์จะมี modifier เป็น `private` เพื่อซ่อนไม่ให้อ็อบเจกต์อื่นๆ เรียกใช้ได้
- เมธอดของอ็อบเจกต์ที่ต้องการให้อ็อบเจกต์อื่นๆ เรียกใช้จะมี modifier เป็น `public`

โปรแกรมที่ 6.9 ตัวอย่างการประกาศเมธอดและคุณลักษณะ

```
public class PrivateStudent {
   private String id;
   private String name;
   private double gpa;
   public void setDetails(String ID,String n,double GPA) {
         id = ID;
         name = n;
         gpa = GPA;
   }
   public void showDetails() {
         System.out.println("ID: "+id);
         System.out.println("Name: "+name);
         System.out.println("GPA: "+gpa);
   }
}
-----------------------------------------------------------------
public class TestPrivateStudent {
   public static void main(String args[]) {
      PrivateStudent ps = new PrivateStudent();
      /*    ps.id = "12345";           illegal
            ps.name = "Thana";         illegal
            ps.gpa = 3.25;             illegal */
      ps.setDetails("12345","Thana",3.25);
      ps.showDetails();
   }
}
```

โปรแกรมที่ 6.10 ตัวอย่างคลาสที่ไม่ได้ใช้หลักการของการห่อหุ้ม

```
public class Student {
   String ID;
   String name;
   public double gpa;
}

-----------------------------------------------------------------
public class NoEncapDemo {
   public static void main(String args[]) {
      Student s1 = new Student();
      double temp = Double.parseDouble(args[0]);
      if ((temp<0) || (temp>4.00)) {
         System.out.println("Incorrect Format!");
      } else {
         s1.gpa = temp;
         System.out.println("GPA: "+s1.gpa);
      }
   }
}
```

```
Command Prompt

C:\_JAVA\Code\Chapter6\build\classes>java NoEncapDemo 2.00
GPA: 2.0

C:\_JAVA\Code\Chapter6\build\classes>java NoEncapDemo -1.59
Incorrect Format!

C:\_JAVA\Code\Chapter6\build\classes>java NoEncapDemo 4.50
Incorrect Format!
```

รูปที่ 6.1 ตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.10

โปรแกรมที่ 6.10 แสดงตัวอย่างคลาสที่ไม่ได้ใช้หลักการของการห่อหุ้ม โดยการกำหนดให้คุณลักษณะ `gpa` ซึ่งมี modifier เป็น `public` ทำให้อ็อบเจกต์อื่นๆ สามารถเรียกใช้ `gpa` ได้โดยตรง คลาส `NoEncapDemo` ได้สร้างอ็อบเจกต์ของคลาส `Student` ที่ชื่อว่า `s1` และได้กำหนดให้ผู้ใช้สามารถป้อนค่า `gpa` ผ่านเข้ามาทาง command line ซึ่งผู้ใช้อาจป้อนค่า `gpa` ที่ไม่ถูกต้องเช่น เป็นค่าตัวเลขจำนวนลบ หรือมีค่ามากกว่า 4.00 ดังนั้นจึงจะต้องมีการตรวจสอบค่าที่ป้อนเข้ามาก่อนที่จะมีการกำหนดค่าให้กับคุณลักษณะ `gpa` ทุกครั้งโดยใช้คำสั่ง `if..else` การเขียนโปรแกรมแบบนี้ทำให้โปรแกรมซับซ้อนขึ้นและยากต่อการแก้ไข ซึ่งตัวอย่างของผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.10 เป็นดังรูปที่6.1

โปรแกรมที่ 6.11 ได้เปลี่ยนคุณลักษณะของ `gpa` ให้มี modifier เป็น `private` และกำหนดเมธอดใหม่ที่ชื่อ `setGpa()` ที่มี modifier เป็น `public` เพื่อให้อ็อบเจกต์อื่นๆ ที่ต้องการกำหนดค่า `gpa` เรียกใช้ได้ ซึ่งการเขียนโปรแกรมแบบนี้เป็นการใช้หลักการของการห่อหุ้ม ดังนั้นการกำหนดค่าให้กับคุณลักษณะ `gpa` ของอ็อบเจกต์ `s1` ในคลาส `EncapDemo` จะทำได้โดยการเรียกใช้เมธอด `setGpa()` เท่านั้น ซึ่งในเมธอดดังกล่าวจะมีการตรวจสอบค่าของ `gpa` อยู่ภายใน ทำให้การเขียนโปรแกรมทำได้ง่ายขึ้นเพราะไม่จำเป็นต้องตรวจสอบค่าทุกครั้ง และง่ายต่อการแก้ไขโปรแกรมในกรณีที่ข้อกำหนดของคุณลักษณะ `gpa` เปลี่ยนแปลงไป เนื่องจากเป็นคำสั่งเฉพาะในเมธอดเท่านั้น

6.2.1 เมธอดแบบ Accessor

การเข้าถึงคุณลักษณะของอ็อบเจกต์ที่มี modifier เป็น `private` นั้นจะต้องใช้เมธอดแบบ accessor ในการกำหนดค่าของคุณลักษณะ หรือดึงค่าของคุณลักษณะออกมาใช้งาน

เมธอดแบบ accessor แบ่งออกได้เป็นสองประเภทคือ

- เมธอดแบบ setter จะใช้ในการกำหนดค่าของคุณลักษณะ ซึ่งเมธอดแบบนี้จะไม่มีการส่งค่ากลับ ดังนั้น

จึงระบุชนิดข้อมูลของค่าที่ต้องการส่งกลับเป็น `void` โดยทั่วไปชื่อของเมธอดแบบนี้จะขึ้นต้นด้วยคำว่า `set` แล้วตามด้วยชื่อของคุณลักษณะ ซึ่งมีรูปแบบดังนี้

```
public void setAttributeName(dataType arg) {
    attributeName = arg;
}
```

โดยที่

- `dataType` คือชนิดข้อมูลของคุณลักษณะที่ต้องการกำหนดค่า
- `arg` คือตัวแปรที่จะรับค่าของคุณลักษณะที่ต้องการกำหนด
- `attributeName` คือชื่อของคุณลักษณะ

ตัวอย่างเช่นคลาส `Student` มีเมธอด `setGpa()` เพื่อกำหนดค่าของคุณลักษณะ `gpa` ซึ่งมีคำสั่งดังนี้

```
public void setGpa(double GPA) {
      gpa = GPA;
}
```

โปรแกรมที่ 6.11 ตัวอย่างคลาสที่ใช้หลักการของการห่อหุ้ม (ตัวอย่างที่ 1)

```
public class Student {
   String ID;
   String name;
   private double gpa;
   public void setGpa(double GPA) {
      if ((GPA<0) || (GPA>4.00)) {
         System.out.println("Incorrect Format!");
      } else {
         gpa = GPA;
      }
   }
   public double getGpa() {
      return gpa;
   }
}
-----------------------------------------------------------------------------
public class EncapDemo {
   public static void main(String args[]) {
      Student s1 = new Student();
      double temp = Double.parseDouble(args[0]);
      s1.setGpa(temp);
      System.out.println("GPA: "+s1.getGpa());
   }
}
```

- เมธอดแบบ getter จะใช้ในการเรียกค่าของคุณลักษณะ ซึ่งเมธอดแบบนี้จะมีการส่งค่ากลับ ดังนั้นจึงต้องระบุชนิดข้อมูลของค่าที่ส่งกลับมาด้วยเสมอ โดยทั่วไปชื่อของเมธอดแบบนี้จะขึ้นต้นด้วยคำว่า `get` แล้วตามด้วยชื่อของคุณลักษณะ ซึ่งมีรูปแบบดังนี้

```
public dataType getAttributeName() {
    return attributeName;
}
```

ตัวอย่างเช่น คลาส `Student` มีเมธอด `getGpa()` เพื่อเรียกค่า ของ `gpa` ที่มีคำสั่งดังนี้

```
public double getGpa() {
      return gpa;
}
```

ชื่อของเมธอดแบบ getter สำหรับคุณลักษณะที่มีชนิดข้อมูลเป็น `boolean` โดยทั่วไปจะกำหนดให้ขึ้นต้นด้วยคำว่า `is` แทนที่จะเป็นคำว่า `get` อาทิเช่น เมธอด `isGraduate()` เป็นเมธอดเพื่อเรียกค่าของคุณลักษณะ `graduate` ของคลาส `Student` ที่มีชนิดข้อมูลเป็น `boolean`

<u>โปรแกรมที่ 6.12</u> <u>ตัวอย่างคลาสที่ใช้หลักการของการห่อหุ้ม (ตัวอย่างที่ 2)</u>

```
public class EncapStudent {
   private String id;
   private String name;
   private double gpa;
   public void setId(String ID) {
      id = ID;
   }
   public void setName(String n) {
      name = n;
   }
   public void setGpa(double GPA) {
      if ((GPA<0) || (GPA>4.00)) {
         System.out.println("Incorrect Format!");
      } else {
         gpa = GPA;
      }
   }
   public String getId() {
      return id;
   }
   public String getName() {
      return name;
   }
   public double getGpa() {
      return gpa;
   }
}
```

โปรแกรมที่ 6.12 แสดงตัวอย่างของโปรแกรมภาษาจาวาที่ใช้หลักการของการห่อหุ้มโดยกำหนดให้คุณลักษณะต่างๆ ของอ็อบเจกต์มี modifier เป็นแบบ `private` และกำหนดให้มีเมธอดแบบ accessor ทั้งที่เป็นแบบ setter และ getter เพื่อทำให้สามารถเข้าถึงคุณลักษณะแต่ละตัวได้

คุณลักษณะของอ็อบเจกต์บางกรณีอาจไม่สามารถกำหนดเมธอดแบบ setter ได้ ทั้งนี้เนื่องจากอาจไม่ต้องการให้มีการกำหนดค่าได้โดยตรง โปรแกรมที่ 6.13 แสดงตัวอย่างคลาส `Account` ซึ่งมีคุณลักษณะ `balance` เพื่อเก็บยอดเงินคงเหลือในบัญชี โปรแกรมนี้ได้กำหนดให้มีเมธอดแบบ getter ที่ชื่อ `getBalance()` เพื่อเรียกดูยอดเงินในบัญชี แต่ไม่มีเมธอดแบบ setter เพื่อกำหนดค่า `balance` ทั้งนี้เนื่องจากคุณลักษณะ `balance` จะเปลี่ยนค่าไปเมื่อมีการเรียกใช้เมธอด `deposit()` และ `withdraw()` ที่ใช้ในการฝากและถอนเงินตามลำดับ

<u>โปรแกรมที่ 6.13</u> <u>ตัวอย่างคลาสที่ใช้หลักการของการห่อหุ้ม (ตัวอย่างที่ 3)</u>

```
public class Account {
   private double balance;
   public double getBalance() {
      return balance;
   }
   public void deposit(double amount) {
      balance += amount;
   }
   public void withdraw(double amount) {
      balance -= amount;
   }
}
```

6.2.2 คีย์เวิร์ด `this`

คีย์เวิร์ด `this` หมายถึงอ็อบเจกต์ของตัวเอง เราสามารถที่จะเรียกใช้เมธอดหรือคุณลักษณะภายในคลาสได้โดยใช้คีย์เวิร์ด `this` ซึ่งมีรูปแบบดังนี้

```
this.methodName();
this.attirbuteName;
```

โดยทั่วไปเราจะไม่ใช้คีย์เวิร์ด `this` ในคำสั่ง ยกเว้นในกรณีที่จำเป็น โปรแกรมที่ 6.14 เป็นตัวอย่างการเรียกใช้คุณลักษณะของออปเจ็ตโดยใช้คีย์เวิร์ด `this` กรณีนี้ถ้าไม่ระบุว่า `this.id`, `this.name` และ `this.gpa` คอมไพเลอร์จะเข้าใจว่าหมายถึงตัวแปรที่เป็น argument ของเมธอด ไม่ใช่ตัวแปรที่เป็นคุณลักษณะของอ็อบเจกต์

โปรแกรมที่ 6.14 ตัวอย่างแสดงการใช้คีย์เวิร์ด this

```
public class ThisStudent {
   private String id;
   private String name;
   private double gpa;
   public void setDetails (String id, String name, double gpa) {
         this.id = id;
         this.name = name;
         this.gpa = gpa;
   }
   public void showDetails() {
         System.out.println("ID: "+id);
         System.out.println("Name: "+name);
         System.out.println("GPA: "+gpa);
   }
}
```

6.3 การเขียนโปรแกรมโดยใช้หลักการของการสืบทอด

ข้อดีของการสืบทอดคือ การนำคลาสที่มีอยู่แล้วมาใช้ใหม่โดยการเพิ่มเติมคุณลักษณะหรือเมธอดในคลาสใหม่ เพื่อให้เข้าใจหลักการของการสืบทอดให้พิจารณาตัวอย่างของคลาส `Student` ที่กำหนดไว้ในโปรแกรมที่ 6.15 ซึ่งถ้าหากจำเป็นที่จะต้องพัฒนาคลาสขึ้นใหม่ที่ชื่อ `GradStudent` เพื่อใช้ในการสร้างอ็อบเจกต์ที่เป็นนักศึกษาระดับบัณฑิตศึกษานั้น เราสามารถที่จะเลือกวิธีการได้สองแบบคือ

1.สร้างคลาสขึ้นมาใหม่โดยไม่อ้างอิงกับคลาสเดิมที่ชื่อ `Student` ดังโปรแกรมที่ 6.16

2.สร้างคลาสที่สืบทอดมาจากคลาสเดิมที่ชื่อ `Student` ดังโปรแกรมที่ 6.17

การสร้างคลาสขึ้นใหม่ตามโปรแกรมที่ 6.16 นั้นจะเห็นได้ว่ามีความซ้ำซ้อนในการกำหนดคุณลักษณะและเมธอดของคลาส นอกจากนี้ยังจะทำให้เกิดความยุ่งยากในการแก้ไขโปรแกรมอีกด้วยอาทิเช่น หากต้องมีการเพิ่มคุณลักษณะ `address` จะทำให้ต้องแก้ไขโปรแกรมของทั้งสองคลาส

โปรแกรมที่ 6.15 ตัวอย่างคลาสที่ชื่อ `Student`

```
public class Student {
    private String id;
    private String name;
    private double gpa;
    public void setId(String id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setGpa(double gpa) {
        this.gpa = gpa;
    }
    public void showDetails() {
        System.out.println("ID: " + id);
        System.out.println("Name: " + name);
        System.out.println("GPA: " + gpa);
    }
}
```

ภาษาจาวาได้กำหนดคีย์เวิร์ดที่ชื่อ `extends` เพื่อระบุการสืบทอดของคลาส โดยมีรูปแบบดังนี้

```
public class SubClass extends SuperClass {
    ...
}
```

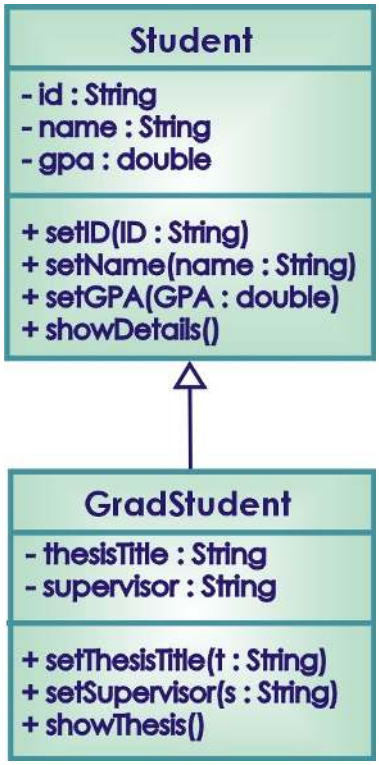
โดยที่ `SuperClass` คือชื่อของคลาสที่ต้องการจะให้มีการสืบทอดคุณลักษณะและเมธอดต่างๆ ไปยัง `SubClass` โปรแกรมที่ 6.16 สามารถที่จะเขียนได้ใหม่โดยใช้หลักการของการสืบทอดดังแสดงในโปรแกรมที่ 6.17 ซึ่งจะเห็นได้ว่ากรณีนี้จะลดความซ้ำซ้อนของคุณลักษณะและเมธอด และจะทำให้การปรับปรุงและแก้ไขโปรแกรมเป็นไปได้ง่ายยิ่งขึ้น ไดอะแกรมคลาสของคลาสทั้งสองสามารถแสดงดังรูปที่ 6.2 คลาส `GradStudent` จะสืบทอดคุณลักษณะและเมธอดของคลาส `Student` มา ดังนั้นอ็อบเจกต์ของคลาส `GradStudent` จะมีคุณลักษณะที่เป็น `id`, `name` และ `gpa` เช่นเดียวกับอ็อบเจกต์ของคลาส `Student`

โปรแกรมที่ 6.16 ตัวอย่างคลาสที่ชื่อ `GradStudent` ที่ไม่ได้ใช้หลักการของการสืบทอด

```
public class GradStudent {
    private String id;
    private String name;
    private double gpa;
    private String thesisTitle;
    private String supervisor;
    public void setId(String id) {
        this.id = id;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setGpa(double gpa) {
        this.gpa = gpa;
    }
    public void setThesisTitle(String thesisTitle) {
        this.thesisTitle = thesisTitle;
    }
    public void setSupervisor(String supervisor) {
        this.supervisor = supervisor;
    }
    public void showThesis() {
        System.out.println("ThesisTitle: " + thesisTitle);
        System.out.println("Supervisor: " + supervisor);
    }
}
```

โปรแกรมที่ 6.17 ตัวอย่างคลาสที่ชื่อ `GradStudent` ที่ใช้หลักการของการสืบทอด

```
public class GradStudent extends Student {
   private String thesisTitle;
   private String supervisor;
   public void setThesisTitle(String t) {
      thesisTitle = t;
   }
   public void setSupervisor(String s) {
      supervisor = s;
   }
   public void showThesis() {
      System.out.println("ThesisTitle: "+thesisTitle);
      System.out.println("Supervisor: "+supervisor);
   }
}
```



รูปที่ 6.2 การใช้หลักการของการสืบทอด

6.3.1 การสืบทอดที่ถูกต้อง

เราสามารถที่จะกำหนดให้คลาสใดๆ สืบทอดคลาสอื่นได้ แต่การออกแบบโปรแกรมเชิงอ็อบเจกต์ที่ดีนั้น ควรจะให้คลาสที่จะสืบทอดกันนั้นมีความสัมพันธ์ที่ถูกต้องกันด้วยตัวอย่างเช่น หากกำหนดให้คลาส `Shirt` และ `Skirt` มีคุณลักษณะของคลาสดังนี้

```
public class Shirt {
      char size;
      float price;
}
public class Skirt {
      char size;
      float price;
      boolean long;
}
```

นักพัฒนาโปรแกรมอาจเห็นว่าคลาสทั้งสองมีคุณลักษณะที่ซ้ำซ้อนกันหลายชื่อ จึงอาจตัดสินใจกำหนดให้คลาส `Skirt` สืบทอดมาจากคลาส `Shirt` โดยมีโปรแกรมดังนี้

```
public class Shirt {
      char size;
      float price;
}
public class Skirt extends Shirt {
      boolean long;
}
```

แต่ในความเป็นจริงแล้ว `Shirt` และ `Skirt` จะไม่มีความสัมพันธ์ในลักษณะการ สืบทอดโดยตรง เราสามารถที่จะทดสอบความสัมพันธ์ของการสืบทอด โดยใช้คำว่า "เป็น" (is a) ได้ คลาสที่จะสืบทอดกันจะต้องสามารถบรรยายด้วยวลีที่ว่า subclass เป็น superclass (A subclass is a superclass) แต่คำว่ากระโปรงเป็นเสื้อ (A skirt is a shirt) เป็นวลีที่ไม่ถูกต้องจึงทำให้คลาสทั้งสองไม่ควรจะสืบทอดกันได้ อย่างไรก็ตามเราสามารถที่จะเขียนโปรแกรมของคลาสทั้งสองได้ใหม่ดังนี้

```
public class Clothing {
      char size;
      float price;
}
public class Shirt extends Clothing {
}
public class Skirt extends Clothing {
      boolean long;
}
```

ทั้งนี้เนื่องจากทั้งเสื้อ (shirt) และกระโปรง (skirt) ต่างก็เป็นเสื้อผ้า (clothing)

6.3.2 คีย์เวิร์ด `protected`

ถึงแม้ว่าคลาสที่เป็น subclass จะสืบทอดคุณลักษณะและเมธอดของคลาสที่เป็น superclass มา แต่คลาสที่เป็น subclass จะไม่สามารถที่จะเรียกใช้คุณลักษณะหรือเมธอดของ superclass ที่มี modifier เป็นแบบ `private` ได้ ภาษาจาวากำหนดให้มี access modifier ที่ชื่อ `protected` ซึ่งจะทำให้คลาสที่เป็น subclass สามารถที่จะเรียกใช้เมธอดหรือคุณลักษณะของ superclass ได้ modifier แบบ `protected` จะแตกต่างจาก modifier แบบ default ในกรณีที่ superclass และ subclass อยู่ต่างแพคเก็จกัน ทั้งนี้กรณีของ default จะไม่สามารถเรียกใช้คุณลักษณะหรือเมธอดของคลาสที่อยู่ต่างแพคเก็จกันได้ แต่ถ้ามี modifier เป็นแบบ `protected` จะสามารถเรียกใช้ได้ในกรณีที่คลาสทั้งสองสืบทอดกัน ตารางที่ 6.1 แสดงการเปรียบเทียบการกำหนด modifier แบบต่างๆ เพื่อเรียกใช้คุณลักษณะหรือเมธอด และ โปรแกรมที่ 6.18 เป็นโปรแกรมที่แสดงตัวอย่างการใช้คีย์เวิร์ด `protected`

ตารางที่ 6.1 ระดับการเข้าถึงคุณลักษณะหรือเมธอดของ modifier แบบต่างๆ

Modifier	คลาสเดียวกัน	คลาสที่อยู่ในแพคเก็จเดียวกัน	คลาสที่เป็น subclass	คลาสใดๆ
public	√	√	√	√
protected	√	√	√	
default	√	√		
Private	√			

โปรแกรมที่ 6.18 ตัวอย่างการใช้คีย์เวิร์ด protected

```
public class Student {
    protected String id;
    protected String name;
    protected double gpa;
    public void setDetails (String ID,String n,double GPA) {
         id = ID;
         name = n;
         gpa = GPA;
    }
    public void showDetails() {
         System.out.println("ID: "+id);
         System.out.println("Name: "+name);
         System.out.println("GPA: "+gpa);
    }
}
```

6.3.3 คลาสที่ชื่อ Object

คลาสทุกคลาสในภาษาจาวาถ้าไม่ได้สืบทอดจากคลาสใดเลย จะถือว่าคลาสนั้นสืบทอดจากคลาสที่ชื่อ Object อาทิเช่น คลาส Student จะสืบทอดมาจากคลาสที่ชื่อ Object และคำสั่งประกาศคลาสจะถือเสมือนว่ามีรูปแบบดังนี้

```
public class Student extends Object {
      ...
}
```

คลาสที่ชื่อ Object จะมีเมธอดที่สำคัญคือ

- `public String toString()` และ
- `public boolean equals(Object o)`

เพื่อใช้ในการแปลงอ็อบเจกต์ของคลาสให้เป็นอ็อบเจกต์ชนิด `String` และใช้ในการเปรียบเทียบอ็อบเจกต์ของคลาสนั้นกับอ็อบเจกต์อื่นๆ

6.3.4 คีย์เวิร์ด super

`super` เป็นคีย์เวิร์ดที่ใช้ในการอ้างอิงถึง superclass เพื่อที่จะเรียกใช้เมธอดหรือ constructor ของ superclass โดยมีรูปแบบคำสั่งดังนี้

```
super.methodName([arguments])
```

ตัวอย่างเช่น คลาส `GradStudent` อาจกำหนดให้มีเมธอดที่ชื่อ `showDetails()` โดยมีคำสั่งที่เรียกใช้เมธอด `showDetails()` ของคลาส `Student` ดังแสดงในโปรแกรมที่ 6.19

โปรแกรมที่ 6.19 ตัวอย่างการใช้คีย์เวิร์ด `super`

```
public class Student {
      private String id;
      private String name;
      private double gpa;
      public void showDetails() {
         System.out.println("ID: "+id);
         System.out.println("Name: "+name);
         System.out.println("GPA: "+gpa);
    }
}

-----------------------------------------------------------------------------------------------------------
public class GradStudent extends Student {
      private String thesisTitle;
      private String supervisor;
      public void showDetails() {
         super.showDetails();
         System.out.println("Thesis Title: "+ thesisTitle);
         System.out.println("Supervisor: "+supervisor);
    }
}
```

6.4 การมีได้หลายรูปแบบ

การมีได้หลายรูปแบบหมายถึงคุณสมบัติของอ็อบเจกต์ของคลาสที่ต่างกัน ที่สามารถจะกำหนดค่าได้หลายรูปแบบ หรือการตอบสนองต่อเมธอดเดียวกันด้วยวิธีการที่ต่างกันได้ หลักการของการมีได้หลายรูปแบบเป็นหลักการที่สืบเนื่องมาจากหลักการของการสืบทอด

6.4.1 Dynamic Binding

Dynamic Binding เป็นหลักการของการกำหนดอ็อบเจกต์ของคลาสที่มีการสืบทอดกันได้หลายรูปแบบ ภาษาจาวากำหนดให้อ็อบเจกต์ของคลาสใดๆ สามารถสร้างมาจากคลาสนั้นหรือคลาสอื่นๆ ที่เป็น subclass ได้ กล่าวคือถ้าประกาศคลาส `SubClass` ซึ่งสืบทอดมาจากคลาส `SuperClass` แล้ว และมีคำสั่งประกาศอ็อบเจกต์ `obj` ให้เป็นอ็อบเจกต์ของคลาส `SuperClass` โดยใช้คำสั่ง

```
SuperClass obj;
```

เราสามารถใช้หลักการของการมีได้หลายรูปแบบในการสร้างอ็อบเจกต์ `obj` ได้ หลายรูปแบบดังนี้

```
      obj = new SuperClass();
หรือ  obj = new SubClass();
```

ในกรณีของตัวอย่างของคลาส `Student` ที่มี subclass เป็นคลาส `GradStudent` โดยที่คลาส `GradStudent` มี subclass อีกคลาสหนึ่งที่ชื่อ `PhDStudent` นั้น เราสามารถที่จะกำหนดอ็อบเจกต์ของคลาสเหล่านี้ได้หลายรูปแบบ ดังแสดงความสัมพันธ์ในรูปที่ 6.3 กล่าวคือ เราสามารถที่ประกาศอ็อบเจกต์ของคลาส `Student` แล้วสร้างอ็อบเจกต์ได้หลายรูปแบบดังนี้

```
      Student s1 = new Student();
หรือ  Student s1 = new GradStudent();
หรือ  Student s1 = new PhDStudent();
```

เช่นเดียวกับกรณีของอ็อบเจกต์ของคลาส `GradStudent` สามารถที่จะสร้างอ็อบเจกต์ได้หลายรูปแบบดังนี้

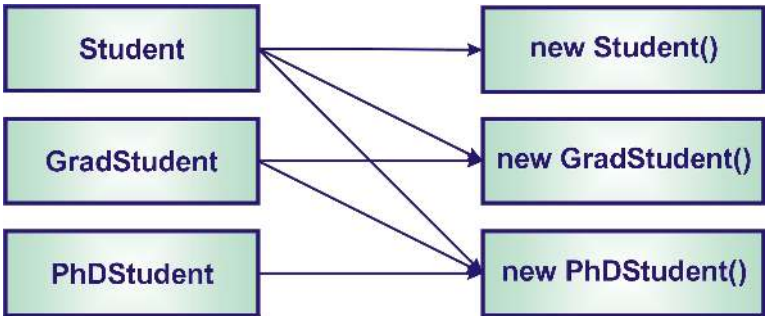
```
      GradStudent s2 = new GradStudent();
      GradStudent s2 = new PhDStudent();
```

แต่ในทางกลับกันเราจะไม่สามารถกำหนดอ็อบเจกต์ของ subclass แล้วสร้างอ็อบเจกต์ของ superclass ได้

กล่าวคือ คำสั่ง

```
GradStudent s3 = new Student();
```

จะเป็นคำสั่งที่ไม่ถูกต้อง



รูปที่ 6.3 อ็อบเจกต์ของคลาสที่มีได้หลายรูปแบบ

6.4.2 การกำหนดเมธอดให้มีวิธีการที่ต่างกัน

หลักการของการมีได้หลายรูปแบบยังรวมไปถึงการที่เมธอดชื่อเดียวกันแต่มีวิธีการ (คำสั่ง) ที่ต่างกัน ซึ่งสามารถกำหนดเมธอดได้สองรูปแบบคือ

1. เมธอดแบบ overloaded
2. เมธอดแบบ overridden

เมธอดแบบ Overloaded

ภาษาจาวาอนุญาตให้คลาสใดๆ มีเมธอดที่ชื่อเดียวกันมากกว่าหนึ่งเมธอดได้ แต่เมธอดเหล่านั้นจะต้องมีจำนวนหรือชนิดข้อมูลของ argument ที่ต่างกัน โปรแกรมที่ 6.20 เป็นตัวอย่างของการเขียนเมธอดแบบ overloaded โดยเมธอด `setDetails()` จะมีสองเมธอด เมธอดแรกจะรับ argument ที่มีสองตัว และเมธอดที่สองจะรับ argument ที่มีสามตัว ซึ่งคอมไพเลอร์จะพิจารณาว่าโปรแกรมเรียกใช้เมธอดใดโดยดูจากจำนวนและชนิดของ argument ทั้งนี้เมธอดแบบ overloaded อาจมีค่าที่ส่งกลับมาเป็นชนิดเดียวกันหรือต่างกันก็ได้

โปรแกรมที่ **6.20** ตัวอย่างของเมธอดแบบ overloaded

```
public class StudentV2 {
   private String id;
   private String name;
   private double gpa;
   public void setDetails(String ID,String n) {
      id = ID;
      name = n;
   }
   public void setDetails(String ID,String n,double GPA) {
      id = ID;
      name = n;
      gpa = GPA;
   }
}
```

ตัวอย่างต่อไปนี้เป็นการเขียนเมธอดแบบ overloaded ที่ถูกต้อง

```
public void setDetails(String ID,String n) {
}
public void setDetails(String ID,double GPA) {
}
public double setDetails(double GPA,String ID){
}
```

ตัวอย่างต่อไปนี้เป็นการเขียนเมธอดแบบ overloaded ที่ไม่ถูกต้อง

```
public void setDetails(String ID,double GPA) {
}
public void setDetails(String n,double GPA) {
}
```

กรณีตัวอย่างที่ไม่ถูกต้องนี้ถึงแม้ว่าจะมีชื่อของ argument ที่ต่างกันก็ตามแต่เนื่องจากเมธอดทั้งสองมีจำนวนและชนิดข้อมูลของ argument ที่เหมือนกัน ดังนั้นเมื่อคอมไพเลอร์เจอคำสั่งเรียกใช้เมธอด `setDetails()` แล้วส่ง argument ที่มีชนิดข้อมูลเป็น `String` และ `double` มา คอมไพเลอร์จะไม่สามารถแยกได้ว่าจะเรียกใช้เมธอดใด ดังนั้นถ้าชุดคำสั่งในตัวอย่างนี้อยู่ในโปรแกรมใดจะทำให้โปรแกรมนั้นไม่สามารถคอมไพล์ผ่านได้

เมธอดแบบ ***Overridden***

การกำหนดเมธอดแบบ overridden เป็นหลักการที่สืบเนื่องมาจากหลักการของการสืบทอด โดยคลาสที่เป็น subclass สามารถที่จะเขียนเมธอดของ superclass ขึ้นใหม่ได้ วิธีการนี้เรียกว่า เมธอดใน subclass เป็นเมธอดแบบ

overridden การเขียนเมธอดแบบ overridden มีข้อกำหนดดังนี้

- จำนวนและชนิดข้อมูลของ argument จะต้องเหมือนเดิม
- ชนิดข้อมูลของค่าที่ส่งกลับจะต้องเหมือนเดิม
- access modifier จะต้องไม่มีระดับต่ำกว่าเดิมอาทิเช่น ถ้าเมธอดเดิมเป็น `public` จะไม่สามารถเปลี่ยนเป็น `private` ได้

โปรแกรมที่ 6.21 แสดงตัวอย่างการกำหนดเมธอดแบบ overridden ที่ชื่อ `showDetails()` โดยการเขียนเมธอด `showDetails()` ของคลาส `Student` ขึ้นใหม่

โปรแกรมที่ 6.21 เมธอดแบบ overridden ที่ชื่อ `showDetails()`

```
public class Student {
   protected String id;
   protected String name;
   protected double gpa;

   public void showDetails() {
      System.out.println("ID: "+id);
      System.out.println("Name: "+name);
      System.out.println("GPA: "+gpa);
   }
}

-------------------------------------------------------
public class GradStudent extends Student {
   private String thesisTitle;
   private String supervisor;

   public void showDetails() {
      System.out.println("ID: "+id);
      System.out.println("Name: "+name);
      System.out.println("GPA: "+gpa);
      System.out.println("ThesisTitle: "+thesisTitle);
      System.out.println("Supervisor: "+supervisor);
   }
}
```

6.4.3 Virtual Method Invocation

การกำหนดอ็อบเจกต์ตามหลักการของ Dynamic Binding ทำให้โปรแกรมภาษาจาวาพิจารณาเรียกใช้เมธอดจากชนิดของอ็อบเจกต์ที่สร้างขึ้น ตัวอย่างเช่น คำสั่ง

```
Student s1 = new GradStudent();
s1.showDetails();
```

เป็นคำสั่งที่เรียกใช้เมธอด `showDetails()` ของคลาส `GradStudent` ไม่ใช่เมธอดของคลาส `Student` ทั้งนี้เนื่องจากอ็อบเจกต์ `s1` เป็นอ็อบเจกต์ของคลาส `GradStudent` แต่คอมไพเลอร์ของภาษาจาวาจะไม่อนุญาตให้เรียกใช้เมธอดใดๆ ก็ตามที่ไม่มีการประกาศอยู่ในเมธอดของ superclass ที่กำหนดไว้ กล่าวคือคำสั่ง

```
Student s1 = new GradStudent();
s1.getSupervisor();
```

เป็นคำสั่งที่ไม่ถูกต้องและจะไม่สามารถคอมไพล์ผ่านได้ เนื่องจากคอมไพเลอร์จะไม่รู้จักเมธอด `getSupervisor()` ที่อยู่ในคลาส `Student` ถึงแม้ว่าอ็อบเจกต์ `s1` จะเป็นอ็อบเจกต์ของคลาส `GradStudent` ที่มีเมธอดดังกล่าวอยู่ก็ตาม

6.4.4 การส่งผ่าน argument ได้หลายรูป

ในกรณีที่เมธอดมี argument เป็นข้อมูลชนิดคลาส เราสามารถที่จะใช้หลักการของการมีได้หลายรูปแบบในการส่ง argument ที่เป็นอ็อบเจกต์ให้กับเมธอดดังกล่าวได้ กล่าวคือเราสามารถที่จะส่งอ็อบเจกต์ของคลาสที่เป็น subclass ของคลาสนั้นแทนได้ ตัวอย่างเช่น คำสั่ง

```
Student s1 = new Student();
Student s2 = new GradStudent();
```

และเมธอด

```
public void printInfo(Student s) {
    ...
}
```

เป็นเมธอดที่มี argument เป็นข้อมูลชนิดคลาส `Student` เราสามารถที่จะเรียกใช้เมธอดนี้โดยการส่งอ็อบเจกต์ที่สืบทอดมาจากคลาส `Student` ได้ อาทิเช่น

```
printInfo(s1)
printInfo(s2)
```

หรือ

```
printInfo(new FullTimeStudent())
```

6.4.5 ตัวดำเนินการ instanceof

คีย์เวิร์ด instanceof เป็นตัวดำเนินการที่ใช้กับอ็อบเจกต์และคลาส เพื่อตรวจสอบว่าอ็อบเจกต์ใดๆ เป็นอ็อบเจกต์ของคลาสที่ระบุหรือไม่ คีย์เวิร์ด instanceof จะให้ผลลัพธ์เป็นชนิดข้อมูลแบบ boolean ซึ่งถ้าได้ผลลัพธ์เป็น true จะหมายถึงว่าอ็อบเจกต์ดังกล่าวเป็นอ็อบเจกต์ของคลาสที่ระบุหรือเป็นอ็อบเจกต์ของคลาสที่เป็น superclass ของคลาสนั้น

ตัวอย่างเช่น

```
GradStudent s1 = new GradStudent();
```

เป็นคำสั่งในการประกาศและสร้างอ็อบเจกต์ s1 ของคลาส GradStudent ดังนั้นนิพจน์

```
(s1 instanceof GradStudent)
```

จะให้ผลลัพธ์เป็น true เช่นเดียวกับนิพจน์

```
(s1 instanceof Student)
```

หรือ

```
(s1 instanceof Object)
```

ที่จะให้ผลลัพธ์เป็น true ทั้งนี้เนื่องจากคลาส GradStudent สืบทอดมาจากคลาส Student และคลาส Student ก็สืบทอดมาจากคลาสที่ชื่อ Object อีกต่อหนึ่ง

ส่วนนิพจน์

```
(s1 instanceof String)
```

จะไม่สามารถคอมไพล์ผ่านได้ ทั้งนี้เนื่องจากอ็อบเจกต์ของคลาส GradStudent จะไม่ใช่อ็อบเจกต์ของคลาสที่สืบทอดมาจากคลาส String

นอกจากนี้ คีย์เวิร์ด instanceof จะให้ผลลัพธ์เป็น false ในกรณีที่จะตรวจสอบว่าอ็อบเจกต์ดังกล่าวเป็นอ็อบเจกต์ของคลาสที่เป็น subclass หรือไม่ ตัวอย่างเช่นคำสั่ง

```
Student s1 = new Student();
```

จะได้นิพจน์

```
(s1 instanceof GradStudent)
```

ที่ให้ผลลัพธ์เป็น false

คีย์เวิร์ด instanceof สามารถใช้ในการตรวจสอบว่า argument ที่รับข้อมูลมาในเมธอดเป็นอ็อบเจกต์ของคลาสใด ตัวอย่างเช่น คำสั่งต่อไปนี้เป็นการใช้คีย์เวิร์ด instanceof เพื่อตรวจสอบว่า argument ของอ็อบเจกต์ s ว่าเป็นอ็อบเจกต์ของคลาสใด

```
public void printInfo(Student s) {
      if (s instanceof PhDStudent) {
            System.out.println("PhD Student");
      } else if (s instanceof GradStudent) {
            System.out.println("Graduate Student");
      } else if (s instanceof FullTimeStudent) {
            System.out.println("Full-Time Student");
      } else if (s instanceof PartTimeStudent) {
            System.out.println("Part-Time Student");
      } else if (s instanceof Student) {
            System.out.println("Student");
      }
}
```

6.4.6 การ Casting อ็อบเจกต์

หลักการของการมีได้หลายรูปแบบ ทำให้เราสามารถจะกำหนดอ็อบเจกต์ของคลาสที่เป็น subclass ให้กับตัวแปรที่กำหนดเป็นคลาสที่เป็น superclass ได้ ตัวอย่างเช่น

```
GradStudent s1 = new GradStudent();
Student s2 = s1;
```

แต่ในทางกลับกันตัวแปรที่กำหนดเป็นคลาสที่เป็น subclass จะไม่สามารถกำหนดค่าให้เป็นอ็อบเจกต์ของคลาสที่เป็น superclass ได้ ตัวอย่างเช่น

```
Student s1 = new GradStudent();
GradStudent s2 = s1;
```

จะเป็นคำสั่งที่ไม่ถูกต้อง และโปรแกรมนี้จะไม่สามารถคอมไพล์ผ่านได้ การกำหนดค่าในลักษณะนี้จะต้องทำการ casting อ็อบเจกต์เพื่อระบุชนิดของอ็อบเจกต์ โดยมีรูปแบบดังนี้

```
(ClassName) objectName
```

โดยที่

- `ClassName` คือชนิดของคลาสที่ต้องการระบุ
- `objectName` คือชื่อของตัวแปรของอ็อบเจกต์

ตัวอย่างที่ผ่านมาสามารถเขียนใหม่ได้เป็น

```
Student s1 = new GradStudent();
GradStudent s2 = (GradStudent) s1;
```

จึงจะทำให้โปรแกรมสามารถคอมไพล์ผ่านได้ เช่นเดียวกับคำสั่งในหัวข้อ 187 ที่เป็นตัวอย่างของการเรียกใช้

เมธอด ก็จะสามารถเขียนใหม่ได้เป็น

```
 Student s1 = new GradStudent();
((GradStudent) s1).getSupervisor();
```

จึงจะทำให้โปรแกรมสามารถคอมไพล์ผ่านได้ จากตัวอย่างข้างต้นอ็อบเจกต์ `s1` จะถูกกำหนดให้เป็นคลาสชนิด `GradStudent` ที่มีเมธอด `getSupervisor()`

ภาษาจาวาจะตรวจสอบชนิดของอ็อบเจกต์ในช่วงของการรันโปรแกรม ดังนั้นถึงแม้ว่าโปรแกรมจะคอมไพล์ผ่านได้ แต่ถ้าเกิดตรวจสอบพบว่าอ็อบเจกต์ที่ทำการ casting ไม่ใช่อ็อบเจกต์ของคลาสนั้นหรืออ็อบเจกต์ของ subclass โปรแกรมจะไม่สามารถรันผ่านได้

ตัวอย่างเช่น

```
Student s1 = new Student();
GradStudent s2 = (GradStudent) s1;
```

จะเป็นคำสั่งที่ทำให้เกิดข้อผิดพลาดในช่วงของการรันโปรแกรม เนื่องจากตัวแปร `s1` ถูกกำหนดค่าให้เป็นอ็อบเจกต์ของ superclass ซึ่งไม่สามารถทำได้ เช่นเดียวกับการเรียกใช้ เมธอดของอ็อบเจกต์ที่ทำการ casting ภาษาจาวาจะตรวจสอบชนิดของอ็อบเจกต์ในช่วงของการรันโปรแกรม ว่าอ็อบเจกต์ดังกล่าวเป็นอ็อบเจกต์ของคลาสใด และคลาสนั้นมีเมธอดที่เรียกใช้หรือไม่ หากไม่พบก็จะเกิดข้อผิดพลาดในช่วงของการรันโปรแกรม ตัวอย่างเช่น

```
Student s1 = new Student();
((GradStudent) s1).getSupervisor();
```

จะเป็นคำสั่งที่สามารถคอมไพล์ผ่านได้ แต่จะเกิดข้อผิดพลาดในช่วงของการรันโปรแกรม เนื่องจากจาวาอินเตอร์พรีตเตอร์จะพบว่า `s1` เป็นอ็อบเจกต์ของคลาส `Student` ซึ่งไม่มีเมธอด `getSupervisor()` โปรแกรมที่ 6.22 แสดงตัวอย่างการเรียกใช้อ็อบเจกต์ในรูปแบบต่างๆ ที่อาจเกิดข้อผิดพลาดในช่วงของการรันโปรแกรมหรือในช่วงของการคอมไพล์โปรแกรม

โปรแกรมที่ 6.22 ตัวอย่างการเรียกใช้อ็อบเจกต์ในรูปแบบต่างๆ ที่อาจเกิดข้อผิดพลาด

```
public class Student {
   protected String id;
   protected String name;
   protected double gpa;
   public String getId() {
      return id;
   }
   public String getName() {
      return name;
   }
   public double getGpa() {
      return gpa;
   }
}

-------------------------------------------------------
public class GradStudent extends Student {
   private String thesisTitle;
   private String supervisor;
   public String getThesisTitle() {
      return thesisTitle;
   }
   public String getSupervisor() {
      return supervisor;
   }
}
public class TestCallingMethods {

   public static void main(String args[]) {
      Student s1 = new GradStudent();
      s1.getSupervisor();
         // compile error

      Student s2 = new Student();
      ((GradStudent) s2).getSupervisor();
         // runtime error
   }
}
```

6.5 Constructor

constructor เป็นเมธอดที่มีชื่อเดียวกับชื่อคลาสซึ่งจะถูกเรียกใช้งานเมื่อมีการสร้างอ็อบเจกต์โดยใช้คำสั่ง `new` โดย constructor มีรูปแบบดังนี้

```
[modifier] ClassName([arguments]) {
     [statements]
}
```

constructor เป็นเมธอดที่ไม่มีค่าที่จะส่งกลับแต่ไม่ต้องระบุคีย์เวิร์ด `void` ซึ่งแตกต่างกับเมธอดทั่วๆ ไป modifier ของ constructor จะเป็น access modifier ซึ่งโดยทั่วไปจะเป็น `public` กรณีที่ constructor มี access modifier เป็น `private` จะทำให้ไม่สามารถสร้างอ็อบเจกต์ของคลาสนั้นจากคลาสอื่นได้ โดยทั่วไปคลาสทุกคลาสจะมี constructor แบบ default แม้จะไม่มีการเขียนคำสั่ง constructor ในโปรแกรม โดย constructor แบบ default จะมีรูปแบบดังนี้

```
public ClassName() {
}
```

constructor แบบ default จะเป็น constructor ที่ไม่มีคำสั่งใดๆ อยู่ภายใน คอมไพเลอร์ของภาษาจาวาจะใส่ constructor แบบ default ให้กับโปรแกรมโดยอัตโนมัติ อาทิเช่น คลาส `Student` ในโปรแกรมที่ 6.15 จะมี constructor แบบ default ที่คอมไพเลอร์ใส่ให้กับโปรแกรม ดังนี้

```
public class Student {
      public Student() {
      }
      ...
}
```

6.5.1 การเขียน Constructor

เราสามารถที่จะเขียน constructor ในคลาสใดๆ ขึ้นมาก็ได้อาทิเช่น คลาส Student อาจกำหนดให้มี constructor เป็น

```
public class Student {
      private String id;
      private String name;
      private double gpa;
      public Student(String ID,String n,double GPA) {
            id = ID;
            name = n;
            gpa = GPA;
      }
      ...
}
```

การกำหนด constructor ขึ้นมาใหม่นั้นจะทำให้ constructor แบบ default หายไป ดังนั้นการสร้างอ็อบเจกต์ของคลาสที่มี constructor ใหม่นี้จะต้องใช้คำสั่ง `new` ที่ส่งผ่านจำนวนและชนิดข้อมูลของ argument ที่สอดคล้องกับที่กำหนดไว้ใน constructor ตัวใหม่ จากตัวอย่างข้างต้น การสร้างอ็อบเจกต์ของคลาส `Student` อาจจะใช้คำสั่ง

```
Student s1 = new Student("1122","Thana",3.00);
```

แต่จะไม่สามารถใช้คำสั่ง

```
Student s1 = new Student();
```

ได้อีก เนื่องจาก constructor แบบ default ของคลาส `Student` ไม่มีแล้ว

6.5.2 ขั้นตอนการทำงานของคำสั่ง `new`

คำสั่ง `new` ที่ใช้ในการสร้างอ็อบเจกต์ของคลาสจะมีลำดับขั้นตอนการทำงานดังนี้

1. กำหนดเนื้อที่ในหน่วยความจำให้กับอ็อบเจกต์
2. กำหนดค่าเริ่มต้นให้กับคุณลักษณะของอ็อบเจกต์
3. กำหนดค่าของคุณลักษณะของอ็อบเจกต์ตามคำสั่งกำหนดค่าที่ประกาศไว้
4. เรียกใช้ constructor

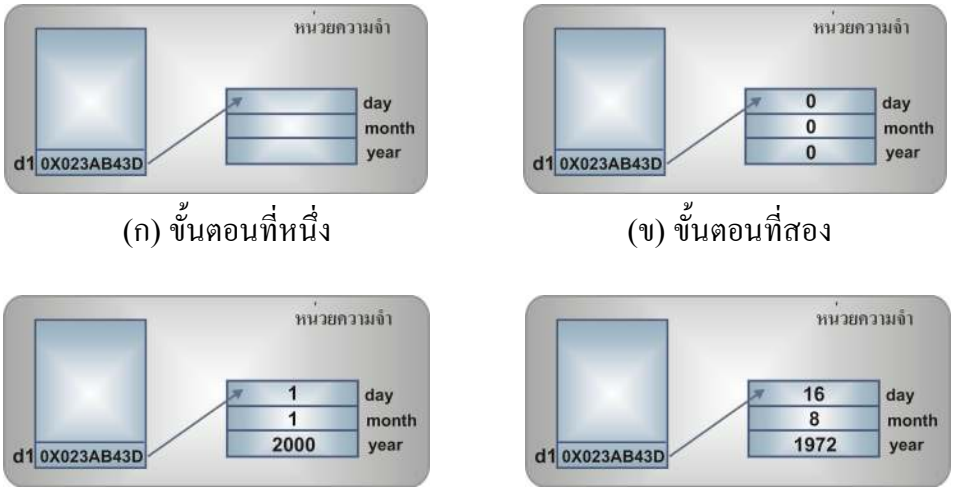
ตัวอย่างเช่น ถ้าคลาส `MyDate` มีโปรแกรมดังนี้

```
public class MyDate {
      private int day = 1;
      private int month = 1;
      private int year = 2000;
      public MyDate(int d,int m,int y) {
            day = d;
            month = m;
            year = y;
      }
}
```

คำสั่ง

```
MyDate d1 = new MyDate(16,8,1972);
```

จะมีขั้นตอนในการทำงานดังรูปที่ 6.4 โดยขั้นตอนที่หนึ่งจะเป็นการกำหนดเนื้อที่ในหน่วยความจำ (รูปที่ 6.4(ก)) ขั้นตอนที่สองเป็นการกำหนดค่าเริ่มต้นโดยคุณลักษณะของ อ็อบเจกต์จะมีค่า default เป็น 0 (รูปที่ 6.4(ข)) ขั้นตอนที่สามเป็นการกำหนดค่าของคุณลักษณะตามคำสั่งที่กำหนดค่าไว้เริ่มต้น โดยที่คุณลักษณะ `day` จะมีค่าเป็น 1 คุณลักษณะ `month` จะมีค่าเป็น 1 และคุณลักษณะ `year` จะมีค่าเป็น 2000 (รูปที่ 6.4(ค)) และขั้นตอนสุดท้ายจะเป็นการเรียกใช้ constructor ทำให้ค่าของคุณลักษณะเป็นไปตามที่กำหนดในคำสั่งของ constructor (รูปที่ 6.4(ง))



(ก) ขั้นตอนที่หนึ่ง (ข) ขั้นตอนที่สอง

ค) ขั้นตอนที่สาม (ง) ขั้นตอนที่สี่

รูปที่ 6.4 ขั้นตอนการทำงานของคำสั่ง `new`

6.5.3 Constructor แบบ Overloaded

ภาษาจาวาสามารถกำหนด constructor แบบ overloaded เช่นเดียวกับการกำหนดเมธอดแบบ overloaded โดย constructor แบบ overloaded จะมีจำนวนหรือชนิดข้อมูลของ argument ที่ต่างกัน ซึ่งคำสั่ง `new` ในการสร้างอ็อบเจกต์ของคลาสจะเรียกใช้ constructor โดยการพิจารณาจาก argument ที่ส่งผ่านมาว่าสอดคล้องกับ constructor ที่กำหนดตัวใด ข้อดีของการสร้าง constructor แบบ overloaded คือทำให้เราสามารถที่จะสร้างอ็อบเจกต์เริ่มต้นได้หลายรูปแบบ

โปรแกรมที่ 6.23 แสดงตัวอย่างของคลาส `Student` ที่มี constructor แบบ overloaded ซึ่งคำสั่งการสร้างอ็อบเจกต์

```
Student s1 = new Student ("1234","Somchai",3.75);
```

จะมีผลทำให้คอมไพเลอร์เรียก constructor ของคลาส `Student` ชุดที่สองซึ่งมี argument สามตัว

โปรแกรมที่ 6.23 ตัวอย่างการเขียน constructor แบบ overloaded

```
public class Student {
      private String id;
      private String name;
      private double gpa;
      public Student(String ID,String n) {
            id = ID;
            name = n;
      }
      public Student(String ID,String n,double GPA) {
            id = ID;
            name = n;
            gpa = GPA;
      }
}
```

6.5.4 เมธอด this()

เมธอดที่ชื่อ `this()` เป็นการเรียกใช้ constructor ของคลาสตัวเอง โดยจะต้องเป็นคำสั่งแรกสุดที่อยู่ใน constructor แบบ overloaded ตัวอย่างเช่น โปรแกรมที่ 6.23 สามารถเขียน constructor ของคลาส `Student` ชุดที่สองใหม่ได้โดยใช้เมธอดที่ชื่อ `this()` ดังแสดงในโปรแกรมที่ 6.24

โปรแกรมที่ 6.24 ตัวอย่างการใช้เมธอดที่ชื่อ `this()`

```
public class Student {
      private String id;
      private String name;
      private double gpa;
      public Student(String ID,String n) {
            id = ID;
            name = n;
      }
      public Student(String ID,String n,double GPA) {
            this(ID,n);
            gpa = GPA;
      }
}
```

6.5.5 เมธอด super()

โดยทั่วไปเมธอดและคุณลักษณะของ superclass จะสืบทอดมายัง subclass แต่จะมีข้อยกเว้นสำหรับ constructor ซึ่งจะไม่สืบทอดมายัง subclass กล่าวคือ ถ้า superclass มี constructor รูปแบบใด จะไม่ได้ทำให้ subclass มี constructor รูปแบบนั้นด้วย ตัวอย่างเช่น

```
public class Parent {
                    ...
}
public class Child extends Parent {
      public Child (String s) {
            ...
            }
}
```

ในกรณีนี้คลาส `Parent` มี constructor แบบ default ส่วนคลาส `Child` ที่สืบทอดมาจากคลาส `Parent` มี constructor ที่มี argument เป็น `String` เพียงรูปแบบเดียว เนื่องจาก constructor แบบ default จะไม่ได้สืบทอดมายังคลาส `Child` ดังนั้นการสร้าง อ็อบเจกต์โดยใช้ constructor แบบ default อาทิเช่น

```
Child c1 = new Child();
```

จึงเป็นคำสั่งที่ไม่ถูกต้อง เนื่องจาก constructor แบบ default ของคลาส `Child` ได้ถูกแทนที่ด้วย constructor แบบใหม่ไปแล้ว

เราสามารถที่จะเรียกใช้ constructor ของ superclass ได้โดยใช้เมธอดที่ชื่อ `super()` โดยส่งผ่าน argument ที่สอดคล้องกัน ทั้งนี้เมธอด `super()` จะต้องเป็นคำสั่งแรกของ constructor เช่นเดียวกับเมธอดที่ชื่อ `this()` โดยทั่วไปการทำงานของ constructor ของคลาสใดๆ จะมีผลทำให้มีการเรียกใช้ constructor ของ superclass นั้น ซึ่งถ้าไม่มีคำสั่ง `super()` อยู่ในคำสั่งแรกของ constructor ของคลาสนั้น ภาษาจาวาจะเรียกใช้ constructor แบบ default ของ superclass นั้นโดยอัตโนมัติ

ในกรณีที่ superclass ไม่มี constructor แบบ default และไม่มีคำสั่ง `super()` อยู่ใน constructor ของ subclass จะทำให้โปรแกรมไม่สามารถคอมไพล์ผ่านได้ดังตัวอย่างในโปรแกรมที่ 6.25 ซึ่งในกรณีนี้จะต้องใช้คำสั่ง `super()` ใน constructor ของคลาส `GradStudentV1` เพื่อที่จะทำให้โปรแกรมสามารถคอมไพล์ผ่านได้ โดยจะต้องกำหนด contructor ของคลาส `GradStudentV1` ใหม่ดังนี้

```
public GradStudentV1(String n){
      super(n);
      }
}
```

โปรแกรมที่ 6.25 ตัวอย่างการเรียกใช้ constructor ของ superclass

```
public class StudentV1 {
   protected String name;
   public StudentV1(String n) {
      name = n;
   }
}

-------------------------------------------------------
public class GradStudentV1 extends StudentV1 {
   public GradStudentV1(String n) {
      name = n;          /* compile error
                            should be super(n) */
   }
}
```

6.5.6 ขั้นตอนการทำงานของ Constructor

อ็อบเจกต์ของคลาสใดๆ จะถูกสร้างขึ้นเมื่อมีการเรียกใช้คำสั่ง `new` ตามขั้นตอนการทำงานที่ได้กล่าวผ่านมาแล้ว โดยขั้นตอนสุดท้ายเป็นการเรียกใช้ constructor ซึ่งจะมีขั้นตอนการทำงานที่ต้องพิจารณาเพิ่มเติมดังนี้

1. ถ้ามีคำสั่ง `this()` ใน constructor ก็จะเรียกใช้ constructor แบบ overloaded ที่สอดคล้องกับคำสั่ง `this()` แล้วข้ามไปขั้นตอนที่ 4
2. เรียกใช้ constructor ของ superclass ถ้าไม่มีคำสั่ง `super()` จะเรียกใช้ constructor แบบ default ยกเว้นคลาสที่ชื่อ `Object` จะไม่มีการเรียกใช้ constructor ของ superclass เนื่องจากคลาสที่ชื่อ `Object` จะไม่มี superclass
3. เรียกใช้คำสั่งกำหนดค่าเริ่มต้นของคุณลักษณะของอ็อบเจกต์
4. เรียกใช้คำสั่งภายใน constructor ของคลาสที่ใช้นั้น

โปรแกรมที่ 6.26 แสดงตัวอย่างคลาสที่มี constructor ในรูปแบบต่างๆ กรณีที่มีการสร้างอ็อบเจกต์ของคลาส `GradStudent` โดยใช้คำสั่ง

```
GradStudent s1 = new GradStudent();
```

จะมีผลทำให้ constructor ของคลาส `GradStudent` ทำงานดังนี้

1. เรียกใช้ constructor ของคลาส `GradStudent` ที่มีรูปแบบคำสั่งเป็น `GradStudent()`
2. เรียกใช้คำสั่ง `this(" ")`

2.1 เรียกใช้ constructor ของคลาส `GradStudent` ที่มีรูปแบบเป็น `GradStudent(String n)` โดยที่จะส่งข้อความ "" ให้กับ argument ที่ชื่อ `n`

2.2 เนื่องจากไม่มีคำสั่ง `this()` จึงข้ามขั้นตอนนี้ไป

2.3 เรียกใช้ constructor ของคลาส `Student` จากคำสั่ง `super(n)`

2.3.1 เรียกใช้ constructor ของคลาส `Student` ที่มีรูปแบบคำสั่งเป็น `Student(String n)`

2.3.2 เนื่องจากไม่มีคำสั่ง `this()` จึงข้ามขั้นตอนนี้ไป

2.3.3 เรียกใช้ constructor ของคลาสที่ชื่อ `Object` ที่เป็น constructor แบบ default

2.3.3.1 เรียกใช้ constructor แบบ default ของคลาสที่ชื่อ `Object`

2.3.3.2 เนื่องจากไม่มีคำสั่ง `this()` จึงข้ามขั้นตอนนี้ไป

2.3.3.3 เนื่องจากคลาสที่ชื่อ `Object` ไม่มี superclass จึงข้ามขั้นตอนของการเรียก constructor ของ superclass

2.3.3.4 ไม่มีคำสั่งกำหนดค่าเริ่มต้นของคุณลักษณะของคลาสที่ชื่อ `Object`

2.3.3.5 ไม่มีคำสั่งภายใน constructor ของคลาสที่ชื่อ `Object`

2.3.4 เรียกใช้คำสั่งกำหนดค่าเริ่มต้นของคุณลักษณะของอ็อบเจกต์ในคลาส `Student`

2.3.5 เรียกใช้คำสั่ง `name = n` ใน constructor ของคลาส `Student`

2.4 เรียกใช้คำสั่งกำหนดค่าเริ่มต้นของคุณลักษณะของอ็อบเจกต์ในคลาส `GradStudent`

2.5 ไม่มีคำสั่งอื่นๆ ภายใน constructor ของคลาส `GradStudent`

<u>โปรแกรมที่ 6.26</u> <u>ตัวอย่างคลาสที่มี constructor ในรูปแบบต่างๆ</u>

```
public class Student {
   protected String name;
   public Student(String n) {
      name = n;
   }
}

-----------------------------------------------------------------------------------------------------------
public class GradStudent extends Student {
   public GradStudent(String n) {
      super(n);
   }
   public GradStudent() {
      this(" ");
   }
}
```

6.6 เมธอดของคลาสที่ชื่อ Object

คลาสที่ชื่อ Object จะเป็นคลาสที่ทุกๆ คลาสจะสืบทอดมา คลาสที่ชื่อ Object มีเมธอดที่สำคัญหลายเมธอด อาทิเช่น toString() และ equals() ในการที่จะสามารถใช้งานเมธอดเหล่านี้ได้จะต้องมีการเขียนคำสั่งเพื่อกำหนดให้เป็นเมธอดแบบ overridden ในคลาสที่ต้องการ

6.6.1 เมธอด toString()

เมธอด toString() เป็นเมธอดที่ใช้ในการแปลงค่าของอ็อบเจกต์ให้เป็นข้อมูลชนิด String คลาสที่อยู่ใน Java API เช่นคลาส Date ได้กำหนดคำสั่งสำหรับเมธอด toString() ไว้แล้ว ดังนั้นการเรียกใช้เมธอด System.out.println() โดยที่ argument เป็นอ็อบเจกต์ของคลาส Date จะทำให้มีการเรียกใช้เมธอด toString() ของคลาส Date โดยอัตโนมัติ ตัวอย่างเช่นคำสั่ง

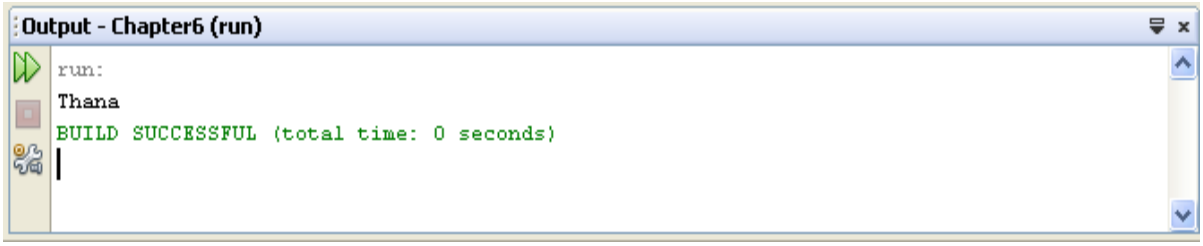
```
Date d = new Date();
System.out.println(d);
```

จะได้ผลลัพธ์เป็นข้อความที่ระบุวันเดือนปีของอ็อบเจกต์ d

โปรแกรมที่ 6.27 แสดงตัวอย่างของคลาส Student ที่มีเมธอดแบบ overridden ที่ชื่อ toString() เพื่อกำหนดข้อความที่ส่งกลับมาเมื่อมีการเรียกใช้เมธอดดังกล่าว คำสั่ง System.out.println(s1) จะเรียกใช้เมธอด toString() เพื่อส่งข้อความที่เป็นคุณลักษณะ name กลับมาแล้วแสดงผลลัพธ์ออกทางจอภาพดังแสดงในรูปที่ 6.5

โปรแกรมที่ 6.27 ตัวอย่างของคลาสที่มีเมธอดแบบ overridden ที่ชื่อ toString()

```
public class Student {
   private String name;
   public Student(String n) {
      name = n;
   }
   public String toString() {
      return name;
   }
}
-------------------------------------------------------
public class TestToString {
   public static void main(String args[]) {
      Student s1 = new Student("Thana");
      System.out.println(s1);
   }
}
```



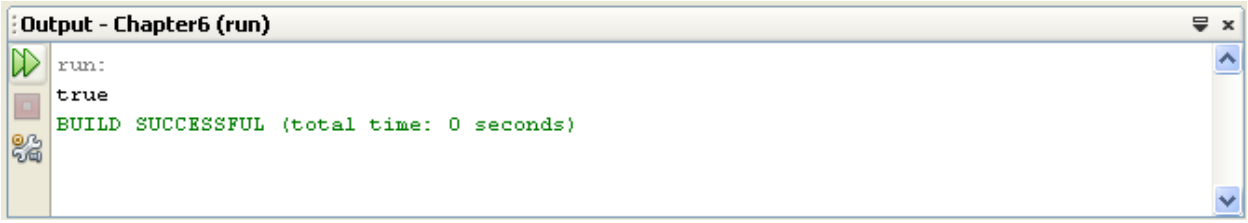
รูปที่ 6.5 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่6.27

6.6.2 เมธอด `equals()`

เมธอด `equals()` เป็นเมธอดในคลาสที่ชื่อ `Object` เพื่อเปรียบเทียบว่าอ็อบเจกต์นี้มีค่าเท่ากับอ็อบเจกต์ของที่ส่งผ่านมาทาง argument หรือไม่ คลาสที่จะสามารถเรียกใช้เมธอดนี้ได้จะต้องกำหนดเมธอดแบบ overridden สำหรับเมธอดนี้อยู่ในคลาสนั้นด้วย โปรแกรมที่ 6.28 แสดงตัวอย่างของคลาส `Student` ที่มีเมธอดแบบ overridden ที่ชื่อ `equals()` เพื่อกำหนดวิธีการเปรียบเทียบอ็อบเจกต์ โดยเมธอดนี้จะให้ค่าเป็น `true` ถ้าอ็อบเจกต์ที่ส่งผ่านมาทาง argument มีค่าของคุณลักษณะ `name` เท่ากับค่าของคุณลักษณะ `name` ของอ็อบเจกต์นี้ โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 6.6

โปรแกรมที่ 6.28 ตัวอย่างของคลาสที่มีเมธอดแบบ overridden ที่ชื่อ `equals()`

```
public class Student {
   private String name;
   public Student(String n) {
      name = n;
   }
   public boolean equals(Object obj) {
        if (obj instanceof Student) {
            if (((Student)obj).name == name) {
                return true;
            }
        }
        return false;
    }
}
-------------------------------------------------------
public class TestEquals {
   public static void main(String args[]) {
      Student s1 = new Student("Thana");
      Student s2 = new Student("Thana");
      System.out.println(s1.equals(s2));
   }
}
```



รูปที่ 6.6 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.28

6.7 คลาสประเภท Wrapper

คลาสประเภท Wrapper เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่มีคุณลักษณะสอดคล้องกับชนิดข้อมูลแบบพื้นฐานทั้งแปดชนิด โดยภาษาจาวาได้กำหนดคลาสที่อยู่ในแพคเก็จ `java.lang` ที่เป็นคลาสประเภท Wrapper ไว้แปดคลาส ดังแสดงในตารางที่ 6.2 คลาสประเภท Wrapper จะช่วยในการสร้างอ็อบเจกต์ที่เก็บชนิดข้อมูลแบบพื้นฐานไว้ในคอลเล็กชั่นแบบ Heterogeneous ที่จะกล่าวถึงในบทที่ 8

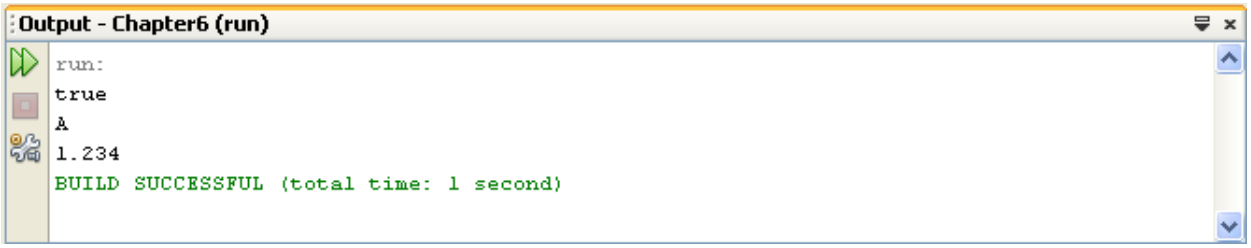
โปรแกรมที่ 6.29 แสดงตัวอย่างการใช้คลาสประเภท Wrapper ซึ่งในตัวอย่างนี้ได้สร้างอ็อบเจกต์ของคลาสประเภท Wrapper เพื่อเก็บชนิดข้อมูลแบบพื้นฐานสามค่า คือ `true`, `'A'` และ `1.234` และได้พิมพ์ค่าของอ็อบเจกต์ทั้งสามออกมา โดยจะได้ผลลัพธ์ดังแสดงในรูปที่ 6.7

ตารางที่ 6.2 คลาสประเภท Wrapper

ชนิดข้อมูล	ชื่อคลาส
boolean	Boolean
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character

โปรแกรมที่ 6.29 ตัวอย่างการใช้คลาสประเภท Wrapper

```
public class ShowWrapper {
   public static void main(String args[]) {
       Boolean b = new Boolean(true);
       Character c = new Character('A');
       Double d = new Double(1.234);
       System.out.println(b);
       System.out.println(c);
       System.out.println(d);
    }
}
```



รูปที่ 6.7 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.29

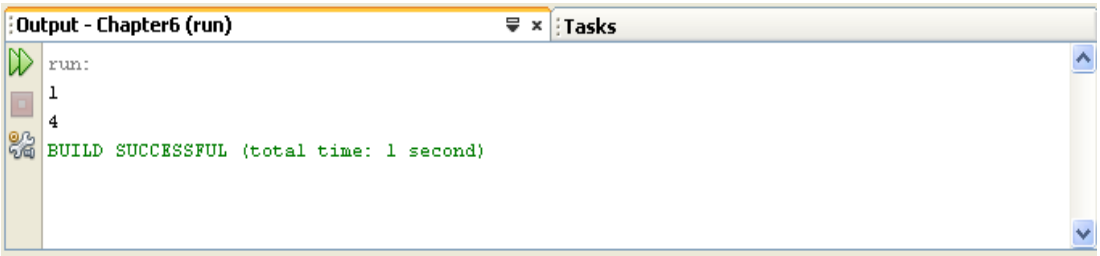
6.7.1 Autoboxing

ภาษาจาวาตั้งแต่เวอร์ชัน Java SE 5 ได้เพิ่มคำสั่ง Auto-boxing และ Auto-unboxing ซึ่งสามารถแปลงค่าระหว่างชนิดข้อมูลพื้นฐานกับ คลาสประเภท Wrapper ที่สอดคล้องกันได้โดยอัตโนมัติ คำสั่ง Autoboxing คือการแปลงจากชนิดข้อมูลพื้นฐานเช่น int ไปเป็นคลาสประเภท Wrapper เช่น Integer ส่วนคำสั่ง Auto-unboxing คือการแปลงจากคลาสประเภท Wrapper เช่น Byte ไปเป็นชนิดข้อมูลพื้นฐานเช่น byte

โปรแกรมที่ แสดงตัวอย่างการใช้คำสั่ง Auto-boxing ในการแปลงค่า int ที่มีค่าเป็น 1 ให้กับอ็อบเจกต์ของคลาส Integer ที่ชื่อ myInt และใช้คำสั่ง Auto-unboxing เพื่อแปลงค่ากลับมาในตัวแปรชนิดข้อมูลพื้นฐานที่ชื่อ i โดยจะได้ผลลัพธ์ดังรูปที่ 6.8

โปรแกรมที่ 6.30 ตัวอย่างการใช้คำสั่ง Auto-boxing/Auto-unboxing

```
public class ShowWrapper {
   public static void main(String []args) {
        Integer myInt = 1; // int into Integer
        System.out.println(myInt);

        int i = myInt + 3; // mix Integer and int
        System.out.println(i);

    }
}
```



รูปที่ 6.8 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.30

6.8 คีย์เวิร์ดอื่นๆ ที่สำคัญ

ภาษาจาวาได้กำหนดคีย์เวิร์ดที่เกี่ยวข้องกับคลาส เมธอด และคุณลักษณะที่สำคัญไว้สองตัวคือ `static` และ `final` คีย์เวิร์ด `static` เป็นคีย์เวิร์ดเพื่อระบุว่าตัวแปรหรือค่าคงที่ใดเป็นคุณลักษณะของคลาสหรือเมธอดใดเป็นเมธอดของคลาส ส่วนคีย์เวิร์ด `final` จะใช้ในคลาส และเมธอดเพื่อกำหนดไม่ให้มีการสืบทอด หรือใช้ในคุณลักษณะเพื่อกำหนดให้เป็นค่าคงที่

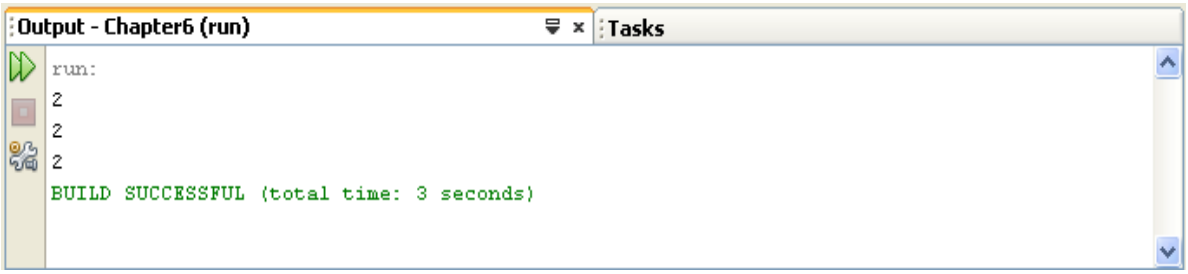
6.8.1 คุณลักษณะแบบ `static`

คุณลักษณะของคลาสจะเป็นคุณลักษณะที่ใช้ร่วมกันในทุกอ็อบเจกต์ และสามารถเรียกใช้งานได้โดยไม่จำเป็นต้องสร้างอ็อบเจกต์ของคลาสขึ้นมาก่อน ตัวแปรหรือค่าคงที่ที่เป็นคุณลักษณะของคลาสคือตัวแปรหรือค่าคงที่ซึ่งมี modifier เป็น `static` คุณลักษณะของคลาสนี้จะเป็นตัวแปรหรือค่าคงที่ซึ่งใช้ร่วมกันในทุกอ็อบเจกต์ โดยจะแตกต่างจากตัวแปรหรือค่าคงที่ที่เป็นคุณลักษณะของอ็อบเจกต์ ซึ่งอาจมีค่าแตกต่างกันได้ในแต่ละอ็อบเจกต์ โปรแกรมที่

6.31 แสดงตัวอย่างการใช้คีย์เวิร์ด `static` เพื่อประกาศตัวแปรที่เป็นคุณลักษณะของคลาส ในกรณีนี้ตัวแปร `counter` จะมีอยู่เพียงตัวเดียวและทุกอ็อบเจกต์ใช้ตัวแปรนี้ร่วมกัน ดังนั้นจึงสามารถที่จะเรียกใช้ได้โดยระบุชื่อคลาส (`Student.counter`) โปรแกรมนี้จะให้ผลลัพธ์ที่มีค่าเท่ากันดังแสดงในรูปที่ 6.9

<u>โปรแกรมที่ 6.31</u> <u>ตัวอย่างการใช้คีย์เวิร์ด `static`</u>

```
public class Student {
   static int counter;
   public Student() {
      counter++;
   }
}

-------------------------------------------------------------
public class TestStatic {
   public static void main(String args[]) {
      Student s1 = new Student();
      Student s2 = new Student();
      System.out.println(Student.counter);
      System.out.println(s1.counter);
      System.out.println(s2.counter);
   }
}
```



รูปที่ 6.9 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.31

6.8.2 เมธอดแบบ `static`

เมธอดแบบ `static` คือเมธอดที่มี modifier เป็น `static` โดยทั่วไปเราจะประกาศเมธอดใดๆ ให้เป็นเมธอดแบบ `static` ในกรณีที่เมธอดนั้นเป็นเมธอดที่ใช้งานทั่วไป (Utility Method) เราสามารถที่จะเรียกใช้เมธอดแบบ `static` ได้โดยเรียกผ่านชื่อคลาสเช่นเดียวกับคุณลักษณะของคลาส ซึ่งมีรูปแบบการใช้งานดังนี้

```
ClassName.methodName()
```

โดยที่

- `className` คือชื่อของคลาส
- `methodName` คือชื่อของเมธอดแบบ `static`

ตัวอย่างของเมธอดแบบ `static` คือเมธอดทุกเมธอดในคลาส `Math` ที่อยู่ใน Java API ทั้งนี้เนื่องจากเมธอดเหล่านี้จัดว่าเป็นเมธอดสำหรับใช้งานทั่วไปจึงกำหนดให้เป็นเมธอดแบบ `static` ทำให้เราสามารถที่จะเรียกใช้เมธอดเหล่านี้ได้โดยใช้ชื่อคลาส เช่นคำสั่ง

```
Math.sqrt(4);
```

เป็นคำสั่งคำนวณหาค่ารากที่สองของ 4 เป็นต้น

เมธอดที่ชื่อ `main()` เป็นอีกตัวอย่างหนึ่งของเมธอดแบบ `static` ทั้งนี้เนื่องจากภาษาจาวาไม่ต้องการให้ JVM ต้องสร้างอ็อบเจกต์ของคลาสที่มีเมธอด `main()` ขึ้นมาก่อนเพื่อรันเมธอด `main()` จึงได้กำหนดให้ `main()` เป็นเมธอดแบบ `static`

ภาษาจาวาได้กำหนดเงื่อนไขของเมธอดแบบ `static` ที่สำคัญไว้ดังนี้

- เมธอดแบบ `static` จะไม่สามารถเรียกคุณลักษณะหรือเมธอดที่ไม่ได้เป็นแบบ `static` (non-static) ได้
- เมธอดแบบ `static` จะไม่สามารถมีเมธอดแบบ overridden ได้

6.8.3 Static Initializer

Static Initializer คือบล็อกในคลาสใดๆ ที่อยู่นอกเมธอด และมีคีย์เวิร์ด `static` เพื่อนิยามให้เป็นบล็อกแบบ `static` โดยมีรูปแบบดังนี้

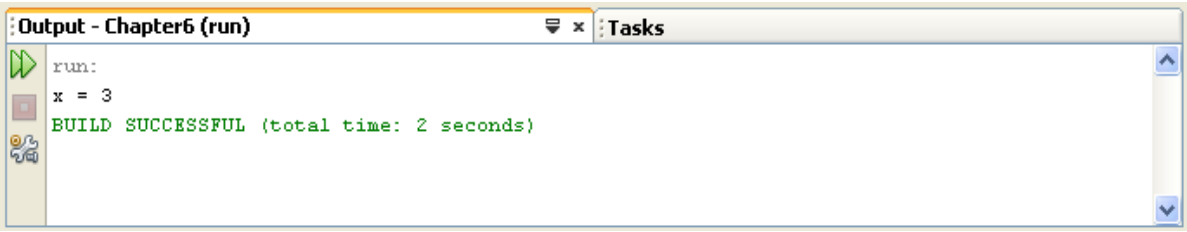
```
static {
    ...
}
```

โปรแกรมภาษาจาวาจะรันคำสั่งที่อยู่ในบล็อกแบบ `static` เพียงครั้งเดียวเมื่อ JVM โหลดคลาสที่มีบล็อกดังกล่าวขึ้นมา Static Initializer จะใช้ในการดีบัก (debug) โปรแกรมหรือใช้ในกรณีที่ต้องการสร้างอ็อบเจกต์ของคลาสขึ้นโดยอัตโนมัติ เราสามารถที่จะกำหนดบล็อกแบบ `static` ได้มากกว่าหนึ่งบล็อกในคลาสแต่ละคลาส โดยการทำงานของโปรแกรมจะเรียกคำสั่งในบล็อกแบบ `static` จากบนลงล่าง โปรแกรมที่ 6.32 แสดงตัวอย่างของคลาสที่มี

Static Initializer ซึ่งโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 6.10

โปรแกรมที่ 6.32 ตัวอย่างของคลาสที่มี Static Initializer

```
public class TestStaticBlock {
   static int x=5;
   static {
       x += 1;
   }
   public static void main(String args[]) {
       System.out.println("x = "+x);
   }
   static {
       x /= 2;
   }
}
```



รูปที่ 6.10 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.32

6.8.4 คีย์เวิร์ด `final`

คีย์เวิร์ด `final` สามารถที่จะใช้เป็น modifier สำหรับคลาส ตัวแปร และเมธอด โดยที่

- คลาสที่มี modifier เป็น `final` จะเป็นคลาสที่ไม่สามารถให้คลาสอื่นสืบทอดได้ตัวอย่างเช่น คลาส `String` ที่กำหนดใน Java API เป็นคลาสแบบ `final` ซึ่งมีรูปแบบคำสั่งประกาศคลาสดังนี้

```
public final class String {
      ...
}
```

ดังนั้นเราจึงไม่สามารถที่จะกำหนดให้คลาสใดๆ สืบทอดคลาส `String` ได้

- เมธอดที่มี modifier เป็น `final` คือเมธอดที่จะไม่สามารถมีเมธอดแบบ overridden ได้ ตัวอย่างเช่นถ้าคลาส `Student` กำหนดให้เมธอด `showDetails()` มี modifier เป็น `final` ดังนี้

```
public class Student{
        ...
    public final void showDetails(){
        ...
    }
}
```

ดังนั้นถ้าคลาส `GradStudent` สืบทอดมาจากคลาส `Student` จะไม่สามารถมีเมธอดแบบ overridden ที่ชื่อ `showDetails()` ได้

- ตัวแปรที่มี modifier เป็น `final` คือค่าคงที่ ค่าคงที่จะสามารถกำหนดค่าได้เพียงครั้งเดียว โดยจะต้องมีการกำหนดค่าก่อนที่จะมีการเรียกใช้งาน ซึ่งเมื่อกำหนดค่าแล้วจะไม่สามารถเปลี่ยนแปลงค่าได้ ถ้าค่าคงที่นั้นเป็นคุณลักษณะของอ็อบเจกต์ เราจะต้องกำหนดค่าในคำสั่งประกาศตัวแปร หรือใช้คำสั่งกำหนดค่าภายใน constructor ของคลาส แต่ถ้าเป็นค่าคงที่ซึ่งอยู่ภายในเมธอด เราจะต้องกำหนดค่าในคำสั่งประกาศตัวแปร หรือใช้คำสั่งกำหนดค่าภายในเมธอดก่อนที่จะมีการเรียกใช้งานค่าคงที่นั้น

6.9 คลาสภายใน

คลาสภายใน (Inner Class) คือคลาสที่ประกาศอยู่ภายในคลาสอื่นๆ ซึ่งบางครั้งเรียกว่าคลาสแบบซ้อน (Nested Class) คลาสภายในอนุญาตให้ประกาศคุณลักษณะหรือเมธอดภายในคลาสอื่นได้ คลาสภายในมีประโยชน์ในกรณีที่ต้องการจะจัดกลุ่มของคลาสที่ต้องทำงานร่วมกัน โดยต้องการควบคุมไม่ให้มีการเข้าถึงโดยตรงจากคลาสอื่นๆ และต้องการเรียกใช้คุณลักษณะหรือเมธอดของคลาสที่อยู่ภายนอกได้โดยตรง

ภาษาจาวาแบ่งคลาสภายในออกเป็นสองแบบคือ

1. คลาสภายในที่อยู่ภายในคลาสโดยตรง
2. คลาสภายในที่อยู่ภายในเมธอด

6.9.1 คลาสภายในที่อยู่ภายในคลาส

กรณีนี้เป็นการประกาศคลาสภายในคลาสอื่นที่เรียกว่า คลาสภายนอก (Outer class) โปรแกรมที่ 6.33 เป็นตัวอย่างของการประกาศคลาสภายในที่ชื่อ `Inner` ซึ่งถูกประกาศอยู่ภายในคลาสที่ชื่อ `Outer` คลาสภายในจะแตกต่างกับคลาสทั่วๆ ไปตรงที่เราสามารถที่จะประกาศให้คลาสภายในมี access modifier เป็น `public`, `private`, default หรือ `protected` ก็ได้ ขณะที่คลาสทั่วๆ ไปจะกำหนดให้เป็นได้เฉพาะ `public` หรือ default เท่านั้น นอกจากนี้การสร้างอ็อบเจกต์ของคลาสภายในยังมีข้อแตกต่างดังนี้

1. การสร้างอ็อบเจกต์ของคลาสภายในนอกคลาสภายนอก จะต้องทำการสร้างอ็อบเจกต์ของคลาสภายนอกก่อน แล้วจึงสร้างอ็อบเจกต์ของคลาสภายในได้ ตัวอย่างเช่นคำสั่ง

```
Outer.Inner in2 = new Outer().new Inner();
```

ในโปรแกรมที่ เป็นตัวอย่างการสร้างอ็อบเจกต์ของคลาสภายในที่ชื่อ `in2` (กรณีนี้จะไม่สามารถสร้างอ็อบเจกต์ของคลาสภายในได้ ถ้าคลาสภายในมี modifier เป็น `private`)

2. การสร้างอ็อบเจกต์ภายในเมธอดที่อยู่ในคลาสภายนอกสามารถทำได้โดยตรง เช่น เดียวกับการสร้างอ็อบเจกต์ทั่วๆ ไป ตัวอย่างเช่นถ้าเมธอด `methodX()` เป็นเมธอดในคลาสภายนอก และต้องการจะสร้างอ็อบเจกต์ของคลาสภายใน เราสามารถทำได้ดังนี้

```
public void methodX() {
     ...
     Inner in1 = new Inner();
}
```

คลาสที่อยู่ภายในสามารถที่จะเรียกใช้คุณลักษณะ และเมธอดของคลาสภายนอกได้ โปรแกรมที่ 6.33 แสดงตัวอย่างของการเรียกใช้และประกาศตัวแปรของคลาสภายในและภายนอก ตัวอย่างนี้จะมีตัวแปรชื่อเดียวกันซึ่งมีขอบเขตที่ต่างกัน

<u>โปรแกรมที่ 6.33</u> <u>ตัวอย่างแสดงคลาสภายในที่อยู่ภายในคลาส</u>

```
public class Outer {
   public void method1() {
       Inner in1 = new Inner();
       in1.method2();
   }
   public class Inner {
       public void method2() {
         System.out.println("Inner Demo");
       }
   }
}

-------------------------------------------------------
public class InnerDemo {
   public static void main(String args[]) {
       Outer.Inner in2 = new Outer().new Inner();
       in2.method2();
   }
}
```

6.9.2 คลาสภายในที่อยู่ภายในเมธอด

กรณีนี้เป็นการประกาศคลาสภายในที่อยู่ภายในเมธอดของคลาสภายนอกดังตัวอย่างที่แสดงในโปรแกรมที่ 6.34 คลาสภายในประเภทนี้จะมี access modifier เป็น default เราจะไม่สามารถที่จะสร้างอ็อบเจกต์ของคลาสภายในประเภทนี้นอกเมธอดที่ประกาศคลาสได้ และจะสามารถเรียกใช้ตัวแปรภายในของเมธอดได้ในกรณีที่ตัวแปรนั้นประกาศเป็น `final` เท่านั้น ส่วนตัวแปรที่เป็นคุณลักษณะของคลาสหรือคุณลักษณะของอ็อบเจกต์ สามารถที่จะเรียกใช้ได้เช่นเดียวกับคลาสภายในที่อยู่ภายในคลาส

คลาสภายในทั้งสองประเภทยังมีคุณสมบัติอื่นๆ ที่สำคัญดังนี้

- คลาสภายในอาจเป็นคลาสแบบ `abstract` หรืออินเตอร์เฟสได้
- คลาสภายในที่อยู่ภายในคลาสภายนอกโดยตรง ถ้ามี modifier เป็น `static` จะกลายเป็นคลาสปกติ และสามารถสร้างอ็อบเจกต์โดยการเรียกชื่อของคลาสภายนอกได้ดังนี้

```
Outer.Inner in3 = new Outer.Inner();
```

- คลาสภายในไม่สามารถที่ประกาศให้มีคุณลักษณะหรือเมธอดเป็นแบบ `static` ได้เว้นแต่คลาสภายในจะเป็นคลาสแบบ `static`
- คลาสภายในสามารถที่จะใช้ตัวแปรที่เป็นคุณลักษณะของคลาสหรือคุณลักษณะของอ็อบเจกต์ของคลาสภายนอกได้

โปรแกรมที่ 6.34 คลาสภายในที่อยู่ภายในเมธอด

```
public class MOuter {
   private int a = 1;
   public void method(final int b) {
       final int c = 2;
       int d = 3;
       class Inner {
         private void iMethod() {
            System.out.println("a = "+a);
            System.out.println("b = "+b);
            System.out.println("c = "+c);
            System.out.println("d = "+d);    //illegal
         }
       }
   }
}
```

6.10 Generic Types

เราสามารถที่จะประกาศคลาสหรืออินเตอร์เฟสใดๆให้มีชนิดข้อมูลเป็นแบบทั่วไป (Generic Type) แล้วค่อยระบุชนิดข้อมูลแบบเจาะจงตอนที่สร้างอ็อบเจกต์ได้ โดยการใช้ตัวแปรชนิดข้อมูล (type variable) ซึ่งมีรูปแบบการใช้งานดังนี้

```
public class ClassName(TypeVariable)
```

โดยที่

- `ClassName` คือชื่อของคลาส
- `TypeVariable` คือชื่อของตัวแปรชนิดข้อมูล

โปรแกรมที่ 6.35 แสดงตัวอย่างคลาสที่ชื่อ Box ที่มีคุณลักษณะที่ใช้ Generic Type โดยประกาศตัวแปรชนิดข้อมูลที่ชื่อ T ซึ่งจะเห็นได้ว่าคุณลักษณะที่ชื่อ t มีชนิดข้อมูลแบบ T ที่ยังไม่ได้ระบุว่าเป็นชนิดใดที่ชัดเจน เราสามารถที่จะระบุชนิดข้อมูลได้เมื่อมีการประกาศและสร้างอ็อบเจกต์ของคลาสนี้ อาทิเช่นคำสั่ง

```
Box<Integer> integerBox;
```

เป็นการประกาศอ็อบเจกต์ที่ชื่อ integerBox ของคลาส Box ที่มีชนิดข้อมูลเป็น Integer ซึ่งเราสามารถสร้างอ็อบเจกต์ของคลาสนี้โดยใช้คำสั่ง

```
integerBox = new Box<Integer>();
```

หรือสามารถที่จะประกาศและสร้างอ็อบเจกต์ในคำสั่งเดียวกันดังนี้

```
Box<Integer> integerBox = new Box<Integer>();
```

หรือหากที่ต้องการจะใช้ชนิดข้อมูลอื่น เราก็สามารถที่จะประกาศและสร้างอ็อบเจกต์โดยระบุชนิดข้อมูลที่ต้องการอาทิเช่น

```
Box<Byte> byteBox = new Box<Byte>();
```

เป็นการประกาศและสร้างอ็อบเจกต์ชื่อ byteBox ของคลาส Box ที่มีชนิดข้อมูลเป็น Byte

โปรแกรมที่ 6.35 ได้แสดงตัวอย่างของคลาส BoxDemo ที่แสดงการใช้งานคลาส Box

โปรแกรมที่ 6.35 ตัวอย่างแสดงคลาสที่ใช้ Generic Type

```
public class Box<T> {
    private T t;

    public void add(T t) {
        this.t = t;
    }

    public T get() {
        return t;
    }
}

------------------------------------------------------------
public class BoxDemo {

    public static void main(String[] args) {
        Box<Integer> integerBox = new Box<Integer>();
        integerBox.add(new Integer(10));
        Integer someInteger = integerBox.get(); // no cast!
        System.out.println(someInteger);
    }
}
```

6.10.1 เมธอดแบบ Generic

เราสามารถที่จะใช้ Generic Type กับเมธอดหรือ constructor ใดๆก็ได้ ในกรณีที่เราต้องการที่จะให้เมธอดหรือ constructor มี argument ที่รับชนิดข้อมูลใดๆก็ได้โดยมีรูปแบบดังนี้

```
[modifier] <TypeVariable> return_type methodName(TypeVaraible varName) {
    [method_body]
}
```

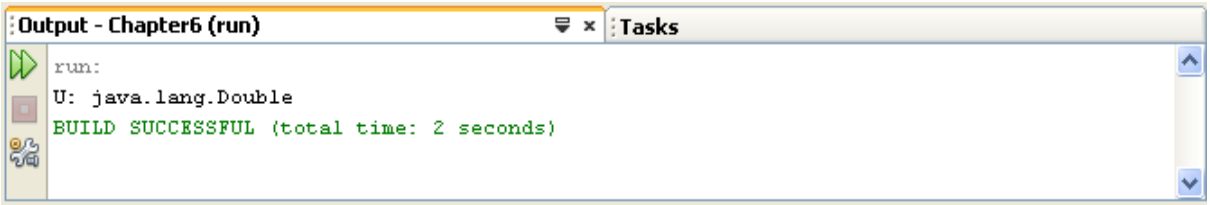
โดยที่

- `TypeVariable` คือชื่อของตัวแปรชนิดข้อมูล
- varName คือชื่อ argument

โปรแกรมที่ 6.36 แสดงตัวอย่างของคลาส BoxMethod ทีมีเมธอดที่ชื่อ inspect ทโดย argument ชื่อ u เป็น Generic Type แล้วคำสั่งของเมธอด main ในคลาส BoxMethodDemo มาเรียกใช้โดยการกำหนดให้เป็นชนิด Double (ค่าทศนิยม 3.2 ถูกแปลงให้เป็นคลาส Double โดยคำสั่ง auto-boxing) โดยจะได้ผลลัพธ์ของการรันโปรแกรมดังรูปที่ 6.11

โปรแกรมที่ 6.36 ตัวอย่างแสดงคลาสที่ประกาศเมธอดแบบ Generic

```
public class BoxMethod {

    public <U> void inspect(U u) {
        System.out.println("U: " + u.getClass().getName());
    }
}
-------------------------------------------------------
public class BoxMethodDemo {
  public static void main(String[] args) {
        BoxMethod box = new BoxMethod();
        box.inspect(3.2);
    }
}
```



รูปที่ 6.11 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 6.36

6.11 Annotation

ภาษาจาวามีคำสั่ง annotation (เครื่องหมาย @)) ซึ่งจะให้ข้อมูลเกี่ยวกับโปรแกรม โดยที่จะไม่มีผลโดยตรงต่อคำสั่งใดๆ ของ code ที่ทำการบันทึก (annotate) อยู่ เราสามารถที่จะกำหนดคำสั่ง annotation ในการประกาศคลาส คุณลักษณะ เมธอด หรือส่วนอื่นๆ ของโปรแกรม โดยทั่วไป annotation จะถูกใช้งานในกรณีต่างๆ ดังนี้

- เป็นข้อมูลสำหรับคอมไพเลอร์ เพื่อตรวจจับข้อผิดพลาดหรือ warning ของโปรแกรม
- ช่วยในกระบวนการขั้นตอนการคอมไพล์และติดตั้ง (deployment) ทั้งนี้ซอฟต์แวร์ในการคอมไพล์สามารถที่จะประมวลผลข้อมูล annotation ในการสร้างโค้ดที่เป็นไฟล์ XML ได้
- ช่วยในกระบวนการขั้นตอน runtime ทั้งนี้จาก annotation บางคำสั่งจะสามารถตวรสอบได้ในช่วง runtime

คำสั่ง annotation @ มักจะถูกประกาศเป็นคำสั่งแรก ตามด้วยชื่อของ annotation ซึ่งอาจมี element

ที่มี named หรือ unnamed valued ตัวอย่างเช่น

```
@Author(
   name = "Benjamin Franklin",
   date = "3/27/2003"
)
class MyClass() { }
```

คือ annotation ชื่อ author ที่มี element สองตัวที่เป็น named valued คือ name และ date

หรือคำสั่ง

```
@SuppressWarnings(value = "unchecked")

void myMethod() { }
```

คือ annotation ที่ชื่อ SuppressWarnings ที่อยู่หน้าเมธอดที่มี element เป็น named value ชื่อ values ซึ่งในกรณีนี้เนื่องจากมี element เดียว ดังนั้นเราอาจไม่จำเป็นต้องกำหนดชื่อ (unnamed value) ดังนี้

```
@SuppressWarnings("unchecked")

void myMethod() { }
```

ในกรณีที่ไม่มี element ใดเลย เราไม่จำเป็นต้องใส่เครื่องหมายวงเล็บหลัง annotation ดังนี้

```
@Override

void mySuperMethod() { }
```

6.11.1 Annotation Document

เราสามารถที่จะนำคำสั่ง annotation มาใช้ในการที่จะเขียน document ของ source code อาทิเช่น แทนที่จะต้องเขียน comment ของคลาสต่อไปนี้ในลักษณะดังนี้

```
public class Generation3List extends Generation2List {

   // Author: John Doe
   // Date: 3/17/2002
   // Current revision: 6
   // Last modified: 4/12/2004
   // By: Jane Doe
   // Reviewers: Alice, Bill, Cindy
```

```
    // class code goes here
}
```

เราสามารถที่จะกำหนด annotation type ของอินเตอร์เฟสขึ้นมาหนึ่งตัวดังนี้

```
@interface ClassPreamble {
   String author();
   String date();
   int currentRevision() default 1;
   String lastModified() default "N/A";
   String lastModifiedBy() default "N/A";
   String[] reviewers();  // Note use of array
}
```

จากนั้นสร้างคลาสโดยกำหนดค่าของ annotation ดังตัวอย่างนี้

```
@ClassPreamble (
   author = "John Doe",
   date = "3/17/2002",
   currentRevision = 6,
   lastModified = "4/12/2004",
   lastModifiedBy = "Jane Doe",
   reviewers = {"Alice", "Bob", "Cindy"} // Note array notation
)
public class Generation3List extends Generation2List {

// class code goes here

}
```

6.11.2 Annotation ที่ใช้โดยคอมไพเลอร์

ภาษาจาวาได้กำหนด annotation สำหรับคอมไพเลอร์ไว้สามคำสั่งคือ `@Depreacted`, `@Override` และ `@Suppresswarnings` โดยคำสั่ง annotation เหล่านี้มีการทำงานดังนี้

- `@Deprecated` เราจะใช้ annotation นี้ ในกรณีที่ต้องการระบุว่าคำสั่งเช่นคลาส เมธอด หรือคุณลักษณะใด เป็นคำสั่งที่ไม่ควรจะใช้แล้ว เพื่อให้คอมไพเลอร์ส่งคำสั่ง warning ออกมาในช่วงการคอมไพล์โปรแกรม ตัวอย่างเช่น

```
// Javadoc comment follows
/**
 * @deprecated
 * explanation of why it was deprecated
 */
```

```
@Deprecated
static void deprecatedMethod() { }
```

- @Override เราจะใช้ annotation นี้เพื่อจะบอกคอมไพเลอร์ว่า element นี้ override คำสั่ง element ของ superclass ตัวอย่างเช่น

```
// mark method as a superclass method
// that has been overridden
@Override
int overriddenMethod() { }
```

- @SuppressWarnings เราจะใช้ annotation นี้เพื่อบอกคอมไพเลอร์ไม่ต้องแสดงคำสั่ง warning ที่อาจปรากฏขึ้นจากการใช้คำสั่งที่ตามมานี้ในช่วงการคอมไพล์โปรแกรม ตัวอย่างเช่น

```
// use a deprecated method and tell
// compiler not to generate a warning
@SuppressWarnings("deprecation")
void useDeprecatedMethod() {
     objectOne.deprecatedMethod(); //deprecation warning-suppressed
}
```

สรุปเนื้อหาของบท

- เมธอดที่กำหนดขึ้นในคลาสใดๆ สามารถเรียกใช้งานได้สองรูปแบบคือ การเรียก ใช้งานจากคลาสที่ต่างกัน และการเรียกใช้งานภายในคลาสเดียวกัน
- เมธอดที่กำหนดขึ้นในคลาสอาจมี argument ที่รับค่าเพื่อนำไปใช้ในเมธอด
- เมธอดใดๆ ของคลาสสามารถที่จะมีค่าที่ส่งกลับมาได้ ทั้งนี้การประกาศเมธอดจะต้องระบุชนิดข้อมูลของค่าที่จะส่งกลับ
- เมธอดสามารถแบ่งเป็นเมธอดของอ็อบเจกต์ (หรือเรียกว่าเมธอดแบบ non-static) และเมธอดของคลาส (หรือเรียกว่าเมธอดแบบ static)
- เมธอดของคลาสใดที่มี modifier เป็นแบบ static จะทำให้สามารถถูกเรียกใช้งาน โดยใช้ชื่อของคลาสนั้นได้เลย
- เมธอดแบบ overloaded คือ มีเมธอดที่มีชื่อเดียวกันแต่จะมีจำนวนหรือชนิดข้อมูลของ argument ที่ต่างกัน
- โดยทั่วไป คุณลักษณะของคลาสจะมี modifier เป็นแบบ private ทั้งนี้เพื่อป้องกันการเข้าถึงจากคลาส

อื่น

- โดยทั่วไป เมธอดของคลาสจะมี modifier เป็นแบบ `public` ทั้งนี้เพื่อให้เมธอดจากคลาสอื่นเรียกใช้ได้
- ชื่อของเมธอดที่เกี่ยวข้องกับคุณลักษณะของคลาส นิยมใช้ `setXxx` สำหรับ เมธอดที่มีไว้เพื่อกำหนดค่าให้กับคุณลักษณะ xxx และใช้ `getXxx` สำหรับ เมธอดที่จะส่งค่ากลับเป็นคุณลักษณะ `xxx`
- คีย์เวิร์ด `protected` จะทำให้คลาสที่เป็น subclass สามารถที่จะอ้างอิงถึงคุณลักษณะหรือเมธอดของ superclass ได้
- คลาสทุกคลาสในภาษาจาวาถ้าไม่ได้สืบทอดจากคลาสใดเลย จะถือว่าคลาสนั้นสืบทอดจากคลาสที่ชื่อ `Object`
- `super` เป็นคีย์เวิร์ดที่ใช้ในการอ้างอิงถึง superclass เพื่อที่จะเรียกใช้เมธอดหรือ constructor ของ superclass
- การมีได้หลายรูปแบบหมายถึง
  - การที่เราสามารถที่จะสร้างหรืออ้างถึงอ็อบเจกต์ของคลาสที่สืบทอดกันได้หลายรูปแบบ
  - การที่เราสามารถที่จะอ้างถึงเมธอดชื่อเดียวกันได้หลายรูปแบบ
- เมธอดแบบ overloaded หมายถึงเมธอดที่มีชื่อเดียวกันมากกว่าหนึ่งเมธอด โดยมีจำนวนหรือชนิดของ argument ที่แตกต่างกัน
- เมธอดแบบ overridden หมายถึงการที่ subclass สร้างเมธอดที่มีอยู่แล้วใน superclass ขึ้นใหม่ โดยใช้ชื่อ argument และชนิดข้อมูลของค่าที่ส่งกลับของเมธอดเหมือนเดิม
- คีย์เวิร์ด `instanceof` เป็นตัวดำเนินการที่ใช้กับอ็อบเจกต์และคลาส เพื่อตรวจสอบว่าอ็อบเจกต์ใดๆ เป็นอ็อบเจกต์ของคลาสที่ระบุหรือไม่
- การ casting อ็อบเจกต์ ทำให้เราสามารถจะกำหนดอ็อบเจกต์ของคลาสที่เป็น superclass ให้กับตัวแปรที่กำหนดเป็นคลาสที่เป็น subclass ได้
- Constructor หมายถึง เมธอดที่มีชื่อเดียวกันกับชื่อคลาส แต่จะไม่มีการส่งค่ากลับและจะไม่มีการใส่คีย์เวิร์ด `void` โดยคลาสทุกคลาสจะมี constructor แบบ default มาให้อยู่แล้ว แต่เราสามารถที่จะกำหนด constructor ขึ้นใหม่เองได้
- Constructor แบบ overloaded หมายถึง constructor ที่มีจำนวนหรือชนิดของ argument ที่แตกต่างกัน
- เมธอดที่ชื่อ `this()` เป็นการเรียกใช้ constructor ของคลาสตัวเอง โดยจะต้องเป็นคำสั่งแรกสุดที่อยู่ใน constructor แบบ overloaded
- Constructor แบบ default ของคลาสที่เป็น superclass จะถูกเรียกใช้โดยอัตโนมัติทุกครั้งที่มีการสร้างอ็อบเจกต์ของคลาสที่เป็น subclass แต่เราสามารถเปลี่ยนให้ไปเรียกใช้ constructor ตัวอื่นของ superclass แทนได้โดยใช้เมธอด `super()`

- คลาสที่ชื่อ `Object` มีเมธอดที่สำคัญอยู่สองเมธอดคือเมธอด `toString()` และเมธอด `equals()`
- เมธอด `toString()` ใช้ในการแปลงค่าของอ็อบเจกต์ให้เป็นข้อมูลชนิด `String` ส่วนเมธอด `equals()` ใช้ในการเปรียบเทียบค่าของอ็อบเจกต์
- Auto-boxing / Auto-unboxing คือการแปลงค่าระหว่างชนิกข้อมูลพื้นฐานกับคลาสประเภท Wrapper ที่สอดคล้องกันโดยอัตโนมัติ
- โดยปกติเราจะประกาศเมธอดใดๆ ให้เป็นเมธอดแบบ `static` ในกรณีที่เมธอดนั้นเป็นเมธอดที่ใช้งานทั่วไป และเมธอดแบบ `static` สามารถถูกเรียกใช้จากชื่อของคลาสได้โดยไม่จำเป็นต้องสร้างอ็อบเจกต์
- คลาสที่มี modifier เป็น `final` จะทำให้คลาสอื่นไม่สามารถสืบทอดคลาสนี้ได้ และเมธอดที่มี modifier เป็น `final` คือเมธอดที่จะไม่สามารถมีเมธอดแบบ overridden ได้ ส่วนตัวแปรที่มี modifier เป็น `final` คือค่าคงที่ ซึ่งจะทำให้สามารถกำหนดค่าได้เพียงครั้งเดียวเท่านั้น
- คลาสภายในหมายถึงคลาสที่อยู่ภายในคลาส หรือคลาสที่อยู่ภายในเมธอด
- `Generic` Type สามารถที่จะทำให้เราประกาศคลาสหรืออินเตอร์เฟสใดๆให้มีชนิดข้อมูลเป็นแบบทั่วไป (Generic Type) แล้วค่อยระบุชนิดข้อมูลแบบเจาะจงตอนที่สร้างอ็อบเจกต์ได้
- เมธอดแบบ Generic คืดเมธอดที่มี argument ที่เป็น Generic Type
- `Annotation` จะให้ข้อมูลเกี่ยวกับโปรแกรม โดยที่จะไม่มีผลโดยตรงต่อคำสั่งใดๆของ code ที่ทำการบันทึก

บทที่ 7 การจัดการกับเหตุการณ์กราฟิก

เนื้อหาในบทนี้เป็นการแนะนำวิธีการเขียนโปรแกรม เพื่อจัดการกับเหตุการณ์กราฟิกในโปรแกรม GUI โดยจะเริ่มต้นจากการแนะนำนิยามความหมายของเหตุการณ์ แนะนำคลาสประเภท `Event` ที่เกี่ยวข้องกับเหตุการณ์กราฟิกต่างๆ แนะนำอินเตอร์เฟสประเภท `Listener` ที่ใช้ในการรับฟังเหตุการณ์ อธิบายวิธีการจัดการกับเหตุการณ์กราฟิกหลายๆ เหตุการณ์ และตอนท้ายของบทเป็นการแนะนำคลาสประเภท Event Adapter

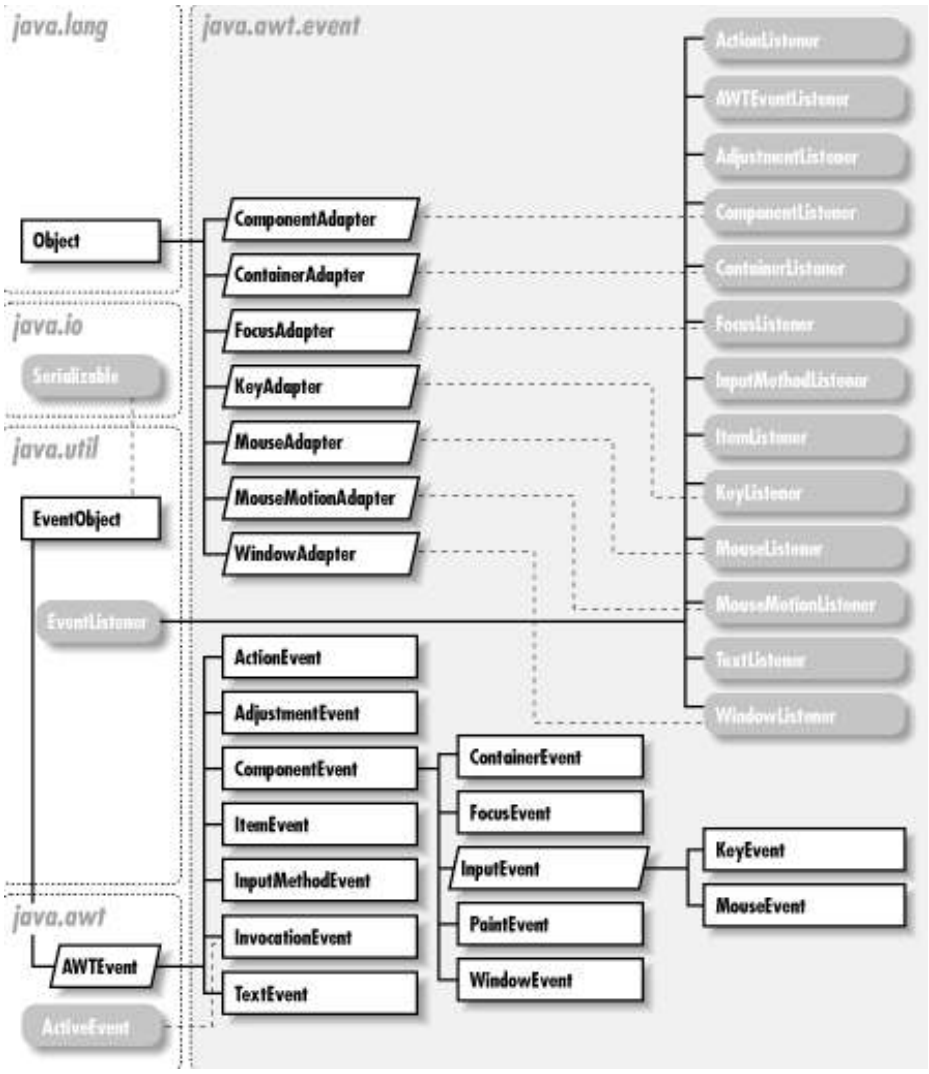
7.1 เหตุการณ์

เหตุการณ์ (Event) เป็นสถานการณ์ที่เกิดขึ้นในขณะรันโปรแกรม อาทิเช่น การใช้อินพุต (เมาส์หรือคีย์บอร์ด) ติดต่อกับโปรแกรม GUI การเกิดเหตุการณ์ในโปรแกรมภาษาจาวาจะเป็นการสร้างอ็อบเจกต์ของคลาสประเภท `Event` ชนิดต่างๆ ขึ้นมาตามประเภทของเหตุการณ์ อาทิเช่น

- เมื่อเลื่อนเมาส์ในเฟรมจะเกิดอ็อบเจกต์ของคลาส `MouseEvent` ขึ้นมา
- เมื่อปิดเฟรมจะเกิดอ็อบเจกต์ของคลาส `WindowEvent` ขึ้นมา
- เมื่อกดปุ่มที่อยู่ในเฟรมจะเกิดอ็อบเจกต์ของคลาส `ActionEvent` ขึ้นมา
- เมื่อพิมพ์ข้อความใน `TextField` จะเกิดอ็อบเจกต์ของคลาส `KeyEvent` ขึ้นมา

คลาสประเภท `Event` จะสืบทอดมาจากคลาส `ObjectEvent` ส่วนคลาสประเภท `Event` สำหรับเหตุการณ์ทางกราฟิกจะมีอยู่สองกลุ่มคือ คลาสในแพคเก็จ `java.awt.event` และ `javax.swing.event` โดยคลาสประเภท Event ที่ใช้โดยพื้นฐานคือคลาสในแพคเก็จ `java.awt.event` ซึ่งส่วนประกอบทางกราฟิกทั้งในแพคเก็จ AWT และ Swing ต่างก็สามารถที่จะใช้คลาสเหล่านั้นได้ รูปที่ 7.1 แสดงคลาสประเภท `Event` และอินเตอร์เฟสต่างๆ ที่อยู่ในแพคเก็จ `java.awt.event`

Java SE ในเวอร์ชัน 1.4 ได้มีการเพิ่มคลาสประเภท Event และอินเตอร์เฟสต่างๆ ที่เกี่ยวข้องสำหรับใช้ในอ็อบเจกต์ของคลาสประเภท Swing มากขึ้น โดยคลาสและอินเตอร์เฟสเหล่านี้จะอยู่ในแพคเก็จ `javax.swing.event` รูปที่ 7.3 แสดงตัวอย่างของคลาสประเภท Event ที่อยู่ในแพคเก็จดังกล่าว แต่เนื้อหาในบทนี้จะเน้นเฉพาะการใช้งานของคลาสในแพคเก็จ `java.awt.event` ที่มีคำสั่งส่วนใหญ่สำหรับใช้ในอ็อบเจกต์ของคลาสประเภท Swing ได้

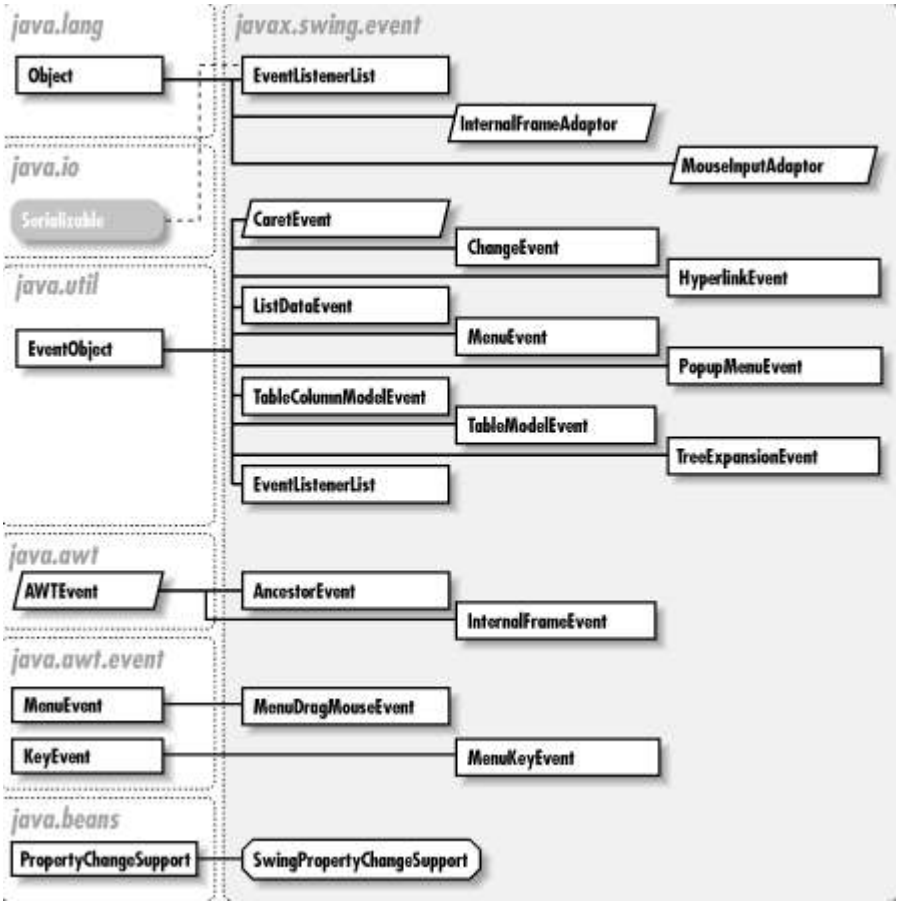


รูปที่ 7.1 คลาสและอินเตอร์เฟสของแพคเก็จ `java.awt.Event`

โดยทั่วไปองค์ประกอบของเหตุการณ์จะมีสามส่วนดังแสดงในรูปที่ 7.2 คือ

1. Event คืออ็อบเจกต์ประเภท `Event` ตามชนิดของเหตุการณ์ที่เกิดขึ้นเช่น อ็อบเจกต์ของคลาส `WindowEvent`
2. Event Source คืออ็อบเจกต์ที่ทำให้เกิดเหตุการณ์เช่น อ็อบเจกต์ของคลาส `JFrame` ที่เป็นส่วนที่ทำให้เกิดเหตุการณ์การปิดเฟรม
3. Event Handler คืออ็อบเจกต์ที่จะทำหน้าที่จัดการกับเหตุการณ์ที่เกิดขึ้น โดยมีเมธอดที่จะรับอ็อบเจกต์ประเภท `Event` ที่เกิดขึ้นและมีคำสั่งในการจัดการกับเหตุการณ์เพื่อโต้ตอบกับผู้ใช้

รูปที่ 7.2 องค์ประกอบของเหตุการณ์



รูปที่ 7.3 คลาสประเภท Event ในแพคเก็จ `javax.swing.event`

7.2 AWTEvent

คลาส `AWTEvent` เป็น superclass ของคลาสประเภท `Event` สำหรับเหตุการณ์ทางด้านกราฟิกซึ่งจะมีอยู่ทั้งหมด 14 คลาสคือ `ActionEvent, AdjustmentEvent, ComponentEvent, ItemEvent, InputMethodEvent, InvocationEvent, TextEvent, FocusEvent, WindowEvent, InputEvent, ContainerEvent, PaintEvent, KeyEvent` และ `MouseEvent` ดังแสดงในรูปที่ เนื้อหาในบทนี้จะกล่าวถึงเมธอดของคลาสประเภท `AWTEvent` ที่สำคัญเท่านั้น ซึ่งคลาส `AWTEvent` และคลาส `ObjectEvent` มีเมธอดที่สำคัญดังนี้

- `Object getSource()`

  เป็นเมธอดในคลาส `ObjectEvent` เพื่อเรียกดูอ็อบเจกต์ประเภท Event Source

- `int geID()`

  เป็นเมธอดในคลาส `AWTEvent` ที่ส่งค่าจำนวนเต็มเพื่อระบุชนิดของเหตุการณ์ เมธอดนี้มีประโยชน์ สำหรับเหตุการณ์ของคลาส `MouseEvent` ซึ่งมีเหตุการณ์ได้หลายชนิด โดยมีค่าจำนวนเต็มระบุชนิดของเหตุการณ์ อาทิเช่น `MOUSE_CLICKED` หรือ `MOUSE_DRAGGED` เป็นต้น

7.2.1 ActionEvent

อ็อบเจกต์ของคลาส `ActionEvent` จะถูกสร้างขึ้นในกรณีที่มีเหตุการณ์เกิดขึ้นในโปรแกรม GUI อาทิเช่น

- เมื่อมีการคลิกเมาส์บนปุ่ม (อ็อบเจกต์ของคลาส `Button`)
- เมื่อมีการป้อนคีย์ Enter ใน `TextField`
- เมื่อมีการเลือกรายการในเมนูของ `MenuItem`
- เมื่อมีการกดดับเบิ้ลคลิก (double click) ใน `List`

คลาส `ActionEvent` มีเมธอดที่เกี่ยวข้องดังนี้

- `String getActionCommand()`

  เป็นเมธอดที่จะส่งชื่อคำสั่งที่เกิดขึ้นจาก `ActionEvent` ทั้งนี้คำสั่งของ `ActionEvent` จะกำหนดขึ้น โดยอ็อบเจกต์ของ Event Source ในกรณีที่เป็นอ็อบเจกต์ของคลาส `Button` หรือ `MenuItem` ก็จะคือ ข้อความ (label) ที่ปรากฏบนปุ่มหรือเมนู นอกจากนี้เราสามารถที่จะกำหนดชื่อคำสั่งของอ็อบเจกต์ของคลาส `Button` หรือ `MenuItem` ให้แตกต่างจากข้อความที่ปรากฏได้โดยใช้เมธอด `setActionCommand()`

- `int getModifiers()`

  เป็นเมธอดที่จะส่งสถานะของคีย์ Modifier (คีย์ Alt, Ctrl, Meta และ Shift) ที่เกิดจากอ็อบเจกต์ของคลาส `ActionEvent` เมธอดนี้จะส่งชนิดข้อมูลแบบ `int` ที่มีค่าคงที่คืนมาคือ `ALT_MASK, CTRL_MASK, META_MASK` และ `SHIFT_MASK`

7.2.2 WindowEvent

อ็อบเจกต์ของคลาส `WindowEvent` จะถูกสร้างขึ้นในกรณีที่มีเหตุการณ์เกิดขึ้นเป็นอ็อบเจกต์ของคลาสประเภท `Window` โดยมีเหตุการณ์ที่เกิดขึ้นได้ดังนี้

- opened เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการเปิด `Window`
- closed เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการปิด `Window`
- closing เป็นเหตุการณ์ที่เกิดขึ้นขณะกำลังปิด `Window`
- iconified เป็นเหตุการณ์เมื่ออ็อบเจกต์ของคลาส `Window` อยู่ในรูปของไอคอน
- deiconified เป็นเหตุการณ์เมื่ออ็อบเจกต์ของคลาส `Window` ไม่ได้อยู่ในรูปของไอคอน
- activated เป็นเหตุการณ์ที่เกิดขึ้นเมื่อ `Window` กำลังทำงานอยู่
- deactivated เป็นเหตุการณ์ที่เกิดขึ้นเมื่อ `Window` ไม่ได้ทำงานอยู่

คลาส `WindowEvent` มีเมธอดที่สำคัญคือ

- `Object getWindow()`

  เป็นเมธอดที่ส่งอ็อบเจกต์ของคลาสประเภท `Window` ที่เป็น Event Source คืนมา

7.2.3 MouseEvent

อ็อบเจกต์ของคลาส `MouseEvent` จะถูกสร้างขึ้นในกรณีที่มีการใช้งานเมาส์เพื่อติดต่อกับผู้ใช้โดยมีเหตุการณ์ที่เกิดขึ้นได้ดังนี้

- dragged เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการเลื่อนเมาส์แล้วกดปุ่มของเมาส์พร้อมกัน
- moved เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการเลื่อนเมาส์
- clicked เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการคลิกเมาส์
- entered เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการเลื่อนเมาส์เข้าไปในขอบเขตของอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกใดๆ
- exited เป็นเหตุการณ์ที่เกิดขึ้นเมื่อตำแหน่งของเมาส์อยู่นอกขอบเขตของอ็อบเจกต์ของคลาสที่เป็นส่วนประกอบกราฟิกใดๆ
- pressed เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการกดปุ่มบนเมาส์
- released เป็นเหตุการณ์ที่เกิดขึ้นเมื่อมีการปล่อยปุ่มที่กดบนเมาส์

คลาส `MouseEvent` มีเมธอดที่สำคัญคือ

- `int getX()`
  เป็นเมธอดที่จะส่งตำแหน่งพิกัดของเมาส์บนแกน `x` ที่มีชนิดข้อมูลเป็น `int` คืนมา

- `int getY()`
  เป็นเมธอดที่จะส่งตำแหน่งพิกัดของเมาส์บนแกน `y` ที่มีชนิดข้อมูลเป็น `int` คืนมา

- `Point getPoint()`
  เป็นเมธอดที่จะส่งตำแหน่งพิกัด `(x,y)` ของเมาส์คืนมา โดยมีชนิดข้อมูลเป็นอ็อบเจกต์ของคลาส `Point`

- `int getClickCount()`
  เป็นเมธอดที่จะส่งจำนวนครั้งของการคลิกเมาส์คืนมา

7.2.4 ItemEvent

อ็อบเจกต์ของคลาส `ItemEvent` จะถูกสร้างขึ้นในกรณีที่มีเหตุการณ์เกิดขึ้นในโปรแกรม GUI ดังนี้

- เมื่อมีการเลือกหรือยกเลิกรายการใน `List` หรือ `Checkbox`
- เมื่อมีการคลิกเมาส์ในรายการใน `Choice`

คลาส `ItemEvent` มีเมธอดที่สำคัญดังนี้

- `ItemSelectable getItemSelectable()`
  เป็นเมธอดที่จะส่งอ็อบเจกต์ของคลาสประเภท `ItemSelectable` ที่เป็น Event Source คืนมา

- `Object getItem()`
  เป็นเมธอดที่จะส่งอ็อบเจกต์ของรายการที่ถูกเลือกคืนมา

- `int getStateChange()`
  เป็นเมธอดที่จะส่งค่าคงที่ชนิด `int` ที่มีค่าเป็น `SELECTED` หรือ `DESELECTED` เพื่อระบุสถานการณ์เลือกของรายการคืนมา

7.2.5 Event อื่นๆ

แพคเก็จ `java.awt.event` ยังมีคลาสที่เป็นเหตุการณ์ทางด้านกราฟิกอื่นๆ ที่สำคัญอาทิเช่น

- `KeyEvent` เป็นคลาสที่มีการสร้างอ็อบเจกต์เมื่อมีเหตุการณ์การกดคีย์บอร์ด
- `FocusEvent` เป็นคลาสที่มีการสร้างอ็อบเจกต์เมื่อผู้ใช้เลื่อนอุปกรณ์อินพุตมาชี้ยังอ็อบเจกต์ของส่วนประกอบกราฟิกใดๆ
- `ComponentEvent` เป็นคลาสที่มีการสร้างอ็อบเจกต์เมื่อมีเหตุการณ์ซึ่งอ็อบเจกต์ของส่วนประกอบกราฟิกมีการเปลี่ยนแปลงเช่น เคลื่อนที่หรือปรับขนาด
- `ContainerEvent` เป็นคลาสที่มีการสร้างอ็อบเจกต์เมื่อมีเหตุการณ์ในการใส่หรือยกเลิกอ็อบเจกต์ของส่วนประกอบกราฟิก ลงในอ็อบเจกต์ของคลาสประเภท `Container`
- `AdjustmentEvent` เป็นคลาสที่มีการสร้างอ็อบเจกต์เมื่อมีเหตุการณ์ในการปรับตำแหน่งชี้ของอ็อบเจกต์ของคลาส `ScrollBar` หรือ `ScrollPane`
- `TextEvent` เป็นคลาสที่มีการสร้างอ็อบเจกต์เมื่อมีเหตุการณ์ในการเปลี่ยนแปลงข้อความในอ็อบเจกต์ของคลาส `TextArea`

7.3 อินเตอร์เฟสประเภท `Listener`

ภาษาจาวาจะจัดการกับเหตุการณ์ โดยการสร้างอ็อบเจกต์ที่สามารถรับฟังเหตุการณ์จากคลาสที่ `implements` อินเตอร์เฟสประเภท `Listener` ที่สอดคล้องกันซึ่งอ็อบเจกต์นี้จะทำหน้าที่เป็น Event Handler อาทิเช่น อ็อบเจกต์ที่จะจัดการกับเหตุการณ์ประเภท `ActionEvent` จะต้อง `implements` อินเตอร์เฟส `ActionListener` โดยต้องเขียนบล็อกคำสั่งในเมธอด `actionPerformed()` อินเตอร์เฟสประเภท `Listener` มีทั้งหมด 13 ชนิด ซึ่งสอดคล้องกับคลาสประเภท `Event` ดังนี้

- ActionListener เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส ActionEvent
- `AdjustmentListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `AdjustmentEvent`
- `ComponentListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `ComponentEvent`
- `ContainerListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `ContainerEvent`
- `FocusListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `FocusEvent`
- `InputMethodListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `InputMethodEvent`

- `ItemListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `ItemEvent`
- `KeyListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `KeyEvent`
- `MouseListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `MouseEvent`
- `MouseMotionListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `MouseEvent`
- `TextListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `TextEvent`
- `WindowListener` เป็นอินเตอร์เฟสสำหรับอ็อบเจกต์ของคลาส `WindowEvent`
- `AWTEventListener` เป็นอินเตอร์เฟสสำหรับให้เราสามารถที่จะไปพัฒนาคลาสประเภท `EventListener` ได้

อ็อบเจกต์ใดๆ ที่ต้องการจัดการกับเหตุการณ์จะต้องลงทะเบียน (register) รับฟังเหตุการณ์นั้นด้วยโดยต้องใช้เมธอดที่สอดคล้องกันดังแสดงในตารางที่ 7.1 ส่วนอินเตอร์เฟสประเภท `Listener` แต่ละชนิดจะมีเมธอดที่ต้องเขียนบล็อกคำสั่งต่างๆ ดังแสดงในตารางที่ 7.2

ตารางที่ 7.1 เมธอดที่ใช้รับฟังเหตุการณ์

อินเตอร์เฟส	เมธอดที่ใช้รับฟังเหตุการณ์
`ActionListener`	`addActionListener()`
`ItemListener`	`addItemListener()`
`KeyListener` `InputMethodListener`	`addKeyListener()` `addInputMethodListener()`
`MouseListener`	`addMouseListener()`
`MouseMotionListener`	`addMouseMotionListener()`
`TextListener`	`addTextListener()`
`FocusListener`	`addFocusListener()`
`AdjustmentListener`	`addAdjustmentListener()`
`ComponentListener`	`addComponentListener()`
`ContainerListener`	`addContainerListener()`
`WindowListener`	`addWindowListener()`

ตารางที่7.2 เมธอดที่ต้อง `implements` สำหรับอินเตอร์เฟสประเภท `Event`

อินเตอร์เฟส	**เมธอดที่ต้อง** `implements`
`ActionListener`	`actionPerformed(ActionEvent)`
`ItemListener`	`itemStateChanged(ItemEvent)`
`MouseMotionListener`	`mouseDragged(MouseEvent)` `mouseMoved(MouseEvent)`

`MouseListener`	`mouseClicked(MouseEvent)`
	`mouseEntered(MouseEvent)`
	`mouseExited(MouseEvent)`
	`mousePressed(MouseEvent)`
	`mouseReleased(MouseEvent)`

`KeyListener`	`keyPressed(KeyEvent)`
	`keyReleased(KeyEvent)`
	`keyTyped(KeyEvent)`

`InputMethodListener`	`caretPositionChanged(InputMethodEvent)` `inputMethodTextChanged(InputMethodEvent)`
`FocusListener`	`focusGained(FocusEvent)` `focusLost(FocusEvent)`
`AdjustmentListener`	`adjustmentValueChanged(AdjustmentEvent)`

`ComponentListener`	`componentMoved(ComponentEvent)`
	`componentHidden(ComponentEvent)`
	`componentResized(ComponentEvent)`
	`componentShown(ComponentEvent)`

`WindowListener`	`windowOpened(WindowEvent)`
	`windowClosed(WindowEvent)`
	`windowClosing(WindowEvent)`
	`windowIconified(WindowEvent)`
	`windowDeiconified(WindowEvent)`
	`windowActivated(WindowEvent)`
	`windowDeactivated(WindowEvent)`

`ContainerListener`	`componentAdded(ContainerEvent)` `componentRemoved(ContainerEvent)`
`TextListener`	`textValueChanged(TextEvent)`

7.4 การจัดการกับเหตุการณ์

ภาษาจาวาจะมีวิธีการจัดการกับเหตุการณ์ที่เรียกว่า Delegation Model โดยจะมีหลักการดังนี้

- อ็อบเจกต์ของส่วนประกอบกราฟิกใดๆ สามารถเป็นอ็อบเจกต์ประเภท Event Source ได้ อาทิเช่น อ็อบเจกต์ของคลาส `Button` สามารถเป็น Event Source ของ `ActionEvent` ได้
- คลาสใดๆ สามารถรับฟังเหตุการณ์ใดๆ ก็ได้ ถ้าคลาสนั้น `implements` อินเตอร์เฟสประเภท `Listener` ที่สอดคล้องกันอาทิเช่น คลาสที่ต้องการรับฟังเหตุการณ์ `ActionEvent` จะต้อง `implements` อินเตอร์เฟสที่ชื่อ `ActionListener`
- อ็อบเจกต์ประเภท `Event` ที่เกิดจาก Event Source จะถูกส่งไปยังอ็อบเจกต์ของคลาสที่สามารถรับฟังเหตุการณ์ประเภทนั้น

จากหลักการข้างต้น โปรแกรมภาษาจาวาจะมีวิธีการเขียนคำสั่งเพื่อจัดการกับเหตุการณ์ต่างๆ ดังนี้

Event Source ใดต้องการที่จะจัดการกับเหตุการณ์ใดต้องลงทะเบียนเพื่อรับฟังเหตุการณ์โดยมีรูปแบบดังนี้

```
eventSource.addXxxListener(listener)
```

โดยที่

- `eventSource` เป็นชื่อของอ็อบเจกต์ที่เป็น Event Source
- `addXxxListener` เป็นเมธอดที่ใช้ในการลงทะเบียนรับฟังเหตุการณ์โดยจะต้องเลือกใช้เมธอดที่สอดคล้องกันในการรับฟังเหตุการณ์แต่ละชนิด ดังแสดงในตารางที่ 7.2 เช่น ใช้เมธอด `addActionListener()` เพื่อรับฟัง `ActionEvent`
- `listener` เป็นอ็อบเจกต์ของคลาส `Xxxlistener` ซึ่งสามารถรับฟังเหตุการณ์ที่ต้องการจัดการได้ และทำหน้าที่เป็น Event Handler
- `XxxListener` เป็นคลาสที่ `implements` อินเตอร์เฟสประเภท `Listener` ที่สอดคล้องกัน

การเขียนโปรแกรมเพื่อจัดการกับเหตุการณ์สามารถที่เขียนคลาสที่เป็น Event Handler ได้หลายรูปแบบดังนี้

- กำหนดคลาสภายนอกคลาสที่ใช้ในการจัดการเหตุการณ์
- กำหนดคลาสที่เป็นคลาสภายในอยู่ในคลาสที่ใช้ในการจัดการเหตุการณ์
- กำหนดให้คลาสที่ใช้ในการจัดการเหตุการณ์ `implements` อินเตอร์เฟสประเภท `Listener` ที่สอดคล้องกัน และสร้างอ็อบเจกต์ของคลาสดังกล่าวภายในคลาสเอง
- กำหนดคลาสภายในเมธอด (คลาสประเภท anonymous)

7.4.1 การสร้างอ็อบเจกต์ของคลาสภายนอก

โปรแกรมที่ 7.1 แสดงตัวอย่างการจัดการกับเหตุการณ์ที่เกิดจากการกดปุ่ม โดยกำหนดคลาส `ActionHandler` ให้เป็นคลาสประเภท Event Handler ซึ่งคลาสนี้จะอยู่ภายนอกคลาส `EventDemo1` ซึ่งเป็นคลาสที่ใช้ในการจัดการกับเหตุการณ์การกดปุ่ม คลาส `EventDemo1` จะมีอ็อบเจกต์ของคลาส `JButton` ที่ชื่อ `bn1` อ็อบเจกต์ `bn1` ได้ลงทะเบียนรับฟังเหตุการณ์ `ActionEvent` โดยใช้คำสั่ง

```
bn1.addActionListener(new ActionHandler())
```

โดยที่คำสั่ง `new ActionHandler()` เป็นการสร้างอ็อบเจกต์ของคลาส `ActionHandler` ที่จะทำหน้าที่เป็น Event Handler ดังนั้นคลาส `ActionHandler` จะต้อง `implements` อินเตอร์เฟส `ActionListener` โดยมีเมธอดเดียวที่จะต้องเขียนบล็อกคำสั่งคือ

```
public void actionPerformed(ActionEvent ev)
```

ซึ่งผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.1 จะได้ดังแสดงในรูปที่ 7.4

<u>โปรแกรมที่ 7.1</u> <u>การจัดการกับเหตุการณ์ที่เกิดจากการกดปุ่ม</u>

```
import java.awt.FlowLayout;
import javax.swing.*;

public class EventDemo1 {
    public void init() {
        JFrame fr = new JFrame("Outer Class Event Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JButton bn1 = new JButton("Exit");
        fr.setLayout(new FlowLayout());
        bn1.addActionListener(new ActionHandler());
        fr.add(bn1);
        fr.setSize(200, 200);
        fr.setVisible(true);
    }
    public static void main(String[] args) {
        EventDemo1 obj = new EventDemo1();
        obj.init();
    }
}

------------------------------------------------------------
public class ActionHandler implements ActionListener {
   public void actionPerformed(ActionEvent ev) {
      System.exit(0);
   }
}
```

รูปที่ 7.4 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.1

7.5 การสร้างอ็อบเจกต์ของคลาสภายใน

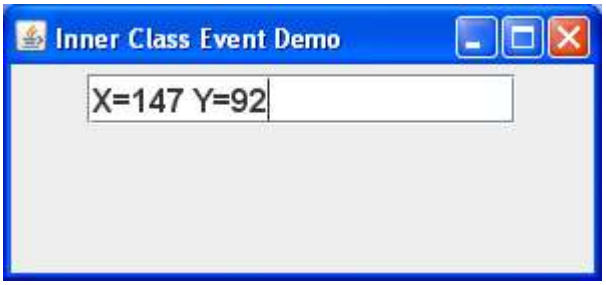
โปรแกรมที่ 7.2 แสดงตัวอย่างการจัดการกับเหตุการณ์ที่เกิดขึ้นจากการเลื่อนเมาส์ โดยกำหนดคลาสประเภท Event Handler ให้อยู่ภายในคลาสที่ต้องการจัดการกับเหตุการณ์ จากตัวอย่างนี้คลาส `MouseHandler` ซึ่งเป็นคลาสที่จะรับฟังเหตุการณ์ชนิด `MouseMotionListener` โดยเป็นคลาสภายในที่อยู่ในคลาส `EventDemo2` ที่มีอ็อบเจกต์ของคลาส `JFrame` ที่ชื่อ `fr` ซึ่งการลงทะเบียนรับฟังเหตุการณ์ `MouseEvent` โดยใช้คำสั่ง

```
fr.addMouseMotionListener(new MouseHandler())
```

โดยที่คำสั่ง `new MouseHandler()` เป็นการสร้างอ็อบเจกต์ของคลาส `MouseHandler` ที่จะทำหน้าที่เป็น Event Handler โปรแกรมนี้จะได้ผลลัพธ์ดังแสดงในรูปที่ 7.5

โปรแกรมที่ 7.2 การจัดการกับเหตุการณ์ที่เกิดขึ้นจากการเลื่อนเมาส์

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class EventDemo2 {

    private JFrame fr;
    private JTextField tf;

    public void init() {
        fr = new JFrame("Same Class Event Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        tf = new JTextField(15);
        fr.setLayout(new FlowLayout());
        fr.add(tf);
        tf.setFont(new Font("TimesRoman", Font.BOLD, 16));
        fr.addMouseMotionListener(new MouseHandler());
        fr.setSize(200, 200);
        fr.setVisible(true);
    }

    public class MouseHandler implements MouseMotionListener {

        public void mouseDragged(MouseEvent ev) {
            tf.setText("X=" + ev.getX() + " Y=" + ev.getY());
        }

        public void mouseMoved(MouseEvent ev) {
        }
    }

    public static void main(String[] args) {
        EventDemo2 obj = new EventDemo2();
        obj.init();

    }
}
```



รูปที่ 7.5 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.2

7.5.1 การสร้างอ็อบเจกต์ภายในคลาสเดียวกัน

เราสามารถที่จะกำหนดให้คลาสที่ต้องการจะจัดการกับเหตุการณ์ เป็นคลาสประเภท Event Handler ได้เอง โดยการกำหนดให้คลาสนั้น `implements` อินเตอร์เฟสประเภท `Listener` ที่สอดคล้องกัน โปรแกรมที่ 7.3 แสดงตัวอย่างของคลาส `EventDemo3` ซึ่งจะมีอ็อบเจกต์ของคลาส `JFrame` ที่ชื่อ `fr` ซึ่งจะรับฟังเหตุการณ์ประเภท `WindowEvent` คลาสนี้จะเป็นคลาสประเภท Event Handler ด้วย โดยการ implements อินเตอร์เฟส `WindowListener` โปรแกรมนี้ได้สร้างอ็อบเจกต์ของคลาส `EventDemo` ขึ้นมา และใช้คำสั่ง

```
fr.addWindowListener(this)
```

เพื่อลงทะเบียนรับฟังเหตุการณ์และให้อ็อบเจกต์ของคลาสนี้ `(this)` ซึ่งเป็นอ็อบเจกต์ประเภท `Event Handler` ในการจัดการกับเหตุการณ์ โปรแกรมนี้จะทำให้สามารถปิดเฟรมได้โดยการคลิกเมาส์ที่เครื่องหมายกากบาทตรง title bar ทั้งนี้เนื่องจากคลาส `EventDemo3` มีคำสั่งในเมธอด `windowClosing()` เพื่อออกจากโปรแกรม (คำสั่ง `System.exit(0)`) โปรแกรมนี้จะได้ผลลัพธ์ดังแสดงในรูปที่ 7.6

โปรแกรมที่ 7.3 คลาสที่มีการรับฟังเหตุการณ์ประเภท `WindowEvent`

```
import java.awt.event.*;
import javax.swing.JFrame;

public class EventDemo3 implements WindowListener{
    public void init() {
        JFrame fr = new JFrame("Same Class Event Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.addWindowListener(this);
        fr.setSize(200, 200);
        fr.setVisible(true);
    }
    public void windowClosing(WindowEvent ev) {
        System.exit(0);
    }
    public void windowOpened(WindowEvent ev) {}
    public void windowClosed(WindowEvent ev) {}
    public void windowIconified(WindowEvent ev) {}
    public void windowDeiconified(WindowEvent ev) {}
    public void windowActivated(WindowEvent ev) {}
    public void windowDeactivated(WindowEvent ev) {}

    public static void main(String[] args) {
        EventDemo3 obj = new EventDemo3();
        obj.init();
    }
}
```

รูปที่ 7.6 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.3

7.5.2 การรับฟังเหตุการณ์หลายเหตุการณ์

ภาษาจาวาอนุญาตให้อ็อบเจกต์ของคลาสที่เป็นส่วนกราฟิกแต่ละอ็อบเจกต์ สามารถที่จะรับฟังเหตุการณ์หลายประเภทได้พร้อมกันอาทิเช่น ถ้าอ็อบเจกต์ `fr` เป็นอ็อบเจกต์ของ `JFrame` เราสามารถที่จะลงทะเบียนรับฟังเหตุการณ์ได้ดังนี้

```
fr.addMouseMotionListener(this);
fr.addWindowListener(this);
```

ซึ่งเป็นการกำหนดให้อ็อบเจกต์ `fr` รับฟังเหตุการณ์ที่เกิดจากการเลื่อนเมาส์และเหตุการณ์ที่เกี่ยวข้องกับเฟรมพร้อมกัน

คลาสใดๆ สามารถที่จะ `implements` อินเตอร์เฟสประเภท `Listener` ได้หลายชนิดเช่น

```
public class EventDemo4 implements
          MouseMotionListener, WindowListener {
     ...
}
```

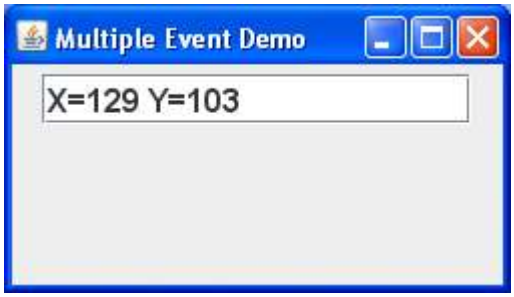
เป็นการกำหนดคลาส `EventDemo4` ให้เป็นคลาสประเภท Event Handler ที่ `implement` อินเตอร์เฟส `MouseMotionListener` และ `WindowListener` ภายในคลาสเดียวกัน

นอกจากนี้คลาสประเภท Event Handler ใดๆ สามารถที่ใช้ในการสร้างอ็อบเจกต์แล้วจัดการกับอ็อบเจกต์ที่เป็น Event Source ได้หลายอ็อบเจกต์ ทั้งนี้ argument ที่เป็นอ็อบเจกต์ของคลาสประเภท `Event` จะมีเมธอดที่ใช้ในการระบุอ็อบเจกต์ของ Event Source ได้อาทิเช่น (เมธอด `getSource()`)

โปรแกรมที่ 7.4 แสดงตัวอย่างของคลาส `EventDemo4` ซึ่งมีอ็อบเจกต์ `fr` ที่ลงทะเบียนรับฟังเหตุการณ์สองชนิด และคลาสนี้ `implements` อินเตอร์เฟสทั้งสองชนิดที่สอดคล้องกัน โปรแกรมนี้จะให้ผลลัพธ์ดังรูปที่ 7.7

โปรแกรมที่ 7.4 คลาสที่ลงทะเบียนรับฟังเหตุการณ์สองชนิด

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;


public class EventDemo4 implements MouseMotionListener, WindowListener {

    private JFrame fr;
    private JTextField tf;

    public void init() {
        fr = new JFrame("Multiple Event Demo");
        tf = new JTextField(15);
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.setLayout(new FlowLayout());
        fr.add(tf);
        tf.setFont(new Font("TimesRoman", Font.BOLD, 16));
        fr.addMouseMotionListener(this);
        fr.addWindowListener(this);
        fr.setSize(200, 200);
        fr.setVisible(true);
    }

    public void mouseDragged(MouseEvent ev) {
        tf.setText("X=" + ev.getX() + " Y=" + ev.getY());
    }

    public void mouseMoved(MouseEvent ev) { }
    public void windowClosing(WindowEvent ev) {
      System.exit(0);
    }
    public void windowOpened(WindowEvent ev) { }
    public void windowClosed(WindowEvent ev) { }
    public void windowIconified(WindowEvent ev) { }
    public void windowDeiconified(WindowEvent ev) { }
    public void windowActivated(WindowEvent ev) { }
    public void windowDeactivated(WindowEvent ev) { }

    public static void main(String[] args) {
        EventDemo4 obj = new EventDemo4();
        obj.init();
    }
}
```



รูปที่ 7.7 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.4

7.5.3 คลาสประเภท Event Adapter

คลาสประเภท Event Adapter คือคลาสที่ได้ `implements` อินเตอร์เฟสประเภท `Listener` ไว้แล้ว โดยได้กำหนดเมธอดต่างๆ ของอินเตอร์เฟสที่ต้องเขียนบล็อกคำสั่งไว้แล้ว แต่จะเป็นบล็อกคำสั่งที่ไม่มีคำสั่งใดๆ อยู่ภายในบล็อก คลาสประเภท Event Adapter จะช่วยทำให้เขียนโปรแกรมที่เป็นคลาสประเภท Event Handler ได้ง่ายขึ้น โดยลดจำนวนเมธอดที่จะต้องเขียนบล็อกคำสั่ง การเขียนคลาสประเภท Event Handler นี้จะต้องสืบทอดมาจากคลาสประเภท Event Adapter สำหรับการรับฟังเหตุการณ์ที่สอดคล้องกันและจะกำหนดเมธอดแบบ `Overridden` เฉพาะเมธอดที่ต้องการจัดการกับเหตุการณ์ คลาสประเภท Event Adapter จะมีอยู่ 7 คลาสดังนี้

- `MouseAdapter` คือคลาสที่ implements อินเตอร์เฟสที่ชื่อ MouseListener
- `MouseMotionAdapter` คือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `MouseMotionListener`
- `ComponentAdapter` คือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `ComponentListener`
- `ContainerAdapter` คือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `ContainerListener`
- `KeyAdapter` คือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `KeyListener`
- `WindowAdapter` คือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `WindowListener`
- `FocusAdapter` คือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `FocusListener`

โปรแกรมที่ 7.5 แสดงตัวอย่างของการกำหนดคลาสประเภท Event Adapter ที่สืบทอดมาจากคลาส `WindowAdapter` โปรแกรมนี้จะทำให้สามารถปิดเฟรม ซึ่งผลลัพธ์ที่ได้จะเป็นเช่นเดียวกับผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.1

โปรแกรมที่ 7.5 คลาสประเภทที่สืบทอดมาจากคลาส `WindowAdapter`

```
import javax.swing.JFrame;

public class EventDemo5 {

    public void init() {
        JFrame fr = new JFrame("Adapter Class Event Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.addWindowListener(new WindowHandler());
        fr.setSize(200, 200);
        fr.setVisible(true);
    }

    public static void main(String[] args) {
        EventDemo5 obj = new EventDemo5();
        obj.init();
    }
}

-----------------------------------------------------------------
import java.awt.event.*;

public class WindowHandler extends WindowAdapter {
    public void windowClosing(WindowEvent ev) {
        System.exit(0);
    }
}
```

7.5.4 การสร้างคลาสแบบ anonymous

เราสามารถสร้างคลาสประเภท Event Handler ภายในเมธอดที่ใช้ในการลงทะเบียนรับฟังเหตุการณ์ (เมธอดประเภท `addXXxListener()`) คลาสประเภทนี้เรียกว่าคลาสแบบ anonymous ซึ่งมักจะใช้คู่กับคลาสประเภท Event Adapter โดยนิยมใช้กับการปิดเฟรมอาทิเช่น คำสั่ง

```
fr.addWindowListener(new WindowAdapter() {
   public void windowClosing(WindowEvent ev) {
      System.exit(0);
   }
});
```

เป็นการสร้างคลาสแบบ anonymous ภายในเมธอด `addWindowListener()` เพื่อปิดเฟรมของอ็อบเจกต์ `fr` โปรแกรมที่ 7.6 แสดงตัวอย่างของคลาสในรูปแบบนี้ ซึ่งผลลัพธ์ที่ได้จะเป็นเช่นเดียวกับผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 7.1

โปรแกรมที่ 7.6 คลาสแบบ anonymous

```
import java.awt.event.*;
import javax.swing.JFrame;

public class EventDemo6 {
    public void init() {
        JFrame fr = new JFrame("Anonymous Class Event Demo");
        fr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fr.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent ev) {
                System.exit(0);
            }
        });
        fr.setSize(200, 200);
        fr.setVisible(true);
    }
    public static void main(String[] args) {
        EventDemo6 obj = new EventDemo6();
        obj.init();
    }
}
```

สรุปเนื้อหาของบท

- อ็อบเจกต์ของคลาสประเภท `Event` ในแพจเก็จ `java.awt.event` จะถูกสร้างขึ้น เมื่อมีเหตุการณ์เกิดขึ้นกับอ็อบเจกต์ของส่วนประกอบกราฟิก เช่น `ActionEvent` เกิดขึ้นเมื่อมีการกด `Button` หรือ `WindowEvent` เกิดขึ้นเมื่อมีการปิด `Frame`
- การจะจัดการกับ Event ประเภทใดนั้น โดยทั่วไปจะต้องสร้างอ็อบเจกต์ของคลาสที่ `implements` อินเตอร์เฟสที่สอดคล้องกันกับ Event นั้นด้วย เช่นถ้าต้องการจัดการกับ `WindowEvent` จะต้องสร้างอ็อบเจกต์ของคลาสที่ `implements` อินเตอร์เฟส `WindowListener` โดยการสร้างอ็อบเจกต์อาจสร้างจากคลาสใหม่ คลาสภายใน คลาสเดียวกัน หรือคลาสประเภท anonymous ซึ่งคลาสประเภทนี้จะถูกเรียกว่าคลาสประเภท Event Handler
- คลาสใดๆ สามารถ `implements` อินเตอร์เฟสได้หลายตัว ทำให้สามารถที่จะขึ้นทะเบียนรับฟังเหตุการณ์ได้หลายเหตุการณ์
- คลาสประเภท Event Adapter คือคลาสที่ `implements` อินเตอร์เฟสประเภท Listener ไว้แล้ว โดยได้กำหนดเมธอดต่างๆ ของอินเตอร์เฟสที่ต้องเขียนบล็อกคำสั่งไว้แล้ว แต่จะเป็นบล็อกคำสั่งที่ไม่มีคำสั่งใดๆ อยู่ภายในบล็อก

บทที่ 8 อะเรย์และคอลเล็กชั่น

เนื้อหาในบทนี้เป็นการแนะนำการใช้อะเรย์ของข้อมูลชนิดพื้นฐาน และชนิดคลาส อธิบายการประกาศและสร้างอะเรย์หลายมิติ แนะนำเมธอดที่เกี่ยวข้องกับอะเรย์ อธิบายความหมายของคอลเล็กชั่น แนะนำอินเตอร์เฟสและคลาสต่างๆ ที่อยู่ใน Collection API และการนำไปใช้งาน

8.1 อะเรย์

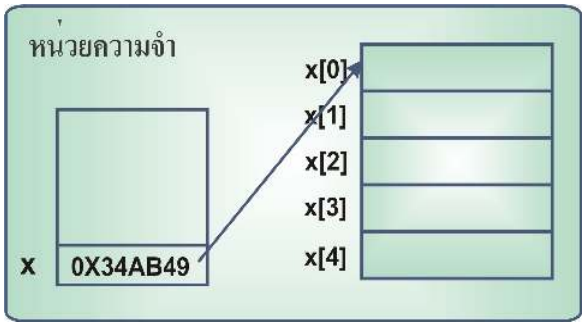
อะเรย์ในภาษาจาวา คือตัวแปรที่เป็นชนิดข้อมูลแบบอ้างอิงที่สามารถใช้เก็บข้อมูลชนิดเดียวกันได้หลายค่า ตัวแปรอะเรย์จะเป็นตัวแปรที่มีชื่อเดียวกันแต่จะมีจำนวนสมาชิกได้หลายตัว โดยตำแหน่งในหน่วยความจำที่เก็บข้อมูลเหล่านี้จะอยู่ติดกัน ตัวอย่างเช่น หากต้องการกำหนดตัวแปรให้มีชนิดข้อมูลเป็น `int` เพื่อเก็บข้อมูลได้ 5 ค่าโดยตั้งชื่อตัวแปรเป็น `x1, x2, x3, x4` และ `x5` ดังนี้

```
int  x1, x2, x3, x4, x5;
```

ตัวแปรดังกล่าวจะถือว่าเป็นตัวแปรคนละตัวกัน โดยมีชื่อต่างกันและไม่มีความสัมพันธ์ซึ่งกันและกัน แต่หากประกาศให้ตัวแปร `x` เป็นตัวแปรอะเรย์ที่มีจำนวนสมาชิก 5 ตัว โดยใช้คำสั่งดังนี้

```
int []x = new int[5];
```

จะทำให้ตัวแปรอะเรย์ `x` เป็นตัวแปรที่มีชนิดข้อมูลเป็น `int` ซึ่งมีสมาชิก 5 ตัว โดยมีหมายเลขสมาชิกตั้งแต่ 0 ถึง 4 และเป็นชนิดข้อมูลแบบอ้างอิง กล่าวคือค่าของ `x` จะเก็บตำแหน่งของหน่วยความจำเพื่ออ้างอิงถึงตำแหน่งที่เก็บข้อมูลสมาชิกของอะเรย์แต่ละตัว ดังแสดงในรูปที่ 8.1



รูปที่ 8.1 การเก็บตำแหน่งอ้างอิงของตัวแปรอะเรย์

ภาษาจาวาแบ่งตัวแปรอะเรย์เป็นสองประเภทคือ

1.อะเรย์ของข้อมูลชนิดพื้นฐาน

2.อะเรย์ของข้อมูลชนิดคลาส

อะเรย์ของข้อมูลชนิดพื้นฐาน คืออะเรย์ที่สามารถใช้เก็บข้อมูลที่มีชนิดข้อมูลแบบพื้นฐานชนิดใดชนิดหนึ่งได้หลายค่า เช่นอะเรย์ของข้อมูลชนิด `int` หรืออะเรย์ของข้อมูลชนิด `boolean` เป็นต้น

อะเรย์ของข้อมูลชนิดคลาส คืออะเรย์ที่สามารถใช้เก็บข้อมูลที่เป็นอ็อบเจกต์ของคลาสใดๆ ได้หลายอ็อบเจกต์ เช่นอะเรย์ของข้อมูลชนิด `String` เป็นต้น

8.2 อะเรย์ของข้อมูลชนิดพื้นฐาน

ภาษาจาวาสามารถที่จะสร้างอะเรย์ เพื่อใช้เก็บข้อมูลที่มีชนิดข้อมูลแบบพื้นฐานได้ การสร้างตัวแปรอะเรย์ จะมีขั้นตอนเช่นเดียวกับการสร้างตัวแปรที่เป็นชนิดข้อมูลแบบอ้างอิงทั่วไปสองขั้นตอนคือ

1. ขั้นตอนแรกเป็นการประกาศชื่อตัวแปร ขั้นตอนนี้จะเป็นการจองเนื้อที่ในหน่วย ความจำเพื่อเก็บค่าของตำแหน่งอ้างอิงที่เก็บข้อมูลสมาชิกในหน่วยความจำ แต่ขั้นตอนนี้จะยังไม่มีการจองเนื้อที่ในหน่วยความจำเพื่อใช้เก็บค่าข้อมูลสมาชิกของอะเรย์แต่ละตัว ดังนั้นค่าของตัวแปรในขั้นตอนนี้จะเป็น `null`
2. ขั้นตอนที่สองเป็นการสร้างตัวแปร เพื่อบอกจำนวนสมาชิกของอะเรย์ และจองเนื้อที่ในหน่วยความจำสำหรับเก็บค่าข้อมูลสมาชิกของอะเรย์แต่ละตัว โดยใช้คำสั่ง `new` เช่นเดียวกับการสร้างอ็อบเจกต์ของคลาส ขั้นตอนนี้จะทำให้ค่าของตัวแปรเปลี่ยนไปเป็นค่าของตำแหน่งอ้างอิงที่เก็บข้อมูลสมาชิกของอะเรย์

8.2.1 การประกาศชื่อตัวแปรอะเรย์ของข้อมูลชนิดพื้นฐาน

การประกาศชื่อของตัวแปรอะเรย์จะมีรูปแบบคล้ายกับการประกาศชื่อตัวแปรของชนิดข้อมูลแบบพื้นฐาน แต่การประกาศตัวแปรอะเรย์จะต้องมีเครื่องหมาย `[]` อยู่ด้านหน้าหรือด้านหลังชื่อตัวแปรดังนี้

```
          [modifier] dataType []variableName;
   หรือ   [modifier] dataType variableName[];
```

ตัวอย่างเช่น คำสั่ง

```
int x[];
private char []ch;
public double y[];
```

ต่างก็เป็นการประกาศให้ตัวแปร x, ch และ y เป็นตัวแปรอะเรย์ โดยมีชนิดข้อมูลเป็น int,char และ double ตามลำดับ

การประกาศชื่อตัวแปรยังเป็นการจองเนื้อที่ในหน่วยความจำ เพื่อเก็บค่าของตำแหน่งอ้างอิงที่เก็บข้อมูลสมาชิกในหน่วยความจำ ซึ่งในขั้นตอนนี้จะเก็บค่าเป็น null ไว้ก่อน เนื่องจากยังไม่มีการจองเนื้อที่ในหน่วยความจำเพื่อเก็บข้อมูลสมาชิกของอะเรย์

ภาษาจาวาแตกต่างจากภาษาซีตรงที่อนุญาตให้ใช้เครื่องหมาย [] อยู่ด้านหน้าชื่อตัวแปรอะเรย์ที่ต้องการประกาศได้ ส่วนข้อแตกต่างของการใช้เครื่องหมาย [] ด้านหน้าและด้านหลังจะเกิดขึ้น ในกรณีที่ต้องการประกาศตัวแปรหลายๆ ตัว ตัวอย่างเช่น คำสั่ง

```
int []x,y;
```

จะเป็นการประกาศให้ตัวแปร x และ y เป็นตัวแปรอะเรย์ชนิดข้อมูล int ทั้งคู่แต่คำสั่ง

```
int x[],y;
```

จะเป็นการประกาศให้ตัวแปร x เป็นตัวแปรอะเรย์ที่มีชนิดข้อมูลเป็น int ส่วนตัวแปร y เป็นตัวแปรปกติที่ไม่ใช่ตัวแปรอะเรย์โดยมีชนิดข้อมูลเป็น int

8.2.2 การสร้างตัวแปรอะเรย์ของข้อมูลชนิดพื้นฐาน

การประกาศชื่อตัวแปรอะเรย์จะไม่มีการจองเนื้อที่ในหน่วยความจำ เพื่อเก็บข้อมูลสมาชิกของอะเรย์ เนื้อที่ในหน่วยความจำดังกล่าวจะถูกจองขึ้นเมื่อมีการใช้คำสั่ง new ซึ่งมีรูปแบบคำสั่งดังนี้

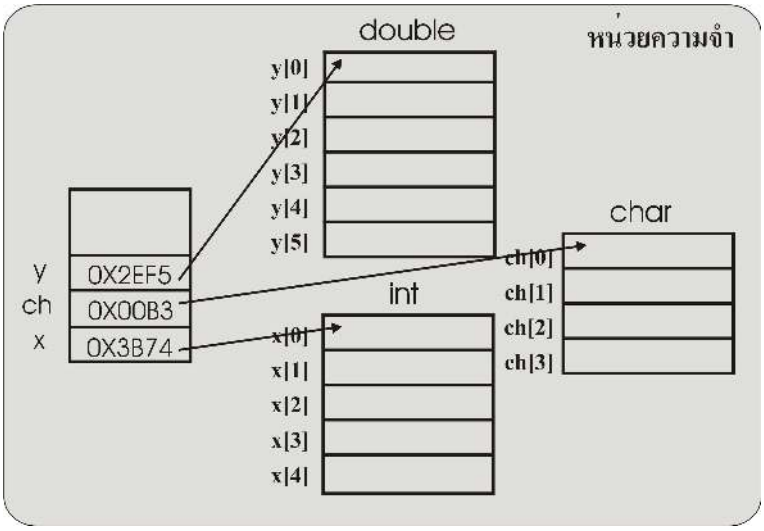
```
variableName = new dataType[size];
```

โดยที่ size คือจำนวนสมาชิกของอะเรย์ที่ต้องการ

คำสั่งนี้เป็นการระบุจำนวนสมาชิกของตัวแปรอะเรย์และเป็นการจองเนื้อที่ในหน่วย ความจำสำหรับสมาชิกของอะเรย์แต่ละตัว นอกจากนี้ยังจะทำให้ค่าในหน่วยความจำที่เก็บตำแหน่งอ้างอิงของตัวแปรนั้นเปลี่ยนค่าจาก null ไปเป็นตำแหน่งที่เก็บข้อมูลดังแสดงในรูปที่ 8.2 ตัวอย่างเช่น คำสั่ง

```
x = new int[5];
ch = new char[4];
y = new double[6];
```

เป็นคำสั่งสร้างตัวแปรอะเรย์ `x`, `ch` และ `y` ให้มีชนิดข้อมูลเป็น `int`, `char` และ `double` และมีจำนวนสมาชิกของอะเรย์เท่ากับ 5, 4 และ 6 ตามลำดับ



รูปที่ 8.2 การเก็บตำแหน่งอ้างอิงของตัวแปรอะเรย์

ในการสร้างตัวแปรอะเรย์ `dataType` ที่อยู่ในคำสั่ง `new` จะต้องเป็นชนิดข้อมูลชนิดเดียวกันกับชนิดข้อมูลของตัวแปรอะเรย์ ดังนั้นคำสั่ง

```
int []x;
x = new double[4];
```

จึงเป็นคำสั่งในการสร้างตัวแปรอะเรย์ที่ไม่ถูกต้อง เนื่องจากตัวแปรอะเรย์ `x` ถูกประกาศให้มีชนิดข้อมูลเป็น `int` แต่คำสั่ง `new` เป็นการมาสร้างอะเรย์ของข้อมูลชนิด `double`

เราสามารถที่จะรวมคำสั่งประกาศชื่อตัวแปรและคำสั่งการสร้างตัวแปรอะเรย์ ไว้ในคำสั่งเดียวกันได้ โดยมีรูปแบบคำสั่งดังนี้

```
          dataType []variableName = new dataType[size];
หรือ      dataType variableName[] = new dataType[size];
```

ตัวอย่างเช่น

```
int []x = new int[5];
```

8.2.3 การเรียกใช้สมาชิกของอะเรย์

ตัวแปรอะเรย์ที่สร้างขึ้นจะมีสมาชิกที่มีหมายเลขตั้งแต่ 0 จนถึง size-1 การอ้างอิงถึงสมาชิกของอะเรย์แต่ละตัวจะมีรูปแบบดังนี้

```
variableName[index]
```

โดยที่ `index` คือตัวเลขระบุหมายเลขที่ของสมาชิกของอะเรย์ ซึ่งจะมีค่าได้ตั้งแต่ค่า 0 จนถึง size-1

ตัวอย่างเช่น

```
x[3]
```

หมายถึงข้อมูลของสมาชิกหมายเลขที่ 3 ของตัวแปรอะเรย์ `x`

ตัวแปรอะเรย์ที่เป็นชนิดข้อมูลแบบพื้นฐาน จะแตกต่างจากตัวแปรปกติที่เป็นชนิดข้อมูลแบบพื้นฐาน โดยตัวแปรอะเรย์จะกำหนดค่าเริ่มต้นให้กับสมาชิกทุกตัวเสมอ ซึ่งมีค่าเริ่มต้นเช่นเดียวกับตัวแปรที่เป็นคุณลักษณะของอ็อบเจกต์ (ตารางที่ 2.5) ขณะที่ตัวแปรปกติจะถูกกำหนดค่าเริ่มต้นเฉพาะตัวแปรที่เป็นคุณลักษณะของอ็อบเจกต์ หรือคุณลักษณะของคลาสเท่านั้น

เราสามารถที่จะเปลี่ยนแปลงค่าของสมาชิกของอะเรย์ได้ โดยการใช้คำสั่งกำหนดค่า โดยต้องอ้างอิงถึงหมายเลขสมาชิกของอะเรย์ ตัวอย่างเช่น

```
x[0] = 4;
x[2] = 5;
```

เป็นการกำหนดค่าให้กับสมาชิกหมายเลขที่ 0 และ 2 ของตัวแปรอะเรย์ `x` ให้มีค่าเป็น 4 และ 5 ตามลำดับ

การอ้างอิงถึงชื่อตัวแปร x โดยไม่ระบุหมายเลขสมาชิกจะเป็นการเรียกดูค่าตำแหน่งอ้างอิงของตัวแปรอะเรย์ โปรแกรมที่ 8.1 แสดงตัวอย่างการประกาศและสร้างตัวแปรอะเรย์ พร้อมทั้งกำหนดค่าต่างๆ โดยผลลัพธ์ที่ได้เป็นดังแสดงในรูปที่ 8.3

<u>โปรแกรมที่ 8.1</u> <u>ตัวอย่างการใช้อะเรย์</u>

```
public class SimpleArrays {
   public static void main(String args[]) {
      int []x;
      x = new int[4];
      x[0] = 0;
      x[1] = 2;
      x[2] = 4;
      x[3] = 6;
      System.out.println("x = "+x);
      System.out.println("x[0] = "+x[0]);
      System.out.println("x[1] = "+x[1]);
      System.out.println("x[2] = "+x[2]);
      System.out.println("x[3] = "+x[3]);
   }
}
```

```
Output - Chapter8 (run)        Tasks
run:
x = [I@a90653
x[0] = 0
x[1] = 2
x[2] = 4
x[3] = 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

รูปที่ 8.3 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.1

8.2.4 การกำหนดค่าเริ่มต้นให้กับสมาชิกของอะเรย์

เราสามารถที่จะประกาศตัวแปรอะเรย์ สร้างตัวแปรอะเรย์ และกำหนดค่าให้กับสมาชิกของอะเรย์ภายในคำสั่งเดียวกัน โดยมีรูปแบบของคำสั่งดังนี้

```
dataType []variableName = {value1,value2,..,valueN};
```

โดยที่ `value1,value2,..,valueN` เป็นค่าที่ต้องการกำหนดให้กับสมาชิกของอะเรย์แต่ละตัว ซึ่งจะต้องเป็นข้อมูลค่าคงที่ที่มีชนิดข้อมูลที่สอดคล้องกับชนิดข้อมูลของตัวแปรอะเรย์

ตัวอย่างเช่น

```
int []x = {4,3,5,1,8};
```

เป็นคำสั่งสร้างตัวแปรอะเรย์ x ซึ่งมีจำนวนสมาชิก 5 ตัว โดยที่ x[0],x[1], x[2],x[3] และ x[4] มีค่าเริ่มต้นเป็น 4, 3, 5, 1 และ 8 ตามลำดับ

การกำหนดค่าเริ่มต้นของตัวแปรอะเรย์ ทำได้เฉพาะในคำสั่งประกาศตัวแปรเท่านั้น ทั้งนี้เราไม่สามารถที่จะกำหนดค่าเริ่มต้นภายหลังจากคำสั่งประกาศตัวแปรอะเรย์ได้ กล่าวคือคำสั่ง

```
int  []x;
x[] = {4, 3, 5, 1, 8};  // illegal
```

เป็นคำสั่งที่ไม่ถูกต้อง

8.2.5 การใช้คำสั่ง for เพื่ออ้างอิงสมาชิกของอะเรย์

โดยทั่วไปเราจะใช้คำสั่ง for ในการอ้างอิงถึงสมาชิกของอะเรย์ที่ต้องเรียกใช้ในคำสั่งที่ซ้ำกันอาทิเช่น คำสั่ง

```
int x[] = {4,3,5,1,8};
```

จะเป็นการประกาศและสร้างตัวแปรอะเรย์ที่มีสมาชิกห้าตัว หากต้องการพิมพ์ค่าข้อมูลสมาชิกของอะเรย์แต่ละตัวโดยใช้คำสั่งดังนี้

```
System.out.println(x[0]);
System.out.println(x[1]);
System.out.println(x[2]);
System.out.println(x[3]);
System.out.println(x[4]);
```

จะเห็นได้ว่าเป็นการเรียกใช้คำสั่งที่ซ้ำกัน ซึ่งเราสามารถจะแทนที่ด้วยคำสั่ง for เพื่อให้โปรแกรมกระชับขึ้นดังนี้

```
for(int i = 0; i < 5; i++) {
       System.out.println(x[i]);
}
```

ทั้งนี้ภาษาจาวากำหนดให้ตัวแปรอะเรย์ทุกตัวมีคุณลักษณะ length เพื่อระบุจำนวนสมาชิกของอะเรย์แต่ละตัว ซึ่งตัวอย่างข้างต้นจะทำให้ x.length มีค่าเป็น 5 ดังนั้นส่วนของโปรแกรมข้างต้นสามารถเขียนใหม่ได้เป็น

```
for(int i = 0; i < x.length; i++) {
        System.out.println(x[i]);
}
```

นอกจากนี้เรายังสามารถที่จะเขียนคำสั่ง for ในรูปแบบที่เรียกว่า enhanced for เพื่อแจกแจงค่าของตัวแปร

อะเรย์โดยอัตโนมัติ ตัวอย่างเช่นคำสั่งข้างต้นสามารถเขียนใหม่อีกครั้งให้กระชับยิ่งขึ้นได้เป็น

```
for(int i : x) {
      System.out.println(x[i]);
}
```

โปรแกรมที่ 8.2 แสดงตัวอย่างการใช้คำสั่ง `for` กับตัวแปรอะเรย์เพื่อกำหนดค่าและพิมพ์ค่าของสมาชิกของอะเรย์แต่ละตัว ซึ่งผลลัพธ์ที่ได้จะเป็นเช่นเดียวกันกับผลลัพธ์ที่ได้จากโปรแกรมที่ 8.1

<u>โปรแกรมที่ 8.2</u> <u>ตัวอย่างการใช้คำสั่ง `for`</u>

```
public class ForWithArrays {
   public static void main(String args[]) {
      int []x;
      x = new int[4];
      for (int i=0; i<x.length; i++) {
         x[i] = i*2;
      }
      System.out.println("x = "+x);
      for (int i=0; i<x.length; i++) {
         System.out.println("x["+i+"] = "+x[i]);
      }
   }
}
```

8.2.6 ข้อผิดพลาดประเภท `ArrayIndexOutOfBoundsException`

การอ้างอิงถึงหมายเลขสมาชิกของอะเรย์ที่ไม่ถูกต้อง จะทำให้เกิดข้อผิดพลาดในตอนรันโปรแกรมโดยโปรแกรมจะส่งข้อผิดพลาด `ArrayIndexOutOfBoundsException` ออกมาในขณะรันโปรแกรม

ตัวอย่างเช่น คำสั่ง

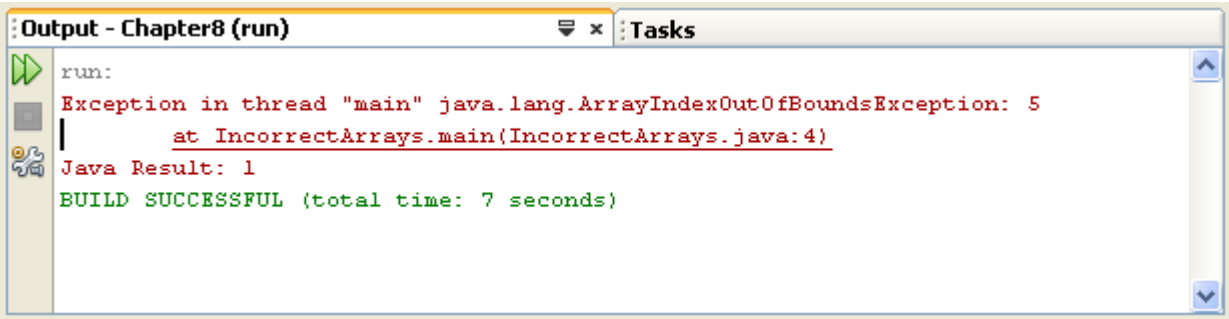
```
int []x = {4,3,5,1,8};
```

เป็นคำสั่งประกาศและสร้างตัวแปรอะเรย์ที่มีจำนวนสมาชิกตั้งแต่หมายเลข 0 ถึง 4 การอ้างอิงถึงสมาชิกตัวอื่น เช่น `x[5]` จะทำให้เกิดข้อผิดพลาดในตอนรันโปรแกรม

โปรแกรมที่ 8.3 แสดงตัวอย่างโปรแกรมที่มีการอ้างอิงสมาชิกของตัวแปรอะเรย์ที่ไม่ถูกต้อง ซึ่งผลลัพธ์ที่ได้เป็นดังแสดงในรูปที่ 8.4

<u>โปรแกรมที่ 8.3</u> <u>ตัวอย่างโปรแกรมที่จะส่งข้อผิดพลาดออกมา</u>

```
public class IncorrectArrays {
   public static void main(String args[]) {
      int []x = {4,3,5,1,8};
      System.out.println(x[5]);
   }
}
```



รูปที่ 8.4 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.3

8.3 อะเรย์ของข้อมูลชนิดคลาส

ตัวแปรอะเรย์ของข้อมูลชนิดคลาสคือ ตัวแปรที่ใช้เก็บกลุ่มของข้อมูลสมาชิกที่เป็น อ็อบเจกต์ของคลาสใดคลาสหนึ่ง ขั้นตอนการสร้างตัวแปรอะเรย์ของข้อมูลชนิดคลาสจะมีขั้นตอนคล้ายกับขั้นตอนการสร้างตัวแปรอะเรย์ของข้อมูลชนิดพื้นฐาน โดยมีขั้นตอนต่างๆ ดังนี้

1. ขั้นตอนแรกเป็นการประกาศตัวแปรอะเรย์ซึ่งจะจองเนื้อที่ในหน่วยความจำ เพื่อเก็บค่าของตำแหน่งอ้างอิงของตำแหน่งในหน่วยความจำของสมาชิกของอะเรย์แต่ละตัว
2. ขั้นตอนที่สองเป็นการใช้คำสั่ง `new` เพื่อสร้างและจองเนื้อที่ในหน่วยความจำเพื่อเก็บค่าของตำแหน่งอ้างอิงของสมาชิกของอะเรย์ซึ่งจะชี้ไปยังตำแหน่งที่เก็บข้อมูลจริงๆ
3. ขั้นตอนสุดท้ายเป็นการสร้างอ็อบเจกต์ของคลาสให้กับสมาชิกแต่ละตัวของอะเรย์โดยใช้คำสั่ง `new` ซึ่งจะเป็นการจองเนื้อที่ในหน่วยความจำเพื่อเก็บข้อมูลจริงๆ ของอ็อบเจกต์

ตัวอย่างเช่น ถ้าคลาส `Student` มีนิยามดังนี้

```
public class Student {
     private String name;
     public Student() {
            name = "NoName";
     }
     public Student(String n) {
            name = n;
     }
     public String getName() {
            return name;
     }
}
```

เราสามารถที่จะประกาศตัวแปรอะเรย์ s ให้มีชนิดข้อมูลเป็นคลาส Student ตามขั้นตอนแรก โดยใช้คำสั่งดังนี้

```
Student  []s;
```

ขั้นตอนที่สองเป็นการกำหนดขนาดและสร้างตัวแปรอะเรย์ ซึ่งหากต้องการให้มีจำนวนสมาชิก 3 ตัว จะทำได้โดยใช้คำสั่งดังนี้

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

ขั้นตอนที่สามจะเป็นการอ้างอิงสมาชิกของอะเรย์แต่ละตัว หรือสร้างอ็อบเจกต์ของคลาส Student โดยใช้คำสั่ง new เพื่อเป็นการเรียกใช้ constructor เหมือนกับการสร้างอ็อบเจกต์ทั่วไปดังนี้

```
s[0] = new Student("Thana");
s[1] = new Student("Somchai");
s[2] = new Student("Somsak");
```

อนึ่งเราสามารถที่จะใช้คำสั่ง for ในกรณีที่ต้องการจะสร้างอ็อบเจกต์ของสมาชิกแต่ละตัว โดยเรียกใช้ constructor แบบ default ได้ดังนี้

```
for(int i=0, i<s.length; i++) {
        s[i] = new Student();
}
```

นอกจากนี้เรายังสามารถที่จะรวมคำสั่งที่ใช้ในการประกาศ และสร้างตัวแปรอะเรย์ของข้อมูลชนิดคลาส และคำสั่งที่ใช้ในการสร้างอ็อบเจกต์ให้กับสมาชิกของอะเรย์แต่ละตัว ดังตัวอย่างต่อไปนี้

```
Student []s = {new Student("Thana");
               new Student("Somchai");
               new Student("Somsak")};
```

8.3.1 การเก็บค่าของตัวแปรอะเรย์ของข้อมูลชนิดคลาส

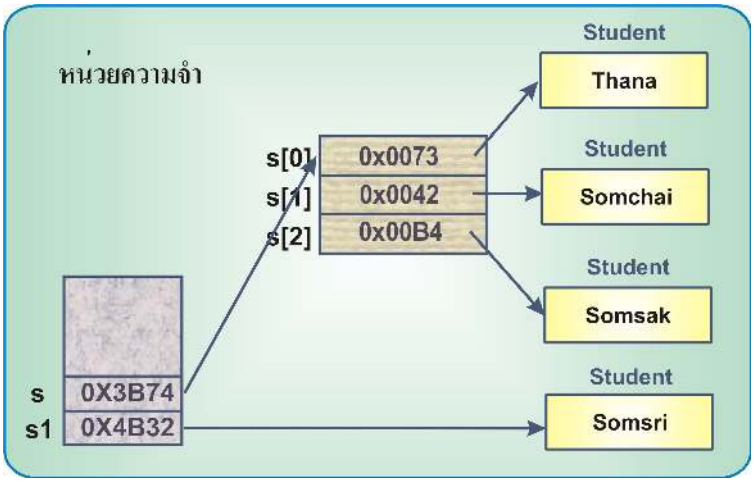
ตัวแปรของข้อมูลชนิดคลาสโดยทั่วไปจะเป็นตัวแปรแบบอ้างอิง อาทิเช่น คำสั่ง

```
Student s1 = new Student("Somsri");
```

จะเป็นการสร้างตัวแปรแบบอ้างอิง `s1` ซึ่งค่าที่ `s1` เก็บในหน่วยความจำจะเป็นตำแหน่งที่อ้างอิงไปยังเนื้อที่ในหน่วยความจำที่เก็บข้อมูลของอ็อบเจกต์ของคลาส `Student`

กรณีของตัวแปรอะเรย์ของข้อมูลชนิดคลาส ข้อมูลสมาชิกของอะเรย์แต่ละตัวก็จะเก็บตำแหน่งอ้างอิงไปยังเนื้อที่ในหน่วยความจำที่เก็บข้อมูลของอ็อบเจกต์ของคลาส `Student` แต่ละอ็อบเจกต์

รูปที่ 8.5 แสดงตัวอย่างการเก็บค่าในหน่วยความจำของตัวแปร `s1` และตัวแปรอะเรย์ `s` ที่สร้างขึ้นจากคำสั่งข้างต้น

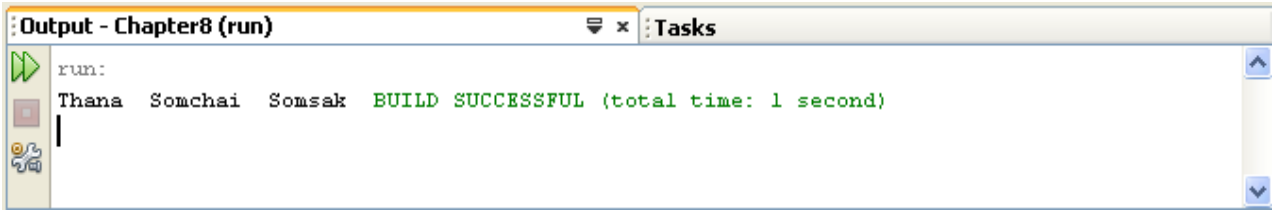


รูปที่ 8.5 ตัวอย่างการเก็บค่าของตัวแปรในหน่วยความจำ

โปรแกรมที่ 8.4 แสดงตัวอย่างโปรแกรมที่มีการสร้างตัวแปรอะเรย์ชนิดข้อมูลคลาส `Student` และมีการเรียกใช้เมธอด `getName()` ของอ็อบเจกต์แต่ละตัว ซึ่งผลลัพธ์ที่ได้เป็นดังแสดงในรูปที่ 8.6

โปรแกรมที่ 8.4 ตัวอย่างอะเรย์ของข้อมูลชนิดคลาส

```
public class Student {
      private String name;
      public Student(String n) {
         name = n;
      }
      public String getName() {
         return name;
      }
}

-------------------------------------------------------------
public class TestClassArrays {
    public static void main(String args[]) {
         Student []s  = {new Student("Thana"),
                         new Student("Somchai"),
                         new Student("Somsak")};
         for(int i=0; i<s.length; i++) {
               System.out.print(s[i].getName()+"  ");
         }
    }
}
```



รูปที่ 8.6 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่8.4

8.4 อะเรย์หลายมิติ

ภาษาจาวากำหนดให้มีตัวแปรอะเรย์ที่เป็นหลายมิติอาทิเช่น การกำหนดตัวแปรที่มีลักษณะเป็นเมตริกซ์ (matrix) หรือตารางจะต้องใช้ตัวแปรอะเรย์ที่เป็นสองมิติ การประกาศ ตัวแปรอะเรย์ที่มีขนาดมากกว่าหนึ่งมิติทำได้โดยการเพิ่มเครื่องหมาย [ ] ในแต่ละมิติ ดังนั้นรูปแบบการประกาศตัวแปรอะเรย์สองมิติมีดังนี้

```
            [modifier] dataType [][]variableName;
หรือ        [modifier] dataType variableName[][];
```

ตัวอย่างเช่น

```
int [][]x;
```

เป็นการประกาศตัวแปรอะเรย์สองมิติ x ซึ่งมีชนิดข้อมูลเป็น int

การสร้างตัวแปรอะเรย์หลายมิติจะต้องระบุจำนวนสมาชิกของอะเรย์ในแต่ละมิติโดยใช้คำสั่ง new ดังนั้นรูปแบบการสร้างตัวแปรอะเรย์สองมิติเป็นดังนี้

```
variableName = new dataType[row][col];
```

โดยที่

- row คือจำนวนสมาชิกในแต่ละแถว
- col คือจำนวนสมาชิกในแต่ละคอลัมน์

ตัวอย่างเช่น

```
x = new int[3][4];
```

เป็นการสร้างตัวแปรอะเรย์สองมิติ x ซึ่งมีขนาด 3 แถว 4 คอลัมน์

การเรียกใช้สมาชิกของอะเรย์สองมิติจะต้องระบุตำแหน่งของแถวและคอลัมน์ โดยมีรูปแบบดังนี้

```
variableName[row_number][col_number]
```

โดยที่

- row_number คือหมายเลขแถวของสมาชิกของอะเรย์สองมิติ
- col_number คือหมายเลขคอลัมน์ของสมาชิกของอะเรย์สองมิติ

ตัวอย่างเช่น

```
x[2][3]
```

หมายถึงสมาชิกของอะเรย์ x ตำแหน่งแถวที่ 2 คอลัมน์ที่ 3 เป็นต้น

8.4.1 การเขียนโปรแกรมเพื่อจัดการกับเมตริกซ์

โปรแกรมทางด้านคณิตศาสตร์ที่เกี่ยวข้องกับการจัดการเมตริกซ์ จะต้องใช้ตัวแปร อะเรย์ขนาดสองมิติ โปรแกรมที่ 8.5 เป็นตัวอย่างแสดงการบวกและคูณเมตริกซ์สองเมตริกซ์โดยการบวกเมตริกซ์สองเมตริกซ์นั้นเมตริกซ์ทั้งสองจะต้องมีขนาดเท่ากัน ผลลัพธ์ที่ได้จะเป็นการรวมสมาชิกแต่ละตัวของเมตริกซ์เข้าด้วยกัน อาทิเช่น

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{13}+b_{13} & a_{14}+b_{14} & a_{15}+b_{15} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} & a_{25}+b_{25} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} & a_{35}+b_{35} \\ a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} & a_{45}+b_{45} \\ a_{51}+b_{51} & a_{52}+b_{52} & a_{53}+b_{53} & a_{54}+b_{54} & a_{55}+b_{55} \end{bmatrix}$$

ส่วนการคูณเมตริกซ์ A กับเมตริกซ์ B นั้นจำนวนคอลัมน์ของเมตริกซ์ A จะต้องมีขนาดเท่ากับจำนวนแถวของเมตริกซ์ B โดยถ้าเมตริกซ์ C เป็นผลลัพธ์ที่ได้จากการคูณ เมตริกซ์ดังนี้

$$\begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \\ c_{31} & c_{32} & c_{33} & c_{34} & c_{35} \\ c_{41} & c_{42} & c_{43} & c_{44} & c_{45} \\ c_{51} & c_{52} & c_{53} & c_{54} & c_{55} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} \text{x} \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{bmatrix}$$

จะได้สมาชิกของเมตริกซ์ C แต่ละตัวมีค่าดังนี้

$$c_{ij} = a_{i1} x\ b_{1j} + a_{i2} x\ b_{2j} + a_{i3} x\ b_{3j} + a_{i4} x\ b_{4j} + a_{i5} x\ b_{5j}$$

โปรแกรมที่ 8.5 จะมีคำสั่ง `int[][]a = new int[5][5];` ที่ใช้ในการสร้างตัวแปรอะเรย์สองมิติ โดยมีค่าของสมาชิกของอะเรย์จากการสุ่มตัวเลขจำนวนระหว่าง 0 ถึง 9 จากคำสั่ง

```
a[i][j] = (int)(Math.random()*10);
```

โปรแกรมนี้จะใช้คำสั่ง `for` ที่มีโครงสร้างแบบซ้อนอยู่หลายที่ ทั้งนี้เพื่อช่วยในการจัดการประมวลผลข้อมูลของสมาชิกตัวแปรอะเรย์คลาส `Matrices` มีเมธอดดังนี้

- `addMatrices(int[][] m1, int[][] m2)` ใช้เพื่อคำนวณหาผลลัพธ์ที่ได้จากการบวกเมตริกซ์ โดยรับ argument เข้ามาเป็นตัวแปรอะเรย์สองมิติ 2 ตัว
- `mulMatrices(int[][] m1, int[][] m2)` ใช้เพื่อคำนวณหาผลลัพธ์ที่ได้จากการคูณเมตริกซ์ โดยรับ argument เข้ามาเป็นตัวแปรอะเรย์สองมิติ 2 ตัว
- `printMatrix(int[][] m)` ใช้เพื่อแสดงค่าสมาชิกแต่ละตัวของเมตริกซ์ที่รับเข้ามาเป็น argument

ตัวอย่างผลลัพธ์ที่ได้จากโปรแกรมนี้เป็นดังแสดงในรูปที่ 8.7

<u>โปรแกรมที่ 8.5</u> <u>ตัวอย่างการคำนวณเมตริกซ์</u>

```
public class CalMatrices {

    public static void main(String args[]) {
        int[][] a = new int[5][5];
        int[][] b = new int[5][5];
        int[][] c = new int[5][5];
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                a[i][j] = (int) (Math.random() * 10);
                b[i][j] = (int) (Math.random() * 10);
            }
        }
        Matrices mt = new Matrices();
        System.out.println("Matrix A:");
        mt.printMatrix(a);
        System.out.println("Matrix B:");
        mt.printMatrix(b);
        c = mt.addMatrices(a, b);
        System.out.println("Matrix A+B:");
        mt.printMatrix(c);
        c = mt.mulMatrices(a, b);
        System.out.println("Matrix A*B:");
        mt.printMatrix(c);
    }
}
```

```
public class Matrices {
  int [][]m = new int[5][5];
  public int[][] addMatrices(int[][] m1, int[][] m2) {
      for (int i=0; i<m1.length; i++) {
         for (int j=0; j<m1[i].length; j++) {
            m[i][j] = m1[i][j] + m2[i][j];
         }
      }
      return m;
  }
  public int[][] mulMatrices(int[][] m1, int[][] m2) {
      int sum;
      for (int i=0; i<m1.length; i++) {
         for (int j=0; j<m1[i].length; j++) {
            sum = 0;
            for (int k = 0;k < m1[i].length; k++) {
               sum += m1[i][k] * m2[k][j];
            }
            m[i][j] = sum;
         }
      }
      return m;
  }
  public void printMatrix(int[][] m) {
      for (int i=0; i<m.length; i++) {
         for (int j=0; j<m[i].length; j++) {
            System.out.print(m[i][j]+"\t");
         }
         System.out.println();
      }
  }
}
```

อนึ่งคุณลักษณะ `length` เมื่อนำมาใช้กับตัวแปรอะเรย์สองมิติจะให้ค่าดังนี้ `x.length` จะได้ค่าเท่ากับจำนวนแถวของตัวแปรอะเรย์ ส่วน `x[i].length` จะได้ค่าเท่ากับจำนวนคอลัมน์ในแถวที่ `i` ของตัวแปรอะเรย์

```
Output - Chapter8 (run)
run:
Matrix A:
2        7        2        2        3
9        1        5        8        5
1        0        8        1        9
6        1        4        4        3
7        6        9        0        5
Matrix B:
5        8        3        4        1
4        5        2        7        8
2        3        7        1        3
0        0        0        6        5
1        1        3        4        5
Matrix A+B:
7        15        5        6        4
13        6        7        15        13
3        3        15        2        12
6        1        4        10        8
8        7        12        4        10
Matrix A*B:
45        60        43        83        89
64        97        79        116        97
30        41        86        54        75
45        68        57        71        61
82        118        111        99        107
BUILD SUCCESSFUL (total time: 1 second)
```

รูปที่ 8.7 ตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.5

8.4.2 อะเรย์สองมิติที่มีจำนวนคอลัมน์ต่างกัน

ภาษาจาวาอนุญาตให้มีการสร้างอะเรย์สองมิติที่มีขนาดของคอลัมน์ในแต่ละแถวไม่เท่ากันได้ โดยจะต้องระบุจำนวนแถวโดยใช้คำสั่ง `new` ก่อน แล้วระบุจำนวนคอลัมน์ในแต่ละแถว ตัวอย่างเช่นคำสั่ง

```
int [][]x;
x    = new int[3][];
x[0] = new int[4]
x[1] = new int[2];
x[2] = new int[3]
```

เป็นการสร้างตัวแปรอะเรย์ `x` ซึ่งมีสมาชิกจำนวน 3 แถว โดยที่แถวที่หนึ่งมี 4 คอลัมน์ แถวที่สองมี 2 คอลัมน์ และแถวที่สามมี 3 คอลัมน์

โปรแกรมที่ 8.6 แสดงการสร้างตัวแปรอะเรย์สองมิติที่แต่ละแถวมีจำนวนคอลัมน์ต่างกัน โปรแกรมนี้จะได้ผลลัพธ์ดังแสดงในรูปที่ 8.8 และรูปที่ 8.9 แสดงข้อมูลที่อยู่ในสมาชิกแต่ละตัวของอะเรย์

<u>โปรแกรมที่ 8.6</u> <u>ตัวอย่างการสร้างอะเรย์สองมิติที่แต่ละแถวมีจำนวนคอลัมน์ต่างกัน</u>

```
public class TwoDimensionArrays {
   public static void main(String args[]) {
      int x[][] = new int[3][];
      x[0] = new int[4];
      x[1] = new int[2];
      x[2] = new int[3];
      for(int i=0; i<x.length; i++) {
         for(int j=0; j<x[i].length; j++) {
            x[i][j] = (i+j)*2;
         }
      }
      for(int i=0; i<x.length; i++) {
         for(int j=0; j<x[i].length; j++) {
            System.out.print(x[i][j]+" ");
         }
         System.out.println();
      }
   }
}
```

```
Output - Chapter8 (run)
run:
0 2 4 6
2 4
4 6 8
BUILD SUCCESSFUL (total time: 0 seconds)
```

รูปที่ 8.8 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.6

		0	1	2	3
	0	0	2	4	6
x	1	2	4		
	2	4	6	8	

รูปที่ 8.9 แสดงตัวอย่างข้อมูลที่อยู่ในอะเรย์ ซึ่งเป็นผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.6

8.4.3 เมธอดที่เกี่ยวข้องกับอะเรย์

ภาษาจาวามีเมธอดหลายเมธอดที่รับ argument เป็นอะเรย์ของข้อมูลชนิดต่างๆ ตัวอย่างเช่นแพคเก็จ `java.lang` มีคลาสที่ชื่อ `Arrays` ซึ่งมีเมธอดที่เกียวข้องกับอะเรย์ที่สำคัญดังนี้

- `sort()` เป็นเมธอดที่ใช้ในการเรียงค่าข้อมูลสมาชิกของอะเรย์จากน้อยไปมาก
- `binarySearch()` เป็นเมธอดที่ใช้ในการค้นหาค่าข้อมูลที่ต้องการจากชุดข้อมูลทั้งหมดของสมาชิกของอะเรย์ ซึ่งถูกเรียงจากน้อยไปมากหรือมากไปน้อยไว้แล้ว
- `fill()` เป็นเมธอดที่ใช้ในการกำหนดค่าข้อมูลเดียวกันให้กับสมาชิกทั้งหมดของอะเรย์

เนื่องจากเมธอดทั้งสามเมธอดที่กล่าวมาข้างต้นเป็นเมธอดแบบ `static` ดังนั้นการเรียกใช้งานเมธอดเหล่านี้จึงสามารถเรียกโดยใช้ชื่อคลาส `Arrays` ได้เลย ไม่จำเป็นต้องสร้างอ็อบเจกต์ของคลาส `Arrays` ขึ้นมาก่อนการเรียกใช้งาน

โปรแกรมที่ 8.7 แสดงตัวอย่างการเรียกใช้เมธอดของคลาส `Arrays` ที่กล่าวไว้ข้างต้น โปรแกรมนี้เรียกใช้เมธอด `sort()` โดยส่ง argument ที่เป็นตัวแปรอะเรย์ `d` ไปเพื่อเรียงค่าข้อมูลของสมาชิกของตัวแปรอะเรย์ `d` จากน้อยไปมาก เมธอด `binarySearch()` จะใช้ในการค้นหาสมาชิกของตัวแปรอะเรย์ `d` ที่มีค่าเท่ากับ 1.65 ส่วนเมธอด `fill()` ใช้ในการกำหนดค่าของสมาชิกของตัวแปรอะเรย์ `d` ทุกตัวให้มีค่าเป็น 1.0 โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 8.10

ภาษาจาวาไม่ยอมให้มีการเปลี่ยนแปลงขนาดของอะเรย์ ดังนั้นการใช้คำสั่ง `new` เพื่อประกาศขนาดของตัวแปรอะเรย์ใหม่ จะทำให้ค่าของข้อมูลเดิมหายไป ตัวอย่างเช่นคำสั่ง

```
int []x = {4,7,9};
x = new int[4];
```

จะเป็นคำสั่งในการประกาศตัวแปรอะเรย์ `x` และสร้างอะเรย์พร้อมกับกำหนดค่าเริ่มต้นให้กับสมาชิกทั้ง 3 ตัวของอะเรย์ โดยมีค่าข้อมูลเป็น 4, 7 และ 9 ตามลำดับ แต่คำสั่งถัดมาจะเป็นการสร้างอะเรย์ `x` ขึ้นมาใหม่โดยไม่สามารถนำค่าของข้อมูลเดิมกลับมาได้ จึงทำให้ค่าข้อมูลสมาชิกของอะเรย์ทั้ง 4 ตัวมีค่าเท่ากับ 0 ทั้งหมด

โปรแกรมที่ 8.7 ตัวอย่างการเรียกใช้เมธอดของคลาส `Arrays`

```
import java.util.Arrays;

public class MethodsArrays {
   public static void main(String args[]) {
      double d[] = {5.3, 3.56, 0.5, 1.65, 7.8};
      Arrays.sort(d);
      for(int i=0; i<d.length; i++) {
         System.out.print(d[i]+" ");
      }
      System.out.println();
      int pos = Arrays.binarySearch(d,1.65);
      System.out.println("arrays["+pos+"] = 1.65");
      Arrays.fill(d,1.0);
      for(int i=0; i<d.length; i++) {
         System.out.print(d[i]+" ");
      }
      System.out.println();
   }
}
```

```
Output - Chapter8 (run)
run:
0.5 1.65 3.56 5.3 7.8
arrays[1] = 1.65
1.0 1.0 1.0 1.0 1.0
BUILD SUCCESSFUL (total time: 1 second)
```

รูปที่ 8.10 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.7

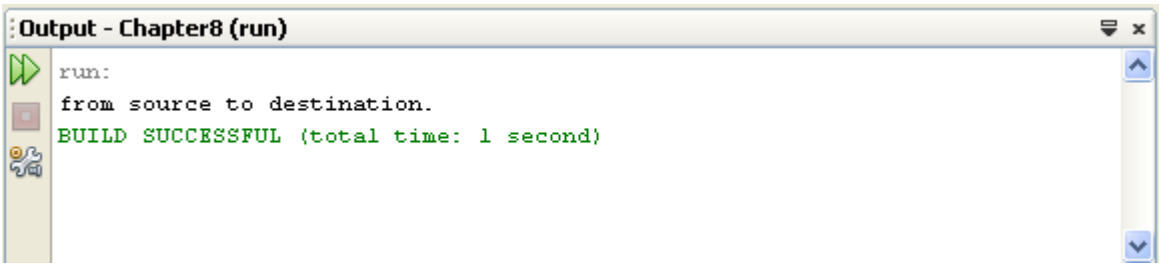
ภาษาจาวามีคำสั่ง `System.arraycopy()` ใช้สำหรับคัดลอกค่าข้อมูลของสมาชิกของอะเรย์โดยมีโครงสร้างรูปแบบคำสั่งดังนี้

```
System.arraycopy(Object src,int src_pos,Object dst,int dst_pos,int length);
```

โปรแกรมที่ 8.8 แสดงตัวอย่างการใช้คำสั่ง `System.arrayCopy()` โดยส่ง argument ซึ่งเป็นตัวแปรอะเรย์ `scr` ซึ่งเป็นอะเรย์ต้นแบบ เริ่มจากค่าข้อมูลของสมาชิกลำดับที่ 3 ของอะเรย์ `src` ไปยังสมาชิกลำดับที่ 0 ของอะเรย์ `dst` โดยให้ทำการคัดลอกข้อมูลทั้งสิ้น 4 ค่าข้อมูล โดยโปรแกรมนี้จะได้ผลลัพธ์ดังแสดงในรูปที่ 8.11

โปรแกรมที่ 8.8 ตัวอย่างการใช้คำสั่ง `System.arraycopy();`

```
public class CopyArrays {
   public static void main(String args[]) {
      String []scr = {"Copy","an","array","from",
               " source"," to"," destination."};
      String []dst = new String[4];
      System.arraycopy(scr,3,dst,0,4);
      for(int i=0; i<dst.length; i++) {
         System.out.print(dst[i]);
      }
      System.out.println();
   }
}
```



รูปที่ 8.11 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.8

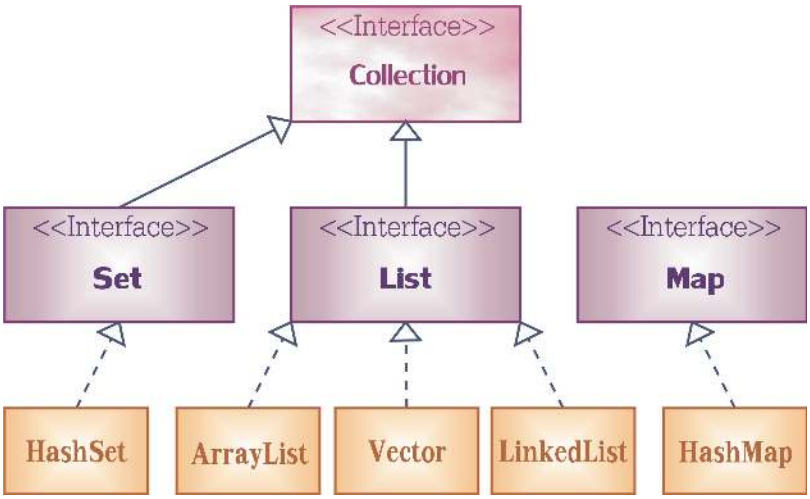
8.5 คอลเล็กชั่น

ตัวแปรอะเรย์เป็นชนิดข้อมูลแบบอ้างอิง ซึ่งเมื่อมีการสร้างอะเรย์โดยใช้คำสั่ง `new` แล้วจะไม่สามารถเปลี่ยนแปลงขนาดของจำนวนสมาชิกที่เก็บในอะเรย์ได้ ในกรณีที่ต้องการจะเก็บกลุ่มของข้อมูลที่สามารถเปลี่ยนแปลงขนาดของจำนวนสมาชิกได้ ภาษาจาวาจะใช้คลาสประเภทคอลเล็กชั่น (Collection) ซึ่งเป็นคลาสที่ใช้ในการเก็บกลุ่มของข้อมูลชนิดคลาสที่ชื่อ `Object` ซึ่งคลาสทุกๆ คลาสในภาษาจาวาจะสืบทอดมาจากคลาสที่ชื่อ `Object` นี้ ดังนั้นคลาสประเภทคอลเล็กชั่นสามารถเก็บอ็อบเจกต์ของคลาสใดๆ ในภาษาจาวาได้ตามหลักการ Dynamic Binding ของการมีได้หลายรูปแบบ ที่ได้กล่าวถึงไปในบทที่ 4

คลาสประเภทคอลเล็กชั่นถูกกำหนดไว้ใน Java API ที่ชื่อ Collection API โดยจะประกอบไปด้วยอินเตอร์เฟสและคลาสต่างๆ ดังแสดงในรูปที่ 8.12 Collection API ได้กำหนดอินเตอร์เฟสต่างๆ ที่มีเงื่อนไขในการใส่ข้อมูลของสมาชิกของคอลเล็กชั่นที่แตกต่างกันดังนี้

- `Collection` เป็นอินเตอร์เฟสที่กำหนดเมธอดในการจัดการข้อมูลของสมาชิก สำหรับคลาสประเภทคอลเล็กชั่นที่จะต้อง `implements` อินเตอร์เฟสนี้
- `Set` เป็นอินเตอร์เฟสที่ใช้ `implements` คลาสประเภทคอลเล็กชั่นที่มีสมาชิกที่มีข้อมูลไม่ซ้ำกัน และไม่มีลำดับการใส่ข้อมูล
- `List` เป็นอินเตอร์เฟสที่ใช้ `implements` คลาสประเภทคอลเล็กชั่นที่มีสมาชิกซึ่งอาจมีข้อมูลซ้ำกันได้ แต่จะมีลำดับของสมาชิกของการใส่ข้อมูล
- `Map` เป็นอินเตอร์เฟสที่ใช้ `implements` คลาสประเภทคอลเล็กชั่นซึ่งสมาชิกจะมีองค์ประกอบสองส่วน คือ ข้อมูลและคีย์ (key) ข้อมูลของสมาชิกประเภทนี้อาจซ้ำกันได้ แต่คีย์ของสมาชิกแต่ละตัวจะต้องไม่ซ้ำกัน

คลาสใน Collection API ที่ `implements` อินเตอร์เฟสเหล่านี้คือ `HashSet, ArrayList, LinkedList, Vector` และ `HashMap`



รูปที่ 8.12 อินเตอร์เฟสและคลาสต่างๆ ใน Collection API

8.5.1 อินเตอร์เฟส `Collection`

`Collection` เป็นชื่ออินเตอร์เฟสที่กำหนดไว้ใน Collection API โดยมีเมธอดที่สำคัญดังนี้

- `boolean add (Object element)`

  เป็นเมธอดที่ใช้ในการใส่สมาชิกลงในคอลเล็กชั่น โดยสมาชิกที่จะใส่ต้องเป็นอ็อบเจกต์ของคลาส

ใดคลาสหนึ่ง เมธอดนี้จะส่งค่ากลับมาเป็นชนิดข้อมูลแบบ `boolean` โดยจะให้ค่าเป็น `true` ถ้าสามารถใส่ข้อมูลได้

- `boolean remove (Object element)`

  เป็นเมธอดที่ใช้ในการลบสมาชิกออกจากคอลเล็กชั่น โดยต้องส่งผ่าน argument ที่เป็นอ็อบเจกต์ที่ต้องการลบออก เมธอดนี้จะส่งค่ากลับมาเป็นชนิดข้อมูลแบบ `boolean` โดยจะให้ค่าเป็น `true` ในกรณีที่ลบข้อมูลได้

- `int size()`

  เป็นเมธอดที่ใช้ในการหาจำนวนสมาชิกที่มีอยู่ในคอลเล็กชั่น

- `boolean isEmpty()`

  เป็นเมธอดที่ใช้ในการตรวจว่าคอลเล็กชั่นมีสมาชิกอยู่หรือไม่ โดยจะส่งค่ากลับมาเป็น `true` ถ้าไม่มีสมาชิกอยู่ในคอลเล็กชั่น

- `boolean contains(Object element)`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่า คอลเล็กชั่นมีสมาชิกที่มีค่าเป็นอ็อบเจกต์ของ argument ที่ส่งผ่านมาหรือไม่

- `Iterator iterator()`

  เป็นเมธอดที่ใช้ในการแจกแจงข้อมูลของสมาชิกในคอลเล็กชั่น

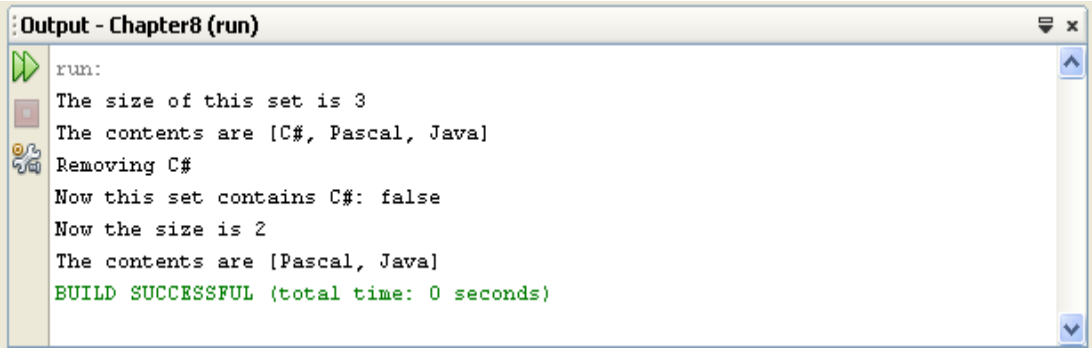
8.5.2 อินเตอร์เฟส `Set`

`Set` เป็นอินเตอร์เฟสที่สืบทอดมาจากอินเตอร์เฟส `Collection` โดยมีคลาสที่สำคัญที่ `implements` อินเตอร์เฟสนี้คือคลาส `HashSet` คลาส `HashSet` ใช้ในการสร้างอ็อบเจกต์ประเภทคอลเล็กชั่นซึ่งจะมีสมาชิกของข้อมูลซ้ำกันไม่ได้ โปรแกรมที่ 8.9 แสดงตัวอย่างการใช้คลาส `HashSet` โปรแกรมนี้จะต้องมีคำสั่ง `import java.util.*;` เพื่อที่จะเรียกใช้คลาสต่างๆ ที่กำหนดใน Collection API โปรแกรมนี้ได้สร้างอ็อบเจกต์ `s` ซึ่งเป็นอ็อบเจกต์ของคลาส `HashSet` และเรียกใช้เมธอด `add()` ในการใส่ข้อมูลลงในอ็อบเจกต์ `s` โดยปกติแล้ว argument ของเมธอด `add()` จะต้องเป็น อ็อบเจกต์ของคลาสที่ชื่อ `Object` แต่โปรแกรมนี้จะส่ง argument ที่เป็นอ็อบเจกต์

ของคลาส String ซึ่งสามารถทำได้เนื่องจากคลาส String เป็นคลาสที่สืบทอดมาจากคลาสที่ชื่อ Object โปรแกรมนี้ต้องการจะแสดงให้เห็นว่าเราไม่สามารถที่จะใส่ข้อมูลที่ซ้ำกัน (ข้อความ “Java”) ลงในอ็อบเจกต์ชนิด HashSet ได้ ซึ่งโปรแกรมนี้จะได้ผลลัพธ์ดังแสดงในรูปที่ 8.13

<u>โปรแกรมที่ 8.9</u> <u>ตัวอย่างการใช้คลาส HashSet</u>

```
import java.util.*;
public class SampleSet {
   public static void main(String args[]) {
      Set s = new HashSet();
      s.add("C#");
      s.add("Java");
      s.add("Pascal");
      System.out.println("The size of this set is " + s.size());
      System.out.println("The contents are " + s);
      System.out.println("Removing C#");
      s.remove("C#");
      System.out.println("Now this set contains C#: " + s.contains("C#"));
      s.add("Java");
      System.out.println("Now the size is " + s.size());
      System.out.println("The contents are " + s);
   }
}
```



รูปที่ 8.13 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.9

8.5.3 อินเตอร์เฟส List

List เป็นอินเตอร์เฟสที่สืบทอดมาจากอินเตอร์เฟส Collection แต่จะแตกต่างจากอินเตอร์เฟส Set ตรงที่จะมีลำดับของสมาชิกอยู่ (index) และสามารถที่มีสมาชิกซึ่งมีข้อมูลซึ่งซ้ำกัน ซึ่งอินเตอร์เฟส List จะเพิ่มเมธอดที่

เกี่ยวข้องกับข้อมูลลำดับของสมาชิก ดังนี้

- `void add(int index,Object element)`

  เป็นเมธอดในการใส่สมาชิกลงในคอลเล็กชั่นที่เป็นอ็อบเจกต์ที่มีค่าเป็น argument ที่ส่งผ่าน โดยมีลำดับที่เป็นเลขจำนวนเต็มใน argument ที่ชื่อ `index`

- `Object remove(int index)`

  เป็นเมธอดในการลบสมาชิกลำดับที่ซึ่งมีเลขจำนวนเต็มใน argument ที่ชื่อ `index` ออกจากคอลเล็กชั่น

- `Object get(int index)`

  เป็นเมธอดนี้ใช้ในการเรียกข้อมูลของสมาชิกลำดับที่มีเลขจำนวนเต็มใน argument ที่ชื่อ `index`

- `int indexOf(Object element)`

  เป็นเมธอดนี้ใช้ในการตรวจสอบว่าอ็อบเจกต์ที่มีค่าใน argument ที่ส่งผ่านมาเป็นสมาชิกลำดับที่เท่าไรของคอลเล็กชั่น

- `ListIterator listIterator()`

  เป็นเมธอดนี้ใช้ในการแจกแจงข้อมูลของสมาชิกในคอลเล็กชั่นแบบ `List`

คลาสสำคัญที่ `implements` อินเตอร์เฟส `List` ซึ่งระบุไว้ใน Collection API คือคลาส `LinkedList` และ `ArrayList` โปรแกรมที่ 8.10 แสดงตัวอย่างการใช้คลาส `LinkedList` โปรแกรมนี้จะมีคำสั่ง `l.get(0)` เป็นการเรียกข้อมูลของสมาชิกลำดับที่ 0 ในอ็อบเจกต์ `l` ที่เป็นอ็อบเจกต์ของคลาส `LinkedList` ส่วนคำสั่ง `l.indexOf("Java")` เป็นคำสั่งเรียกดูลำดับที่ของสมาชิกในอ็อบเจกต์ `l` ซึ่งมีข้อมูลเป็นอ็อบเจกต์ `String` ที่มีข้อความใน `"Java"` โปรแกรมนี้แสดงตัวอย่างของข้อแตกต่างระหว่างคลาสประเภท `Set` และ `List` ทั้งนี้จะเห็นได้จากคำสั่ง `l.add("Java")` สามารถที่จะใส่ข้อมูลที่ซ้ำกันได้โดยโปรแกรมจะได้ผลลัพธ์ดังแสดงในรูปที่ 8.14

โปรแกรมที่ 8.10 ตัวอย่างการใช้คลาส `LinkedList`

```
import java.util.*;

public class SampleList {
   public static void main(String args[]) {
      List l = new LinkedList();
      l.add("C#");
      l.add("Java");
      l.add("Pascal");
      System.out.println("The size is "+l.size());
      System.out.println("The contents are "+l);
      System.out.println("The first one is "+l.get(0));
      l.add("Java");
      System.out.println("The contents are "+l);
      System.out.println("The index of Java is "+
      l.indexOf("Java"));
   }
}
```

```
Output - Chapter8 (run)
run:
The size is 3
The contents are [C#, Java, Pascal]
The first one is C#
The contents are [C#, Java, Pascal, Java]
The index of Java is 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

รูปที่ 8.14 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.10

8.5.4 อินเตอร์เฟส `Map`

`Map` เป็นอินเตอร์เฟสที่สืบทอดมาจากอินเตอร์เฟส `Collection` ซึ่งสมาชิกแต่ละตัวของคอลเล็กชั่นที่เป็น `Map` จะมีข้อมูลอยู่สองส่วนคือ ส่วนที่เป็นคีย์และส่วนที่เป็นข้อมูลซึ่งเป็นอ็อบเจกต์ของคลาสใดๆ คีย์จะทำหน้าที่ช่วยในการสืบค้นส่วนที่เป็นข้อมูลสมาชิกของ `Map` โดยจะต้องมีคีย์ที่ไม่ซ้ำกันแต่อาจมีส่วนที่เป็นข้อมูลซ้ำกันได้ อินเตอร์เฟส `Map` จะมีเมธอดต่างๆ ที่เพิ่มขึ้นมาเพื่อช่วยสนับสนุนการทำงานของคีย์ดังนี้

- `Object put(Object key, Object value)`

  เป็นเมธอดที่ใช้ในการใส่สมาชิกลงในคอลเล็กชั่น โดยต้องใส่อ็อบเจกต์ทั้งส่วนที่เป็นคีย์และส่วนที่เป็นข้อมูลตาม argument ที่ชื่อ `key` และ `value` ตามลำดับ

- `Object remove(Object key)`

  เป็นเมธอดที่ใช้ในการลบสมาชิกออกจากคอลเล็กชั่น โดยสมาชิกที่ถูกลบจะมีค่าของคีย์เป็นอ็อบเจกต์ที่มีค่าตาม argument ที่ชื่อ `key`

- `Object get(Object key)`

  เป็นเมธอดที่ใช้ในการเรียกดูข้อมูลของสมาชิกในคอลเล็กชั่นที่มีค่าของคีย์เป็นค่าของอ็อบเจกต์ที่ชื่อ `key` ที่ผ่านมายัง argument

- `Set entrySet()`

  เป็นเมธอดที่ใช้เรียกดูข้อมูลของสมาชิกทั้งหมดในคอลเล็กชั่น

- `Set keySet()`

  เป็นเมธอดที่ใช้เรียกดูคีย์ของสมาชิกทั้งหมดในคอลเล็กชั่น

- `int size()`

  เป็นเมธอดที่ใช้ในการหาจำนวนสมาชิกของคอลเล็กชั่น

คลาสที่สำคัญที่ `implements` อินเตอร์เฟส `Map` คือคลาส `HashMap` โปรแกรมที่ 8.11 แสดงตัวอย่างการสร้างอ็อบเจกต์ของคลาสนี้แล้วเรียกใช้เมธอด `put()` ในการใส่สมาชิกลงในคอลเล็กชั่น ทั้งนี้เมธอดนี้จะต้องส่งผ่าน argument สองตัว ในที่นี้ทั้งสองตัวจะส่งผ่านอ็อบเจกต์ชนิด `String` เมธอด `remove()` ในที่นี้จะใช้ในการลบสมาชิกที่มีค่าของคีย์เป็น `String` ที่มีค่าเป็น “3” โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 8.15

โปรแกรมที่ 8.11 ตัวอย่างการใช้คลาส `HashMap`

```
import java.util.*;

public class SampleMap {
   public static void main(String args[]) {
      Map m = new HashMap();
      m.put("1","C#");
      m.put("2","Java");
      m.put("3","Pascal");
      System.out.println("Removing Pascal");
      m.remove("3");
      System.out.println("The size is "+m.size());
      System.out.println("The first one is "+
      m.get("1"));
      m.put("3","Java");
      System.out.println("The key of this map are "+
      m.keySet());
      System.out.println("The contents are "+ m.entrySet());
   }
}
```

```
Output - Chapter8 (run)
run:
Removing Pascal
The size is 2
The first one is C#
The key of this map are [3, 2, 1]
The contents are [3=Java, 2=Java, 1=C#]
BUILD SUCCESSFUL (total time: 0 seconds)
```

รูปที่ 8.15 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.11

8.5.5 อินเตอร์เฟส `Iterator`

`Iterator` เป็นอินเตอร์เฟสที่ใช้ในการเรียกดูสมาชิกของคอลเล็กชั่น ทั้งนี้อินเตอร์เฟส `Collection` จะมีเมธอด `iterator()` ซึ่งจะส่งอ็อบเจกต์ของคลาสประเภท `Iterator` กลับคืนมา อินเตอร์เฟส `Iterator` จะมีเมธอดต่างๆ ที่ใช้ในการเรียกดูข้อมูลดังนี้

- `boolean hasNext()`

  เป็นเมธอดที่จะตรวจสอบว่ายังมีข้อมูลอยู่ใน `Iterator` อีกหรือไม่

- `Object next()`

  เป็นเมธอดที่จะเรียกดูค่าของอ็อบเจกต์ของสมาชิกตัวถัดไปของ `Iterator` โดยจะส่งค่ากลับมาเป็นอ็อบเจกต์ของคลาสที่ชื่อ `Object`

- `void remove()`

  เป็นเมธอดที่ใช้ในการลบสมาชิกตำแหน่งปัจจุบันออกจาก `Iterator`

โดยทั่วไปตำแหน่งเริ่มต้นของ `Iterator` จะชี้ไปที่ตำแหน่งก่อนสมาชิกของ `Iterator` ตัวแรกและจะใช้เมธอด `next()` ในการเรียกดูสมาชิกตัวต่อๆ ไป โปรแกรมที่ 8.12 แสดงตัวอย่างการใช้อินเตอร์เฟส `Iterator` ในการเรียกดูข้อมูลของสมาชิกของอ็อบเจกต์ชนิด `ArrayList` โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 8.16

<u>โปรแกรมที่ 8.12</u> <u>ตัวอย่างการใช้อินเตอร์เฟส `Iterator`</u>

```
import java.util.*;

public class SampleIterator {
   public static void main(String args[]) {
      Set scrSet = new HashSet();
      scrSet.add("C#");
      scrSet.add("Java");
      scrSet.add("Pascal");
      Iterator it = scrSet.iterator();
      Set dstSet = new HashSet();
      for(int i=0; i<scrSet.size(); i++) {
         if(it.hasNext()) {
            dstSet.add(it.next());
         }
      }
      System.out.println(dstSet);
   }
}
```

```
Output - Chapter8 (run)
run:
[C#, Pascal, Java]
BUILD SUCCESSFUL (total time: 1 second)
```

รูปที่ 8.16 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.12

Collection API ยังมีอินเตอร์เฟสประเภท `Iterator` ที่สำคัญอีกสองตัวคือ `ListIterator` และ `Enumeration` อินเตอร์เฟส `ListIterator` จะใช้ในการเรียกดูข้อมูลของคอลเล็กชั่นประเภท `List` โดยมีเมธอดที่สำคัญที่เพิ่มมาจากเมธอดของอินเตอร์เฟส `Iterator` ดังนี้

- `boolean hasPrevious()`

  เป็นเมธอดที่จะตรวจสอบว่ามีข้อมูลในตำแหน่งก่อนหน้านี้ใน `ListIterator` หรือไม่

- `Object previous()`

  เป็นเมธอดที่จะเรียกดูค่าของอ็อบเจกต์ของสมาชิก ในตำแหน่งก่อนหน้านี้ของ `ListIterator` โดยจะส่งค่ากลับมาเป็นอ็อบเจกต์ของคลาสที่ชื่อ `Object`

- `void add(Object element)`

  เป็นเมธอดในการใส่สมาชิกตัวใหม่ลงใน `ListIterator`

- `void set(Object element)`

  เป็นเมธอดที่ใช้ในการแทนค่าสมาชิกของ `ListIterator` ในตำแหน่งปัจจุบันด้วยค่าที่ส่งผ่านมาทาง argument

อินเตอร์เฟส `List` จะมีเมธอด `listIterator()` ซึ่งเป็นเมธอดที่ส่งค่ากลับเป็นอ็อบเจกต์ประเภท `ListIterator()` เพื่อใช้ในการแจกแจงข้อมูลของสมาชิกของ คอลเล็กชั่นประเภท `List`

อินเตอร์เฟส `Enumeration` จะมีลักษณะคล้ายกับอินเตอร์เฟส `Iterator` โดยจะใช้ในการแจกแจงของข้อมูลของคอลเล็กชั่นต่างๆ อินเตอร์เฟสนี้จะมีเมธอดที่สำคัญสองเมธอดคือ

- `boolean hasMoreElement()`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่ายังมีสมาชิกใน `Enumeration` อีกหรือไม่

- `Object nextElement()`

  เป็นเมธอดที่ใช้ในการเรียกดูค่าของสมาชิกใน `Enumeration` ที่อยู่ในตำแหน่งถัดไป

โดยทั่วไปการเรียกดูค่าของสมาชิกของอ็อบเจกต์ประเภท Enumeration จะมีรูปแบบของคำสั่งดังนี้

```
while(e.hasMoreElements()){
      System.out.print(e.nextElement() + " ");
}
```

8.5.6 คลาส Vector

Vector เป็นคลาสประเภทคอลเล็กชั่นที่กำหนดไว้ใน Collection API คลาส Vector ใช้เก็บกลุ่มของอ็อบเจกต์ของคลาสใดๆ โดยไม่จำกัดจำนวน คลาส Vector เป็นคลาสที่ implements อินเตอร์เฟส List เราสามารถสร้างอ็อบเจกต์ของคลาส Vector โดยเรียกใช้ constructor ในรูปแบบต่างๆ ดังนี้

- Vector()
- Vector(int initialCapacity)
- Vector(int initialCapacity, int capacityIncrement)

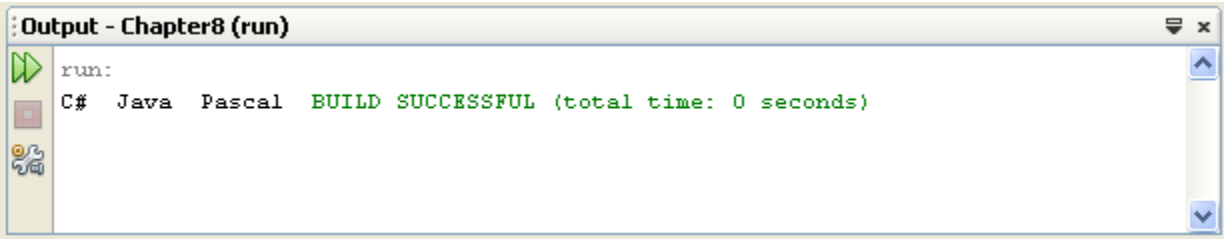
โดยที่

- initialCapacity คือขนาดเริ่มต้นของ Vector
- capacityIncrement คือขนาดที่จะเพิ่มขึ้นเมื่อ Vector มีขนาดเต็มแล้ว

โปรแกรมที่ 8.13 เป็นตัวอย่างการใช้อ็อบเจกต์ของคลาส Vector และใช้อินเตอร์เฟส Enumeration ในการเรียกดูข้อมูล โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 8.17

<u>โปรแกรมที่ 8.13</u> <u>ตัวอย่างการใช้อ็อบเจกต์ของคลาส Vector</u>

```
import java.util.*;

public class SampleEnumeration {
   public static void main(String args[]) {
      Vector    v = new Vector();
      v.add("C#");
      v.add("Java");
      v.add("Pascal");
      Enumeration e = v.elements();
      while (e.hasMoreElements()) {
            System.out.print(e.nextElement()+"  ");

      }
   }
}
```



รูปที่ 8.17 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.13

8.5.7 การใช้คำสั่ง for และ Generic

เราสามารถที่จะใช้คำสั่ง Generic ในการกำหนดชนิดข้อมูลของอ็อบเจกต์ที่อยู่ในคอลเล็กชั่น อาทิเช่นถ้าต้องการกำหนดให้อ็อบเจกต์ `myList` ชนิด `LinkedList` เป็นอ็อบเจกต์ที่เก็บข้อมูลชนิด `String` เราสามารถที่จะประกาศอ็อบเจกต์นี้โดยใช้คำสั่ง

```
List<String> myList = new LinkedList<String>();
```

และเราสามารถที่จะใช้คำสั่ง `for` ในการที่จะแจกแจงค่าของอ็อบเจกต์ประเภทคอลเล็กชั่นแทนที่จะใช้อินเตอร์เฟส `Iterator` อาทิเช่นอ็อบเจกต์ `myList` ที่เป็นมีค่าของข้อมูลภายในคอลเล็กชั่นเป็น `String` สามารถที่จะแจงข้อมูลโดยใช้คำสั่ง `for` ดังนี้

```
for (String stringList : myList) {
      System.out.println(stringList);
}
```

โปรแกรมที่ 8.14 เป็นตัวอย่างการใช้ Generic ในการสร้างอ็อบเจกต์ของคลาส `HashSet` ที่จะใส่อ็อบเจกต์ชนิด `Integer` และใช้คำสั่ง for ในการแจกแจงข้อมูล โปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 8.18

โปรแกรมที่ 8.14 ตัวอย่างการใช้ Generic

```
public class SampleGeneric {

    public static void main(String args[]) {
        Set<Integer> scrSet = new HashSet<Integer>();
        int[] myInt = {1, 3, 5, 7, 11};
        for (int i : myInt) {
            scrSet.add(i);
        }
        for (Integer num : scrSet) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

```
Output - Chapter8 (run)
run:
[C#, Pascal, Java]
BUILD SUCCESSFUL (total time: 1 second)
```

รูปที่ 8.18 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 8.14

สรุปเนื้อหาของบท

- โดยทั่วไปโครงสร้างข้อมูลแบบอะเรย์จะถูกนำมาใช้ เมื่อต้องการเก็บข้อมูลชนิดเดียวกันหลายค่า แต่ใช้ตัวแปรอะเรย์ตัวเดียวกัน
- อะเรย์แบ่งออกได้เป็น 2 ประเภท คืออะเรย์ของข้อมูลชนิดพื้นฐาน และอะเรย์ของข้อมูลชนิดคลาสขั้นตอนในการสร้างอะเรย์ของข้อมูลชนิดพื้นฐานคือ การประกาศตัวแปรอะเรย์ และการสร้างอะเรย์
- ขั้นตอนในการสร้างอะเรย์ของข้อมูลชนิดคลาสคือ การประกาศตัวแปรอะเรย์ การสร้างอะเรย์ และการสร้างอ็อบเจกต์ให้กับสมาชิกของอะเรย์แต่ละตัว
- ขนาดของอะเรย์สามารถหาได้จากคุณลักษณะ `length`
- คำสั่ง `for` นิยมนำมาใช้ในการอ้างถึงสมาชิกของอะเรย์

- เราสามารถสร้างอะเรย์หลายมิติได้ โดยจำนวนเครื่องหมาย `[]` บ่งบอกถึงจำนวนมิติของอะเรย์
- สำหรับอะเรย์สองมิติ จำนวนคอลัมน์ในแต่ละแถวไม่จำเป็นต้องเท่ากัน
- ในคลาสที่ชื่อ `Arrays` มีเมธอดสำคัญที่เกี่ยวข้องกับอะเรย์คือ `sort()`, `binarySearch()` และ `fill()`
- เมธอด `arraycopy()` จากคลาส `System` ใช้ในการคัดลอกค่าของสมาชิกของอะเรย์
- `Collection`, `Set`, `List` และ `Map` เป็นอินเตอร์เฟสสำคัญที่อยู่ใน Collection API โดยที่อินเตอร์เฟส `Set` และ `List` สืบทอดมาจากอินเตอร์เฟส `Collection`
- อินเตอร์เฟส `Set` จะไม่สามารถมีค่าข้อมูลของสมาชิกที่ซ้ำกันได้ และไม่มีลำดับของสมาชิก
- ส่วนอินเตอร์เฟส `List` จะสามารถมีค่าข้อมูลของสมาชิกที่ซ้ำกันได้ และมีลำดับของสมาชิก
- สำหรับอินเตอร์เฟส `Map` จะมีการเก็บค่าคีย์คู่กับค่าข้อมูลของสมาชิก โดยที่ค่าคีย์ของสมาชิกจะต้องไม่ซ้ำกัน แต่ค่าข้อมูลของสมาชิกสามารถซ้ำกันได้
- เราสามารถนำคลาสที่ `implements` อินเตอร์เฟสเหล่านี้ไปใช้ในการเก็บข้อมูลที่เป็นอ็อบเจกต์ได้หลายตัว คล้ายกับอะเรย์ แต่สามารถเปลี่ยนแปลงขนาดได้
- อินเตอร์เฟส `Iterator`,`ListIterator` และ `Enumeration` ใช้ในการอ้างถึงข้อมูลสมาชิกของคลาสที่อยู่ใน Collection API
- คลาส `Vector` เป็นคลาสที่ใช้ในการเก็บกลุ่มของอ็อบเจกต์ของคลาสใดๆ โดยไม่จำกัดจำนวน
- เราสามารถที่ใช้ `Generic` ในการกำหนดชนิดของอ็อบเจกต์ที่จะใส่ข้อมูลลงอ็อบเจกต์คลาสประเภทคอลเล็กชั่น
- คำสั่ง `for` สามารถที่จะใช้ในการแจกแจงข้อมูลแทนการใช้ `Iterator` หรือ `Enumeration` ได้
- เราสามารถใช้คำสั่ง Generic ในการกำหนดชนิดข้อมูลของอ็อบเจกต์ที่อยู่ในคอลเล็กชั่นได้

บทที่ 9 การจัดการกับข้อผิดพลาด

เนื้อหาในบทนี้ เป็นการแนะนำหลักการของการจัดการกับข้อผิดพลาดในภาษาจาวา แนะนำคลาสที่เกี่ยวข้องกับการจัดการกับข้อผิดพลาดที่กำหนดไว้ใน Java API อธิบายคำสั่งที่ใช้ในการจัดการกับข้อผิดพลาดคือคำสั่ง `try`, `catch` และ `finally` อธิบายกฎการจัดการกับข้อผิดพลาด และตอนท้ายของบทจะเป็นการแนะนำการสร้างคลาสประเภทข้อผิดพลาดขึ้นมาใหม่

9.1 ข้อผิดพลาด

โปรแกรมภาษาจาวาอาจเกิดข้อผิดพลาด (Exception) ขึ้นในขั้นตอนการรันโปรแกรม โดยที่ข้อผิดพลาดเหล่านี้จะไม่สามารถตรวจสอบได้ในขั้นตอนการคอมไพล์โปรแกรม ตัวอย่าง เช่น คำสั่ง

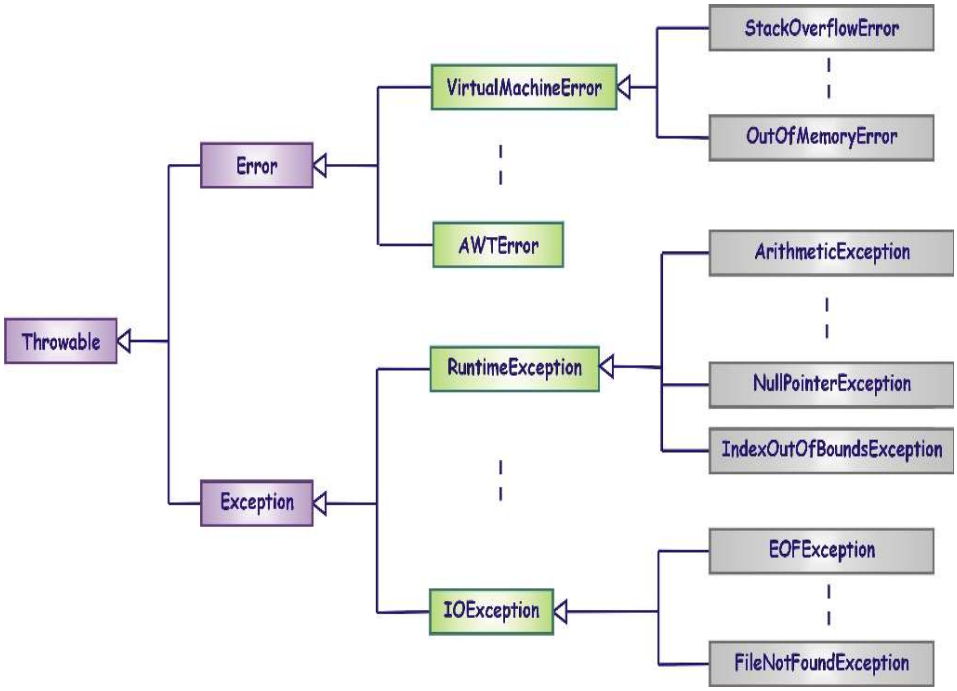
```
offset = x/n;
```

สำหรับตัวแปรที่มีชนิดข้อมูลเป็น `int` ที่ชื่อ `offset`, `x` และ `n` จะเป็นคำสั่งที่สามารถคอมไพล์ผ่านได้ แต่ถ้าค่าของ `n` เป็นศูนย์ โปรแกรมนี้จะเกิดข้อผิดพลาดขึ้นในขั้นตอนการรันโปรแกรม ซึ่งโปรแกรมภาษาจาวาจะตรวจสอบพบข้อผิดพลาดในการหารด้วยจำนวนเต็มศูนย์ และจะส่งข้อผิดพลาด (throwing an exception) ที่เกิดขึ้นกลับมา

โปรแกรมภาษาจาวาแบ่งข้อผิดพลาดที่อาจเกิดขึ้นขณะรันโปรแกรมเป็นสองประเภทคือ

1. Error เป็นข้อผิดพลาดที่ไม่สามารถแก้ไขและจัดการได้ เช่น `Virtual MemoryError` และ `OutOfMemoryError` เป็นต้น เราจะไม่กล่าวถึง Error เนื่องจาก Error เป็นข้อผิดพลาดที่เราไม่สามารถเขียนโปรแกรมเพื่อแก้ไขและจัดการได้
2. Exception เป็นข้อผิดพลาดที่สามารถแก้ไขและจัดการได้ เช่น ข้อผิดพลาดจากการเปิดไฟล์ที่ไม่มีอยู่ในไดเร็กทอรี่ `(FileNotFoundException)` หรือข้อผิดพลาดจากการอ้างอิงหมายเลขของสมาชิกของอะเรย์ที่ไม่ถูกต้อง คือไม่ได้มีอยู่จริง `(ArrayIndexOutOfBoundsException)`

ข้อผิดพลาดในภาษาจาวาจะกำหนดเป็นอ็อบเจกต์ของคลาสต่างๆ มากกว่าหกสิบคลาส โดยมีลำดับการสืบทอดดังแสดงในรูปที่ 9.1 ซึ่งคลาสของข้อผิดพลาดเหล่านี้จะสืบทอดมาจากคลาส `Throwable` ซึ่งเป็นคลาสราก



รูปที่ 9.1 คลาสของข้อผิดพลาดต่างๆที่สืบทอดมาจากคลาสที่ชื่อ `Throwable`

9.2 Exception

คลาสที่ชื่อ `Exception` เป็นคลาสที่กำหนดใน Java API เพื่อระบุข้อผิดพลาดที่เกิดขึ้นในขณะรันโปรแกรมภาษาจาวา Exception แบ่งออกเป็นสองประเภทคือ

1. `RuntimeException` เป็นข้อผิดพลาดที่อาจหลีกเลี่ยงได้หากมีการเขียนโปรแกรมที่ถูกต้อง ตัวอย่างเช่น `ArrayIndexOutOfBoundsException` ซึ่งเป็นข้อผิดพลาดเพราะมีการอ้างอิงสมาชิกของอะเรย์ที่ไม่ถูกต้อง สามารถแก้ไขโปรแกรมให้ดีขึ้นได้ถ้ามีการตรวจสอบหมายเลขสมาชิกของอะเรย์ก่อนที่จะอ้างอิงโดยใช้คำสั่ง `if`
2. `IOException` เป็นข้อผิดพลาดที่อาจจะไม่สามารถแก้ไขโดยการปรับปรุงโปรแกรมให้สมบูรณ์ขึ้นได้ เช่น `UnknownHostException` ที่อาจเกิดขึ้นในขณะที่โปรแกรมกำลังพยายามติดต่อกับระบบอินเทอร์เน็ต แต่เครื่องคอมพิวเตอร์ไม่สามารถติดต่อกับระบบเครือข่ายได้ ซึ่งข้อผิดพลาดประเภทนี้ไม่สามารถตรวจสอบได้โดยใช้คำสั่ง `if` แต่การเรียกใช้เมธอดที่อาจเกิดข้อผิดพลาดประเภท `IOException` จะต้องมีการเขียนคำสั่ง `try/catch` ในการจัดการกับข้อผิดพลาดที่เกิดขึ้น

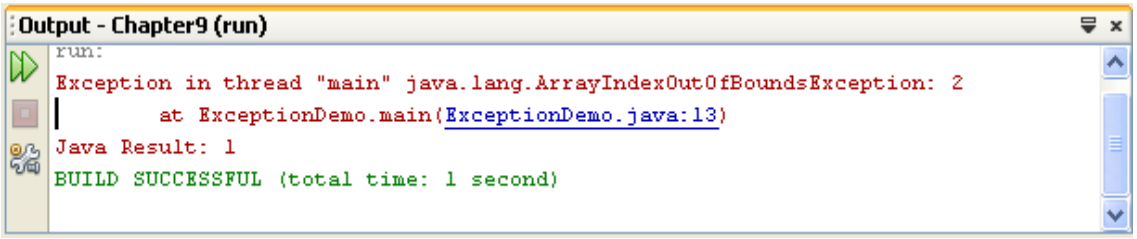
คลาสประเภท Exception ที่สำคัญและพบบ่อยในโปรแกรมภาษาจาวามีดังนี้

- `ArithmeticException` เป็นการระบุว่ามีข้อผิดพลาดในนิพจน์คณิตศาสตร์เช่น การหารด้วยเลขจำนวนเต็มศูนย์
- `ArrayIndexOutOfBoundsException` เป็นการระบุว่ามีการอ้างอิงสมาชิกของอะเรย์ไม่ถูกต้อง (เป็นเลขจำนวนเต็มลบหรือมีค่ามากกว่าหมายเลขสมาชิกของอะเรย์ที่มีอยู่)
- `EOFException` เป็นการระบุว่าตำแหน่งสิ้นสุดของไฟล์ได้ผ่านมาแล้ว
- `FileNotFoundException` เป็นการระบุว่าไม่พบไฟล์ที่ต้องการอ้างอิง
- `InterruptedException` เป็นการระบุว่า process บาง process ถูกระงับ (interrupt)
- `IOException` เป็นการระบุข้อผิดพลาดที่เกิดขึ้นจากขบวนการอินพุตหรือเอาต์พุตใดๆ
- `NullPointerException` เป็นการระบุว่ามีการเรียกใช้เมธอดของคลาสจากอ็อบเจกต์ที่ยังมีตำแหน่งอ้างอิงเป็น `null` อยู่ (อ็อบเจกต์ยังไม่ได้ถูกสร้าง)
- `NumberFormatException` เป็นการระบุว่ารูปแบบของตัวเลขที่ใช้อยู่ไม่ถูกต้อง

โปรแกรมที่ 9.1 แสดงตัวอย่างโปรแกรมที่เกิดข้อผิดพลาดที่ชื่อ `ArrayIndexOutOfBoundsException` แต่เนื่องจากโปรแกรมนี้ไม่มีการจัดการกับข้อผิดพลาดดังกล่าว จึงหยุดทำงานเมื่อพบข้อผิดพลาดขณะรันโปรแกรม โดยจะให้ผลลัพธ์ดังแสดงในรูปที่ 9.2

<u>โปรแกรมที่ 9.1</u> <u>ตัวอย่างแสดงข้อผิดพลาดที่ชื่อ `ArrayIndexOutOfBoundsException`</u>

```
public class ExceptionDemo {
   public static void main(String args[]) {
      System.out.println(args[2]);
      System.out.println("Hello");
   }
}
```



รูปที่ 9.2 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 9.1

9.3 คำสั่ง `try..catch`

ภาษาจาวามีคีย์เวิร์ด `try` ที่เป็นคำสั่งที่ใช้ในการจัดการกับเมธอดหรือคำสั่งที่อาจเกิดข้อผิดพลาดซึ่งจะส่งอ็อบเจกต์ประเภท Exception ในขณะรันโปรแกรม คีย์เวิร์ด `try` จะมีชุดคำสั่งอยู่ภายในบล็อกโดยมีรูปแบบดังนี้

```
try {
    [statements]
}
```

โปรแกรมภาษาจาวาจะสั่งงานชุดคำสั่งที่อยู่ในบล็อกทีละคำสั่ง และหากเกิดข้อผิดพลาดขึ้นในคำสั่งใด ก็จะมีการส่งอ็อบเจกต์ของข้อผิดพลาดประเภท Exception นั้นขึ้นมา ซึ่งโปรแกรมจะยกเลิกการทำงานคำสั่งที่อยู่ในบล็อกที่เหลือทั้งหมด แต่หากทุกคำสั่งที่อยู่ในบล็อกไม่มีข้อผิดพลาดใดเกิดขึ้น โปรแกรมจาวาก็จะทำงานต่อไปตามปกติเสมือนว่าไม่มีคีย์เวิร์ด `try` อยู่ ในกรณีที่ต้องการจัดการกับข้อผิดพลาดที่เกิดขึ้น โปรแกรมจะต้องมีชุดคำสั่งอยู่ในบล็อกของคีย์เวิร์ด `catch` ที่จะระบุชนิดของอ็อบเจกต์ในคลาสประเภท Exception ที่ต้องการจัดการ โดยมีรูปแบบคำสั่งดังนี้

```
catch(ExceptionType argumentName){
    [statements]
}
```

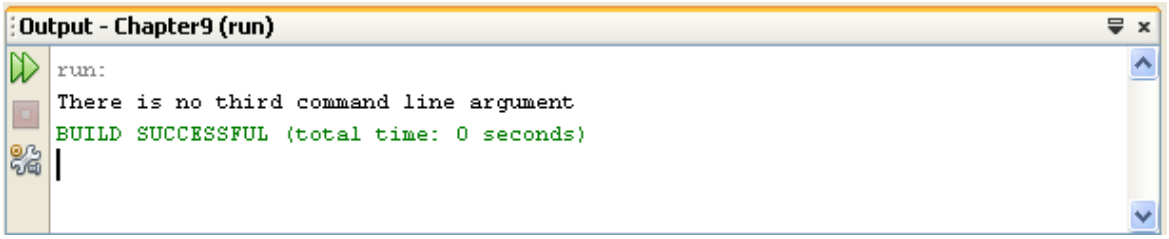
โดยที่

- `ExceptionType` คือชื่อคลาสประเภท Exception ที่ต้องการจะจัดการเมื่อมีข้อผิดพลาดเกิดขึ้น
- `argumentName` คือชื่อของอ็อบเจกต์ที่จะเป็น argument ที่ใช้ในบล็อกคำสั่งของ `catch`

โปรแกรมภาษาจาวาจะทำชุดคำสั่งในบล็อก `catch` ถ้ามีคำสั่งในบล็อก `try` คำสั่งใดคำสั่งหนึ่งเกิดข้อผิดพลาดโดยส่งอ็อบเจกต์ประเภท Exception ตามชนิดที่สอดคล้องกับคลาสที่ระบุใน `ExceptionType` และโปรแกรมจะข้ามคำสั่งที่เหลืออยู่ในบล็อก `try` ทั้งหมด ส่วนในกรณีที่ไม่มีคำสั่งใดในบล็อก `try` เกิดข้อผิดพลาดขึ้น โปรแกรมภาษาจาวาจะไม่มีการทำชุดคำสั่งในบล็อก `catch` โปรแกรมที่ 9.2 แสดงตัวอย่างการจัดการกับข้อผิดพลาดที่อาจเกิดขึ้นในโปรแกรมที่ 9.1 โปรแกรมนี้ได้เพิ่มชุดคำสั่ง `try` และ `catch` ขึ้นมา ซึ่งโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 9.3

<u>โปรแกรมที่ 9.2</u> <u>ตัวอย่างการจัดการกับข้อผิดพลาดที่อาจเกิดขึ้น</u>

```
public class ExceptionHandlingDemo {
   public static void main(String args[]) {
      try {
         System.out.println(args[2]);
      } catch(ArrayIndexOutOfBoundsException ex) {
         System.out.println("There is no third command line argument");
      }
   }
}
```



รูปที่ 9.3 *ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 9.2*

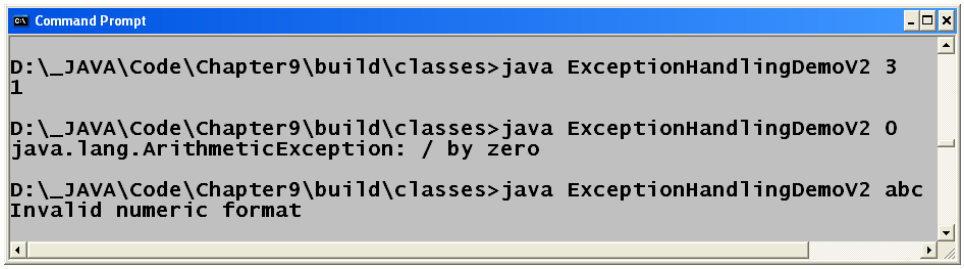
9.4 การจัดการกับข้อผิดพลาดหลายๆ ประเภท

โปรแกรมภาษาจาวาสามารถจะมีชุดคำสั่งของบล็อก `catch` ได้มากกว่าหนึ่งชุดสำหรับในแต่ละบล็อกคำสั่ง `try` โดยที่ชนิดของอ็อบเจกต์ประเภท Exception ที่อยู่ในชุดคำสั่งของบล็อก `catch` จะต้องเรียงตามลำดับการสืบทอด โปรแกรมที่ 9.3 แสดงตัวอย่างการจัดการกับข้อผิดพลาดมากกว่าหนึ่งประเภท โดยมีชุดคำสั่งในบล็อก `catch` สองชุดเพื่อจัดการกับข้อผิดพลาดสองชนิดคือ `ArithmeticException` และ `ArrayIndexOutOfBoundsException` โปรแกรมนี้จะรับ argument ผ่านทาง command line ซึ่งจะเป็นตัวเลขที่มีชนิดข้อมูลเป็น `String` แล้วจึงจะถูกแปลงให้มีชนิดข้อมูลเป็น `int` ซึ่งหากข้อมูลเป็นจำนวนเต็มศูนย์ก็จะทำให้เกิดข้อผิดพลาดชนิด `ArithmeticException` ขึ้นได้ โดยโปรแกรมจะข้ามมาทำชุดคำสั่งในบล็อก `catch` ที่ตรวจจับอ็อบเจกต์ชนิดดังกล่าว และโปรแกรมจะจัดการกับข้อผิดพลาดชนิด `NumberFormatException` ถ้า argument ที่รับผ่านทาง command line ไม่ใช่ข้อความที่แปลงเป็นตัวเลขจำนวนเต็มได้ โดยโปรแกรมจะข้ามมาทำชุดคำสั่งในบล็อก `catch` ที่ตรวจจับอ็อบเจกต์ชนิด `NumberFormatException` โดยมีผลลัพธ์ของโปรแกรมดังตัวอย่างที่แสดงในรูปที่ 9.4

ในกรณีที่มีข้อผิดพลาดเกิดขึ้น ภาษาจาวาจะพิจารณาว่าเป็นข้อผิดพลาดชนิดใด ซึ่งการที่จะจัดการกับอ็อบเจกต์ประเภท Exception นั้นจะพิจารณาจากคลาสที่มีการสืบทอดตาม ลำดับชั้น ทั้งนี้เราสามารถจะจัดการกับอ็อบเจกต์ประเภท Exception โดยใช้คลาสที่เป็น superclass ของอ็อบเจกต์นั้นได้ อาทิเช่นอ็อบเจกต์ชนิด `FileNotFoundException` สามารถจัดการได้โดยใช้คลาส `IOException` หรือ `Exception` แทนได้เนื่องจากคลาส `FileNotFoundException` สืบทอดมาจากคลาส `IOException` ซึ่งสืบทอดมาจากคลาส `Exception` อีกชั้นหนึ่ง

<u>โปรแกรมที่ 9.3</u> <u>ตัวอย่างการจัดการกับข้อผิดพลาดมากกว่าหนึ่งประเภท</u>

```
public class ExceptionHandlingDemoV2 {
   public static void main(String args[]) {
      try {
         int i = Integer.parseInt(args[0]);
         System.out.println(4 / i);
      } catch(ArithmeticException ex) {
         System.out.println(ex.toString());
      } catch(NumberFormatException ex) {
         System.out.println("Invalid numeric format");
      }
   }
}
```



รูปที่ 9.4 *ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 9.3*

ภาษาจาวากำหนดให้ชุดคำสั่งในบล็อก `catch` จะต้องเรียงอ็อบเจกต์ของคลาสประเภท `Exception` ตามลำดับการสืบทอด ตัวอย่างเช่น โปรแกรมที่ 9.3 ถึงแม้ว่าจะมีคลาสประเภท Exception ที่จะตรวจจับสองชนิด แต่เนื่องจากคลาสทั้งสองต่างก็สืบทอดมาจาก `RunTimeException` และอยู่ในลำดับชั้นเดียวกันจึงสามารถที่จะสลับชุดคำสั่งในบล็อก `catch` ทั้งสองได้ แต่ในกรณีของโปรแกรมที่ 9.4 ชุดคำสั่งในบล็อก `catch` ที่จะจัดการกับคลาส `RunTimeException` อยู่ลำดับก่อนหน้าชุดคำสั่งในบล็อก `catch` ที่จะจัดการกับคลาส

ArrayIndexOutOfBoundsException ดังนั้นโปรแกรมนี้จึงไม่มีโอกาสที่จะทำชุดคำสั่งที่สองได้เนื่องจากคลาส ArrayIndexOutOfBoundsException สืบทอดมาจากคลาส RunTimeException จึงทำให้โปรแกรมนี้ไม่สามารถคอมไพล์ผ่านได้

โปรแกรมที่ 9.4 ตัวอย่างการจัดวางลำดับคลาสที่จะมาจัดการกับข้อผิดพลาดที่ไม่ถูกต้อง

```
public class ExceptionHandlingDemoV3 {
   public static void main(String args[]) {
      try {
         int i = Integer.parseInt(args[0]);
         System.out.println(4 / i);
         System.out.println(args[2]);
      } catch(RuntimeException ex) {
         System.out.println(ex.toString());
      } catch(ArrayIndexOutOfBoundsException ex) {
         System.out.println("There is no third command
         line argument");
      }
   }
}
```

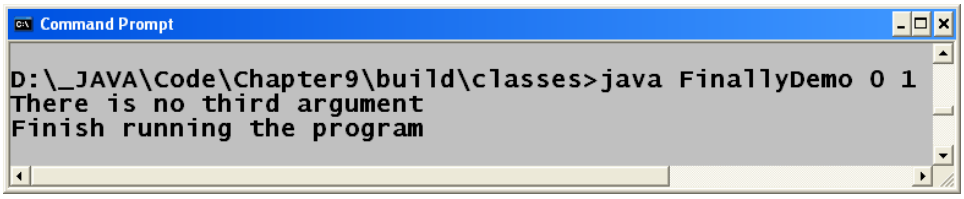
9.5 บล็อก finally

ภาษาจาวามีคีย์เวิร์ด finally ที่จะมีชุดคำสั่งอยู่ในบล็อก เพื่อระบุให้โปรแกรมทำชุดคำสั่งดังกล่าวหลังจากสิ้นสุดการทำงานของชุดคำสั่งในบล็อก try หรือ catch โปรแกรมที่ 9.5 แสดงตัวอย่างการกำหนดชุดคำสั่งในบล็อก finally ซึ่งโปรแกรมนี้จะให้ผลลัพธ์ดังแสดงในรูปที่ 9.5

ภาษาจาวาจะทำชุดคำสั่งในบล็อก finally เสมอ แม้ว่าจะมีคำสั่ง return ในบล็อก try หรือ catch ก่อนก็ตาม กรณีเดียวที่จะไม่ทำชุดคำสั่งในบล็อก finally คือมีคำสั่ง System.exit(); เพื่อที่จะออกจากโปรแกรม โปรแกรมที่ 9.6 แสดงตัวอย่างที่มีคำสั่ง return อยู่ก่อนชุดคำสั่งในบล็อก finally โดยโปรแกรมนี้จะได้ผลลัพธ์ดังรูปที่ 9.6

โปรแกรมที่ 9.5 ตัวอย่างการกำหนดชุดคำสั่งในบล็อก `finally`

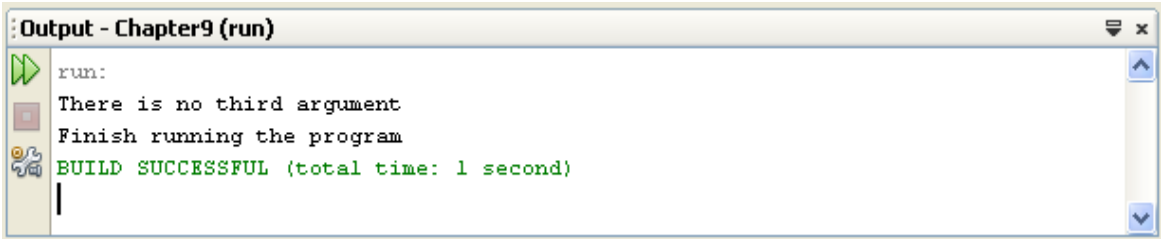
```
public class FinallyDemo {
   public static void main(String args[]) {
      try {
         System.out.println(args[2]);
         System.out.println("Hello");
      } catch(ArrayIndexOutOfBoundsException ex) {
         System.out.println("There is no third argument");
      } finally {
         System.out.println("Finish running the program");
      }
   }
}
```



รูปที่ 9.5 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 9.5

โปรแกรมที่ 9.6 ตัวอย่างที่มีคำสั่ง `return` อยู่ก่อนชุดคำสั่งในบล็อก `finally`

```
public class FinallyDemoV2 {
    public static void main(String args[]) {
       FinallyDemoV2 obj = new FinallyDemoV2();
       obj.myMethod(args);
    }
    public int myMethod(String args[]) {
       try {
            System.out.println(args[2]);
            return 0;
       } catch(ArrayIndexOutOfBoundsException ex) {
            System.out.println("There is no third argument");
       } finally {
            System.out.println("Finish running the program");
            return 1;
       }
    }
}
```



รูปที่ 9.6 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 9.6

9.6 การจัดการกับเมธอดที่ส่งอ็อบเจกต์ประเภท Exception

เมธอดบางเมธอดที่กำหนดใน Java API อาจส่งอ็อบเจกต์ประเภท Exception เมื่อเกิดข้อผิดพลาดขึ้นในการเรียกใช้คำสั่ง อาทิเช่น constructor ของคลาส `FileInputStream` อาจส่งอ็อบเจกต์ของคลาส `FileNotFoundException` ถ้าไม่พบไฟล์ดังกล่าว หรือเมธอด `getLocalHost()` ของคลาส `InetAddress` อาจส่งอ็อบเจกต์ของคลาส `UnknownHostException` ถ้าไม่ทราบ IP address ของเครื่อง ภาษาจาวากำหนดให้เราต้องเขียนโปรแกรมจัดการกับข้อผิดพลาด เมื่อมีการเรียกใช้เมธอดที่อาจส่งอ็อบเจกต์ประเภท `IOException` สำหรับเมธอดซึ่งการจัดการกับข้อผิดพลาดแบ่งออกเป็น

1. ใช้คำสั่ง `try/catch` ดังที่กล่าวไว้ในหัวข้อที่ผ่านมา
2. ใช้คำสั่ง `throws` ในการประกาศเมธอดที่จะมีการเรียกใช้เมธอดใดๆที่อาจส่งอ็อบเจกต์ประเภท Exception ในกรณีนี้หมายความว่าเมธอดที่ประกาศไม่ต้องการจัดการกับอ็อบเจกต์ประเภท Exception ดังกล่าวเอง แต่ต้องการจะให้เมธอดอื่นๆที่เรียกใช้เมธอดนี้เป็นตัวจัดการแทน รูปแบบการใช้คำสั่ง `throws` มีดังนี้

```
[modifier] return_type methodName([arguments]) throws ExceptionType
                                                      [,ExceptionType2]{
        ...
}
```

ตัวอย่างเช่น

```
public void openFile(String s)throws FileNotFoundException {
}
```

เมธอดใดๆ สามารถที่จะจัดการกับอ็อบเจกต์ประเภท Exception โดยใช้คำสั่ง `throws` ได้มากกว่าหนึ่งชนิด ตัวอย่างเช่น

```
public void openFile(String s)throws FileNotFoundException,
                                                UnknownHostException {
}
```

กรณีที่มีการใช้คำสั่ง throws แล้วส่งต่อให้เมธอดอื่นๆที่เรียกใช้เป็นตัวจัดการกับอ็อบเจกต์ประเภท Exception ดังกล่าวไปเรื่อยๆ แต่ถ้าเมธอดที่ชื่อ main() ซึ่งเรียกใช้เมธอดสุดท้ายที่ใช้คำสั่ง throws ไม่มีการจัดการกับอ็อบเจกต์ประเภท Exception ดังกล่าว โปรแกรมจะเกิดข้อผิดพลาดในขั้นตอนการรันโปรแกรม เมื่อมีข้อผิดพลาดของอ็อบเจกต์ประเภท Exception ดังกล่าวเกิดขึ้น ตัวอย่างเช่นโปรแกรมที่ 9.7 ซึ่งผลลัพธ์ที่ได้จากการรันเป็นดังแสดงในรูปที่ 9.7

<u>โปรแกรมที่ 9.7</u> <u>ตัวอย่างที่ไม่มีการจัดการกับอ็อบเจกต์ประเภท Exception</u>

```
public class ExceptionDemo1 {
   public static void main(String args[]) {
      ExceptionDemo1 ex1 = new ExceptionDemo1();
      ex1.method1();
   }
   public void method1() throws ArithmeticException {
      method2();
   }
   public void method2() throws ArithmeticException {
      System.out.println(2/0);
   }
}
```

```
Output - Chapter9 (run)
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo1.method2(ExceptionDemo1.java:22)
        at ExceptionDemo1.method1(ExceptionDemo1.java:18)
        at ExceptionDemo1.main(ExceptionDemo1.java:14)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)
```

รูปที่ 9.7 *ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 9.7*

เราสามารถที่จะใช้คำสั่ง throws ในเมธอดที่ชื่อ main() ได้ แต่จะเป็นการยกเลิกการจัดการใดๆกับข้อผิดพลาดทั้งหมดที่อาจเกิดขึ้น นอกจากนี้ภาษาจาวาไม่ได้กำหนดให้เราจะต้องเขียนคำสั่งในการจัดการกับเมธอด ที่อาจส่งข้อผิดพลาดของอ็อบเจกต์ประเภท RuntimeException แต่จะกำหนดไว้เฉพาะเมธอดที่อาจส่งข้อผิดพลาดของอ็อบเจกต์ประเภท IOException

เมธอดที่มีคำสั่ง `throws` จะมีผลต่อการสืบทอด ทั้งนี้เนื่องจากกฎของการกำหนดเมธอดแบบ overridden จะไม่อนุญาตให้มีการจัดการอ็อบเจกต์ประเภท Exception โดยใช้คำสั่ง `throws` มากชนิดกว่าที่เมธอดเดิมจัดการอยู่ โปรแกรมที่ 9.8 และโปรแกรมที่ 9.9 แสดงตัวอย่างโปรแกรมที่มีเมธอดแบบ overridden ที่ถูกต้องและไม่ถูกต้องตามลำดับ

<u>โปรแกรมที่ 9.8</u> <u>ตัวอย่างที่มีเมธอดแบบ overridden ที่ถูกต้อง</u>

```
import java.io.*;

public class Parent {
      public void myMethods() throws IOException { }
}

public class OverrideException extends Parent{
      public void myMethods() throws IOException {
            new FileInputStream("temp.txt");
      }
}
```

<u>โปรแกรมที่ 9.9</u> <u>ตัวอย่างที่มีเมธอดแบบ overridden ที่ไม่ถูกต้อง</u>

```
import java.io.*;

public class Parent {
  public void myMethods() throws FileNotFoundException { }
}

public class OverrideExceptionV2 extends Parent{
    public void myMethods() throws
                      FileNotFoundException,IOException {
        new FileInputStream("temp.txt");
    }
}
```

9.7 การสร้างคลาสประเภท Exception ขึ้นใหม่

Java API มีคลาสประเภท Exception อยู่หลายชนิดแต่ในบางครั้งเราอาจต้องการที่จะกำหนดคลาสประเภท Exception ขึ้นมาใหม่ เพื่อใช้ในการระบุข้อผิดพลาดเฉพาะเจาะจงเช่น โปรแกรมระบบทะเบียนนักศึกษาอาจมีคลาสที่ชื่อ `StudentIDNotFoundException` เพื่อใช้ในการสร้างอ็อบเจกต์ที่จะส่งข้อผิดพลาด เมื่อไม่สามารถตรวจพบรหัสนักศึกษาที่ต้องการ การสร้างคลาสประเภท Exception ขึ้นมาใหม่ สามารถทำได้โดยนิยามคลาสใดๆให้สืบทอดมาจาก

คลาสที่ชื่อ `Exception` ในกรณีที่ต้องการบังคับให้เมธอดใดๆ จัดการกับอ็อบเจกต์ของคลาสนั้นในกรณีที่เกิดข้อผิดพลาดขึ้น หรือนิยามคลาสใดๆให้สืบทอดมาจากคลาส `Runtime Exception` ในกรณีที่ไม่ต้องการให้เมธอดใดๆ จำเป็นต้องจัดการกับข้อผิดพลาดดังกล่าว

โดยทั่วไปคลาสที่ชื่อ `Exception` จะมี constructor สองรูปแบบคือ

- `public Exception()`
- `public Exception(String s)`

ดังนั้นคลาสที่สืบทอดมาจากคลาสที่ชื่อ `Exception` ควรจะมี constructor ทั้งสอง โดยรูปแบบหนึ่งจะมี argument ที่มีชนิดข้อมูลเป็น `String` เพื่อรับข้อความที่จะอธิบายข้อผิดพลาดและคำสั่งแรกใน constructor ดังกล่าวควรเป็นคำสั่ง `super(s);` เพื่อส่งข้อความดังกล่าวให้กับ constructor ของ superclass (คลาสที่ชื่อ `Exception`) โปรแกรมที่ 9.10 แสดงตัวอย่างของคลาส `MyOwnException` ซึ่งเป็นคลาสประเภท Exception ที่กำหนดขึ้นใหม่

<u>โปรแกรมที่ 9.10</u> <u>ตัวอย่างคลาสประเภท Exception ที่กำหนดขึ้นใหม่</u>

```
public class MyOwnException extends Exception {
   public MyOwnException (String s) {
      super(s);
   }
}
```

9.7.1 การเขียนเมธอดเพื่อส่งอ็อบเจกต์ประเภท Exception

เมธอดที่ต้องการส่งอ็อบเจกต์ประเภท Exception เมื่อเกิดข้อผิดพลาดขึ้นในคำสั่งใด จะต้องเรียกใช้คำสั่งที่ชื่อ `throw` เพื่อจะสร้างอ็อบเจกต์ของคลาสประเภท Exception ขึ้นมา โดยมีรูปแบบคำสั่งดังนี้

```
throw new ExceptionType([arguments])
```

โดยที่

- `ExceptionType` คือชื่อของคลาสประเภท Exception ที่ต้องการจะสร้างอ็อบเจกต์โดยมี argument สอดคล้องกับที่ระบุใน constructor ของคลาสดังกล่าว

นอกจากนี้คำสั่งประกาศเมธอดนั้นจะต้องมีคำสั่ง `throws` เพื่อกำหนดให้คำสั่งในเมธอดอื่นๆที่เรียกใช้เมธอดนี้ต้องเขียนคำสั่งในการจัดการกับข้อผิดพลาดนี้ โปรแกรมที่ 9.11 แสดงตัวอย่างการเขียนคลาส `FileHandler` โดยมีเมธอด `openFile()` ซึ่งจะส่งอ็อบเจกต์ของคลาส `MyOwnException` ขึ้นมาเมื่อไม่พบไฟล์ที่ระบุ ส่วนโปรแกรมที่

9.12 แสดงตัวอย่างโปรแกรมที่มีการจัดการกับข้อผิดพลาดดังกล่าว

โปรแกรมที่ 9.11 ตัวอย่างคลาส `FileHandler`

```
import java.io.*;

public class FileHandler {
   public static void openFile(String filename) throws
                                          MyOwnException {
      File f = new File(filename);
      if (!f.exists()) {
         throw new MyOwnException("File Not Found");
      }
   }
}
```

โปรแกรมที่ 9.12 ตัวอย่างที่มีการจัดการกับข้อผิดพลาด

```
public class FileOpener {
   public static void main(String args[]) {
      try {
            FileHandler.openFile(args[0]);
            System.out.println("Open successful");
      } catch (MyOwnException ex) {
            System.out.println(ex);
      }
   }
}
```

สรุปเนื้อหาของบท

- ข้อดีประการหนึ่งของภาษาจาวาคือ เราสามารถเขียนโปรแกรมให้มีการตรวจจับและจัดการกับข้อผิดพลาดที่อาจเกิดขึ้นได้ โดยที่การทำงานไม่ต้องหยุดลง
- Error เป็นข้อผิดพลาดที่ไม่สามารถแก้ไขและจัดการได้ ส่วน Exception เป็นข้อผิดพลาดที่สามารถแก้ไขหรือจัดการได้
- คำสั่ง `try` และ `catch` เป็นคำสั่งที่ใช้ในการตรวจจับและจัดการกับข้อผิดพลาดที่อาจเกิดขึ้นได้ โดยบล็อกคำสั่ง `catch` สามารถมีได้มากกว่าหนึ่งบล็อกสำหรับในแต่ละบล็อกคำสั่ง `try`
- คำสั่ง `finally` เป็นคำสั่งที่อยู่ต่อจากคำสั่ง `try/catch` ถูกใช้เมื่อมีบางคำสั่งที่ต้องการให้มีการทำเสมอ

ยกเว้นเจอคำสั่ง `System.exit(0);` ก่อนเท่านั้น

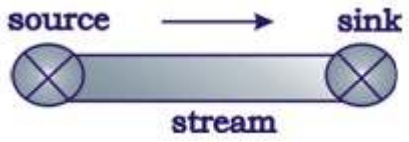
- คำสั่ง `throws` จะใส่ไว้ตรงคำสั่งประกาศเมธอด สำหรับเมธอดที่ยังไม่ต้องการจัดการกับข้อผิดพลาดแต่จะให้เมธอดที่เรียกใช้เมธอดนี้เป็นตัวจัดการแทน
- เราสามารถสร้างคลาสประเภท `Exception` ชนิดใหม่ขึ้นได้ โดยจะต้องสืบทอดมาจากคลาส `Exception` และต้องมีการเรียกใช้ `constructor` ของคลาส `Exception` ด้วย

บทที่ 10 คลาสอินพุตและเอาต์พุต

เนื้อหาในบทนี้เป็นการแนะนำคลาสและเมธอดต่างๆ ที่เกี่ยวกับอินพุตและเอาต์พุต และอธิบายความหมายของ stream โดยอธิบายคลาสที่เกี่ยวข้องกับอินพุตและเอาต์พุตที่อยู่ในแพคเก็จ `java.io` แนะนำคลาส `InputStream`, `OutputStream`, `Reader` และ `Writer` อธิบายการสร้างและใช้ stream แบบต่างๆ แนะนำคลาสและเมธอดของคลาส `File` และ `RandomAccessFile` อธิบายการใช้อินเตอร์เฟสที่ชื่อ `Serializable` และอธิบายวิธีการเขียนและอ่านข้อมูลของอ็อบเจกต์ผ่าน stream

10.1 Stream

ภาษาจาวาจัดการกับขบวนการอินพุตและเอาต์พุตโดยใช้หลักการของ stream ซึ่งเปรียบเสมือนท่อส่งข้อมูลจากต้นทาง (source) ไปยังปลายทาง (sink) โดยที่ต้นทางเป็นตำแหน่งเริ่มต้นของ stream ที่เรียกว่า input stream และปลายทางเป็นตำแหน่งสิ้นสุดของ stream ที่เรียกว่า output stream ดังแสดงในรูปที่ 10.1 stream จะช่วยให้นักพัฒนาโปรแกรมสามารถที่เขียนโปรแกรมเพื่อส่งข้อมูลจากต้นทางไปยังปลายทางได้ โดยไม่จำเป็นต้องทราบรายละเอียดภายในของการติดต่อกับฮาร์ดแวร์หรือซอฟต์แวร์ที่ในการส่งข้อมูล ซึ่งสามารถเขียนโปรแกรมได้โดยเพียงแต่รู้เมธอดที่ใช้ในการรับหรือส่งข้อมูลเท่านั้น



รูปที่ 10.1 หลักการของ stream

ต้นทางและปลายทางของ stream อาจเป็นฮาร์ดแวร์หรือซอฟต์แวร์ โดยจะเรียกว่า โหนด(node) เช่น ไฟล์ หน่วยความจำ หรือ socket เป็นต้น ซึ่งต้นทางและปลายทางอาจมีชนิดของโหนดที่ต่างกัน อาทิเช่น การอ่านข้อมูลจากตัวแปรที่เก็บในหน่วยความจำที่เป็นต้นทางผ่าน stream ลงไปเก็บไว้ในไฟล์ที่เป็นปลายทาง

stream ที่ใช้ในการส่งข้อมูลของภาษาจาวาจะแบ่งออกเป็นสองประเภทคือ

– byte stream เป็น stream ที่ใช้ในการรับหรือส่งข้อมูลชนิด `byte`

– character stream เป็น stream ที่ใช้ในการรับหรือส่งข้อมูลชนิด `char`

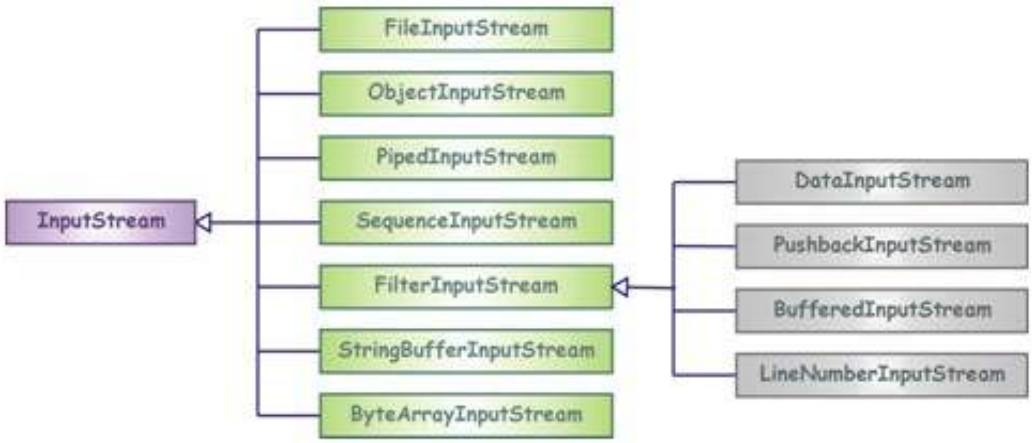
โดยทั่วไปคำว่า stream จะหมายถึง byte stream ส่วน character stream ภาษาจาวาจะใช้คำว่า reader และ write สำหรับ stream ในการรับและส่งชนิดข้อมูล `char` ตามลำดับ

10.1.1 แพคเก็จ `java.io`

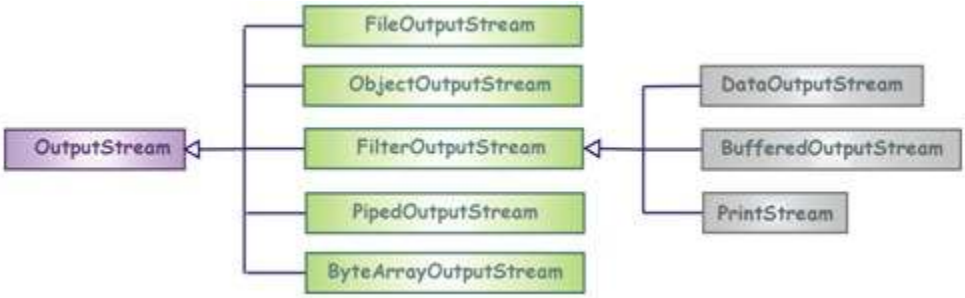
คลาสที่เกี่ยวกับอินพุตและเอาต์พุตจะถูกกำหนดโดย Java API ในแพคเก็จ `java.io` ซึ่งจะมีคลาสพื้นฐานอยู่ 4 คลาสคือ

- `InputStream` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็น stream ในการรับชนิดข้อมูลแบบ `byte`
- `OutputStream` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็น stream ในการส่งชนิดข้อมูลแบบ `byte`
- `Reader` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็น stream ในการรับชนิดข้อมูลแบบ `char`
- `Writer` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็น stream ในการส่งชนิดข้อมูลแบบ `char`

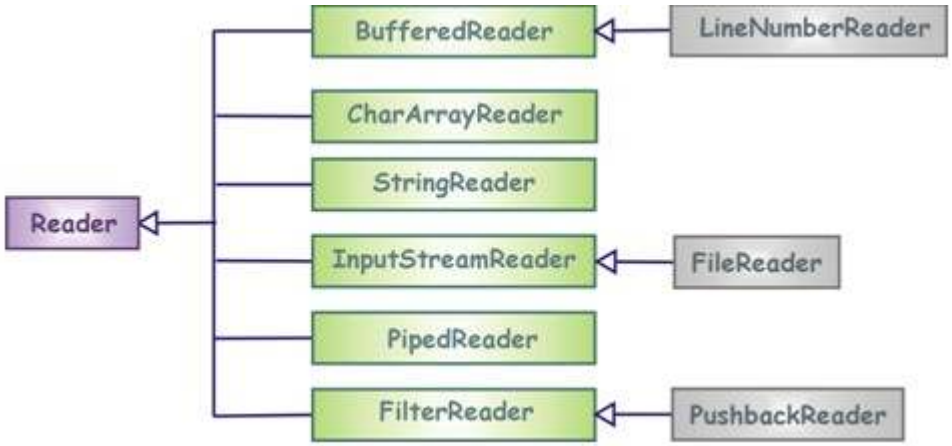
ซึ่งคลาสทั้งสี่นี้จะเป็นคลาสที่เป็นคลาสรากของคลาสอื่นๆที่จะสืบทอดมา ดังแสดงในรูปที่ 10.2 ถึงรูปที่ 10.5



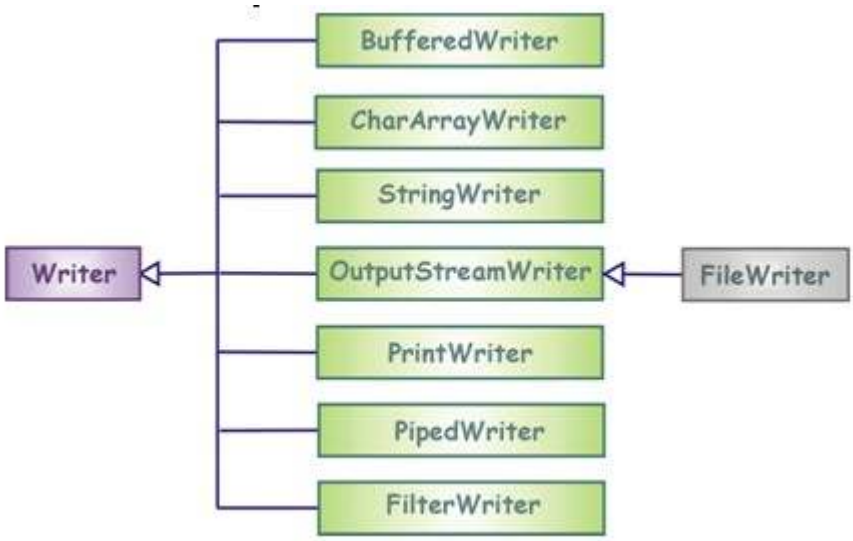
รูปที่ 10.2 คลาสประเภท `InputStream`



รูปที่ 10.3 คลาสประเภท `OutputStream`



รูปที่ 10.4 คลาสประเภท `Reader`



รูปที่ 10.5 คลาสประเภท `Writer`

10.1.2 คลาสประเภท Byte Stream

ภาษาจาวาจะมีคลาสพื้นฐานในการจัดการกับอินพุตและเอาต์พุต ที่เป็นชนิดข้อมูลแบบ `byte` อยู่สองคลาสที่คู่กันคือ `InputStream` และ `OutputStream` คลาสทั้งสองเป็นคลาสแบบ `abstract` ซึ่งเราไม่สามารถที่จะสร้างอ็อบเจกต์ของคลาสทั้งสองได้ แต่คลาสทั้งสองจะมีคลาสที่เป็น subclass ซึ่งจะใช้ในการสร้างอ็อบเจกต์สำหรับการรับและส่งข้อมูลแบบ `byte` ของโหนดที่มีต้นทางและปลายทางแบบต่างๆอาทิเช่น

- `FileInputStream` และ `FileOutputStream` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์สำหรับต้นทางและปลายทางที่เป็นไฟล์
- `ByteArrayInputStream` และ `ByteArrayOutputStream` เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์สำหรับต้นทางและปลายทาง ที่เป็นอะเรย์ของชนิดข้อมูลแบบ `byte`

10.1.3 คลาส `InputStream`

คลาส `InputStream` จะใช้ในการอ่านข้อมูลของ stream ที่เป็นชนิดข้อมูลแบบ `byte` คลาส `InputStream` จะนำข้อมูลจากโหนดต้นทางเข้ามาใน stream และการอ่านข้อมูลจาก stream จะเป็นการลบข้อมูลที่อ่านออกจาก stream ดังแสดงในรูปที่ 10.6 โดยมีเมธอดที่ใช้สำหรับการอ่านข้อมูลที่เป็น `byte` หรืออะเรย์ของ `byte` เท่านั้นดังนี้

- `int read()`

  เป็นเมธอดที่ใช้ในการอ่านข้อมูลจาก stream มาหนึ่งไบท์โดยจะส่งค่ากลับมาเป็นค่าที่อ่านได้ระหว่าง 0 ถึง 255 และจะส่งค่ากลับเป็น –1 ถ้าไม่มีข้อมูลจากโหนดต้นทาง เมธอดนี้จะส่งอ็อบเจกต์ชนิด `IOException` ในกรณีที่มีข้อผิดพลาดเกิดขึ้นจากการอ่านข้อมูล ตัวอย่างเช่นไฟล์ที่อ่านข้อมูลเสีย เมธอดนี้เป็นเมธอดแบบ `abstract` ซึ่งคลาสที่เป็น subclass จะต้องกำหนดบล็อกคำสั่งสำหรับเมธอดนี้

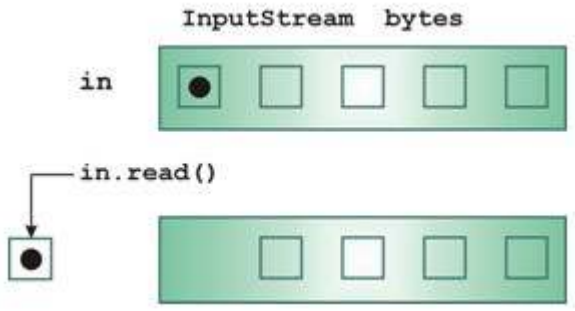
- `int read(byte []b)`

  เป็นเมธอดที่ใช้ในการอ่านข้อมูลจากโหนดต้นทางทีละหลายไบท์ โดยจะอ่านข้อมูลครั้งละไม่เกินจำนวนสมาชิกของอะเรย์ของ argument ที่ชื่อ b โดยจะเก็บข้อมูลที่อ่านเข้าไว้ในอะเรย์ของ byte เมธอดนี้จะส่งจำนวนไบท์ที่อ่านได้คืนมาและจะมีค่าเป็น –1 ถ้าไม่สามารถอ่านข้อมูลได้

- `int read (byte []b, int offset, int length)`

  เป็นเมธอดที่ใช้ในการอ่านข้อมูลจากโหนดต้นทางทีละหลายไบท์เช่นกัน แต่จะสามารถกำหนดตำแหน่งในการเก็บข้อมูลที่อ่านลงในอะเรย์ของ argument ที่ชื่อ `b` ได้ โดยเริ่มที่ตำแหน่ง `offset` และอ่านครั้ง

ละ `length` ไบท์ ซึ่งเมธอดจะส่งจำนวนข้อมูลที่อ่านได้จริงคืนมา การอ่านข้อมูลเป็นอะเรย์ของชนิดข้อมูล `byte` จะช่วยเพิ่มประสิทธิภาพ ทั้งนี้จะอ่านข้อมูลได้เร็วกว่า



รูปที่ 10.6 การอ่านข้อมูลจาก stream

คลาส `InputStream` ยังมีเมธอดอื่นๆ ที่สำคัญอีกดังนี้

- `int available()`

  เมธอดนี้จะส่งจำนวนไบท์ที่ยังสามารถอ่านได้จาก stream คืนมา

- `void close()`

  เมธอดนี้ใช้ในการปิด stream การเรียกใช้เมธอดจะช่วยทำให้ลดการใช้ทรัพยากรในระบบ (หน่วยความจำหรือจำนวนไฟล์ที่เปิด)

- `int skip(long n)`

  เมธอดนี้จะเป็นการลบข้อมูลจำนวน `n` ไบท์ออกจาก stream และข้ามไปข้อมูลตำแหน่งถัดไป

- `void mark(int readlimit)`

  เมธอดนี้ใช้ระบุตำแหน่งของข้อมูลใน stream ที่ต้องการจะระบุไว้เพื่อที่จะต้องการให้ stream กลับมาที่ตำแหน่งนี้หลังจากเรียกใช้เมธอด `reset()`

- `void reset()`

  เมธอดนี้ใช้เพื่อให้ stream กลับมาชี้ในตำแหน่งที่ระบุไว้ในเมธอด `mark()`

- `boolean markSupported()`

  เมธอดนี้ใช้ตรวจสอบว่า stream ที่ใช้อยู่สนับสนุนการใช้เมธอด `mark()` และ `reset()` หรือไม่

10.1.4 คลาส OutputStream

คลาส OutputStream จะใช้การส่งข้อมูลของ stream ที่เป็นชนิดข้อมูลแบบ byte การส่งข้อมูลของอ็อบเจกต์ชนิด Outputstream จะเป็นการเพิ่มข้อมูลลงใน stream ดังแสดงในรูปที่ 10.7 คลาส OutputStream จะมีเมธอดในการส่งข้อมูลชนิด byte ที่สอดคล้องกับเมธอด read() ในคลาส InputStream โดยคลาสนี้จะมีเมธอด write() ที่เป็นเมธอดแบบ abstract ในรูปแบบต่างๆ ดังนี้

- void write(int i)

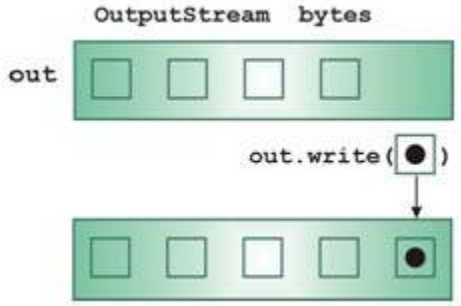
  เมธอดนี้ใช้การเขียน (ส่ง) ข้อมูลลงใน stream โดยจะส่งข้อมูลเพียงแปดบิตท้ายของ argument ที่ชื่อ i อาทิเช่น ถ้า i มีค่าเป็น 781 ก็จะเขียนข้อมูล 00001101 ลงใน stream ทั้งนี้เพราะ 781 มีค่าเป็น 00000000 00000000 00000011 00001101 ในเลขฐานสอง เมธอดนี้จะส่งอ็อบเจกต์ประเภท IOException คืนมากรณีที่เกิดข้อผิดพลาดในการเขียนข้อมูล

- void write(byte [] b)

  เมธอดนี้ใช้ในการเขียนอะเรย์ของชนิดข้อมูล byte ลงใน stream ซึ่งเมธอดนี้อาจส่งอ็อบเจกต์ชนิด NullPointerException ในกรณีที่ตัวแปรอะเรย์ b ยังไม่มีการสร้างขึ้นมาโดยใช้คำสั่ง new

- int write (byte []b, int offset, int length)

  เมธอดนี้ใช้ในการเขียนอะเรย์ของชนิดข้อมูล byte ลงใน stream โดยจะเขียน ข้อมูลของอะเรย์ของ argument ที่ชื่อ b ตั้งแต่ตำแหน่งที่ offset ไปจำนวน length ไบท์ เมธอดจะส่งจำนวนไบท์ที่สามารถเขียนลงใน stream ได้จริงกลับคืนมา และอาจส่งอ็อบเจกต์ชนิด IndexOutOfBoundsException เพิ่มขึ้นมาในกรณีที่หมายเลขของสมาชิกของอะเรย์ใน argument ที่ชื่อ b ไม่อยู่ในช่วงที่ถูกต้อง



รูปที่ 10.7 การเพิ่มข้อมูลลงใน stream

นอกจากนี้คลาส `OutputStream` ยังมีเมธอดอื่นๆที่สำคัญอาทิเช่น

- `void close()`

  เป็นเมธอดที่ใช้ในการปิด stream

- `void flush()`

  เป็นเมธอดที่ใช้ในกรณีที่ stream มีบัฟเฟอร์ (buffer) ในการเก็บข้อมูลที่เขียนไว้ชั่วคราว ซึ่งการเรียกใช้เมธอดนี้จะเป็นการเขียนข้อมูลที่อยู่ในบัฟเฟอร์ลง stream

10.1.5 คลาสประเภท Character Stream

ภาษาจาวากำหนดคลาสพื้นฐานในการจัดการกับอินพุตและเอาต์พุตที่เป็นชนิดข้อมูลแบบ `char` อยู่สองคลาส คือ `Reader` และ `Writer` คลาสทั้งสองเป็นคลาสแบบ `abstract` โดยมี subclass ที่สืบทอดมาเพื่อใช้ในการสร้างอ็อบเจกต์สำหรับจัดการกับโหนดต้นทางและปลายทางในรูปแบบต่างๆเช่น ไฟล์ หน่วยความจำ และไบท์ เป็นต้น

`Reader` เป็นคลาสที่ใช้ในการอ่านข้อมูลของ stream ที่เป็นชนิดข้อมูลแบบ `char` `Reader` จะมีเมธอดที่เหมือนกับคลาส `InputStream` และมีหลักการทำงานที่สอดคล้องกันแต่จะรับข้อมูลหรือ argument ที่เป็นชนิดข้อมูล `char` โดยมีเมธอดต่างๆ ดังนี้

- `int read()`
- `int read(char []c)`
- `int read(char []c, int offset, int length)`
- `void close()`
- `void skip(long n)`
- `boolean markSupported()`
- `void mark(int readAheadlimit)`
- `void reset()`
- `boolean ready()`

`Writer` เป็นคลาสที่ใช้ในการเขียนข้อมูลของ stream ที่เป็นชนิดข้อมูลแบบ `char` `Writer` จะมีเมธอดที่เหมือนกับคลาส `OutputStream` และมีหลักการทีสอดคล้องกันแต่จะรับข้อมูลหรือ argument ที่เป็นชนิดข้อมูล `char` โดยมีเมธอดต่างๆ ดังนี้

- `void write(int c)`

- `void write(char []c)`
- `void write(char []c, int offset, int length)`
- `void close()`
- `void flush()`

นอกจากนี้คลาส `Writer` จะมีเมธอดเพิ่มเติมขึ้นมาเพื่อเขียนชนิดข้อมูลที่เป็น `String` ดังนี้

- `void write(String s)`
- `void write(String s, int offset, int length)`

10.2 โหนดสำหรับ Stream

ภาษาจาวากำหนดโหนดที่เป็นต้นทางและปลายทางของ stream ไว้สามแบบคือ ไฟล์ หน่วยความจำ และไปป์ (pipe)

- ไฟล์คือโหนดสำหรับ stream ที่เป็นไฟล์สำหรับอ่านหรือเขียนข้อมูลชนิด `byte` คลาสที่เป็นโหนดแบบไฟล์สำหรับ byte stream คือ `FileInputStream` และ `FileOutputStream` ส่วนคลาสที่เป็นโหนดแบบไฟล์สำหรับ character stream คือ `FileReader` และ `FileWriter`
- หน่วยความจำคือโหนดสำหรับ stream ที่ใช้สำหรับอ่านหรือเขียนข้อมูลที่เป็น อะเรย์หรือ `String` คลาสที่เป็นโหนดแบบหน่วยความจำ สำหรับ byte stream คือ `ByteArrayInputStream` และ `ByteArrayOutputStream` ส่วนคลาสที่เป็นโหนดแบบหน่วยความจำสำหรับ character stream ในภาษาจาวาจะมีสองแบบคือ อะเรย์ของชนิดข้อมูลแบบ `char` และ `string` โดยมีคลาสที่ชื่อ `CharArrayReader, CharArrayWriter, StringReader` และ `StringWriter` ตามลำดับ
- ไปป์คือโหนดสำหรับ stream ที่จะส่งหรือรับข้อมูลระหว่าง process หรือโปรแกรมเธรด อาทิเช่น stream แบบไปป์ที่เป็นเอาต์พุต (output pipe stream) ของ process หนึ่งอาจเชื่อมต่อกับ stream แบบไปป์ที่เป็นอินพุต (input pipe stream) ของอีก process หนึ่ง คลาสที่เป็นโหนดแบบไปป์สำหรับ byte stream คือ `PipedInputStream` และ `PipedOutputStream` ส่วนคลาสที่เป็นโหนดแบบไปป์สำหรับ character stream คือ `PipedReader` และ `PipedWriter`

ตารางที่ 10.1 สรุปชื่อคลาสที่เป็นโหนดสำหรับ stream ต่างๆ ของโหนดทั้งสามแบบ และโปรแกรมที่ 10.1 แสดงตัวอย่างของการก็อบปี้ข้อมูลของไฟล์หนึ่งไฟล์ไปยังไฟล์อื่น โดยใช้ byte stream โปรแกรมนี้จะเปิดไฟล์ที่มีชื่อตาม argument ที่ชื่อ `args[0]` สำหรับอ่านข้อมูลชนิด `byte` โดยสร้างอ็อบเจกต์ของคลาส `FileInputStream` ที่

ชื่อ `fin` และโปรแกรมนี้จะเปิดไฟล์ที่มีชื่อตาม argument ที่ชื่อ `args[1]` สำหรับเขียนข้อมูลชนิด `byte` โดยสร้างอ็อบเจกต์ของคลาส `FileOutputStream` ที่ชื่อ `fout` จากนั้นโปรแกรมจะอ่านข้อมูลจากอ็อบเจกต์ `fin` ทีละ 100 ไบท์ แล้วนำไปเขียนลงในอ็อบเจกต์ `fout` จนกระทั่งไม่มีข้อมูลที่จะอ่านต่อไปจึงทำการปิดไฟล์ทั้งสอง

โปรแกรมที่ 10.2 แสดงตัวอย่างของก็อปปี้ข้อมูลของไฟล์เช่นเดียวกับโปรแกรมที่ 10.1 แต่จะใช้ character stream โดยสร้างอ็อบเจกต์ของคลาส `FileReader` และ `FileWriter` แทน

ตารางที่ 10.1 คลาสที่เป็นโหนดสำหรับ stream ต่างๆ

ชนิด	Byte Stream	Character Stream
File	FileInputStream FileOutputStream	FileReader FileWriter
Memory: Array	ByteArrayInputStream ByteArrayOutputStream	CharArrayReader CharArrayWriter
Memory: String	-	StringReader StringWriter
Pipe	PipedInputStream PipedOutputStream	PipedReader PipedWriter

โปรแกรมที่ 10.1 การก็อบปี้ข้อมูลของไฟล์หนึ่งไฟล์ไปยังไฟล์อื่นโดยใช้ byte stream

```
import java.io.*;

public class FileCopy {
    public static void main(String args[]) {
        try {
            FileInputStream fin = new FileInputStream(args[0]);
            FileOutputStream fout = new FileOutputStream(args[1]);
            byte b[] = new byte[100];
            int i = fin.read(b);
            while (i != -1) {
                fout.write(b, 0, i);
                i = fin.read(b);
            }
            fin.close();
            fout.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

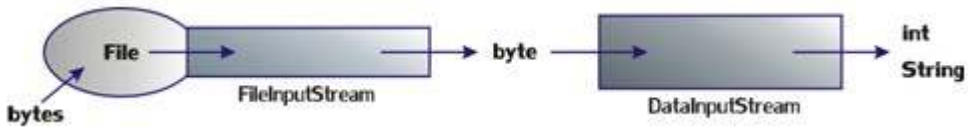
โปรแกรมที่ 10.2 การก็อบปี้ข้อมูลของไฟล์หนึ่งไฟล์ไปยังไฟล์อื่นโดยใช้ character stream

```
import java.io.*;

public class FileCopyReader {
    public static void main(String args[]) {
        try {
            FileReader fr = new FileReader(args[0]);
            BufferedReader br = new BufferedReader(fr);
            FileWriter fw = new FileWriter(args[1]);
            BufferedWriter bw = new BufferedWriter(fw);
            String line = br.readLine();
            while (line != null) {
                bw.write(line, 0, line.length());
                bw.newLine();
                line = br.readLine();
            }
            br.close();
            bw.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

10.3 คลาสประเภท Stream ระดับสูง

โดยทั่วไปโปรแกรมภาษาจาวาจะใช้อ็อบเจกต์ประเภท stream มากกว่าหนึ่งอ็อบเจกต์โดยจะเชื่อมอ็อบเจกต์ของ stream ต่างๆ เข้าด้วยกันเพื่อใช้ในการแปลงชนิดข้อมูลประเภทต่างๆ ตัวอย่างเช่นรูปที่ 10.8 เป็นการเชื่อมต่ออ็อบเจกต์ของคลาส `FileInputStream` เข้ากับอ็อบเจกต์ของคลาส `DataInputStream` โดยอ็อบเจกต์ของคลาส `FileInputStream` จะอ่านข้อมูลของไฟล์เข้ามาโดยมีชนิดข้อมูลแบบ `byte` ส่วนอ็อบเจกต์ของคลาส `DataInputStream` จะทำการแปลงชนิดข้อมูลแบบ `byte` ให้เป็นชนิดข้อมูลอื่นๆทำให้มีเมธอดในการอ่านข้อมูลมากขึ้น การเชื่อมต่ออ็อบเจกต์ประเภท stream สามารถทำได้โดยการเขียนคำสั่งในโปรแกรม เพื่อนำอ็อบเจกต์ของคลาสประเภท stream ชนิดหนึ่งไปเป็น argument ใน constructor ของคลาสประเภท `stream` ที่ต้องการเชื่อมต่อ อาทิเช่น ตัวอย่างในรูปที่ 10.8 สามารถเขียนเป็นคำสั่งได้ดังนี้

```
FileInputStream fin = new FileInputStream("test.dat");
DataInputStream din = new DataInputStream(fin);
```



รูปที่ 10.8 การเชื่อมต่ออ็อบเจกต์ของคลาส

อ็อบเจกต์ที่ใช้ในการเชื่อมต่อ stream จะเป็นอ็อบเจกต์ของคลาสประเภท steam ระดับสูง (high-level stream) ซึ่งสามารถที่จะอ่านหรือเขียนข้อมูลที่เป็นชนิดข้อมูลอื่นๆแล้วแปลงข้อมูลให้เป็นชนิดข้อมูลแบบ `byte` หรือมีบัฟเฟอร์ในการเพิ่มประสิทธิภาพในการอ่านหรือเขียนข้อมูล คลาสเหล่านี้จะไม่สามารถอ่านหรือเขียนข้อมูลไปยังโหนดต้นทางหรือปลายทางที่เป็นไฟล์ หน่วยความจำ หรือ Pipe ได้โดยตรง แต่จะรับข้อมูลมาจาก stream อื่นๆ ที่เป็นคลาสพื้นฐานในการอ่านหรือเขียนข้อมูลตามตารางที่ 10.1 แทน ตารางที่ 10.2 แสดงคลาสประเภท steam ระดับสูงที่กำหนดไว้ใน Java API ซึ่งมีคลาสที่สำคัญดังนี้

- `DataInputStream` และ `DataOutputStream`

  เป็นคลาสที่ใช้ในการแปลงชนิดข้อมูลระหว่างชนิดข้อมูลแบบ `byte` กับชนิดข้อมูลแบบอื่นๆ

- `BufferedInputStream` และ `BufferedOutputStream`

  เป็นคลาสที่มีบัฟเฟอร์สำหรับชนิดข้อมูล `byte` อยู่ภายในเพื่อให้สามารถอ่านหรือเขียนข้อมูลขนาดใหญ่ได้ ซึ่งจะช่วยเพิ่มประสิทธิภาพในการอ่านหรือเขียนข้อมูล

- `PrintStream`

  เป็นคลาสที่ใช้ในการเขียนข้อความที่เป็น `String` ที่แปลงมาจากชนิดข้อมูลแบบ `byte` อ็อบเจกต์ `out` และ `err` ที่อยู่ในคลาส `System` เป็นตัวอย่างของอ็อบเจกต์ที่ใช้คลาสนี้

- `PushbackInputStream`

  เป็นคลาสที่อนุญาตให้ส่งข้อมูลชนิด `byte` ที่เพิ่งอ่านมากลับไปยัง stream ได้

- `BufferedReader` และ `BufferedWriter`

  เป็นคลาสที่มีบัฟเฟอร์สำหรับชนิดข้อมูลแบบ `char` เพื่อให้สามารถอ่านหรือเขียนข้อมูลข้อใหญ่ได้

- `InputStreamReader` และ `OutputStreamWriter`

  เป็นคลาสที่ใช้ในการแปลงชนิดข้อมูลระหว่างชนิดข้อมูลแบบ `char` กับชนิดข้อมูลแบบอื่นๆ

- `PrintWriter`

  เป็นคลาสที่ใช้ในการเขียนข้อความที่เป็น `String` ที่แปลงมาจากชนิดข้อมูลแบบ `char`

- `PushbackReader`

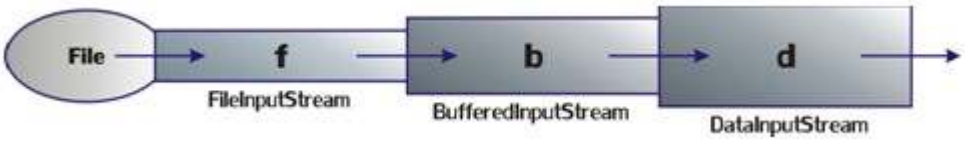
  เป็นคลาสที่อนุญาตให้ส่งข้อมูลชนิด `char` ที่เพิ่งอ่านมากลับไปยัง stream ได้

ตารางที่ 10.2 คลาสประเภท stream ระดับสูง

ชนิด	Byte Stream	Character Stream
Buffering	BufferedInputStream BufferedOutputStream	BufferedReader BufferedWriter
Filtering	FilterInputStream FilterOutputStream	FilterReader FilterWriter
Data conversion	DataInputStream DataOutputStream	-
Printing	PrintStream	PrintWriter
Peeking ahead	PushbackInputStream	PushbackReader

การเชื่อมต่ออ็อบเจกต์ของคลาสประเภท stream ระดับสูงเหล่านี้สามารถที่จะเชื่อมต่อกันได้หลายๆ ชั้น อาทิเช่น รูปที่ 10.9 แสดงตัวอย่างของการเชื่อมอ็อบเจกต์ของคลาส `FileInputStream` เข้ากับอ็อบเจกต์ของคลาส `BufferedInputStream` เพื่อเพิ่มประสิทธิภาพในการอ่านข้อมูลแล้วจึงไปเชื่อมต่อกับอ็อบเจกต์ของคลาส `DataInputStream` เพื่อใช้ในการแปลงชนิดข้อมูลแบบ `byte` ให้เป็นชนิดข้อมูลแบบอื่นๆ โดยจะมีรูปแบบคำสั่งดังนี้

```
FileInputStream f = new FileInputStream("test.dat");
BufferedInputStream b = new BufferedInputStream(f);
DataInputStream d = new DataInputStream(b);
```



รูปที่ 10.9 การเชื่อมอ็อบเจกต์ของคลาสหลายๆ ชั้น

10.3.1 DataInputStream และ DataOutputStream

คลาส DataInputStream เป็นคลาสที่ใช้ในการแปลงชนิดข้อมูลแบบ byte ให้เป็นชนิดข้อมูลแบบอื่นๆ คลาส DataInputStream เป็นคลาสที่สืบทอดมาจากคลาส FilterInputStream และ implements อินเตอร์เฟส DataInput โดยจะมีเมธอดในการอ่านชนิดข้อมูลแบบต่างๆเพิ่มขึ้นมาดังนี้

- boolean readBoolean() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ boolean มาหนึ่งค่า
- byte readByte() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ byte มาหนึ่งไบท์
- char readChar() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ char มาหนึ่งตัวอักษร
- double readDouble() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ double มาหนึ่งค่า
- float readFloat() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ float มาหนึ่งค่า
- int readInt() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ int มาหนึ่งค่า
- long readLong() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ long มาหนึ่งค่า
- short readShort() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ short มาหนึ่งค่า
- String readUTF() throws IOException

  เป็นเมธอดที่ใช้ในการอ่านชนิดข้อมูลแบบ String มาหนึ่งค่าโดยใช้รูปแบบการเข้ารหัสข้อมูลเป็น

UTF-8

คลาส DataOutputStream เป็นคลาสที่ใช้ในการแปลงชนิดข้อมูลใดๆมาเป็นชนิดข้อมูลแบบ byte DataOutputStream เป็นคลาสที่สืบทอดมาจากคลาส FilterOutputStream และ implements อินเตอร์เฟส DataOutput โดยจะมีเมธอดในการอ่านชนิดข้อมูลแบบต่างๆ เพิ่มขึ้นมาดังนี้

- void writeBoolean(boolean b) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ boolean หนึ่งค่า
- void writeByte(int b) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ byte หนึ่งไบท์
- void writeBytes(String s) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ String หนึ่งค่า
- void writeChar(int c) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ char หนึ่งตัวอักขระ
- void writeDouble(double b) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ double หนึ่งค่า
- void writeFloat(float f) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ float หนึ่งค่า
- void writeInt(int i) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ int หนึ่งค่า
- void writeLong(long l) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ long หนึ่งค่า
- void writeShort(int s) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ short หนึ่งค่า
- void writeUTF(String s) throws IOException
  เป็นเมธอดที่ใช้ในการเขียนชนิดข้อมูลแบบ String หนึ่งค่าโดยใช้รูปแบบการเข้ารหัสแบบ UTF-8

โปรแกรมที่ 10.3 แสดงตัวอย่างของการใช้คลาส DataOutputStream เพื่อเก็บชนิดข้อมูลแบบต่างๆ ลงในไฟล์ ส่วนโปรแกรมที่ 10.4 แสดงตัวอย่างของการใช้คลาส DataInputStream เพื่อนำข้อมูลที่เก็บไว้ในไฟล์ดังกล่าว

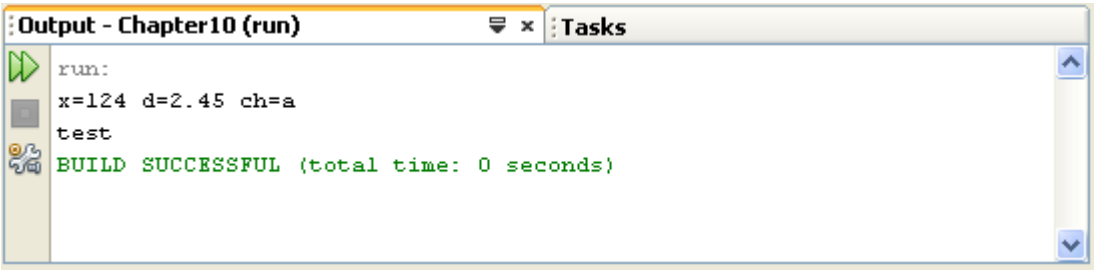
มาแสดงผล โดยจะได้ผลลัพธ์ดังแสดงในรูปที่ 10.4

โปรแกรมที่ 10.3 ตัวอย่างของการใช้คลาส `DataOutputStream`

```
import java.io.*;

public class DataWriterDemo {
    public static void main(String args[]) {
        try {
            FileOutputStream fout = new FileOutputStream("c:/data.dat");
            DataOutputStream dout = new DataOutputStream(fout);
            dout.writeInt(124);
            dout.writeDouble(2.45);
            dout.writeChar('a');
            dout.writeUTF("test");
            dout.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

โปรแกรมที่ 10.4 ตัวอย่างของการใช้คลาส `DataInputStream`

```
import java.io.*;

public class DataReaderDemo {
    public static void main(String args[]) {
        try {
            FileInputStream fin = new FileInputStream("c:/data.dat");
            DataInputStream din = new DataInputStream(fin);
            int x = din.readInt();
            double d = din.readDouble();
            char ch = din.readChar();
            String line = din.readUTF();
            System.out.println("x=" + x + " d=" + d + " ch=" + ch);
            System.out.println(line);
            din.close();
        } catch (IOException ex) {
            System.out.println(ex.toString());
        }
    }
}
```



รูปที่ 10.10 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 10.3 และโปรแกรมที่ 10.4

10.3.2 InputStreamReader และ OutputStreamWriter

เราสามารถที่จะแปลง stream ระหว่างชนิดข้อมูลแบบ byte และชนิดข้อมูลแบบ char ได้โดยใช้คลาส InputStreamReader ซึ่งจะอ่านชนิดข้อมูลแบบ byte แล้วแปลงเป็นชนิดข้อมูลแบบ char และคลาส OutputStreamWriter ซึ่งจะอ่านชนิดข้อมูลแบบ char แล้วแปลงเป็นชนิดข้อมูลแบบ byte

โปรแกรมที่ 10.5 แสดงตัวอย่างของการใช้คลาสดังกล่าว เพื่อเก็บข้อความลงในไฟล์ที่เป็นชนิดข้อมูลแบบ byte โปรแกรมนี้จะสร้างอ็อบเจกต์ของคลาส FileOutputStream แล้วนำไปเชื่อมต่อกับอ็อบเจกต์ของคลาส OutputStreamWriter เพื่อจะแปลงชนิดข้อมูลแบบ char ให้มาเป็นชนิดข้อมูลแบบ byte ส่วนอ็อบเจกต์ของคลาส PrintWriter ที่เชื่อมต่อกับอ็อบเจกต์ของคลาส OutputStreamWriter จะทำหน้าที่แปลงชนิดข้อมูลแบบ String ให้เป็นชนิดข้อมูลแบบ char ส่วนโปรแกรมที่ 11.6 จะเป็นตัวอย่างในการอ่านข้อมูลจากไฟล์แล้วมาแสดงผลโดยใช้อ็อบเจกต์ของคลาส BufferedReader

โปรแกรมที่ 10.5 ตัวอย่างของการใช้คลาสเพื่อเก็บข้อความลงในไฟล์

```
import java.io.*;

public class FileWriter {
    public static void main(String args[]) {
        String line1 = "This is a test message";
        String line2 = "This is another line";
        try {
            FileOutputStream fout = new FileOutputStream("c:\\data.dat");
            OutputStreamWriter oout = new OutputStreamWriter(fout);
            PrintWriter p = new PrintWriter(oout);
            p.println(line1);
            p.println(line2);
            p.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

10.4 คลาส File

คลาสที่ชื่อ File เป็นคลาสที่อยู่ในแพคเก็จ java.io เป็นคลาสที่ใช้ในการสร้างอ็อบเจกต์ที่เป็นไฟล์หรือไดเร็กทอรี่ คลาส File จะมีเมธอดในการจัดการกับไฟล์หรือไดเร็กทอรี่ และเมธอดในการสืบค้นข้อมูลต่างๆ อยู่หลายเมธอด อ็อบเจกต์ของคลาส File จะสร้างมาจาก constructor ที่มีรูปแบบดังนี้

- public File(String name)
- public File(String dir, String name)
- public File(File dir, String name)

โดยที่

- name คือชื่อของไฟล์หรือไดเร็กทอรี่ที่ต้องการเปิดขึ้นมา
- dir คือชื่อของไดเร็กทอรี่ในกรณีที่ไฟล์หรือไดเร็กทอรี่นั้นอยู่ภายในไดเร็กทอรี่อื่น

ในกรณีที่ไม่พบไฟล์หรือไดเร็กทอรี่ตามชื่อที่ระบุ constructor จะส่งอ็อบเจกต์ของคลาส FileNotFoundException เพื่อระบุว่ามีข้อผิดพลาดจากการเปิดไฟล์หรือไดเร็กทอรี่

โปรแกรมที่ 10.6 ตัวอย่างของการใช้คลาสเพื่ออ่านข้อมูลจากไฟล์

```
import java.io.*;

public class FileReader {
    public static void main(String args[]) {
        try {
            FileInputStream fin = new FileInputStream("c:\\data.dat");
            InputStreamReader in = new InputStreamReader(fin);
            BufferedReader bin = new BufferedReader(in);
            System.out.println(bin.readLine());
            System.out.println(bin.readLine());
            bin.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

เมธอดของคลาส `File` ที่ใช้ในการสืบค้นข้อมูลหรือจัดการกับไฟล์ที่สำคัญมีดังนี้

- `boolean exists()`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่าไฟล์หรือไดเร็กทอรี่ว่ามีปรากฏอยู่จริงหรือไม่

- `boolean isDirectory()`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่าอ็อบเจกต์ของคลาส `File` นี้เป็นไดเร็กทอรี่หรือไม่

- `boolean isFile()`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่าอ็อบเจกต์ของคลาส `File` นี้เป็นไฟล์หรือไม่

- `boolean canRead()`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่าไฟล์นี้สามารถใช้ในการอ่านข้อมูลได้หรือไม่

- `boolean canWrite()`

  เป็นเมธอดที่ใช้ในการตรวจสอบว่าไฟล์นี้สามารถใช้ในการเขียนข้อมูลได้หรือไม่

- `String getName()`

  เป็นเมธอดที่ใช้ในการเรียกดูชื่อไฟล์หรือไดเร็กทอรี่ของอ็อบเจกต์ของคลาส `File` นี้

- `String getParent()`

  เป็นเมธอดที่ใช้ในการเรียกดูชื่อไดเร็กทอรี่รากของอ็อบเจกต์ของคลาส `File` นี้

- `String []list()`

  เป็นเมธอดที่ใช้ในการเรียกดูชื่อไฟล์หรือไดเร็กทอรี่ย่อยทั้งหมด ที่อยู่ในอ็อบเจกต์ของคลาส `File` นี้ในกรณีที่เป็นไดเร็กทอรี่

- `boolean mkdir()`

  เป็นเมธอดที่ใช้ในการสร้างไดเร็กทอรี่ย่อย

- `boolean renameTo(File newName)`

  เป็นเมธอดที่ใช้ในการเปลี่ยนชื่ออ็อบเจกต์ของคลาส `File` นี้ให้เป็นชื่อใหม่ตามที่ระบุไว้ในอ็อบเจกต์ของ argument ที่ชื่อ `newName`

คลาส `File` สามารถใช้ในการจัดการกับการเปิดไฟล์เพื่อตรวจสอบว่ามีไฟล์ดังกล่าวหรือไม่ ซึ่ง constructor ของคลาส `FileInputStream` และ `FileOutputStream` หรือ `FileReader` และ `FileWriter` จะมีรูปแบบที่มี argument เป็นอ็อบเจกต์ของคลาส `File` เพื่อช่วยในการตรวจสอบไฟล์ก่อนที่จะใช้ในการอ่านหรือเขียนข้อมูลดังตัวอย่างต่อไปนี้

```
File f = new File("Test.dat");
if (f.exists()) {
      FileInputStream fir = new FileInputStream(f);
}
```

10.4.1 คลาส RandomAccessFile

ภาษาจาวามีคลาสที่สนับสนุนการอ่านหรือเขียนไฟล์เป็นแบบ `random access` กล่าวคือไม่จำเป็นจะต้องอ่านหรือเขียนข้อมูลของไฟล์เรียงตามลำดับจากตำแหน่งแรกสุดไปยังตำแหน่งสุดท้ายของไฟล์ คลาสประเภทนี้คือคลาส `RandomAccessFile` ซึ่งจะใช้เวลาในการอ่านหรือเขียนข้อมูลในตำแหน่งต่างๆเท่ากัน คลาส `RandomAccessFile` จะมี constructor อยู่สองรูปแบบดังนี้

- `public RandomAccessFile(String name, String mode)`
- `public RandomAccessFile(File file, String mode)`

โดยที่

- `name` คือชื่อของไฟล์

- file คืออ็อบเจกต์ของคลาสไฟล์
- mode คือรูปแบบในการเปิดไฟล์ที่มีดังนี้ “r” เป็นการเปิดไฟล์หรืออ่านข้อมูลหรือ “rw” เป็นการเปิดไฟล์เพื่ออ่านและเขียนข้อมูล

ตัวอย่างเช่นคำสั่ง

```
RandomAccessFile myFile = new RandomAccessFile("test.dbf","rw");
```

เป็นการเปิดไฟล์ที่ชื่อ test.dbf สำหรับการอ่านและเขียนข้อมูล

คลาส RandomAccessFile จะมีเมธอดสำหรับการอ่านและเขียนข้อมูลที่เป็นชนิดข้อมูลแบบพื้นฐานชนิดต่างๆ คล้ายกับคลาส DataInputStream และ DataOutputStream และมีเมธอดอื่นๆที่สำคัญเพิ่มเติมมาดังนี้

- long getFilePointer()

  เป็นเมธอดที่ใช้ในการเรียกดูตำแหน่งชี้ของไฟล์
- void seek(long pos)

  เป็นเมธอดที่ใช้ในการเลื่อนตำแหน่งชี้ของไฟล์ไปที่ตำแหน่ง pos
- long length()

  เป็นเมธอดที่ใช้ในการเรียกดูความยาวของไฟล์

10.4.2 ObjectInputStream และ ObjectOutputStream

ภาษาจาวามีคลาสที่ใช้ในการรับและส่งข้อมูลของอ็อบเจกต์ ObjectInputStream และ ObjectOutputStream ทั้งนี้อ็อบเจกต์ที่จะสามารถส่งผ่าน stream เหล่านี้ได้จะต้องเป็นอ็อบเจกต์ของคลาสที่ implements อินเตอร์เฟส Serializable ซึ่งเป็นอินเตอร์เฟสในแพคเก็จ java.io ซึ่งไม่มีเมธอดใดๆ อยู่ แต่จะเป็นอินเตอร์เฟสที่ใช้ระบุว่า อ็อบเจกต์ดังกล่าวสามารถเก็บข้อมูลหรือดึงข้อมูลของอ็อบเจกต์กลับคืนมาได้ อ็อบเจกต์ที่สามารถเก็บหรือดึงข้อมูลได้เป็นอ็อบเจกต์ประเภทที่เรียกว่า persistence ซึ่งข้อมูลที่จะเก็บหรือเรียกดูได้จะมีเฉพาะคุณลักษณะของอ็อบเจกต์หรือคุณลักษณะของคลาสเท่านั้น ส่วนเมธอดจะไม่สามารถเก็บได้

อ็อบเจกต์บางประเภทจะไม่สามารถที่จะ implements อินเตอร์เฟส Serializable ได้ เนื่องจากข้อมูลของ

อ็อบเจกต์อาจเปลี่ยนแปลงได้ตลอดเวลา อาทิเช่น อ็อบเจกต์ของคลาส `Thread` หรือ `FileInputStream` ในกรณีที่อ็อบเจกต์ใดเป็นอ็อบเจกต์ประเภท `Serializable` แล้วมีสมาชิกที่เป็นอ็อบเจกต์ประเภท non-serializable อยู่ การเก็บหรือเรียกดูข้อมูลของอ็อบเจกต์จะไม่สามารถทำได้และจะเกิดข้อผิดพลาดประเภท `NotSerializableException` ขึ้นมา

เราสามารถจะระบุคุณลักษณะของอ็อบเจกต์หรือคุณลักษณะของคลาสที่ไม่ต้องการจะเก็บหรือเรียกดูข้อมูลผ่าน stream ได้โดยการระบุให้คุณลักษณะดังกล่าวมี modifier เป็นคีย์เวิร์ด `transient` ตัวอย่างเช่น

```
public class Student implements Serializable {
      private String id,name;
      private double gpa;
      private transient String parent;
}
```

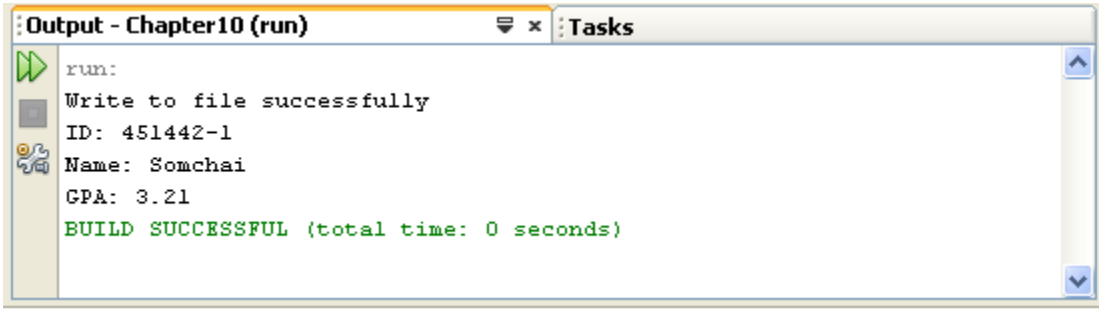
เป็นคลาสที่มีคีย์เวิร์ด `transient` ในคุณลักษณะชื่อ `parent` ซึ่งจะมีผลทำให้คุณลักษณะนี้ไม่สามารถเก็บหรือเรียกดูข้อมูลผ่าน stream คีย์เวิร์ด `transient` สามารถใช้กับคุณลักษณะที่เป็นอ็อบเจกต์ประเภท non-serializable เพื่อป้องกันไม่ให้เกิดข้อผิดพลาดในการเก็บหรือเรียกดูข้อมูลเช่น

```
public class Student implements Serializable {
      public transient Thread myThread;
      private String id,name;
      private double gpa;
}
```

คลาส `ObjectOutputStream` เป็นคลาสที่ใช้ในการเขียนอ็อบเจกต์ใดๆลงใน stream โดยมีเมธอด `writeObject()` ที่ใช้ในการเขียนข้อมูล ส่วนคลาส `ObjectInputStream` เป็นคลาสที่ใช้ในการรับอ็อบเจกต์ใดๆมาจาก stream โดยมีเมธอด `readObject()` ที่ใช้ในการอ่านข้อมูลซึ่งจะได้อ็อบเจกต์ของคลาสที่ชื่อ `Object` คืนมา ดังนั้นจึงจำเป็นที่จะต้องทำการ casting อ็อบเจกต์ให้สอดคล้องกับชนิดข้อมูลของคลาสที่ต้องการ โปรแกรมที่ 10.6 แสดงตัวอย่างของการเขียนและอ่านข้อมูลของอ็อบเจกต์ลงในไฟล์ `data.dat` โดยจะให้ผลลัพธ์ดังแสดงในรูปที่ 10.3

โปรแกรมที่ 10.7 การเขียนและอ่านข้อมูลของอ็อบเจกต์ลงในไฟล์

```
import java.io.*;

public class ObjectPersistenceDemo {
    public static void main(String args[]) {
        Student s = new Student("451442-1", "Somchai", 3.21);
        try {
            FileOutputStream fout = new FileOutputStream("c:/data.dat");
            ObjectOutputStream oout = new ObjectOutputStream(fout);
            oout.writeObject(s);
            System.out.println("Write to file successfully");
            oout.close();

            FileInputStream fin = new FileInputStream("c:/data.dat");
            ObjectInputStream oin = new ObjectInputStream(fin);
            Student s2 = (Student) oin.readObject();
            oin.close();

            s2.showAllDetails();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```



รูปที่ 10.11 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 10.6

สรุปเนื้อหาของบท

- stream เปรียบเสมือนท่อส่งข้อมูล สามารถแบ่งออกได้เป็น 2 ประเภทคือ byte stream และ character stream
- byte stream มีคลาสพื้นฐานคือ คลาส `InputStream` และ `OutputStream`

- character stream มีคลาสพื้นฐานคือ คลาส `Reader` และ `Writer`
- คลาส `InputStream` และ `OutputStream` เป็น low-level stream ที่สามารถอ่านหรือเขียนข้อมูลชนิด `byte` ได้เท่านั้น
- คลาส `Reader` และ `Writer` เป็น low-level stream ที่สามารถอ่านหรือเขียนข้อมูลชนิด `char` ได้เท่านั้น
- คลาสที่เป็น high-level stream สำหรับ byte stream จะสืบทอดมาจากคลาส `FilterInputStream` หรือ `FilterOutputStream`
- คลาสที่เป็น high-level stream สำหรับ character stream คือคลาส `BufferedReader`, `BufferedWriter`, `InputStreamReader`, `OutputStreamReader`, `LineNumberReader`, `PrintWriter` และ `PushbackReader`
- การเชื่อมต่อ stream หลายชั้น จะทำให้สามารถเรียกใช้เมธอดได้หลายรูปแบบตามความต้องการในการใช้งาน
- คลาส `File` ใช้ในการสร้างอ็อบเจกต์ของไฟล์หรือไดเร็กทอรี่ โดยจะมีเมธอดที่ใช้ในการจัดการกับไฟล์
- คลาส `RandomAccessFile` จะมีรูปแบบในการอ่านและเขียนข้อมูลเป็นแบบ random access
- อ็อบเจกต์ที่จะสามารถรับหรือส่งผ่าน stream ได้จะต้องเป็นอ็อบเจกต์ของคลาสที่ `implements` อินเตอร์เฟส `Serializable`

บทที่ 11 โปรแกรมแบบเธรด

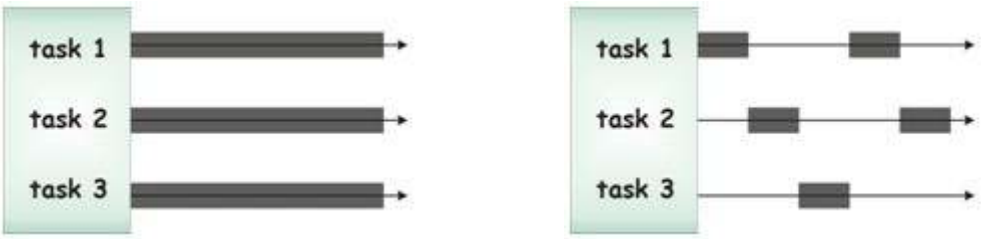
เนื้อหาในบทนี้เป็นการแนะนำการเขียนโปรแกรมแบบเธรด โดยเริ่มต้นจากการอธิบายความหมายและองค์ประกอบของเธรด แนะนำวิธีการสร้างคลาสแบบเธรดโดยการสืบทอดจากคลาสที่ชื่อ `Thread` หรือ `implements` อินเตอร์เฟสที่ชื่อ `Runnable` อธิบายขั้นตอนการทำงานของโปรแกรมแบบเธรด เมธอดสำคัญต่างๆ ที่เกี่ยวข้องกับการทำงานของคลาส `Thread` และตอนท้ายของบทเป็นการอธิบายการใช้คีย์เวิร์ด `synchronized` เพื่อป้องกันการผิดพลาดของการอ่านหรือเขียนข้อมูลที่อาจเกิดขึ้นจากโปรแกรมแบบเธรด

11.1 โปรแกรมแบบมัลติเธรด

คุณลักษณะเด่นข้อหนึ่งของภาษาจาวาคือ ความสามารถในการพัฒนาโปรแกรมแบบมัลติเธรด (Multithreaded) ที่สามารถทำงานพร้อมกันได้ ซึ่งโปรแกรมแบบมัลติเธรดนี้ใช้หลักการของระบบปฏิบัติการแบบ Multitasking ในการกำหนดให้โปรแกรมสามารถทำงานหลายงาน (task) ได้พร้อมกัน ระบบปฏิบัติการแบบ Multitasking จะเป็นระบบปฏิบัติการที่อนุญาตให้ผู้ใช้สามารถส่งโปรแกรมให้ระบบปฏิบัติการทำงานได้มากกว่าหนึ่งโปรแกรมพร้อมกัน ซึ่งโปรแกรมแต่ละโปรแกรมที่รันอยู่ในระบบปฏิบัติการจะสร้าง process ขึ้นมา และระบบปฏิบัติการแบบ Multitasking จะมี process หลายๆ process เข้าคิวรอการทำงาน โดยระบบปฏิบัติการจะกำหนดลำดับของการทำงานของ process เอง

โปรแกรมมัลติเธรดจะเป็นการทำงานหลายงานพร้อมกันโดยแต่ละงานจะเรียกว่าเธรด (Thread) ซึ่งเป็นแบบ lightweight process แตกต่างจาก process ที่ทำงานภายใต้ระบบปฏิบัติการแบบ Multitasking ตรงที่ process แต่ละ process จะมีความเป็นอิสระต่อกัน แต่เธรดแต่ละเธรดอาจใช้ข้อมูลร่วมกันได้

โปรแกรมแบบเธรดจะมีหลักการทำงานที่แตกต่างกัน ทั้งนี้ขึ้นอยู่กับระบบปฏิบัติการและจำนวนซีพียู ในระบบคอมพิวเตอร์ที่มีซีพียูหลายตัว โปรแกรมแบบเธรดแต่ละโปรแกรมสามารถที่จะรันพร้อมกันได้ดังแสดงในรูปที่ 11.1 (ก) ส่วนในระบบคอมพิวเตอร์ที่มีซีพียูตัวเดียว ระบบปฏิบัติการจะเป็นส่วนที่จะจัดการตารางการทำงานของโปรแกรมเธรด ซึ่งอาจแบ่งเวลาการทำงานของซีพียูดังแสดงในรูปที่ 11.1 (ข)

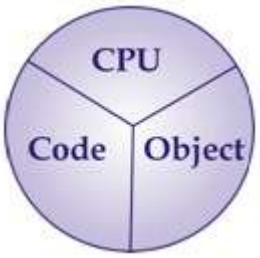


(ก) (ข)

รูปที่ 11.1 การทำงานของโปรแกรมแบบเธรด

องค์ประกอบของโปรแกรมแบบเธรดจะมีสามส่วนดังแสดงในรูปที่ 11.2 คือ

- ซีพียู (CPU) คือการจัดการทำงานของโปรแกรมแบบเธรด
- โค้ด (Code) คือคลาสในภาษาจาวาที่เป็นคลาสแบบเธรด
- อ็อบเจกต์ (Object) คืออ็อบเจกต์ของคลาสแบบเธรด



รูปที่ 11.2 องค์ประกอบของโปรแกรมแบบเธรด

โปรแกรมภาษาจาวาสามารถกำหนดให้อ็อบเจกต์ของคลาสใดๆ ทำงานเป็นแบบเธรดได้ ซึ่งก็จะทำให้สามารถรันโปรแกรมของอ็อบเจกต์แบบเธรดหลายๆ อ็อบเจกต์พร้อมกันได้ โดยภาษาจาวาจะมีตัวจัดตารางเวลาเธรด (Thread Scheduler) เพื่อจัดลำดับการทำงานของอ็อบเจกต์แบบเธรด ทั้งนี้โปรแกรมบางประเภทจำเป็นต้องพัฒนาเป็นแบบเธรด อาทิเช่น โปรแกรมภาพกราฟิกแบบเคลื่อนไหว (Graphics Animation) โปรแกรมนาฬิกา และโปรแกรมแม่ข่าย (Application Server) เป็นต้น

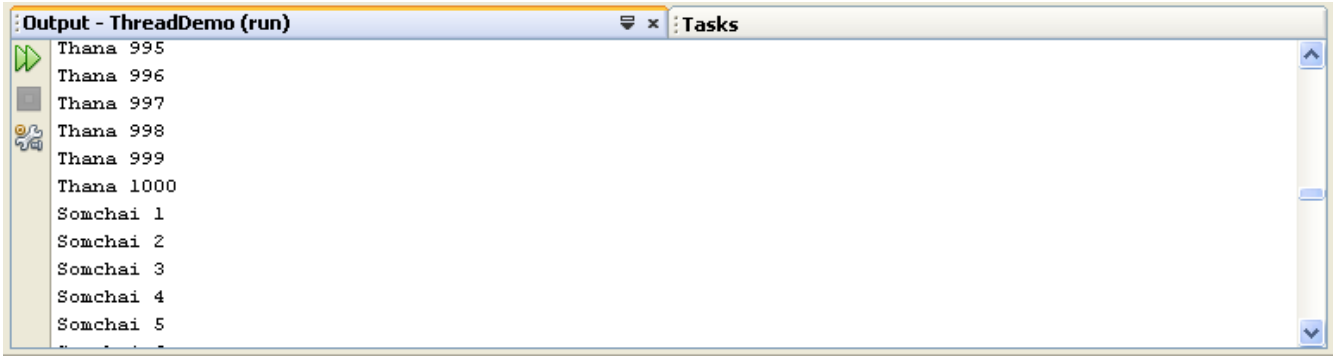
11.2 การสร้างคลาสแบบเธรด

โปรแกรมแบบเธรดในภาษาจาวาจะใช้หลักการของการสร้างอ็อบเจกต์ของคลาสแบบเธรด แล้วเรียกใช้เมธอดเพื่อส่งให้อ็อบเจกต์ดังกล่าวทำงานพร้อมกัน คลาสแบบเธรดในภาษาจาวาคือคลาสที่สืบทอดมาจากคลาสที่ชื่อ `Thread` หรือคลาสที่ `implements` อินเตอร์เฟส `Runnable` อ็อบเจกต์ที่สร้างมาจากคลาสแบบเธรดจะเรียกว่าอ็อบเจกต์ประเภท runnable

โปรแกรมที่ 11.1 โปรแกรมเป็นโปรแกรมที่ทำงานในรูปแบบปกติ ซึ่งจะมีคลาส `PrintName` ที่มีเมธอด `run()` ซึ่งจะพิมพ์ค่าของคุณลักษณะ `name` และหมายเลขรอบ (ตัวแปร i) จำนวน 1000 ครั้ง โดยคลาส `TestPrintName` จะสร้างอ็อบเจกต์ของคลาส `PrintName` ขึ้นมาสองอ็อบเจกต์คือ `p1` และ `p2` จากนั้นจะเรียกใช้เมธอด `run()` เพื่อพิมพ์ข้อความออกมา เนื่องจากโปรแกรมนี้ไม่ได้เป็นโปรแกรมแบบเธรด ดังนั้นซีพียูจะทำคำสั่ง `p1.run()` ให้เสร็จก่อนแล้วจึงจะทำคำสั่ง `p2.run()` ดังแสดงผลลัพธ์ที่ได้จากการรันโปรแกรมในรูปที่ 11.3 และอธิบายการทำงานได้ดังแสดงในรูปที่ 11.4

<u>โปรแกรมที่ 11.1</u> <u>ตัวอย่างโปรแกรมที่ทำงานในรูปแบบปกติ</u>

```
class PrintName extends Thread {
    private String name;
    public PrintName(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 1000; i++) {
            System.out.println(name + " " + i);
        }
    }
}
public class PrintNameThread {
    public static void main(String args[]) {
        PrintName p1 = new PrintName("Thana");
        PrintName p2 = new PrintName("Somchai");
        p1.run();
        p2.run();
    }
}
```



รูปที่ 11.3 แสดงผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.1

รูปที่ 11.4 อธิบายการทำงานของโปรแกรมที่ 11.1

11.2.1 การ extends คลาสที่ชื่อ Thread

คลาสแบบเธรดสามารถสร้างได้โดยการสืบทอดคลาสที่ชื่อ `Thread` ซึ่งมีรูปแบบดังนี้

```
public class ClassName extends Thread {
      public void run(){
            [statements]
      }
}
```

ซึ่งในกรณีนี้จะต้องทำการ override เมธอด `run()` ที่มีชุดคำสั่งที่ต้องการให้โปรแกรมทำงานในลักษณะแบบเธรดอยู่ภายในบล็อกคำสั่ง โดยเมื่อมีการเรียกใช้เมธอด `start()` ซึ่งเป็นการลงทะเบียนอ็อบเจกต์ดังกล่าวในตัวจัดตารางเวลาเธรด และจะทำให้ซีพียูเรียกรันคำสั่งในเมธอด `run()` ของอ็อบเจกต์นี้

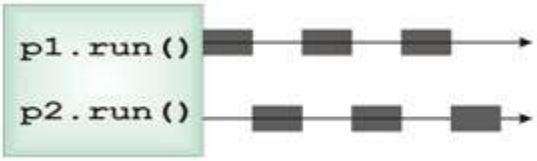
โปรแกรมที่ 11.2 แสดงตัวอย่างการปรับปรุงคลาส `PrintName` ให้เป็นคลาสแบบเธรด โดยการสืบทอดมาจากคลาสที่ชื่อ `Thread` ซึ่งคลาส `PrintNameThread` จะเป็นการสร้างอ็อบเจกต์ของคลาส `PrintName` ขึ้นมาสองอ็อบเจกต์คือ `p1` และ `p2` ซึ่งทั้งสองเป็นอ็อบเจกต์ประเภท `Thread` จากนั้นจะเป็นการเรียกใช้เมธอด `p1.start()` และ `p2.start()` จะเป็นการส่งอ็อบเจกต์แบบเธรดทั้งสองเข้าไปจองคิวกับตัวจัดตารางเวลาเธรด ซึ่งซีพียูก็จะแบ่งเวลาให้อ็อบเจกต์ทั้งสองมีโอกาสสลับกันทำงานได้ ดังแสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมในรูปที่ 11.5 และอธิบายการทำงานได้ดังแสดงในรูปที่ 11.6

<u>โปรแกรมที่ 11.2</u> <u>ตัวอย่างแสดงคลาสที่สืบทอดมาจากคลาสที่ชื่อ `Thread`</u>

```
class PrintName extends Thread {
    private String name;
    public PrintName(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 1000; i++) {
            System.out.println(name + " " + i);
        }
    }
}
public class PrintNameThread {
    public static void main(String args[]) {
        PrintName p1 = new PrintName("Thana");
        PrintName p2 = new PrintName("Somchai");
        p1.start();
        p2.start();
    }
}
```

```
Output - ThreadDemo (run)        Tasks
Thana 84
Thana 85
Thana 86
Thana 87
Thana 88
Thana 89
Somchai 1
Somchai 2
Somchai 3
Somchai 4
Somchai 5
```

รูปที่ 11.5 แสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.2



รูปที่ 11.6 อธิบายการทำงานของโปรแกรมที่ 11.2

11.2.2 การ implements อินเตอร์เฟส Runnable

คลาสแบบเธรดสามารถสร้างได้โดยการ implements อินเตอร์เฟส Runnable ซึ่งเมธอดเดียวที่จะต้องเขียนบล็อกคำสั่งคือเมธอด run() ซึ่งมีรูปแบบดังนี้

```
public class ClassName implements Runnable {
      public void run(){
            [statements]
      }
}
```

คำสั่งที่อยู่ในเมธอด run() คือชุดคำสั่งที่ต้องการให้โปรแกรมทำงานในลักษณะแบบเธรด โปรแกรมจาวาที่รันอ็อบเจกต์ของคลาสแบบเธรด ในลักษณะนี้จะต้องมีการสร้างอ็อบเจกต์ของคลาสที่ชื่อ Thread ก่อนโดย constructor ของคลาสที่ชื่อ Thread จะมี argument เพื่อรับอ็อบเจกต์ของคลาสที่ implements อินเตอร์เฟส Runnable โดยมีรูปแบบดังนี้

```
public Thread(Runnable obj) {
}
```

โดยที่

- obj คืออ็อบเจกต์ของคลาสที่ implements อินเตอร์เฟส Runnable

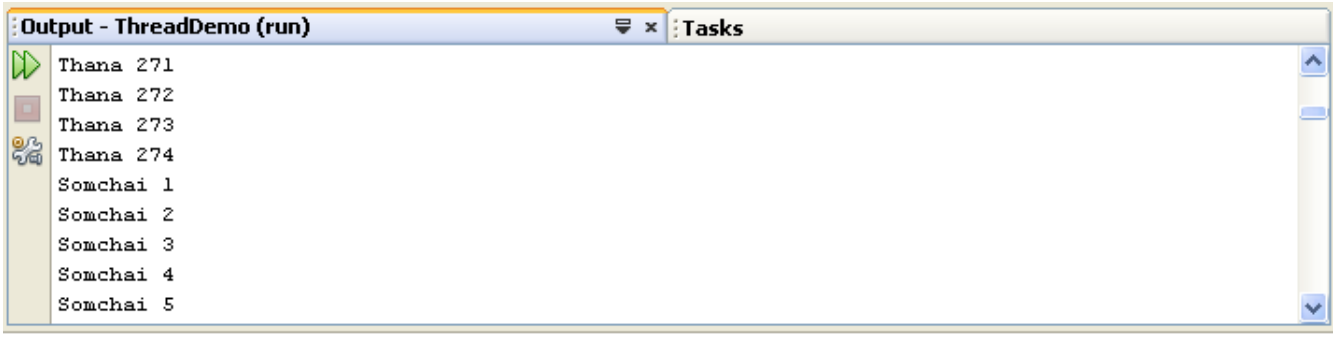
ภายหลังจากที่สร้างอ็อบเจกต์ของคลาสแบบเธรดแล้ว เราสามารถจะสั่งงานให้อ็อบเจกต์ obj เริ่มทำงานได้โดยการเรียกใช้เมธอด start() ที่อยู่ในคลาสที่ชื่อ Thread ซึ่งคลาสที่ชื่อ Thread จะลงทะเบียนอ็อบเจกต์ดังกล่าว

ลงในตัวจัดตารางเวลาเธรด และเมื่อซีพียูเรียกรันโปรแกรมของอ็อบเจกต์ดังกล่าว คำสั่งในเมธอด `run()` จะถูกสั่งงานตามลำดับ

โปรแกรมที่ 11.3 เป็นตัวอย่างการปรับปรุงคลาส `PrintName` ให้เป็นคลาสแบบเธรด โดยการ `implements` อินเตอร์เฟส `Runnable` คลาส `PrintNameThread` จะสร้างอ็อบเจกต์ของคลาส `PrintName` ขึ้นมาสองอ็อบเจกต์คือ `p1` และ `p2` ซึ่งทั้งสองเป็นอ็อบเจกต์ประเภท `Runnable` จากนั้นโปรแกรมได้สร้างอ็อบเจกต์ของคลาสที่ชื่อ `Thread` ขึ้นมาสองอ็อบเจกต์คือ `t1` และ `t2` ซึ่งอ็อบเจกต์ทั้งสองจะมีซอร์ดโค้ดเดียวกันคือ `PrintName` แต่จะมีอ็อบเจกต์แบบ runnable ต่างกันคือ `p1` และ `p2` การเรียกใช้เมธอด `t1.start()` และ `t2.start()` จะเป็นการส่งอ็อบเจกต์แบบเธรดทั้งสองเข้าไปจองคิวกับตัวจัดตารางเวลาเธรด ซึ่งจะได้ผลลัพธ์ในลักษณะเดียวกับโปรแกรมที่ 11.2 ดังแสดงตัวอย่างผลลัพธ์ในรูปที่ 11.7

<u>โปรแกรมที่ 11.3</u> <u>ตัวอย่างแสดงคลาสที่ `implements` อินเตอร์เฟส `Runnable`</u>

```
class PrintName implements Runnable {
    private String name;
    public PrintName(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 1000; i++) {
            System.out.println(name + " " + i);
        }
    }
}
public class PrintNameThread {
    public static void main(String args[]) {
        PrintName p1 = new PrintName("Thana");
        PrintName p2 = new PrintName("Somchai");
        Thread t1 = new Thread(p1);
        Thread t2 = new Thread(p2);
        t1.start();
        t2.start();
    }
}
```



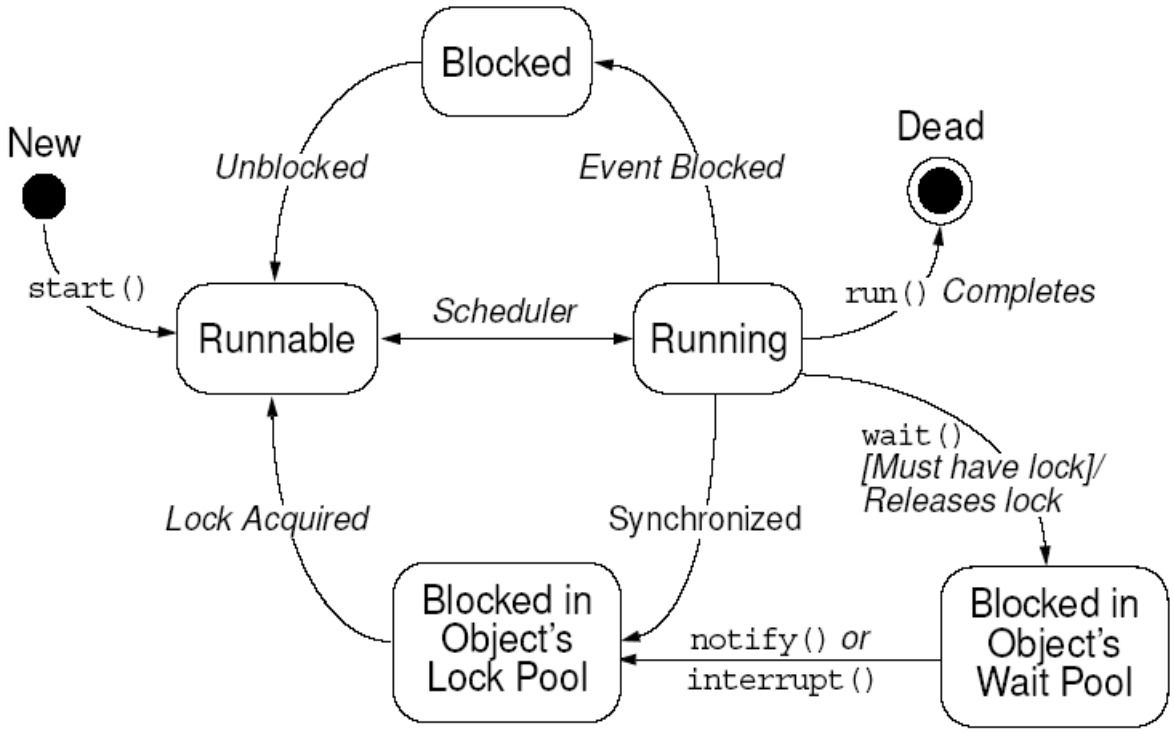
รูปที่ 11.7 แสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.3

การสร้างคลาสแบบเธรดทั้งสองวิธีมีข้อดีต่างกันดังนี้

- การสร้างคลาสโดยการสืบทอดคลาสที่ชื่อ `Thread` จะเป็นการเขียนโปรแกรมที่สั้นกว่า
- การสร้างคลาสโดย `implements` อินเตอร์เฟส `Runnable` จะมีหลักการเชิงอ็อบเจกต์ที่ดีกว่า เนื่องจากคลาสดังกล่าวไม่ใช่คลาสที่สืบทอดมาจากคลาสที่ชื่อ `Thread` โดยตรง นอกจากนี้ยังทำให้การเขียนโปรแกรมมีรูปแบบเดียวกันเสมอ เพราะคลาสในจาวาจะสืบทอดจากคลาสอื่นได้เพียงคลาสเดียวเท่านั้น แต่สามารถที่จะ `implements` อินเตอร์เฟสได้หลายอินเตอร์เฟสพร้อมกัน

11.3 วงจรการทำงานของเธรด

อ็อบเจกต์แบบเธรดเมื่อลงทะเบียนไว้กับตัวจัดตารางเวลาเธรดแล้ว อาจยังไม่มีการรันโปรแกรมโดยทันที แต่ทั้งนี้จะขึ้นอยู่กับสถานะของอ็อบเจกต์ ซึ่งวงจรการทำงานของเธรดจะมีการทำงานดังแสดงในรูปที่ 11.8 โดยจะมีสถานะต่างๆ ดังนี้



รูปที่ 11.8 วงจรการทำงานของเธรด

- สถานะ New
  - ➢ เป็นสถานะเมื่อมีการสร้างอ็อบเจกต์ของคลาสแบบเธรด ก่อนที่จะมีการเรียกใช้เมธอด `start()`
- สถานะ Runnable และสถานะ Running
  - ➢ Runnable เป็นสถานะที่อ็อบเจกต์แบบเธรดพร้อมที่จะทำงาน ซึ่งอาจเกิดจากการเรียกใช้เมธอด `start()` หรือเกิดจากการกลับมาจากสถานะที่ถูกบล็อกหรือสถานะ Running โดยอ็อบเจกต์แบบเธรดที่อยู่ในสถานะ Runnable จะเข้าสู่สถานะ Running ซึ่งเป็นสถานะที่ซีพียูรันคำสั่งของอ็อบเจกต์แบบเธรด เมื่อตัวจัดตารางเวลาเธรดจัดเวลาทำงานให้ โดยจะทำงานตามชุดคำสั่งในเมธอด `run()`
  - ➢ อ็อบเจกต์แบบเธรดจะหยุดการทำงานและกลับเข้าสู่สถานะ Runnable อีกครั้ง เมื่อตัวจัดตารางเวลาเปลี่ยนให้อ็อบเจกต์แบบเธรดอื่นๆ ที่อยู่ในสถานะ Runnable เข้ามาทำงานแทน โดยทั่วไปตัวจัดตารางเวลาจะพยายามจัดการทำงานของอ็อบเจกต์แบบเธรดตามค่าลำดับความสำคัญ (Priority) ที่กำหนดไว้ ซึ่งปกติจะมีค่าเริ่มต้น (Default) เป็น 5 ค่าต่ำสุดเป็น 1 และค่าสูงสุดเป็น 10 โดยอ็อบเจกต์แบบเธรดที่มีค่าลำดับความสำคัญสูงกว่าจะได้รับโอกาสในการถูกเรียกไปรันมากกว่า

- สถานะ Blocked
  - ➢ อ็อบเจกต์แบบเธรดที่อยู่ในสถานะ Running อาจถูกบล็อกแล้วย้ายไปอยู่ในสถานะ Blocked ได้ เนื่องจากเหตุการณ์ต่างๆ ดังนี้
    - มีการเรียกใช้เมธอด `sleep()`
    - อ็อบเจกต์แบบเธรดรอรับการทำงานที่เป็นอินพุตหรือเอาต์พุต
  - ➢ อ็อบเจกต์แบบเธรดที่อยู่ในสถานะ Blocked จะกลับเข้าสู่สถานะ Runnable ได้อีกครั้งหนึ่งจากเหตุการณ์ต่างๆ ดังนี้
    - เมื่อหมดระยะเวลาการพักที่กำหนดในเมธอด `sleep()`
    - เมื่อมีการส่งข้อมูลอินพุตหรือเอาต์พุต ในกรณีที่อ็อบเจกต์แบบเธรดรอรับหรือส่งข้อมูล
- สถานะ Blocked in Object's Lock Pool
  - ➢ เมื่ออ็อบเจกต์แบบเธรดที่ทำงานอยู่ในสถานะ Running เจอคำสั่ง `synchronized` แต่ไม่ได้รับล็อค (Lock) ของอ็อบเจกต์ที่มีคำสั่ง `synchronized` อยู่ เนื่องจากล็อคอยู่กับอ็อบเจกต์แบบเธรดตัวอื่น (อ็อบเจกต์ของคลาสใดๆ ในจาวา จะมีได้เพียงแค่หนึ่งล็อคเท่านั้น) ทำให้อ็อบเจกต์ของเธรดจะต้องเข้ามาอยู่ในสถานะนี้ก่อน จนกว่าจะได้ล็อค แล้วจึงเข้าสู่สถานะ Runnable ต่อไป
  - ➢ เมื่อถูกระงับการทำงาน (Interrupt)
- สถานะ Blocked in Object's Wait Pool
  - ➢ เมื่ออ็อบเจกต์แบบเธรดที่ทำงานอยู่ในสถานะ Running เจอคำสั่ง `synchronized` และได้รับล็อค ของอ็อบเจกต์ที่มีคำสั่ง `synchronized` อยู่ ต่อมาเมื่อเจอเมธอด `wait()` จะต้องปล่อยล็อค แล้วเข้าสู่สถานะนี้ เพื่อรอให้อ็อบเจกต์แบบเธรดตัวอื่นเรียกเมธอด `notify()` หรือ `notifyAll()` แล้วจึงเข้าสู่สถานะ Blocked in Object's Lock Pool ต่อไป
- สถานะ Dead
  - ➢ อ็อบเจกต์แบบเธรดจะเข้าสู่สถานะ Dead เมื่อสิ้นสุดการทำงานในชุดคำสั่งของเมธอด `run()` หรือเมื่อมีการส่งอ็อบเจกต์ประเภท `Exception` ที่ไม่ได้มีการตรวจจับ (`catch`) ในเมธอด `run()`

11.4 เมธอดของคลาส `Thread`

คลาสที่ชื่อ `Thread` มีเมธอดต่างๆ ที่สำคัญดังนี้

- `void start()`

  เป็นเมธอดที่เรียกใช้งานเพื่อกำหนดให้อ็อบเจกต์แบบเธรด เริ่มทำงานโดยเข้าสู่สถานะ Runnable โดยเมื่อมีการเรียกใช้เมธอด `start()` จะมีการสร้างสแต๊ก (Stack) ขึ้นมาใหม่เพิ่มขึ้นมาจากสแต๊กสำหรับเมธอด `main()` เพื่อให้สามารถทำงานสลับกันกับเธรดที่ทำงานเมธอด `main()` ได้

- `void run()`

  เป็นเมธอดที่จะถูกเรียกใช้เมื่อซีพียูรันโปรแกรมอ็อบเจกต์แบบเธรด

- `boolean isAlive()`

  เป็นเมธอดที่ใช้ตรวจสอบว่าอ็อบเจกต์แบบเธรดยังมีสถานะ Running อยู่หรือไม่

- `void setPriority(int p)`

  เป็นเมธอดที่ใช้ในการกำหนดลำดับความสำคัญของอ็อบเจกต์แบบเธรด ซึ่งจะมีค่าได้ระหว่าง 1 (MIN_PRIORITY) ถึง 10 (MAX_PRIORITY) โดยที่ค่าปกติอยู่ที่ 5 (NORM_PRIORITY)

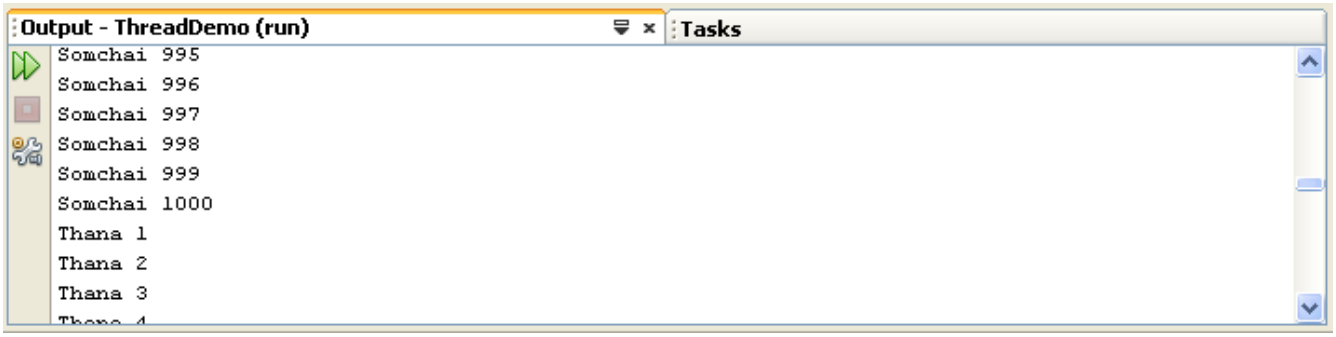
- `int getPriority()`

  เป็นเมธอดที่ใช้ในการเรียกดูค่าลำดับความสำคัญของอ็อบเจกต์แบบเธรด

ตัวอย่างโปรแกรมที่ 11.4 แสดงการกำหนดค่าลำดับความสำคัญให้กับอ็อบเจกต์แบบเธรด โดยเมื่อเปรียบเทียบผลลัพธ์ของโปรแกรมที่ 11.3 กับผลลัพธ์ของโปรแกรมที่ 11.4 จะพบว่าอ็อบเจกต์แบบเธรดที่ถูกกำหนดค่าลำดับความสำคัญที่สูงกว่าจะมีโอกาสถูกเลือกจากตัวจัดตารางเวลาเธรดให้ได้รันมากกว่าอ็อบเจกต์แบบเธรดที่มีลำดับความสำคัญที่ต่ำกว่า ดังแสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมในรูปที่ 11.9

โปรแกรมที่ 11.4 ตัวอย่างแสดงการกำหนดค่าลำดับความสำคัญให้กับอ็อบเจกต์แบบเธรด

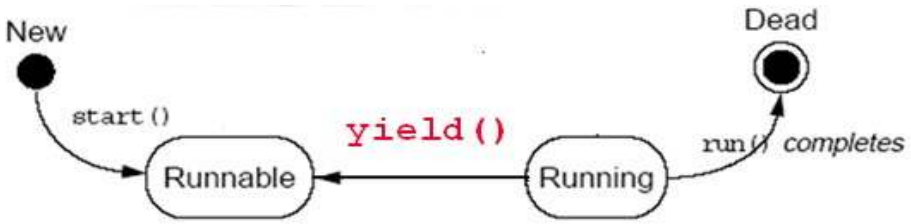
```
class PrintName implements Runnable {
    private String name;
    public PrintName(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 1000; i++) {
            System.out.println(name + " " + i);
        }
    }
}
public class PrintNameThread {
    public static void main(String args[]) {
        PrintName p1 = new PrintName("Thana");
        PrintName p2 = new PrintName("Somchai");
        Thread t1 = new Thread(p1);
        Thread t2 = new Thread(p2);
        t1.start();
        t2.start();
        t2.setPriority(10);
    }
}
```



รูปที่ 11.9 แสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.4

- `static void yield()`

  เป็นเมธอดที่กำหนดให้อ็อบเจกต์แบบเธรดที่กำลังรันอยู่ (อยู่ในสถานะ Running) หยุดการทำงานชั่วคราว แล้วกลับเข้าสู่สถานะ Runnable เพื่อเปิดโอกาสให้อ็อบเจกต์แบบเธรดตัวอื่นๆ ที่รออยู่ในสถานะ Runnable ได้มีโอกาสรัน ดังอธิบายการทำงานได้ในรูปที่ 11.10

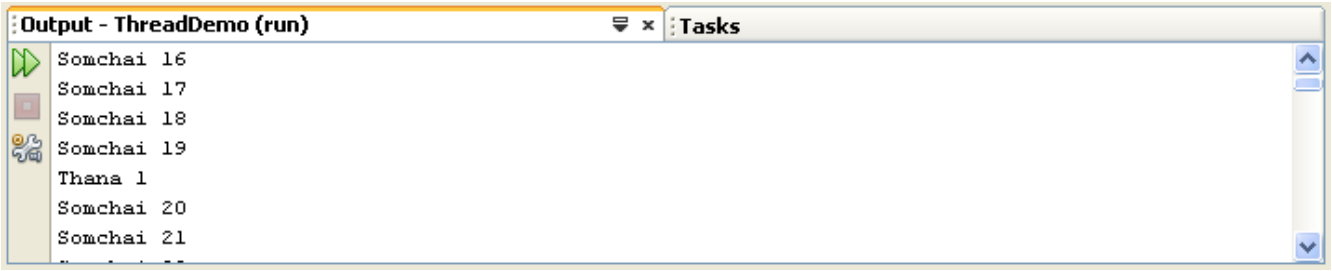


รูปที่ 11.10 อธิบายการทำงานของเมธอด `yield()`

ตัวอย่างโปรแกรมที่ 11.5 แสดงการเรียกใช้เมธอด `yield()` กับอ็อบเจกต์แบบเธรดที่กำลังรันอยู่ จะพบว่าทำให้อ็อบเจกต์แบบเธรดทั้งสองตัวมีโอกาสสลับกันทำงานได้ดีขึ้น ดังแสดงตัวอย่างของผลลัพธ์ที่ได้จากการรันโปรแกรมในรูปที่ 11.11

<u>โปรแกรมที่ 11.5</u> <u>ตัวอย่างแสดงการเรียกใช้เมธอด `yield()`</u>

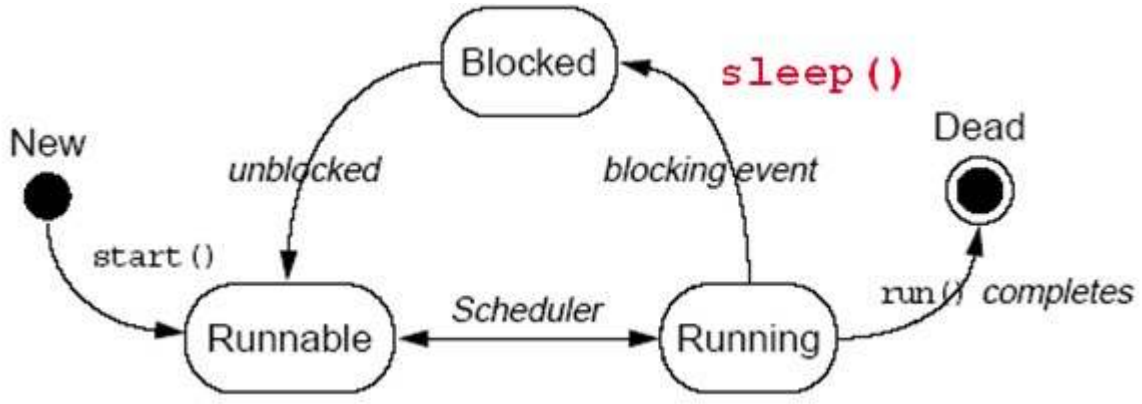
```
class PrintName implements Runnable {
    private String name;
    public PrintName(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 1000; i++) {
            System.out.println(name + " " + i);
            Thread.yield();
        }
    }
}
public class PrintNameThread {
    public static void main(String args[]) {
        PrintName p1 = new PrintName("Thana");
        PrintName p2 = new PrintName("Somchai");
        Thread t1 = new Thread(p1);
        Thread t2 = new Thread(p2);
        t1.start();
        t2.start();
        t2.setPriority(10);
    }
}
```



รูปที่ 11.11 แสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.5

- `static void sleep(long mills)`

  เป็นเมธอดที่กำหนดให้อ็อบเจกต์แบบเธรดที่กำลังรันอยู่ หยุดการทำงานชั่วคราวเป็นเวลา `mills` มิลลิวินาที โดยจะเข้าสู่สถานะ Blocked เพื่อเปิดโอกาสให้เธรดอื่นๆ ที่รออยู่ในสถานะ Runnable ได้มีโอกาสรัน ดังอธิบายการทำงานได้ในรูปที่ 11.12

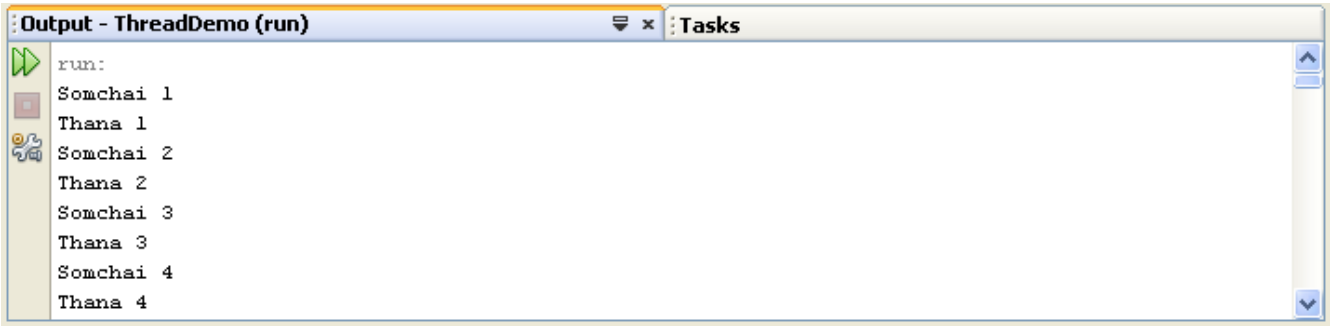


รูปที่ 11.12 อธิบายการทำงานของเมธอด `sleep()`

ตัวอย่างโปรแกรมที่ 11.6 แสดงการเรียกใช้เมธอด `sleep()` กับอ็อบเจกต์แบบเธรดที่กำลังรันอยู่ พบว่าทำให้อ็อบเจกต์แบบเธรดทั้งสองตัวสลับกันทำงานทุกๆ 10 มิลลิวินาที ดังแสดงตัวอย่างผลลัพธ์ที่ได้จากการรันในรูปที่ 11.13

โปรแกรมที่ 11.6 ตัวอย่างแสดงการเรียกใช้เมธอด `sleep()`

```
class PrintName implements Runnable {
    private String name;
    public PrintName(String name) {
        this.name = name;
    }
    public void run() {
        for (int i = 1; i <= 1000; i++) {
            System.out.println(name + " " + i);
            try {
                Thread.sleep(10);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
    }
}
public class PrintNameThread {
    public static void main(String args[]) {
        PrintName p1 = new PrintName("Thana");
        PrintName p2 = new PrintName("Somchai");
        Thread t1 = new Thread(p1);
        Thread t2 = new Thread(p2);
        t1.start();
        t2.start();
        t2.setPriority(10);
    }
}
```



รูปที่ 11.13 แสดงตัวอย่างผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.6

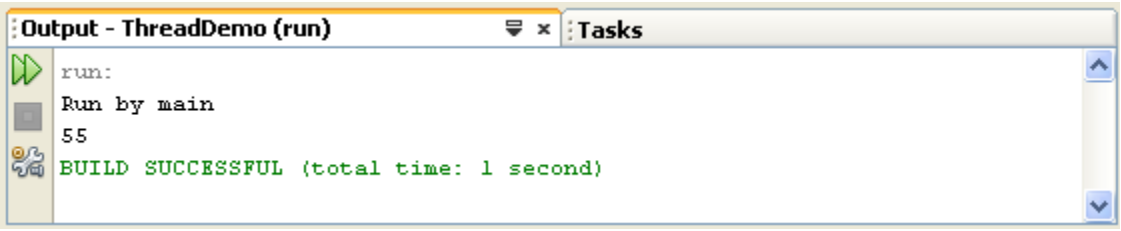
- `void join()`

  เป็นเมธอดที่หยุดอ็อบเจกต์แบบเธรดที่กำลังรันอยู่ (อยู่ในสถานะ Running) แล้วให้อ็อบเจกต์แบบเธรดที่มีการเรียกใช้เมธอด `join()` ทำงานในเมธอด `run()` จนจบ (เข้าสู่สถานะ Dead)

ตัวอย่างโปรแกรมที่ 11.7 แสดงการเรียกใช้เมธอด `join()` เพื่อหยุดอ็อบเจกต์แบบเธรดที่ชื่อ main ที่กำลังรันอยู่ แล้วให้อ็อบเจกต์แบบเธรดที่ชื่อ `t` ทำงานในเมธอด `run()` จนจบ ซึ่งก็คือการบวกเลขตั้งแต่ 1 จนถึง 10 แล้วได้ค่า `sum` เป็น 55 แล้วจึงกลับมาทำงานของอ็อบเจกต์แบบเธรดที่ชื่อ main ต่อโดยการพิมพ์ค่าของ `sum` ออกมาทางจอภาพ ดังแสดงผลลัพธ์ที่ได้จากการรันโปรแกรมในรูปที่ 11.14 และอธิบายการทำงานของโปรแกรมได้ในรูปที่ 11.15

<u>โปรแกรมที่ 11.7</u> <u>ตัวอย่างแสดงการเรียกใช้เมธอด `join()`</u>

```
class Cal implements Runnable {
    int sum;
    public void run() {
        for (int i=1; i<=10; i++) {
            sum += i;
        }
    }
}
public class CalThread {
    public static void main(String args[]) {
        Cal c = new Cal();
        Thread t = new Thread(c);
        t.start();
        System.out.println("Run by " + Thread.currentThread().getName());
        try {
            t.join();
        } catch (InterruptedException ex) {
            ex.printStackTrace();
        }
        System.out.println(c.sum);
    }
}
```



รูปที่ 11.14 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.7

```
                                Cal c = new Cal();
                                Thread t = new Thread(c);
main is running ———————         t.start();
t is running
t is running                    // main และ t สลับกันทำงาน
main is running
main is running
main is running
t is running
main is running ———————          t.join ();// main หยุดการทำงานชั่วคราว
t is running                              // และรอจนกว่า t จะทำงานเสร็จ
t is running
t is running
t is running
t is running
t is running
t is running ———————            // t ทำงานเสร็จแล้ว
main is running ———————         // main กลับมาทำงานอีกครั้ง
main is running
```

รูปที่ 11.15 อธิบายการทำงานของโปรแกรมที่ 11.7

11.5 Synchronization

กรณีที่อ็อบเจกต์แบบเธรดใช้ข้อมูลร่วมกันอาจเกิดปัญหาที่เรียกว่า race condition ขึ้นได้ ซึ่งเป็นกรณีที่อ็อบเจกต์แบบเธรดต่างแย่งกันจัดการข้อมูลทำให้ได้ข้อมูลที่ผิดพลาดได้ โปรแกรมที่ 11.8 เป็นตัวอย่างกรณีของโปรแกรมจำลองระบบการใส่และดึงข้อมูลออกจากสแต๊ก (Stack) ซึ่งประกอบไปด้วยคลาสทั้งหมด 4 คลาสคือ คลาส `MyStack`, `Producer`, `Consumer` และ `MyStackThread` โดยที่คลาส `MyStack` จะเป็นคลาสที่มีเมธอด `push()` สำหรับใส่ข้อมูลเข้าไปในสแต๊ก และเมธอด `pop()` สำหรับดึงข้อมูลออกจากสแต๊ก โดยคลาส `Producer` ซึ่งเป็นคลาสแบบเธรด จะทำหน้าที่ในการใส่ตัวอักขระตั้งแต่ตัว 'A' จนถึงตัว 'Z' (วนไปเรื่อยๆ จนกว่าจะครบ 1000 ครั้ง) เข้าไปในสแต๊กโดยเรียกใช้เมธอด `push()` ของคลาส `MyStack` และคลาส `Consumer` ซึ่งเป็นคลาสแบบเธรด จะทำหน้าที่ในการดึงตัวอักขระตั้งแต่ตัว 'A' จนถึงตัว 'Z' ออกจากสแต๊ก โดยเรียกใช้เมธอด `pop()` ของคลาส `MyStack`

จากการรันโปรแกรมนี้ พบว่ามีโอกาสที่ Consumer และ Producer จะทำงานไม่สัมพันธ์กันเกิดขึ้นได้ เช่น กรณีที่ Consumer ดึงตัวอักขระออกมาจากสแต๊กมากเกินกว่า Producer ใส่ตัวอักขระเข้าไปในสแต๊ก ทำให้มีโอกาสเกิด ArrayIndexOutOfBoundsException ขึ้นได้ ดังแสดงตัวอย่างผลลัพธ์ในรูปที่ 11.16

<u>โปรแกรมที่ 11.8</u> <u>ตัวอย่างแสดงการจำลองระบบการใส่และดึงข้อมูลออกจากสแต๊ก (Stack)</u>

```
import java.util.ArrayList;

class MyStack {
    ArrayList list = new ArrayList();
    int index;
    public void push(char ch) {
        list.add(ch);
        index++;
    }
    public char pop() {
        return (Character) list.remove(--index);
    }
}
class Producer implements Runnable {
    MyStack s;
    char ch = 'A';
    Producer(MyStack s) {
        this.s = s;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
            System.out.println("PUSH: "+ch++);
            if (ch > 'Z') ch = 'A';
            s.push(ch);
        }
    }
}
class Consumer implements Runnable {
    MyStack s;
    Consumer(MyStack s) {
        this.s = s;
    }
    public void run() {
        for (int i = 0; i < 1000; i++) {
            System.out.println("POP: "+s.pop());
        }
    }
}
```

```
public class MyStackThread {
    public static void main(String[] args) {
        MyStack s = new MyStack();
        Producer p = new Producer(s);
        Thread t1 = new Thread(p);
        t1.start();
        Consumer c = new Consumer(s);
        Thread t2 = new Thread(c);
        t2.start();
    }
}
```

```
Output - ThreadDemo (run)                                   Tasks
POP: T
POP: S
POP: R
Exception in thread "Thread-1" java.lang.ArrayIndexOutOfBoundsException: -1
        at java.util.ArrayList.remove(ArrayList.java:390)
        at MyStack.pop(MyStackThread.java:11)
        at Consumer.run(MyStackThread.java:35)
        at java.lang.Thread.run(Thread.java:619)
POP: Q
POP: P
POP: O
```

รูปที่ 11.16 ผลลัพธ์ที่ได้จากการรันโปรแกรมที่ 11.8

เราสามารถที่จะใช้คีย์เวิร์ด `synchronized` เพื่อล็อคชุดคำสั่งที่อ็อบเจกต์แบบเธรดต้องทำทั้งหมดไว้ได้ เพื่อแก้ปัญหา race condition ซึ่งการใช้คีย์เวิร์ด `synchronized` เพื่อล็อคคำสั่งทำได้สองรูปแบบคือ

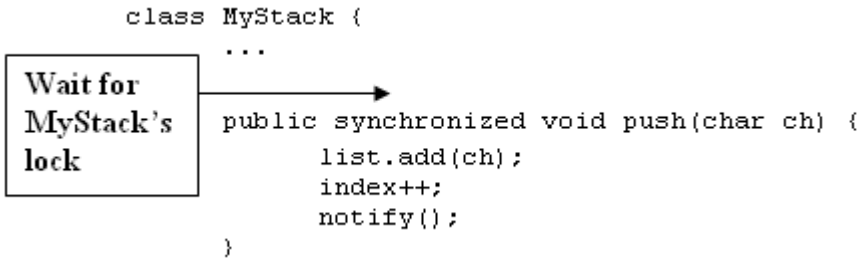
1. กำหนดคีย์เวิร์ด `synchronized` ที่เมธอด เรียกว่าเมธอดแบบ `synchronized` ตัวอย่างเช่น

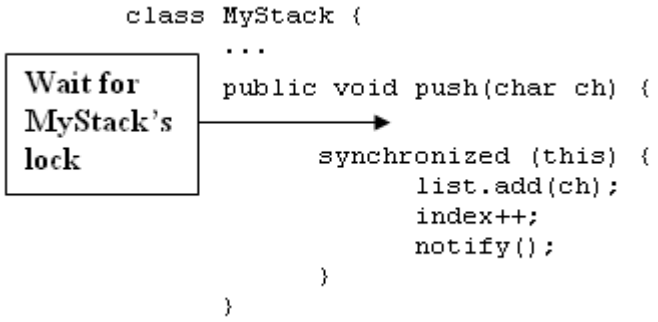
```
synchronized void push(char ch) {
        ...
}
```

2. กำหนดบล็อคโดยใช้คีย์เวิร์ด `synchronized` เรียกว่าบล็อค `synchronized` ตัวอย่างเช่น

```
void push(char ch) {
   synchronized(this) {
       ...
   }
}
```

เนื่องจากอ็อบเจกต์ของคลาสใดๆ ในจาวาจะมีล็อคได้เพียงแค่หนึ่งล็อคเท่านั้น และอ็อบเจกต์แบบเธรดที่จะสามารถทำงานคำสั่งที่อยู่ในเมธอดแบบ `synchronized` หรือบล็อค `synchronized` ได้นั้น จะต้องได้รับล็อคของอ็อบเจกต์ของคลาสที่มีคีย์เวิร์ด `synchronized` อยู่ ซึ่งแสดงได้ดังตัวอย่างต่อไปนี้



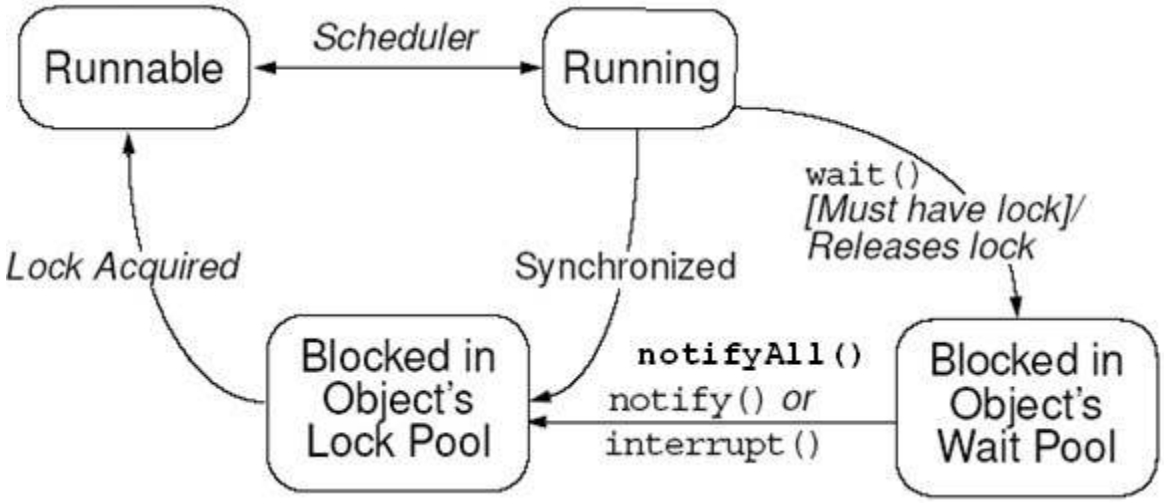


ทำให้อ็อบเจกต์แบบเธรดตัวอื่นๆ ที่ไม่ได้ล็อค ต้องรอให้อ็อบเจกต์แบบเธรดที่ทำงานอยู่ ทำงานในบล็อกคำสั่งที่ถูกล็อคไว้ให้เสร็จก่อน แล้วจึงเข้ามาทำงานต่อได้ ซึ่งอ็อบเจกต์แบบเธรดจะปล่อยล็อคก็ต่อเมื่อสิ้นสุดการทำงานของทุกคำสั่ง หรือเมื่อได้รับข้อผิดพลาดจากอ็อบเจกต์แบบ `Exception` เท่านั้น

11.5.1 เมธอด wait() และ notify()

เมธอด `wait()` และ `notify()` หรือ `notifyAll()` เป็นเมธอดที่ใช้กับเมธอดหรือบล็อกคำสั่งที่เป็นแบบ `synchronized` โดยเมธอด `wait()` จะมีผลทำให้อ็อบเจกต์แบบเธรดที่อยู่ในสถานะ Running เข้าสู่สถานะ Blocked in Object's Wait Pool เพื่อรอให้อ็อบเจกต์แบบเธรดตัวอื่นๆ เรียกใช้เมธอด `notify()` หรือ `notifyAll()` เพื่อย้ายเข้าสู่สถานะ Blocked in Object's Lock Pool ต่อไป ซึ่งความแตกต่างระหว่างเมธอด `notify()` และเมธอด `notifyAll()` คือเมธอด `notify()` จะทำการสุ่ม `(random)` อ็อบเจกต์แบบเธรดเพียงแค่หนึ่งตัวออกมาจาก

สถานะ Blocked in Object's Wait Pool แต่สำหรับเมธอด `notifyAll()` จะทำให้ อ็อบเจกต์แบบเธรดทุกตัวออกมาจากสถานะ Blocked in Object's Wait Pool ดังอธิบายได้ในรูปที่ 11.17



รูปที่ 11.17 อธิบายการทำงานของคำสั่ง `synchronized` เมธอด `wait()`, `notify()` และ `notifyAll()`

โปรแกรมที่ 11.9 แสดงตัวอย่างของการปรับปรุงโปรแกรมที่ 11.8 โดยใช้คีย์เวิร์ด `synchronized` กับเมธอด `pop()` และเมธอด `push()` โดยเมธอด `pop()` จะมีการเรียกใช้เมธอด `wait()` เพื่อให้รอจนกว่าจะมีตัวอักขระอยู่ในสแต๊ก แล้วจึงจะสามารถดึงตัวอักขระออกมาจากสแต๊กได้ และในเมธอด `push()` จะมีการเรียกใช้เมธอด `notify()` เพื่อระบุให้อ็อบเจกต์แบบเธรดที่ได้รับล็อคแล้วเจอเมธอด `wait()` สามารถทำงานต่อได้ ซึ่งจากการรันโปรแกรมนี้ จะได้ผลลัพธ์คือไม่เกิดการผิดพลาดระหว่างการใส่และดึงตัวอักขระออกจากสแต๊ก

โปรแกรมที่ 11.9 ตัวอย่างแสดงการคีย์เวิร์ด `synchronized`

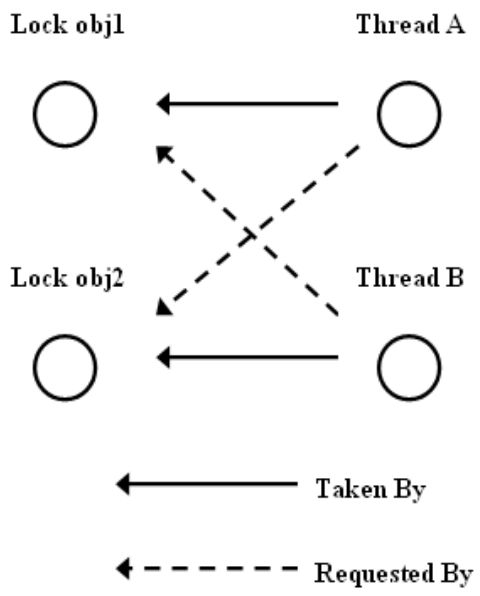
```
import java.util.ArrayList;

class MyStack {
    ArrayList list = new ArrayList();
    int index;
    synchronized void push(char ch) {
        list.add(ch);
        index++;
        notify();
    }
    char pop() {
        while (list.size() == 0) {
            try {
                synchronized (this) {
                    wait();
                }
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
        return (Character) list.remove(--index);
    }
}
```

11.5.2 Deadlock

dead lock คือไม่มีการทำงานใดๆ เกิดขึ้น มักเกิดจากกรณีที่อ็อบเจกต์แต่ละตัวรอการปลดล็อคจากกันและกัน ตัวอย่างเช่น อ็อบเจกต์แบบเธรด `A` ได้ล็อคของอ็อบเจกต์ `obj1` และกำลังรอเพื่อที่จะได้รับล็อคของอ็อบเจกต์ `obj2` ส่วนอ็อบเจกต์แบบเธรด `B` ได้ล็อคของอ็อบเจกต์ `obj2` และกำลังรอเพื่อที่จะได้รับล็อคของอ็อบเจกต์ `obj1` ซึ่งกรณีนี้เป็นทำให้เกิดสถานการณ์ที่เรียกว่า deadlock ดังแสดงตัวอย่างในโปรแกรมที่ 11.10 และอธิบายการทำงานในรูปที่ 11.18 ดังนั้นการป้องกันการเกิด deadlock สามารถทำได้โดยการจัดลำดับการล็อคอ็อบเจกต์แบบเธรดให้เหมาะสม

โปรแกรมที่ 11.10 ตัวอย่างแสดงส่วนของโปรแกรมที่มีโอกาสเกิด deadlock

```
public Object obj1 = new Object();
public Object obj2 = new Object();

...

public void read() {
  synchronized (obj1) {
    synchronized (obj2) {
      doSomething();
    }
  }
}
public void write() {
  synchronized (obj2) {
    synchronized (obj1) {
      doSomethingElse();
    }
  }
}
```



รูปที่ 11.18 อธิบายการทำงานของโปรแกรมที่ 11.10

สรุปเนื้อหาของบท

- เธรดจะแตกต่างจาก process ที่ทำงานภายใต้ระบบปฏิบัติการแบบ Multitasking ตรงที่ process แต่ละ process จะมีความเป็นอิสระต่อกัน แต่เธรดแต่ละเธรดอาจใช้ข้อมูลร่วมกันซึ่งเปรียบเสมือนซีพียูจำลอง (Virtual CPU)
- เธรดประกอบไปด้วยซีพียู ซอร์ดโค้ดและอ็อบเจกต์
- คลาสแบบเธรดในภาษาจาวาคือคลาสที่สืบทอดมาจากคลาสที่ชื่อ `Thread` หรือคลาสที่ `implements` อินเตอร์เฟสที่ชื่อ `Runnable`
- ภายในคลาสแบบเธรดจะต้องมีเมธอด `run()` ที่ไม่มีการรับ argument ใดๆเข้ามา
- สถานะของเธรดอาจจะเป็น New, Runnable, Running, Blocked, Blocked in Object's Lock Pool, Blocked in Object's Wait Pool หรือ Dead โดยการเข้าสู่สถานะ Running จะขึ้นอยู่กับตัวจัดตารางเวลาเธรด
- เมธอดที่สำคัญของคลาส `Thread` มีเมธอด `start()`, `run()`, `isAlive()`, `setPriority()`, `getPriority()`, `yield()`, `sleep()` และ `join()`
- เราสามารถที่จะใช้คีย์เวิร์ดที่ชื่อ `synchronized` เพื่อล็อคชุดคำสั่งที่อ็อบเจกต์แบบเธรดต้องทำพร้อมกันไว้ได้
- เมธอด `wait()` และ `notify()` หรือ `notifyAll()` เป็นเมธอดที่ใช้กับเมธอดหรือบล็อกคำสั่งที่เป็นแบบ `synchronized`
- Deadlock หมายถึงการที่อ็อบเจกต์แต่ละตัวรอการปลดล็อกจากกันและกัน ทำให้ไม่มีการทำงานใดๆ เกิดขึ้น