บทที่ ๙ แพคเกจ *java.lang*

แพคเกจ java.lang บรรจุคลาสต่าง ๆ ที่เป็นแกนในการดำเนินการของภาษาจาวา คลาสที่บรรจุในแพคเกจนี้เป็นคลาสพื้นฐานที่ใช้บ่อย ดังนั้นเพื่ออำนวยความสะดวกในการเขียนโปรแกรม คอมไพเลอร์จะนำเข้า (import) คลาสทุกคลาสในแพคเกจนี้ไปไว้ในโปรแกรมต้นฉบับทุกแฟ้มโดยอัตโนมัติในตอนที่คอมไพเลอร์ภาษาจาวากำลังทำงานโดยที่ผู้เขียนไม่จำเป็นต้องเขียนข้อความสั่ง import แพคเกจนี้ไว้ในโปรแกรมเหมือนอย่างแพคเกจอื่น

ในบทนี้จะกล่าวถึงเฉพาะบางคลาสที่มีความสำคัญและมีโอกาสใช้ค่อนข้างมากในแพคเกจ java.lang ซึ่งในแต่ละคลาสมักมีรายละเอียดมาก ซึ่งไม่สามารถหยิบยกมาอธิบายได้ทั้งหมด จึงขออธิบายให้รู้จักเพียงบางเมท็อดของบางคลาสเท่านั้น สำหรับรายละเอียดเพิ่มเติมสามารถศึกษาได้จากคู่มือเอพีไอที่ให้มากับชุดพัฒนาซอฟท์แวร์จาวา

คลาส Object

คลาส Object เป็นคลาสบนสูงสุดของทุกคลาสในภาษาจาวา ในการประกาศคลาส ถ้าคลาสใดไม่ได้ระบุชื่อคลาสอื่นไว้หลังคำสำคัญ extends คอมไพเลอร์จะสร้างคลาสนั้นโดยถือว่าต่อเติมมาจากคลาส Object นี้โดยตรงเสมอ คลาสอื่นทุกคลาสสามารถรับทอดเมท็อดต่าง ๆ ทุกเมท็อดของคลาส Object ไปใช้

ต่อไปนี้จะกล่าวถึงเมท็อดบางเมท็อดของคลาส Object

เมท็อดแรกที่จะกล่าวถึงคือเมท็อด equals ซึ่งประกาศไว้ดังนี้

```
public boolean equals(Object obj)
```

เมท็อดนี้ใช้สำหรับเปรียบเทียบ"ในทางลึก"ว่าอ็อบเจกต์ ๒ ตัวมีค่าเท่ากันหรือไม่ ต่างจากการใช้ตัวดำเนินการ == ในการเปรียบเทียบซึ่งเปรียบเทียบค่าของตัวแปรอย่างง่ายเพียงผิวเผินเท่านั้น การใช้ตัวดำเนินการ == เปรียบเทียบนั้นไม่ได้เปรียบเทียบสิ่งที่อยู่ในอ็อบเจกต์ว่ามีค่าเท่ากันหรือไม่ ยกตัวอย่างเช่น มีตัวแปรอ้างอิง 2 ตัว d1 และ d2 ต่างก็อ้างอิงไปยังอ็อบเจกต์ของคลาส java.util.Date การเปรียบเทียบโดยใช้วิธีการต่อไปนี้คือ

```
if (d1 == d2)
```

เป็นการเปรียบเทียบค่าของตัวแปรอ้างอิง d1 และ d2 ว่ามีค่าเท่ากันหรือไม่ ไม่ได้เปรียบเทียบว่า อ็อบเจกต์ที่อ้างถึงโดย d1 และ อ็อบเจกต์ที่อ้างถึงโดย d2 มีค่าเท่ากันหรือเปล่า ผลลัพธ์จากการเปรียบเทียบนี้จะมีค่าเป็นจริงก็ต่อเมื่อ d1 และ d2 ต่างก็อ้างถึงอ็อบเจกต์เดียวกัน หรือไม่ก็ d1 และ d2 ต่างก็มีค่าเป็น null ซึ่งหมายความว่าไม่ได้อ้างอิงไปยังอ็อบเจกต์ใดเลย

บางครั้งเราอยากเปรียบเทียบโดยที่ไม่สนใจว่า d1 หรือ d2 อ้างถึงอ็อบเจกต์เดียวกันหรือไม่ หรือบางครั้งเรารู้ว่าตัวแปรอ้างอิงเหล่านี้อ้างอิงไปยังอ็อบเจกต์คนละตัวกัน แต่สิ่งที่เราสนใจอยากจะเปรียบเทียบก็คืออ็อบเจกต์ ๒ ตัวนี้มีค่าภายในเท่ากันหรือไม่ ในกรณีเช่นนี้ เราจะใช้ตัวดำเนินการ == ในการเปรียบเทียบไม่ได้ เราต้องใช้วิธีการมองลึกลงไปในอ็อบเจกต์ทั้งสองว่ามีตัวแปรอะไรอยู่ในอ็อบเจกต์บ้าง ตัวแปรในอ็อบเจกต์เหล่านั้นมีค่าเท่ากันหรือไม่ เพื่ออำนวยความสะดวกในการเปรียบเทียบแบบนี้ ผู้ออกแบบและพัฒนาภาษาจาวาจึงได้เขียนคลาส Object ขึ้นมาให้มีเมท็อด equals() สำหรับทำหน้าที่นี้ ดังนั้น ถ้าเราต้องการเปรียบเทียบอ็อบเจกต์ที่อ้างถึงโดย d1 และอ้างถึงโดย d2 ว่าเท่ากันหรือไม่จึงทำได้โดยการเรียกใช้เมท็อด equals() ดังนี้

```
if ( d1.equals(d2) )
```

ความจริงแล้วเมท็อด equals() ที่เขียนไว้ในคลาส Object ไม่ค่อยมีประโยชน์อะไรมากนัก มันไม่สามารถตรวจสอบได้ว่าอ็อบเจกต์ ๒ ตัวมีค่าเท่ากันจริงหรือไม่ กรณีเดียวที่มันยืนยันได้ว่าอ็อบเจกต์มีค่าเท่ากันก็คือ เมื่ออ็อบเจกต์ที่กำลังเปรียบเทียบอยู่นั้นเป็นตัวเดียวกัน สิ่งที่เมท็อด equals() ของคลาส Object ทำก็คือเพียงแค่เปรียบเทียบว่า ตัวแปรอ้างอิงทั้ง ๒ มีค่าเท่ากันหรือไม่ ถ้าเท่ากันจะคืนค่าเป็น true ถ้าไม่เท่าจะคืนค่าเป็น false

เหตุที่เมท็อด equals() ของคลาส Object ไม่กล้ายืนยันว่าอ็อบเจกต์สองตัวเท่ากันหรือไม่ก็เพราะว่าจะเปรียบเทียบค่าของตัวแปรต่าง ๆที่อยู่ในอ็อบเจกต์ทั้ง ๒ อย่างตรงไปตรงมาไม่ได้ ในบางครั้งการที่ตัวแปรในอ็อบเจกต์มีค่าไม่เท่ากันมิได้หมายความว่าอ็อบเจกต์จะไม่เท่ากัน หรือการที่ค่าของตัวแปรทุกตัวในอ็อบเจกต์ที่กำลังเปรียบเทียบมีค่าเท่ากันก็มิได้หมายความว่าอ็อบเจกต์ทั้งสองนั้นจะเท่ากัน เราต้องพิจารณาให้ลึกลงไปกว่านั้น เพราะว่าในอ็อบเจกต์อาจมีตัวแปรอ้างอิงอยู่ภายใน เราจะเปรียบเทียบเฉพาะค่าของตัวแปรในอ็อบเจกต์เพียงอย่างเดียวไม่ได้ จะต้องตามไปเปรียบเทียบข้อมูลหรืออ็อบเจกต์อื่นที่ตัวแปรในอ็อบเจกต์อ้างอิงไปถึงด้วย ซึ่งการที่จะเปรียบเทียบในทางลึกอย่างนี้จะใช้ตัวดำเนินการ == เพื่อเปรียบเทียบค่าของตัวแปรอย่างเดียวไม่เพียงพอ ต้องเขียนข้อความสั่งบางอย่างขึ้นเป็นการเฉพาะเป็นกรณีกรณีไป นักเขียนโปรแกรมที่เขียนคลาสใหม่ขึ้นมา ถ้าต้องการให้สามารถเปรียบเทียบอ็อบเจกต์ของคลาสที่เขาเขียนขึ้นมาได้ ควรเขียนเมท็อด equals() ขึ้นมาเองด้วย ซึ่งเท่ากับว่าทำการยกเลิก(override) ไม่ใช้เมท็อด equals() ที่อยู่ในคลาส Object ไปโดยปริยาย ส่วนจะเปรียบเทียบอย่างไรนั้นเป็นหน้าที่ของนักเขียนโปรแกรมที่จะต้องเขียนขึ้นเองเพราะว่าไม่มีผู้อื่นใดที่จะล่วงรู้ได้ว่าจะเปรียบเทียบอ็อบเจกต์ได้อย่างไรนอกจากผู้ที่เขียนคลาสนั้นขึ้นมา เมท็อด equals() จะต้องคืนค่าเป็น true ถ้าอ็อบเจกต์มีค่าเท่ากัน และคืนค่าเป็น false ถ้าไม่เท่ากัน

เมท็อดที่สองของคลาส Object ที่จะกล่าวถึงคือเมท็อด toString() ซึ่งมีการประกาศไว้ดังนี้

```
public String toString()
```

วัตถุประสงค์ของเมท็อด toString() ก็เพื่อใช้สำหรับแปลงค่าสถานะของอ็อบเจกต์ในรูปของสายอักขระเพื่อนำไปใช้ประโยชน์ในการแสดงผลหรือหาข้อบกพร่องของโปรแกรม (debugging)

ทำนองเดียวกันกับเมท็อด `equals()` ของคลาส `Object` เมท็อด `toString()` ที่รับทอดมาจากคลาส `Object` ไม่ค่อยมีประโยชน์นักเช่นกัน เมท็อดที่รับทอดมานี้ทำงานเพียงแค่ให้ชื่อคลาสและรหัสแฮช (hash code) ของอ็อบเจกต์ในรูปแบบของเลขฐานสิบหกคั่นด้วยอักขระ `@` เท่านั้น คลาสต่าง ๆ ของเอพีไอส่วนใหญ่จะมีเมท็อด `toString()` อยู่ด้วยซึ่งเป็นการยกเลิกไม่ใช้เมท็อด `toString()` ของคลาส `Object` ทั้งนี้ก็เพื่อให้สารสนเทศอื่นที่เป็นประโยชน์มากกว่านั่นเอง

หมายเหตุ

รหัสแฮช (hash code) เป็นค่าตัวเลขที่สามารถใช้ระบุถึงอ็อบเจกต์แต่ละตัว อ็อบเจกต์คนละตัว (ถึงแม้จะสร้างมาจากคลาสเดียวกันก็ตาม) จะมีรหัสแฮชที่ต่างกัน รหัสแฮชนี้นิยมใช้เป็นคีย์ในการค้นหาอ็อบเจกต์ สำหรับคลาส Object จะใช้เลขที่อยู่ของอ็อบเจกต์เป็นรหัสแฮช

ในนิพจน์สายอักขระ (string expression) การต่อสายอักขระด้วยอ็อบเจกต์นั้น จาวาเรียกใช้เมท็อด `toString()` ของอ็อบเจกต์เพื่อแปลงข้อมูลในอ็อบเจกต์เป็นสายอักขระก่อนแล้วจึงนำสายอักขระที่ได้นั้นไปต่อท้ายสายอักขระที่อยู่ด้านซ้ายของตัวดำเนินการบวก `(+)` ตัวอย่างเช่น ถ้า `d` เป็นตัวแปรอ้างอิงที่อ้างอิงไปยังอ็อบเจกต์ของคลาส `java.util.Date` ที่สร้างขึ้นมา การเขียนนิพจน์โดยใช้ตัวดำเนินการ*บวก*ต่อไปนี้

```
"today is " + d
```

จะได้ผลลัพธ์เหมือนกับ

```
"today is " + d.toString()
```

ซึ่งได้ผลลัพธ์เป็นสายอักขระ ๑ ตัว ขึ้นต้นด้วยข้อความ `"today is "` แล้วตามด้วยข้อความแสดงวันที่และเวลาอันเป็นผลมาจากการเรียกเมท็อด `toString()` ของอ็อบเจกต์ `d` การเรียกเมท็อด `toString()` นี้ผู้เขียนโปรแกรมไม่จำเป็นต้องเขียนเพื่อเรียกเมท็อดนี้เอง เนื่องจากที่ใดก็ตามที่มีการใช้ต้องการใช้ตัวแปรอ้างอิงในฐานะของสายอักขระ คอมไพเลอร์จะเปลี่ยนตัวแปรอ้างอิงนั้นเป็นการเรียกเมท็อด `toString()` ของอ็อบเจกต์ที่อ้างถึงโดยตัวแปรอ้างอิงนั้นแทน ซึ่งช่วยให้เขียนโปรแกรมได้สั้นและง่ายขึ้น

คลาส *Math*

คลาส `Math` ของจาวามีเมท็อดและค่าคงที่ซึ่งช่วยในการคำนวณทางคณิตศาสตร์ คลาสนี้ถูกกำหนดให้เป็น final จึงไม่สามารถสร้างคลาสลางของคลาสนี้ได้ ตัวสร้าง (constructor) เป็น `private` ดังนั้นเราจึงไม่สามารถสร้างอ็อบเจกต์ของคลาสนี้ได้เอง แต่เมท็อดและตัวแปรต่าง ๆ กำหนดให้เป็น `static` เราจึงสามารถเข้าถึงเมท็อดและตัวแปรเหล่านี้ได้โดยใช้ชื่อคลาสไม่จำเป็นต้องสร้างอ็อบเจกต์แต่อย่างใด

คลาส Math มีค่าคงที่ให้ใช้ ๒ ตัวคือ

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

ทั้งคู่ต่างก็ได้รับการประกาศให้เป็น `public static final` และมีประเภทเป็น `double`

เมท็อดของคลาส Math ครอบคลุมการทำงานด้านต่าง ๆทางคณิตศาสตร์อย่างกว้างขวาง เช่น การคำนวณทางตรีโกณมิติ ลอการิธึม และการปัดตัวเลขให้เป็นค่ากลม ๆ (rounding) ต่อไปนี้คือเมท็อดของคลาส Math ที่ควรรู้จัก

`int abs(int i)`	คืนค่าสัมบูรณ์ของ `i`
`long abs(long l)`	คืนค่าสัมบูรณ์ของ `l`
`float abs(float f)`	คืนค่าสัมบูรณ์ของ `f`
`double abs(double d)`	คืนค่าสัมบูรณ์ของ `d`
`int max(int i1, int i2)`	คืนค่าสูงสุดของ `i1` และ `i2`
`long max(long l1, long l2)`	คืนค่าสูงสุดของ `l1` และ `l2`
`float max(float f1, float f2)`	คืนค่าสูงสุดของ `f1` และ `f2`
`double max(double d1, double d2)`	คืนค่าสูงสุดของ `d1` และ `d2`
`int min(int i1, int i2)`	คืนค่าต่ำสุดของ `i1` และ `i2`
`long min(long l1, long l2)`	คืนค่าต่ำสุดของ `l1` และ `l2`
`float min(float f1, float f2)`	คืนค่าต่ำสุดของ `f1` และ `f2`
`double min(double d1, double d2)`	คืนค่าต่ำสุดของ `d1` และ `d2`
`double random()`	คืนค่าเป็นตัวเลขแบบสุ่มระหว่าง 0.0 ถึง 1.0
`int round(float f)`	คืนค่าเป็นเลขจำนวนเต็มที่ใกล้เคียงกับ `f` ที่สุด
`long round(double d)`	คืนค่าเป็นเลขจำนวนเต็มที่ใกล้เคียงกับ `d` ที่สุด
`double sin(double d)`	คืนค่าเป็น sin ของ `d`
`double cos(double d)`	คืนค่าเป็น cos ของ `d`
`double tan(double d)`	คืนค่าเป็น tan ของ `d`
`double sqrt(double d)`	คืนค่าเป็นรากสองของ `d`

คลาสสำหรับห่อข้อมูลพื้นฐาน

ในบางครั้งเราต้องการที่จะแปลงข้อมูลพื้นฐานเช่น `int` ไปเป็นอ็อบเจกต์ ข้อมูลพื้นฐานแต่ละประเภทของจาวาจะมี*คลาสตัวห่อ (wrapper class)* ควบคู่กันไปด้วย เช่น คลาส `Integer` ใช้สำหรับห่อหุ้มข้อมูลพื้นฐาน `int` เป็นต้น คลาสตัวห่อจะมีชื่อที่บอกได้ชัดเจนว่าห่อหุ้มข้อมูลพื้นฐานชนิดใดไว้ คลาสตัวห่อเป็นคลาสที่ห่อหุ้มค่าพื้นฐานหนึ่งค่า ค่าที่ห่อหุ้มไว้นั้นไม่สามารถเปลี่ยนแปลงได้ และตัวคลาสเองก็เป็นคลาสสุดท้าย (final) จึงไม่สามารถมีคลาสล่างหรือถูก override ได้ ตัวอย่างเช่น คลาส `Integer` ห่อค่า `int` ไว้หนึ่งตัว ส่วนคลาส `Float` ห่อค่า `float` ไว้หนึ่งค่าเช่นกัน

เหตุผลหลักที่มีการสร้างคลาสตัวห่อให้ใช้ก็คือเพื่อสนับสนุนแนวคิดการ*สร้างโปรแกรมเพื่อใช้งานทั่วไป (generic programming)* คลาสบางอย่างเช่นคลาสประเภท*บรรจุภัณฑ์ (container class)* เป็นคลาสที่สามารถ

บรรจุอ็อบเจกต์ใด ๆ ก็ได้ไว้ในตัวอ็อบเจกต์ของคลาสเหล่านี้ แต่มันไม่สามารถบรรจุข้อมูลพื้นฐานใด ๆ ได้ เว้นเสียแต่ว่าจะแปลงข้อมูลพื้นฐานเหล่านั้นให้เป็นอ๊อเจ็กท์เสียก่อน

นอกจากเหตุผลเรื่องการเขียนโปรแกรมเพื่อใช้งานทั่วไปแล้ว ยังมีเหตุผลอื่นที่เราจะใช้คลาสตัวห่อ เช่นใช้คลาสตัวห่อในการแปลงประเภทของข้อมูล ผู้ออกแบบภาษาจาวาได้บรรจุเมท็อดจำนวนหนึ่งสำหรับแปลงประเภทของข้อมูล เช่นแปลงสายอักขระเป็นตัวเลข ไว้ในคลาสตัวห่อเพื่อความสะดวกในการแปลงประเภทข้อมูลด้วย

ตารางต่อไปนี้แสดงชื่อคลาสตัวห่อที่เข้าคู่กับค่าพื้นฐาน

ประเภทของข้อมูลพื้นฐาน	คลาสตัวห่อ
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

อ็อบเจกต์ของคลาสตัวห่อทุกคลาสสามารถสร้างโดยการส่งผ่านค่าที่ต้องการห่อไปให้ตัวสร้างที่เหมาะสม ตัวอย่างบางส่วนของโปรแกรมต่อไปนี้แสดงวิธีการสร้างอ็อบเจกต์ตัวอย่างของตัวห่อแต่ละประเภท

```
boolean    primitiveBoolean = true;
Boolean    wrappedBoolean   = new Boolean(primitiveBoolean);

byte       primitiveByte    = 50;
Byte       wrappedByte      = new Byte(primitiveByte);

char       primitiveChar    = 'T';
Character  wrappedChar      = new Character(primitiveChar);

short      primitiveShort   = 3333;
Short      wrappedShort     = new Short(primitiveShort);

int        primitiveInt     = 12345678;
Integer    wrappedInt       = new Integer(primitiveInt);

long       primitiveLong    = 1234567890;
```

```
Long        wrappedLong      = new Long(primitiveLong);

float       primitiveFloat   = 1.1f;
Float       wrappedFloat     = new Float(primitiveFloat);

double      primitiveDouble  = 2.22222222;
Double      wrappedDouble    = new Double(primitiveDouble);
```

นอกจากคลาสตัวห่อจะมีตัวสร้างที่รับพารามิเตอร์เป็นข้อมูลพื้นฐานประเภทเดียวกับชื่อคลาสแล้ว คลาสเหล่านี้ยังมีตัวสร้างที่สามารถรับพารามิเตอร์เป็นสายอักขระอีกด้วย จะมีข้อยกเว้นก็แต่เพียงคลาส `Character` เท่านั้นที่ไม่มีตัวสร้างที่รับพารามิเตอร์เป็นสายอักขระ สายอักขระที่ส่งเป็นพารามิเตอร์ไปให้ตัวสร้างควรมีค่าของข้อมูลที่ต้องการห่อให้สัมพันธ์กับคลาสแต่ละประเภทแต่เขียนอยู่ในรูปของสายอักขระแทนค่าพื้นฐาน ตัวสร้างจะทำการแปลงสายอักขระให้เป็นข้อมูลพื้นฐานตามประเภทของคลาสตัวห่อ ถ้าไม่สามารถแปลงได้จะถือว่ามีสิ่งผิดปรกติชนิด `NumberFormatException` เกิดขึ้น ตัวสร้างของคลาสตัวห่อเกือบทุกตัวสามารถโยนสิ่งผิดปรกติชนิดนี้ได้ เพราะว่ามีความเป็นไปได้ที่ในสายอักขระจะมีค่าที่ไม่ถูกต้อง มีเพียงคลาส `Boolean` เท่านั้นที่ไม่โยนสิ่งผิดปรกติ ตัวสร้างของคลาส `Boolean` รับพารามิเตอร์เป็นสายอักขระที่มีค่าเป็นอะไรก็ได้ ถ้าเป็น "`true`" ไม่ว่าจะเป็นตัวอักษรตัวใหญ่หรือตัวเล็กก็ตามจะถือว่าได้ค่าเป็น `true` ถ้าสายอักขระมีค่าอื่นนอกเหนือจากนี้ คลาส `Boolean` จะถือว่าได้ค่า `false`

นอกจากคลาสตัวห่อที่ใช้ห่อหุ้มข้อมูลพื้นฐานที่กล่าวมาแล้ว ยังมีคลาส `BigInteger` และ `BigDecimal` สำหรับห่อหุ้มข้อมูลขนาดใหญ่ที่ข้อมูลพื้นฐานไม่สามารถรองรับได้อีกด้วย คลาส `BigInteger` ใช้สำหรับเก็บเลขจำนวนเต็มซึ่งอาจมีขนาดใหญ่กว่าชนิด `int` ส่วนคลาส `BigDecimal` ใช้กับค่าตัวเลขแบบมีจุดทศนิยมที่สามารถรองรับค่าได้มากกว่าชนิด `double` คลาสทั้ง 2 นี้มีวิธีการจัดเก็บข้อมูลภายในอ็อบเจกต์ที่ใหญ่โดยไม่จำกัดขนาด มันจะเตรียมเนื้อที่หน่วยความจำในอ็อบเจกต์ให้เพียงพอกับค่าของข้อมูลที่จะจัดเก็บ เมื่อจัดเก็บข้อมูลในอ็อบเจกต์แล้วจะเปลี่ยนค่าไม่ได้ (immutable)

ส่วนของโปรแกรมต่อไปนี้แสดงให้เห็นถึงวิธีการสร้างอ็อบเจกต์ของคลาสตัวห่อ โดยใช้ตัวสร้างซึ่งรับพารามิเตอร์เป็นสายอักขระ

```
Boolean    wrappedBoolean   = new Boolean("True");
  try {
    Byte       wrappedByte   = new Byte("50");
    Short      wrappedShort  = new Short("3333");
    Integer    wrappedInt    = new Integer("12345678");
    Long       wrappedLong   = new Long("1234567890");
    Float      wrappedFloat  = new Float("1.1f");
    Double     wrappedDouble = new Double("2.22222222");
    BigInteger wBigInteger   = new BigInteger("12345678901234567890");
```

```
    BigDecimal wBigDecimal   = new BigDecimal("12345678901234.567");
  }
  catch (NumberFormatException e) {
    System.out.println("Invalid Number Format");
  }
```

หลังจากที่สร้างอ็อบเจกต์แล้ว เราสามารถเรียกข้อมูลที่อยู่ในอ็อบเจกต์กลับมาในรูปของข้อมูลพื้นฐานได้ สำหรับอ็อบเจกต์ของคลาส `Boolean` เราสามารถเรียกข้อมูลคืนมาเป็น `boolean` โดยการเรียกเมท็อด `booleanValue()` สำหรับอ็อบเจกต์ของคลาส `Character` เราสามารถเรียกข้อมูลคืนมาเป็น `char` โดยการเรียกเมท็อด `charValue()`

การเรียกค่าตัวเลขพื้นฐานจากตัวห่อ

สำหรับค่าตัวเลขที่ได้รับการห่อไว้ในอ็อบเจกต์ประเภทตัวห่อแล้ว เราสามารถดึงค่าตัวเลขของมันกลับออกมาใช้ในรูปของข้อมูลตัวเลขประเภทอื่นใด ๆ ก็ได้ ไม่ว่าจะเป็นชนิด `byte short int long float` หรือ `double` ก็ตาม เมท็อดที่ใช้ในการดึงข้อมูลกลับออกมาเป็นประเภทต่าง ๆ มีดังนี้

- `public byte byteValue()`
- `public short shortValue()`
- `public int intValue()`
- `public long longValue()`
- `public float floatValue()`
- `public double doubleValue()`

ทั้ง 6 เมท็อดข้างต้นนี้กำหนดไว้ในคลาสนามธรรม (abstract class) `Number` ซึ่งเป็นคลาสบนของคลาส `Byte Short Integer Long Float Double BigInteger` และ `BigDecimal` มีประโยชน์สำหรับการแปลงประเภทของข้อมูลตัวเลขจากชนิดหนึ่งไปเป็นอีกชนิดหนึ่ง การแปลงประเภทนี้อาจมีการปัดหรือตัดเศษทิ้งหรือทำให้ข้อมูลสูญค่าไปบ้างก็เป็นได้เช่นเดียวกับการลดระดับ (demotion) ถ้าชนิดของต้นทางใหญ่กว่าชนิดของปลายทางที่ต้องการแปลงค่าไปหา

การแปลงสายอักขระเป็นข้อมูลประเภทตัวเลข

ในกรณีของการแปลงข้อมูลที่อยู่ในรูปของสายอักขระให้เป็นข้อมูลตัวเลขนั้น เราสามารถใช้คลาสตัวห่อให้เป็นประโยชน์ได้ เช่นมีข้อมูลที่เป็นสายอักขระเป็น "123456" ข้อมูลนี้ถึงแม้จะมีตัวเลขอยู่ในสายอักขระแต่นำไปใช้ในฐานะตัวเลขไม่ได้ ถ้าต้องการนำไปเก็บหรือคำนวณในฐานะของเลขจำนวนเต็มประเภท `int` จำเป็นต้องแปลงจากสายอักขระให้เป็น `int` เสียก่อน โดยการส่งสายอักขระนี้ให้เป็นพารามิเตอร์ในการสร้างอ็อบเจกต์ของคลาส `Integer` ต่อจากนั้นใช้เมท็อด `intValue()` เรียกข้อมูลคืนมาในรูปของ `int` ดังตัวอย่าง

```
String s = "123456";
Integer  oi = new Integer(s);
```

```
int i = oi.intValue();
```

เราสามารถแปลงข้อมูลตัวเลขที่ฝังอยู่ในสายอักขระให้เป็นตัวเลขประเภทต่าง ๆ ได้โดยเลือกใช้ตัวห่อที่เหมาะสม

การแปลงข้อมูลที่อยู่ในรูปของสายอักขระให้เป็นประเภทตัวเลขนั้น นอกจากใช้วิธีสร้างอ็อบเจกต์ของตัวห่อขึ้นมาก่อนแล้วใช้เมท็อดข้างต้นในการคืนค่าเป็นประเภทข้อมูลพื้นฐานที่ต้องการแล้ว เรายังสามารถแปลงข้อมูลโดยที่ไม่ต้องสร้างอ็อบเจกต์ของคลาสตัวห่อก็ได้ ในคลาสตัวห่อมีเมท็อดของคลาส (static method) ที่สามารถนำมาใช้ในการแปลงสายอักขระให้เป็นข้อมูลพื้นฐานตัวเลขชนิดต่าง ๆได้ ดังนี้

คลาส	**นิยามของเมท็อด**
Byte	`public static byte` **`parseByte`**`(String s)`
Sort	`public static short` **`parseShort`**`(String s)`
Integer	`public static int` **`parseInt`**`(String s)`
Long	`public static long` **`parseLong`**`(String s)`
Float	`public static float` **`parseFloat`** `(String s)`
Double	`public static double` **`parseDouble`**`(String s)`

เราสามารถนำเมท็อดเหล่านี้มาแปลงสายอักขระให้เป็นข้อมูลประเภทตัวเลขต่าง ๆได้ทันทีโดยไม่ต้องสร้างอ็อบเจกต์ดังตัวอย่าง

```
String s = "123456";
int i = Integer.parseInt(s);
```

ตัวห่อ Character

คลาสตัวห่อ `Character` ซึ่งใช้ห่อหุ้มตัวอักขระนั้น มีเมท็อดให้ใช้เป็นจำนวนมาก ในที่นี่จะกล่าวถึงบางเมท็อดที่มีโอกาสใช้บ่อย

`char` **`charValue`**`()`

คืนค่าตัวอักขระที่ถูกห่อหุ้มอยู่ในอ็อบเจกต์นี้

`static boolean` **`isLetter`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นตัวอักษรใช่หรือไม่

`static boolean` **`isDigit`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นตัวเลขใช่หรือไม่

`static boolean` **`isLetterOrDigit`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นตัวอักษรหรือตัวเลขใช่หรือไม่

`static boolean` **`isLowerCase`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นอักษรโรมันตัวเล็กใช่หรือไม่

`static boolean` **`isUpperCase`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นอักษรโรมันใช่หรือไม่

`static boolean` **`isSpaceChar`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นตัว space ใช่หรือไม่

`static boolean` **`isWhiteSpace`**`(char ch)`

ตรวจสอบอักขระที่ส่งเข้ามาให้พารามิเตอร์ว่าเป็นตัวพื้นที่สีขาวใช่หรือไม่

`static char` **`toLowerCase`**`(char ch)`

จะตรวจสอบว่าอักขระที่รับเข้ามาทางพามามิเตอร์เป็นอักษรโรมันตัวใหญ่ใช่หรือไม่ ถ้าใช่จะคืนค่าโดยแปลงให้เป็นอักษรโรมันตัวเล็ก ถ้าไม่ใช่จะคืนค่าเป็นอักระตัวเดิมที่รับเข้ามา

`static char` **`toUpperCase`**`(char ch)`

จะตรวจสอบว่าอักขระที่รับเข้ามาทางพามามิเตอร์เป็นอักษรโรมันตัวเล็กใช่หรือไม่ ถ้าใช่จะคืนค่าโดยแปลงให้เป็นอักษรโรมันตัวใหญ่ ถ้าไม่ใช่จะคืนค่าเป็นอักระตัวเดิมที่รับเข้ามา

เมท็อดที่ขึ้นต้นด้วย `is` เป็นเมท็อดที่ตรวจสอบแล้วคืนค่าเป็น `boolean` ถ้าผลการตรวจสอบเป็นจริงจะคืนค่าเป็น `true` ถ้าเป็นเท็จจะคืนค่าเป็น `false`

สรุป

ข้อเท็จจริงที่สำคัญเกี่ยวกับคลาสผู้ห่อต่าง ๆ สรุปได้ดังนี้

- คลาสตัวห่อต่าง ๆ มีประโยชน์ในกรณีที่เราต้องการความสะดวกในการดำเนินการต่อข้อมูลพื้นฐานราวกับว่ามันเป็นอ็อบเจกต์ เพราะว่ามีบางเมท็อดของบางคลาสรับแต่พารามิเตอร์ที่เป็นอ็อบเจกต์เท่านั้น ไม่รับพารามิเตอร์ที่เป็นข้อมูลพื้นฐาน เราจึงต้องเอาข้อมูลพื้นฐานทำให้เป็นอ็อบเจกต์ โดยใช้คลาสตัวห่อที่เหมาะสมกับมัน นอกจากนี้คลาสตัวห่อยังมีเมท็อดต่าง ๆ ที่เป็นประโยชน์ในการเปลี่ยนชนิดของข้อมูล จากชนิดหนึ่งไปเป็นอีกชนิดหนึ่งที่แปลงกันได้
- ข้อมูลพื้นฐานทุกประเภทมีคลาสตัวห่อที่ตรงกับประเภทของมัน
- ตัวห่อทุกตัวสามารถสร้างมาจากข้อมูลพื้นฐาน และ สายอักขระ(string) ยกเว้นคลาส `Character` เพียงคลาสเดียวที่ไม่สามารถสร้างจากสายอักขระได้
- ค่าที่ได้รับการห่อไว้แล้วสามารถทดสอบว่าเท่ากันหรือไม่ได้โดยใช้เมท็อด `equals()`

ค่าที่ได้รับการห่อสามารถดึงกลับออกมาใช้เป็นข้อมูลพื้นฐานประเภทต่าง ๆ ได้ด้วยเมท็อด `XXXvalue()`