ตัวอย่างการใช้ โครงสร้างข้อมูลเชิงเส้นเบื้องต้น

ใน collection มักประกอบด้วยส่วนย่อย(element)จำนวนมากซึ่งบ่อยครั้งที่เราจำเป็นต้องไล่หาส่วนย่อยใน collection เหล่านี้ จาวามีตัวช่วยในการไล่หาอ็อบเจกต์ที่อยู่ใน collection เรียกว่า Iterator (iterate มีรากศัพท์มาจากภาษาลาตินมีความหมายตรงกับภาษาอังกฤษคำว่า again ซึ่งหมายถึงทำอีก iterate จึงหมายถึงการทำซ้ำหลายครั้ง หรือการวนทำซ้ำในลูปนั่นเอง) Iterator เป็นอ็อบเจกต์ที่ใช้ในการเดินทางหรือ"เคลื่อนผ่าน" (traverse) อ็อบเจกต์ต่าง ๆ ที่อยู่ใน collection โครงสร้างข้อมูลแต่ละชนิดจะมีวิธีการ"เคลื่อนผ่าน" ส่วนย่อยแตกต่างกัน Java Collection Framework จึงได้เตรียม Iterator ไว้อำนวยความสะดวกในการ "เคลื่อนผ่าน" ส่วนย่อยของโครงสร้างข้อมูลแต่ละชนิด การที่จะใช้ Iterator ได้เราจำเป็นต้องถามหาอ็อบเจกต์ Iterator ให้เจอก่อนซึ่งโครงสร้างข้อมูลแต่ละชนิดจะมีเมท็อดให้ถามหา Iterator ที่เขาจัดเตรียมไว้ให้ใช้กับโครงสร้างข้อมูลชนิดนั้น ๆ

<u>หมายเหตุ</u> ตัวอย่างที่ให้ ชื่อคลาส(แสดงด้วยตัวหนา) จะตรงกับชื่อไฟล์ของ source code

ตัวอย่างที่ ๑ แสดงการใช้ LinkedList และ Iterator

```
import java.util.*;

class IteratorDemo {
      public static void main(String args[]) {
            LinkedList <String> l = new LinkedList <String> ();
            l.add("Bangkok");
            l.add("Nontaburi");
            l.add("Pathumthani");
            l.add("Samuthprakarn");

            Iterator <String> i = l.iterator();
            while (i.hasNext()) {
                  String s = i.next();
                  System.out.println(s + " length="+ s.length());
                  // remove elements having length > 10
                  if (s.length() > 10)
                        i.remove();
            }
            System.out.println(l);
      }
}
```

ตัวอย่างนี้แสดงการใช้ `LinkedList` อย่างง่าย โดยใส่สตริงซึ่งเป็นอ็อบเจกต์ลงใน `LinkedList` ด้วยเมท็อด `add()` หลังจากนั้นจะวนลูปพิมพ์ค่าและความยาวของสตริงแต่ละตัวออกมา ขณะที่วนลูปจะตรวจไปด้วยว่าความยาวของสตริงมากกว่า ๑๐ หรือไม่ ถ้าใช่จะลบออกจาก `LinkedList` ด้วยเมท็อด `remove()` เมื่อวนลูปเสร็จสิ้นแล้วออกจากลูปมาพิมพ์ค่าของ `LinkedList` ซึ่งจะแสดงค่าของสตริงที่ยังคงเหลืออยู่ใน `LinkedList` ไว้ในวงเล็บ [ ] ในการวนลูปนั้น เราใช้ Iterator เป็นตัวช่วยในการเคลื่อนผ่านไปยังส่วนย่อยต่าง ๆ จะมีการถามหา `Iterator` ด้วยเมท็อด `iterator()` ซึ่งจะคืนค่ามาเป็นค่าอ้างอิงถึงอ็อบเจกต์ `Iterator` ที่ต้องการ เมื่อได้ Iterator แล้วจึงใช้เมท็อด `next()` ในการเคลื่อนผ่านไปยังส่วน

ย่อยที่ละตัว แต่ก่อนเคลื่อนผ่านเข้าไปควรตรวจสอบเสียก่อนว่ายังมีส่วนย่อยใน collection ให้เคลื่อนผ่านไปหรือไม่ด้วยการถามจากเมท็อด `hasNext()` ว่ามีส่วนย่อยตัวถัดไปให้เคลื่อนผ่านเข้าไปหรือไม่

ผลลัพธ์จากการรันโปรแกรมเป็นดังนี้

```
Bangkok length=7
Nontaburi length=9
Pathumthani length=11
Samuthprakarn length=13
[Bangkok, Nontaburi]
```

ตัวอย่างที่ ๒ การใช้ LinkedList และ ListIterator อย่างง่าย

```
import java.util.LinkedList;
import java.util.ListIterator;

class LinkedListDemo {
      public static void main(String [] args) {
            LinkedList <String> fruit =  new LinkedList<String>();
            fruit.add("Lemon");
            fruit.add("Coconut");
            fruit.add("Mango");
            fruit.add("Banana");
            fruit.add("Apple");
            fruit.add("Chery");

            ListIterator <String> i = fruit.listIterator();
            while (i.hasNext() ) {
                  String s = i.next();
                  System.out.println(s);
                  if (s.equals("Apple"))
                        i.remove(); // remove "Apple" from the list
            }
            // Start iterate again
            System.out.println("Start over after remove an element");
             i = fruit.listIterator();
            while (i.hasNext() )
                  System.out.println(i.next());
      }
}
```

ตัวอย่างนี้ แสดงการสร้าง `LinkedList` แล้วเพิ่มส่วนย่อยลงใน `LinkedList` หลังจากนั้นจะทำการ iterate เพื่อหาสตริง Apple เพื่อลบออกจากรายการ ก่อนลบจะแสดงรายการทั้งหมดออกมาก่อน เมื่อลบออกแล้วจะวนลูปเพื่อแสดงรายการที่เหลืออีกครั้ง ตัวอย่างนี้คล้ายตัวอย่างที่ ๑ ต่างกันที่ตัวอย่างนี้ใช้ `ListIterator` ซึ่งมีเมท็อดที่จำเพาะเจาะจงที่จะใช้กับ `LinkedList` มากกว่า `Iterator` แต่ตัวอย่างนี้ไม่ได้แสดงการใช้เมท็อดที่แตกต่างกันแต่อย่างใด เพียงแต่เจตนาให้เห็นว่าใช้แทนกันได้เท่านั้น ถ้ารันโปรแกรมนี้แล้วจะได้ผลลัพธ์ดังนี้

```
Lemon
Coconut
Mango
Banana
Apple
Chery
```

```
      Start over after remove an element
      Lemon
      Coconut
      Mango
      Banana
      Chery
```

ตัวอย่างที่ ๓

```
import java.util.*;

class LinkedListDemo1 {
      public static void main(String args[]) {
            LinkedList <String> l = new LinkedList <String> ();
            l.add("Bangkok");
            l.add("Nontaburi");
            l.add("Pathumthani");
            l.add("Samuthprakarn");
            System.out.println(l);
            System.out.println("First Element: " + l.getFirst());
            System.out.println("Last Element: " + l.getLast());
            l.add(2,"Nakornphathom"); // Insert the third element
            l.addFirst("List of Provinces");
            l.addLast("End of List");
            System.out.println(l);
            l.remove(4);
            System.out.println("Size of list: " + l.size());
            System.out.println(l);
            l.removeFirst();
            l.removeLast();
            l.set(3,"Samuthsakhon"); // replace element
            System.out.println(l);
      }
}
```

ตัวอย่างที่ ๓ แสดงการใช้ `LinkedList` **แสดงให้เห็นการใช้เมท็อดเกี่ยวกับ** `LinkedList` **เพิ่มเติมจากตัวอย่างก่อนหน้านี้ เมท็อดที่แสดงให้เห็นเพิ่มเติมคือ** `getFirst()` **และ** `getLast()` **สำหรับเข้าถึงส่วนย่อยตัวแรกและตัวสุดท้ายของ** `LinkedList` **ตามลำดับ**

`addFirst()` **และ** `addLast()` **สำหรับเพิ่มส่วนย่อยลงในตำแหน่งแรกและสุดท้าย ตามลำดับ**

`removeFirst()` **และ** `RemoveLast()` **สำหรับลบส่วนย่อยตัวแรก และตัวสุดท้ายออกจาก** `LinkedList` **ตามลำดับ**

`size()` **สำหรับถามขนาดปัจจุบันของ** collection

`set()` **สำหรับนำส่วนย่อยตัวใหม่เข้าไปแทนที่ตัวเดิมในตำแหน่งที่ต้องการ**

ผลลัพธ์จากการรันโปรแกรมเป็นดังนี้

```
[Bangkok, Nontaburi, Pathumthani, Samuthprakarn]
First Element: Bangkok
Last Element: Samuthprakarn
[List of Provinces, Bangkok, Nontaburi, Nakornphathom, Pathumthani, Samuthprakarn, End of
List]
Size of list: 6
[List of Provinces, Bangkok, Nontaburi, Nakornphathom, Samuthprakarn, End of List]
[Bangkok, Nontaburi, Nakornphathom, Samuthsakhon]
```

ตัวอย่างที่ ๔ คลาส SaleItem ซึ่งจะนำไปใช้ในตัวอย่างที่ ๕

```
import java.io.Serializable;
public class SaleItem implements Serializable {
	protected int productId;
	protected String description;
	protected int qty;
	protected double price;
	protected double cost;

	SaleItem() { }  // default constructor
	SaleItem(int pid, String desc, int q, double p, double c) {
		productId = pid;
		description = desc;
		qty = q;
		price = p;
		cost = c;
	}

	public void setProductId (int pid) { productId = pid; }
	public void setDescription(String desc) { description = desc;}
	public void setQty (int q) {qty = q;}
	public void setPrice (double p) { price = p; }
	public void setCost (double c) { cost = c; }

	public int getProductId() { return productId; }
	public String getDescription() { return description; }
	public int getQty() { return qty; }
	public double getPrice() { return price; }
	public double getCost() { return cost; }

	public String toString() {
		return "\nProduct " + productId + " " + description + " qty=" + qty +
				" price=" + price + " cost=" + cost;
	}
}
```

คลาส `SaleItem` **ทำการอิมพลิเม้นท์อินเทอร์เฟส** `Serializable` **เพราะต้องการให้อ็อบเจกต์ของคลาสนี้สามารถส่งข้อมูลในอ็อบเจกต์ออกไปทาง** output stream **ได้เมื่อมีการสั่ง** `writeObject()` **อินเทอร์เฟสนี้เป็นอินเทอร์เฟสพิเศษผู้เขียนโปรแกรมแค่บอกว่าต้องการอิมพลิเม้นท์อินเทอร์เฟสนี้เท่านั้นไม่ต้องเขียนเมท็อดใด ๆ ของอินเทอร์เฟสนี้เลย ที่เหลือคอมไพเลอร์จะจัดการให้**

ตามตัวอย่างนี้มี constructor 2 **ตัว ตัวแรกเป็น** default constructor **คือไม่มีพารามิเตอร์ กับ** constructor **ตัวที่** 2 **ซึ่งจะรับพารามิเตอร์จำนวน** 5 **ตัวคือ รหัสสินค้า ชื่อสินค้า จำนวน ราคาขาย และต้นทุน ตามลำดับ เพื่อนำไปใส่ไว้ในตัวแปรสมาชิกของอ็อบเจกต์ เนื่องจากต้องการปกป้องข้อมูลไม่ให้คลาสอื่นที่ไม่ใช่คลาสมันเองและซับคลาสเข้าถึงข้อมูลของตัวแปรสมาชิก จึงได้ประกาศให้ตัวแปรสมาชิกทุกตัวเป็น** protected **เมื่อประกาศเช่นนี้แล้ว คลาสอื่นใดนอกจากคลาสมันเองและซับคลาสของมันจะไม่สามารถเข้าถึงตัวแปรที่** protected **ได้ จึงได้เตรียมเมท็อดสำหรับเข้าถึงข้อมูลของตัวแปรสมาชิกเอาไว้ให้ โดยมีเมท็อดที่ขึ้นต้นด้วย** get **สำหรับสอบถามค่าข้อมูล และมีเมท็อดที่ชื่อขึ้นต้นด้วย** set **สำหรับกำหนดค่าใหม่ให้กับตัวแปรสมาชิก นอกจากนี้ยังมีเมท็อด** `toString()` **ทำหน้าที่ในการเตียมข้อมูลในอ็อบเจกต์ให้อยู่ในรูปของสตริงเพื่อให้ผู้นำคลาสนี้ไปใช้สามารถดูค่าข้อมูลในอ็อบเจกต์ได้สะดวก**

ตัวอย่างที่ ๕ นำคลาส SaleItem **มาสร้างอ็อบเจกต์แล้วใส่ลงใน** LinkedList

```
import java.util.*;

class LinkedListDemo2 {
      public static void main(String args[]) {
            LinkedList <SaleItem> l = new LinkedList <SaleItem> ();

            // demonstration of using setter to set internal values to an object
            SaleItem si = new SaleItem();
            si.setProductId(212);
            si.setDescription("Desktop Computer Toyoba DP111");
            si.setQty(5);
            si.setPrice(35000);
            si.setCost(32000);
            l.add(si);

            // demonstration of using constructor to initilze objects
            l.add(new SaleItem(101, "Netbook P333", 10, 12000, 10000));
            l.add(new SaleItem(102, "LCD Monitor Sunny S150", 34, 7800, 6100));
            l.add(new SaleItem(131, "Printer Canue P131", 18, 18000, 17000));
            l.add(new SaleItem(101, "DVD writer Samsun DW444", 5, 1300, 1220));

            System.out.println("Number of original items in the collection: " +
l.size());

            // System.out.println(l);

            Iterator <SaleItem> i = l.iterator();
            while (i.hasNext()) {
                  SaleItem s = i.next();
                  // remove SaleItem having quantity  > 10
                  if (s.getQty() > 10) {
                        i.remove();
                        System.out.println(s.getDescription() + " removed");
                        }
                  else
                        // change price up if profit less than 10%
                        if ((s.getPrice() - s.getCost()) / s.getCost() < 0.1) {
                              System.out.printf("change price of %s from %.2f to ",
                                          s.getDescription(),     s.getPrice());
                              s.setPrice( s.getCost() * 1.1);
                              System.out.printf("%.2f\n", s.getPrice());
                        }
            }
            System.out.println("\nList of items in collection after change" + l);
      }
}
```

ตัวอย่างนี้แสดงการนำอ็อบเจกต์ใส่ลงใน `LinkedList` **คล้ายกับตัวอย่างที่ให้ดูไปแล้ว แต่แทนที่จะใส่สตริงซึ่งเป็นอ็อบเจกต์อย่างง่าย คราวนี้ใส่อ็อบเจกต์จากคลาสที่เราเขียนขึ้นเอง คือคลาส** `SaleItem`

ตัวอย่างนี้จะสร้างอ็อบเจกต์ของคลาส `SaleItem` **ขึ้นมาหลายตัวแล้วใส่ลงใน** `LinkedList` **อ็อบเจกต์แรกที่สร้างขึ้นใช้** default Constructor **ซึ่งไม่มีการส่งพารามิเตอร์ไปให้แต่อย่างใด จึงใช้เมท็อด** `set` **ต่าง ๆ เพื่อกำหนดค่าให้กับตัวแปรสมาชิกของอ็อบเจกต์ ก่อนที่จะใส่อ็อบเจกต์ลงใน** `LinkedList` **ส่วนการสร้างอ็อบเจกต์ตัวอื่น ๆ ในลำดับถัดไปจะสร้างอ็อบเจกต์ด้วย** constructor **ที่รับพารามิเตอร์เป็นข้อมูลต่าง ๆ ตามที่ต้องการให้นำไปใส่ไว้ในอ็อบเจกต์**

หลังจากใส่อ็อบเจกต์ลงใน `LinkedList` เรียบร้อยแล้ว จะแสดงการไล่อ็อบเจกต์ใน `LinkedList` ถ้าพบอ็อบเจกต์ที่มีจำนวนมากกว่า 10 จะลบอ็อบเจกต์นั้นออกจาก `LinkedList` แต่ถ้าจำนวนไม่มากกว่า 10 จะตรวจว่าราคาขายมีกำไรน้อยกว่า 10% หรือไม่ ถ้าใช่จะเปลี่ยนราคาขายเป็นราคาใหม่ให้สูงกว่าต้นทุน 10%

ผลลัพธ์จากการรันโปรแกรมเป็นดังนี้

```
Number of original items in the collection: 5
change price of Desktop Computer Toyoba DP111 from 35000.00 to 35200.00
LCD Monitor Sunny S150 removed
Printer Canue P131 removed
change price of DVD writer Samsun DW444 from 1300.00 to 1342.00

List of items in collection after change[
Product 212 Desktop Computer Toyoba DP111 qty=5 price=35200.0 cost=32000.0,
Product 101 Netbook P333 qty=10 price=12000.0 cost=10000.0,
Product 101 DVD writer Samsun DW444 qty=5 price=1342.0 cost=1220.0]
```

ตัวอย่างที่ ๖ แสดงการใช้ Queue

```
import java.util.*;

class QueueDemo {
      public static void main(String args[]) {
            Queue <String> q = new LinkedList <String> ();

            q.add("Bangkok");
            q.add("Nontaburi");
            q.add("Pathumthani");
            q.add("Samuthprakarn");
            System.out.println(q);
            System.out.println(q.remove());
            System.out.println(q);
            q.add("Nakorn Sawan");
            System.out.println(q);
            System.out.println(q.peek());
            System.out.println(q);
      }
}
```

ตัวอย่างนี้แสดงการใช้คิว โดยสาธิตให้เห็นการเพิ่มสตริงลงในคิวด้วยเมท็อด `add()` มีการดึงออกจากคิวด้วยเมท็อด `remove()` และมีการ “ดู” หัวคิวโดยไม่ดึงออกจากคิวด้วยเมท็อด `peek()` ก่อนและหลังการทำอะไรกับคิวจะพิมพ์ค่าสตริงที่อยู่ในคิวออกมาดูเพื่อให้เห็นว่าข้อมูลในคิวเปลี่ยนไปอย่างไร

ถ้าสังเกตุจะเห็นว่า คิวสร้างมาจากคลาส `LinkedList` แต่เรากำลังจะใช้ในฐานะที่เป็น `Queue` ซึ่งเป็นอินเทอร์เฟสที่ `LinkedList` ได้ทำการอิมพลิเม้นท์

ผลลัพธ์เป็นดังนี้

```
[Bangkok, Nontaburi, Pathumthani, Samuthprakarn]
Bangkok
[Nontaburi, Pathumthani, Samuthprakarn]
[Nontaburi, Pathumthani, Samuthprakarn, Nakorn Sawan]
Nontaburi
[Nontaburi, Pathumthani, Samuthprakarn, Nakorn Sawan]
```

ตัวอย่างที่ ๗ แสดงการใช้ PriorityQueue

```
import java.util.*;

class PriorityQueueDemo {
      public static void main(String args[]) {
            PriorityQueue <String> q = new PriorityQueue <String> ();

            q.add("Samuthprakarn");
            q.add("Bangkok");
            q.add("Pathumthani");
            q.add("Nontaburi");

            System.out.println(q);
            System.out.println(q.remove());
            System.out.println(q);
            q.add("Nakorn Sawan");
            System.out.println(q);
            System.out.println(q.peek());
            System.out.println(q);
      }
}
```

ตัวอย่างนี้คล้ายกับตัวอย่างที่ ๖ แต่เปลี่ยนจากคิวธรรมดาซึ่งใช้คลาส `LinkedList` เป็นคิวแบบจัดอันดับความสำคัญซึ่งจะเป็นคลาส `PriorityQueue` ซึ่งถ้าเรา `compareTo()` ไม่บอกให้จัดลำดับเป็นอย่างอื่นมันจะจัดลำดับความสำคัญตามค่าของข้อมูลโดยเรียงลำดับจากน้อยไปหามาก ตัวอย่างนี้ปรับแก้จากตัวอย่างที่ ๖ เล็กน้อย คือเปลียนจากคลาส `LinkedList` เป็น `PriorityQueue` และเปลี่ยนลำดับของข้อมูลสตริงที่ใส่ในคิวไม่ให้เรียงลำดับตามตัวอักษร ผลลัพธ์การทำงานของโปรแกรมเป็นดังนี้

```
[Bangkok, Nontaburi, Pathumthani, Samuthprakarn]
Bangkok
[Nontaburi, Samuthprakarn, Pathumthani]
[Nakorn Sawan, Nontaburi, Pathumthani, Samuthprakarn]
Nakorn Sawan
[Nakorn Sawan, Nontaburi, Pathumthani, Samuthprakarn]
```

จะเห็นว่าข้อมูลที่อยู่ในคิวจะเรียงตามลำดับตัวอักษรจากน้อยไปหามาก ไม่ได้เป็นไปตามลำดับที่เราใส่ข้อมูลเข้าไป

การใช้ตัวเปรียบเทียบข้อมูลของเราเอง

ตัวอย่างที่ 7 เราไม่ต้องบอกว่าให้เรียงลำดับข้อมูลในคิวอย่างไร `PriorityQueue` จะเรียกใช้เมท็อด `compareTo()` ของอ็อบเจกต์ที่อยู่ในคิวเปรียบเทียบข้อมูลให้หลังจากนั้น `PriorityQueue` จึงจัดลำดับอ็อบเจกต์เสียใหม่ตามผลลัพธ์ที่เมท็อด `compareTo()` ส่งมาให้ แต่ถ้าเราไม่ต้องการให้เรียงลำดับตามแบบที่เมท็อด เปรียบเทียบข้อมูลให้ละ เราจะเรียงแบบอื่นได้หรือไม่ คำตอบคือได้ แต่ผู้เขียนโปรแกรมต้องบอก `PriorityQueue` ว่าจะให้เรียงลำดับอย่างไร วิธีการคือจะต้องเตรียมอ็อบเจกต์ขึ้นมาให้บริการเปรียบเทียบอ็อบเจกต์ ส่วนจะเปรียบเทียบอย่างไรขึ้นอยู่กับการเขียนคำสั่งใส่ไว้ในเมท็อด `compare()` ของอ็อบเจกต์นี้ อ็อบเจกต์ที่จะทำหน้าที่เปรียบเทียบอ็อบเจกต์ให้ `PriorityQueue` จะต้องมาจากคลาสที่อิมพลิเม้นท์อินเทอร์เฟส `Comparator` ซึ่งบังคับว่าจะต้องมีเมท็อด `compare()` ให้ใช้นั่นเอง

ตัวอย่างที่ ๘ ปรับปรุงมาจากตัวอย่างที่ ๗ ให้คิวเรียงลำดับความสำคัญจากความยาวของสตริงแทนที่จะเป็นค่าของสตริง

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

class PriorityQueueDemo2 {
      public static void main(String args[]) {
            Comparator<String> comparator = new StringLengthComparator();
            PriorityQueue <String> q = new PriorityQueue <String> (10,comparator);

            q.add("Samuthprakarn");
            q.add("Bangkok");
            q.add("Pathumthani");
            q.add("Nontaburi");

            System.out.println(q);
            System.out.println(q.remove());
            System.out.println(q);
            q.add("Nakorn Sawan");
            System.out.println(q);
            System.out.println(q.peek());
            System.out.println(q);
      }
}

class StringLengthComparator implements Comparator<String>
{
    public int compare(String s1, String s2)
    {
        // Assume neither string is null.
        return s1.length() - s2.length();
    }
}
```

ตัวอย่างนี้มีรายละเอียดมากกว่าตัวอย่างที่ ๗ เนื่องจากต้องมีคลาสใหม่ คือ `StringLengthComparator` **เพิ่มขึ้นมาเพื่อทำหน้าที่เปรียบเทียบความยาวของสตริง** `PriorityQueue` **จะส่งอ็อบเจกต์สตริง 2 ตัวมาให้เมท็อด** `compare()` **ทำการเปรียบเทียบ เนื่องจากเราต้องการเปรียยเทียบความยาวของสตริง เมท็อด** `compare()` **จึงคืนค่าเป็นผลต่างของความยาวของสตริงทั้งสองที่ส่งมาให้เปรียบเทียบ**

เนื่องจากเราต้องการใช้ตัวเปรียบเทียบอ็อบเจกต์ของเราเองจึงต้องบอก `PriorityQueue` **ว่าเราต้องการให้อ็อบเจกต์ใดดำเนินการเปรียบเทียบให้ ซึ่งสามารถบอกได้ตอนสร้างอ็อบเจกต์** `PriorityQueue` **โดยส่งชื่ออ็อบเจกต์ที่ต้องการให้เป็นผู้เปรียบเทียบไปให้เป็นพารามเตอร์ตอนสั่ง new ดังนี้**

```
new PriorityQueue <String> (10,comparator)
```

กรณีนี้เราใช้ constructor ซึ่งรับพารามิเตอร์ 2 ตัว ตัวแรกเป็นตัวเลขบอกขนาดของคิวว่าให้เตรียมรองรับส่วนย่อยที่อยู่ในคิวได้กี่ตัว ส่วนพารามิเตอร์ตัวที่ 2 เป็นชื่ออ็อบเจกต์ที่เราเตรียมไว้สำหรับเปรียบเทียบอ็อบเจกต์ซึ่งสร้างรอไว้ก่อนแล้วในบรรทัดก่อนหน้านี้ เราอาจยุบรวม 2 บรรทัดต่อไปนี้

```
Comparator<String> comparator = new StringLengthComparator();
PriorityQueue <String> q = new PriorityQueue <String> (10,comparator);
```

ให้เหลือเพียงบรรทัดเดียวได้ดังนี้

```
PriorityQueue <String> q = new PriorityQueue <String> (10, new
StringLengthComparator());
```

โดยไม่ต้องตั้งตัวแปร comparator ก็ได้

ผลลัพธ์จากการรันโปรแกรมนี้ได้เป็นดังนี้

```
[Bangkok, Nontaburi, Pathumthani, Samuthprakarn]
Bangkok
[Nontaburi, Samuthprakarn, Pathumthani]
[Nontaburi, Nakorn Sawan, Pathumthani, Samuthprakarn]
Nontaburi
[Nontaburi, Nakorn Sawan, Pathumthani, Samuthprakarn]
```

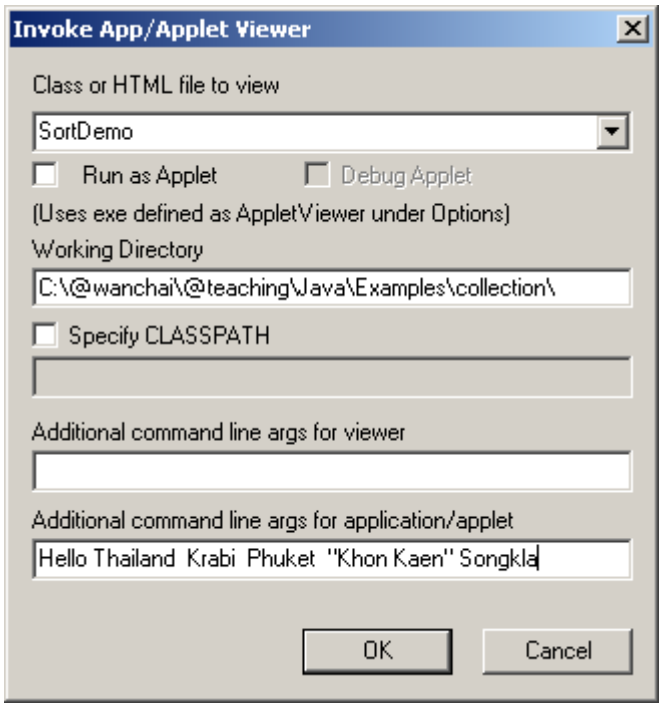
จะเห็นว่าข้อมูลเรียงตามความยาวของสตริง ไม่ใช่ค่าของสตริง

การเตรียมตัวเปรียบเทียบอ็อบเจกต์ให้ `PriorityQueue` นั้น นอกจากใช้ในกรณีที่เราไม่ต้องการให้มันเรียงลำดับตามการเปรียบเทียบของเมท็อด `compareTo()` ซึ่งต้องอิมพลิเม้นท์ อินเทอร์เฟส `Compareble` แล้ว เราอาจใช้วิธีนี้แก้ปัญหากรณีที่เรานำคลาสที่ไม่มีการอิมพลิเม้นท์ `Comparable` มาใส่ใน `PriorityQueue` ได้ด้วย

ตัวอย่างที่ ๙ แสดงการสั่งให้เรียงลำดับข้อมูล (sort) ข้อมูลที่อยู่ใน `ArrayList`

```
import java.util.*;

public class SortDemo {
    public static void main( String args[] ) {
      List <String> l = new  ArrayList <String> ();

      for ( int i = 0; i < args.length; i++ )
          l.add( args[ i ] );

      Collections.sort( l );

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

ตัวอย่างนี้แสดงการนำข้อมูลจาก command line ไปใส่ไว้ใน `ArrayList` หลังจากนั้นสั่งให้เรียงลำดับข้อมูลภายในด้วยเมท็อด sort ของคลาส `Collections` โปรแกรมนี้รับข้อมูลเข้าทาง command line ถ้ารันด้วยโปรแกรม JeditPlus สามารถส่งข้อมูลให้โปรแกรมในช่อง Additional command line args for application/applet ดังตัวอย่างต่อไปนี้



ผลลัพธ์ที่ได้

```
[Hello, Khon Kaen, Krabi, Phuket, Songkla, Thailand]
```

ตัวอย่างที่ ๑๐ การใช้เมท็อด sort ของคลาส Arrays เพื่อเรียงลำดับข้อมูลในแถวลำดับ

```
import java.util.*;

public class ArraysSortDemo {
    public static void main( String args[] ) {
      Arrays.sort( args );

      List l = Arrays.asList( args );

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

ตัวอย่างนี้แสดงการใช้คลาส Array ซึ่งเป็น คลาสอรรถประโยชน์สำหรับทำอะไรบางอย่างกับ array มีเมท็อด sort ช่วยในการเรียงลำดับข้อมูลใน array ตัวอย่างนี้ใช้เมท็อด sort ของคลาส Arrays เข้าไปเรียงลำดับข้อมูลใน array args โดยตรง เมื่อเรียงลำดับเรียบร้อยแล้วจึงสร้าง List จาก array นี้เพื่อความสะดวกในการนำไปแสดงผล

ตัวอย่างข้างต้นเป็นตัวอย่างอย่างง่ายที่หยิบเอาคลาส String มาใช้ ต่อไปจะให้ดูตัวอย่างการใช้คลาสอื่นบ้าง โดยจะยกตัวอย่างเป็นคลาสที่เราเขียนขึ้นเองว่าทำอย่างไรจึงสามารถนำไป sort ได้

อ็อบเจกต์ของคลาสที่สามารถนำไป sort หรือค้นหา ได้นั้นจำเป็นต้องอิมพลิเม้นท์อินเทอร์เฟส `Comparable` ตัวอย่างต่อไปนี้เป็นคลาส Name ที่เราต้องการให้ใช้งานได้เทียบเท่าคลาสมาตรฐานของจาวาทั่วไป เราต้องการให้สามารถ sort หรือค้นได้จึงต้องอิมพลิเม้นท์อินเทอร์เฟส `Comparable` นอกจากนี้ยังใส่เมท็อดที่คลาสทั่วไปมักจะมีคือเมท็อดสำหรับเข้าถึงข้อมูล (getter – ขึ้นต้นชื่อเมท็อดด้วย get แล้วตามด้วยชื่อตัวแปรสมาชิก) กับเมท็อดสำหรับเปลี่ยนค่าข้อมูลในอ็อบเจกต์ (setter – ขึ้นต้นชื่อเมท็อดด้วย set แล้วตามด้วยชื่อตัวแปรสมาชิก) และเมท็อดที่คลาสทั่วไปควรมีดังต่อไปนี้ด้วย

- `equals()` สำหรับเปรียบเทียบอ็อบเจกต์ว่าเท่ากันหรือไม่
- `hashCode()` สำหรับนำข้อมูลในอ็อบเจกต์มาคำนวณให้เป็นตัวเลขชนิด `int` หนึ่งค่า วิธีคำนวณอยู่นอกเหนือขอบเขตวิชานี้ แต่ตามตัวอย่างไม่ได้คำนวณเองแต่ใช้ค่าจาก `hashCode()` ของสตริงมาคำนวณต่อยอด
- `toString()` สำหรับแปลงข้อมูลต่าง ๆ ในอ็อบเจกต์ให้อยู่ในรูปของสตริง 1 ตัวเพื่อความสะดวกในการนำค่าข้อมูลในอ็อบเจกต์ไปแสดงผล

ตัวอย่างที่ ๑๑ แสดงคลาส **Name** ที่เราเขียนขึ้นเองเพื่อให้สามารถนำไป sort ได้

```
import java.util.*;

public class Name implements Comparable {

    private String first;
    private String last;

    public Name( String firstName, String lastName ) {
```

```
      first = firstName;
      last = lastName;
    }

    public String getFirst() {
      return first;
   }

    public String getLast() {
      return last;
   }
public boolean equals( Object o ) {
          boolean retval = false;

           if (o !=null && o instanceof Name ) {
          Name n = ( Name )o;
          retval = n.getFirst().equals( first ) &&
                 n.getLast().equals( last );
         }

      return retval;
  }

  public int hashCode() {
      return first.hashCode() + last.hashCode();
  }

  public String toString() {
      return first + " " + last;
  }

public int compareTo( Object o ) throws ClassCastException {
      int retval;

      Name n = ( Name ) o;

      retval = last.compareTo( n.getLast() );

      if ( retval == 0 )
          retval = first.compareTo( n.getFirst() );

      return retval;
    }

} //Name
```

ตัวอย่างที่ ๑๒ แสดงการเรียงลำดับอ็อบเจกต์ที่สร้างจากคลาสที่เราเขียนขึ้นเอง

```
public class SortNameDemo {  // run this class to test Name class
    public static void main( String args[] ) {
        List <Name> l = new  ArrayList <Name> ();

             l.add( new Name("Sombat", "Maimee"));
             l.add( new Name("Somsri", "Deejing"));
             l.add( new Name("Amorn", "Nonnan"));
        l.add( new Name("Pichai", "Maimee"));

       Collections.sort( l );

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

ตัวอย่างนี้ทำการเรียงลำดับข้อมูลด้วยเมท็อด `sort` ของคลาส `Collections` เช่นเดียวกับตัวอย่างที่ ๙ แต่แทนที่ใน `List` จะเป็นสตริง จะใช้อ็อบเจกต์จากคลาสที่เราเขียนขึ้นเองแทน

อ็อบเจกต์ของคลาสที่สามารถนำเรียงลำดับได้ต้องอิมพลิเม้นท์อินเทอร์เฟส `Comparable` ดังนั้นคลาส `Name` ที่เราเขียนขึ้นนี้จึงต้องมีเมท็อด `compareTo()` ซึ่งจะถูกเรียกใช้โดยเมท็อด `sort` ของ `Collections` นั่นเอง เมท็อด `compareTo()` ของคลาส `Name` ให้ความสำคัญกับนามสกุลมากว่าชื่อ จึงทำการเปรียบเทียบสกุลก่อน ถ้าพบว่าสกุลไม่ตรงกันก็คือค่าส่วนต่างกลับไปได้เลย แต่ถ้าสกุลเท่ากันจะต้องไปเปรียบเทียบชื่อเพิ่มเติมแล้วใช้ส่วนต่างของการเปรียบเทียบชื่อเป็นผลลัพธ์ของการเปรียบเทียบอ็อบเจกต์ ตัวอย่างนี้ต้องใช้สองคลาสอยู่ในคนละ source file กัน ให้คอมไพล์ Name.java ก่อนแล้วจึงคอมไพล `SortNameDemo` หลังจากนั้นให้สั่งรันคลาส `SortNameDemo` ซึ่งมีเมท็อด `main` จะได้ผลลัพธ์ดังนี้

```
[Somsri Deejing, Pichai Maimee, Sombat Maimee, Amorn Nonnan]
```

จะเห็นว่าผลลัพธ์เรียงตามนามสกุล นามสกุลที่มีค่าน้อยกว่า (ตัวอักษรมาก่อน) จะขึ้นก่อน กรณีที่นามสกุลเท่ากันตัวอย่างนี้คือคือ Maimee ผู้ที่มีนามสกุลเดียวกันข้อมูลอยู่ติดกัน โดยเรียงตามชื่อในกลุ่มของคนที่มีนามสกุลเดียวกัน

ถ้าเราไม่ต้องการให้เรียงลำดับอันเป็นผลมาจากการใช้เมท็อด `compareTo()` เราต้องบอกวิธีการเปรียบเทียบเองทำนองเดียวกับตัวอย่างที่ ๘

ตัวอย่างที่ ๑๓ ปรับปรุงจากตัวอย่างที่ ๑๒ แสดงการ sort โดยใช้ Comparator เป็นตัวเปรียบเทียบอ็อบเจกต์

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

class SortNameLengthDemo {  // run this class to test Name class
    public static void main( String args[] ) {
      List <Name> l = new  ArrayList <Name> ();

           l.add( new Name("Sombat", "Maimee"));
           l.add( new Name("Somsri", "Deejing"));
           l.add( new Name("Amorn",  "Nonnan"));
      l.add( new Name("Pichai", "Maimee"));

      Collections.sort( l ,new NameLengthComparator() );

      for ( Name n: l)
            System.out.println( n );
    }
}

class NameLengthComparator implements Comparator<Name>
{
    public int compare(Name n1, Name n2)
    {
        // Assume neither Name is null.
            return n1.toString().length() - n2.toString().length();
    }
}
```

ตัวอย่างนี้ปรับปรุงมาจากตัวอย่างที่ ๑๒ โดยเพิ่มตัวเปรียบเทียบอ็อบเจกต์ให้ชื่อคลาสว่า `NameLengthComparator` ซึ่งจะต้องอิมพลิเม้นท์ `Comparator` โดยจัดให้มีเมท็อด `compare()` รับพารามิเตอร์เป็นอ็อบเจกต์ของคลาส Name 2 ตัวเข้ามาเปรียบเทียบกัน การเปรียบเทียบในที่นี่จะเปรียบเทียบความยาวของชื่อเต็ม อันประกอบด้วย ชื่อและนามสกุล ซึ่งมีเมท็อด `toString()` ของ Name ช่วยต่อเชื่อมชื่อกับนามสกุลให้อยู่แล้ว จึงเรียกใช้ได้เลย

เมท็อดนี้จะให้ผลลัพธ์เป็นสตริง 1 ตัว แต่สิ่งที่ต้องการเปรียบเทียบคือความยาวของมันจึงเรียกใช้เมท็อด `length()` ของมันอีกทอดเพื่อหาความยาว

ตรงบรรทัด `Collections.sort( l ,new NameLengthComparator() );` เป็นการเรียกเมท็อด Sort ของ คลาส Collections คราวนี้ต้องบอกพารามิเตอร์เพิ่มคือพารามิเตอร์ตัวที่สองจะเป็นอ็อบเจกต์ที่ทำหน้าที่เปรียบเทียบอ็อบเจกต์ซึ่ง ในที่นี่คืออ็อบเจกต์ของคลาส `NameLengthComparator` ที่เราเขียนขึ้นนั่นเอง

หลังจาก sort แล้วจึงพิมพ์ผลลัพธ์ออกมาดูว่าได้ผลตามที่ต้องการหรือไม่ด้วยการใช้ลูป for แบบใหม่ที่เขียนสั้นกระชับกว่าเดิม โดยพิมพ์บรรทัดละชื่อเพื่อให้เปรียบเทียบความยาวได้ง่าย

ผลลัพธ์จากการรันโปรแกรมนี้เป็นดังนี้

```
Amorn Nonnan
Sombat Maimee
Pichai Maimee
Somsri Deejing
```

จะเห็นว่าชื่อที่พิมพ์ออกมาเรียงตามความยาวของชื่อ ไม่ได้เรียงตามตัวอักษร

การเรียงลำดับจากมากไปหาน้อย

จากตัวอย่างการเรียงลำดับข้อมูลที่แสดงมาก่อนหน้านี้ทุกตัวอย่างเป็นการเรียงลำดับจากน้อยไปหามาก(Ascending Order) เรียกว่าเป็นการเรียงลำดับตามธรรมชาติ (Natural order) แต่บางครั้งเราต้องการเรียงลำดับจากมากไปหาน้อย (Descending Order) แล้วจะใช้ความสามารถของจาวาที่มี algorithm สำหรับการเรียงลำดับให้ใช้อยู่แล้วได้อย่างไร จากตัวอย่างที่ ๑๒ แสดงให้เห็นว่าเราสามารถเรียงลำดับอย่างไรก็ได้ที่ไม่เป็นไปตามลำดับตามธรรมชาติแต่เราต้องเตรียมอ็อบเจกต์หรือเมท็อดสำหรับเปรียบเทียบข้อมูลเอง ถ้าเราต้องการเรียงลำดับจากมากไปหาน้อยเราก็ต้องเตรียม comparator เองเช่นกัน ตัวอย่างที่ ๑๔ แสดงเมท็อด `compare()` เพื่อให้เรียงข้อมูลจากมากไปหาน้อย ซึ่งทำได้ด้วยการใช้ เมท็อด `compareTo()` ที่คลาส `Name` มีอยู่เดิมมาทำให้ผลลัพธ์จากการเปรียบเทียบออกมาเป็นตรงกันข้ามด้วยการคูณด้วย -1 หรืออาจทำได้อีกวิธีคือใช้อ็อบเจกต์ `n2` เป็นตัวตั้งในการเปรียบเทียบข้อมูล แทนที่จะใช้ `n1` เป็นตัวตั้ง

ตัวอย่างที่ ๑๔ ปรับแก้ Comparator ให้เปรียบเทียบข้อมูลเพื่อให้เรียงลำดับจากมากไปหาน้อย

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

public class SortNameDescendingOrder {  // run this class to test Name class
    public static void main( String args[] ) {
      List <Name> l = new  ArrayList <Name> ();

           l.add( new Name("Sombat", "Maimee"));
           l.add( new Name("Somsri", "Deejing"));
           l.add( new Name("Amorn",  "Nonnan"));
      l.add( new Name("Pichai", "Maimee"));

      Collections.sort( l ,new NameDescendingComparator());
```

```
      for ( Name n: l)
            System.out.println( n );
    }
}

class NameDescendingComparator implements Comparator<Name>
{
    public int compare(Name n1, Name n2)
    {
            // return -1*(n1.compareTo(n2));
            return n2.compareTo(n1);
    }
}
```

ผลลัพธ์จากการรันโปรแกรมเป็นดังนี้

```
Amorn Nonnan
Sombat Maimee
Pichai Maimee
Somsri Deejing
```

จะเห็นว่าข้อมูลเรียงจากมากไปหาน้อย อย่าลืมนะครับว่าการเรียงลำดับชื่อของคลาส Name **ในที่นี่ให้ความสำคัญกับนามสกุลมากกว่าชื่อ จึงเรียงนามสกุลก่อน ถ้าพบว่านามสกุลเท่ากันจึงจะเรียงตามชื่อ**

ตัวอย่างที่ ๑๔ แสดงให้เห็นการอิมพลิเม้นท์ `Comparator` **เพื่อให้การเปรียบเทียบข้อมูลเป็นไปตามแบบที่เราต้องการ ซึ่งความจริงแล้วถ้าเราต้องการเปลี่ยนเพียงแค่ลำดับให้เป็นไปในทิศทางตรงกันข้ามกับลำดับตามธรรมชาติ เราสามารถใช้ตัวช่วยคือเมท็อด** `reverseOrder()` **ของคลาส** `Collections` **ช่วยสร้าง** `Comparator` **ให้เราได้ เมท็อด** `reverseOrder()` **จะคืนค่าเป็นอ็อบเจกต์ซึ่งมีเมท็อด** `compare()` **ซึ่งทำหน้าที่เปรียบเทียบข้อมูลแบบย้อนลำดับให้เรา ทำให้เราไม่ต้องอิมพลิเม้นท์** `Comparator` **เองตามตัวอย่างที่ ๑๕**

ตัวอย่างที่ ๑๕ แสดงการใช้เมท็อด `Collections.reverseOrder()`

```
import java.util.*;

public class SortNameReverseOrder {  // run this class to test Name class
    public static void main( String args[] ) {
      List <Name> l = new  ArrayList <Name> ();

    l.add( new Name("Sombat", "Maimee"));
    l.add( new Name("Somsri", "Deejing"));
    l.add( new Name("Amorn",  "Nonnan"));
      l.add( new Name("Pichai", "Maimee"));

      Collections.sort( l ,Collections.reverseOrder());

      for ( Name n: l)
            System.out.println( n );
    }
}
```

ตัวอย่างนี้จะให้ผลลัพธ์เช่นเดียวกับตัวอย่างที่ ๑๔ แต่โปรแกรมสั้นกว่าเพราะอาศัยความสามารถของจาวาที่เตรียมวิธีการเรียงลำดับย้อนหลังไว้ให้ใช้แล้ว