Assignment 7 TreeSet

- ศึกษาการใช้คลาส Set สำเร็จรูปที่มีอยู่ใน JAVA เช่น TreeSet
  - ไม่สามารถใส่ object ที่มีคีย์ที่ซ้ำได้ (ถ้าซ้ำจะทับตัวเดิม)
  - ต้องสร้าง Comparable ให้กับข้อมูลที่จะใช้เป็นคีย์ด้วย // public int compareTo(...) {...}
- คลาส TreeSet
  - ใช้ดำเนินงานเกี่ยวกับกลุ่มข้อมูล โดยจัดเก็บข้อมูลแบบ Binary Tree
  - ข้อมูลที่ใส่จะถูกเรียงลำดับอัตโนมัติ เมื่อเรียกใช้
  - ใช้ Iterator ในการเข้าถึงข้อมูลทีละตัวตามลำดับ
- ตัวอย่างคำสั่งที่เกี่ยวข้องกับคลาส TreeSet
  - การจองตัวแปรในคลาส TreeSet
    - TreeSet<Bnode> dict = new TreeSet<Bnode> ();
  - การเพิ่มข้อมูล dict.add(x);
  - การลบข้อมูล dict.remove(x);
  - การตรวจสอบสมาชิกว่ามีอยู่หรือไม่ dict.contains(x)
  - นับจำนวนข้อมูล size() i = dict.size();
  - การอ้างข้อมูลน้อยสุด และมากสุด x = dict.first(); x = dict.last(); // ตัวแรก , ตัวสุดท้าย
  - การอ้างถึงข้อมูล(คัน) บางส่วน(sub set) ตั้งแต่ x ถึง y ต้องใช้ TreeSet z มารับค่าที่ได้

```
TreeSet<Bnode> z = (TreeSet<Bnode>)dict.subSet(x, true, y, true);
// จะได้ข้อมูลมาตั้งแต่ z.first() จนถึง z.last()
// y, true หมายถึงรวมตัว y ถ้า y,false จะไม่รวม y ถ้า (x,true,y,false) จะใช้เป็น (x,y)
for (String x : z.first().mean) // z.first คือ ตัวแรกของ subset z
```

Assignment 7 Test Case

- คำแนะนำ ดัดแปลง Assignment 6ใหม่เปลี่ยนเป็นใช้คลาส TreeSet
  - ออกแบบโหนดให้มี word และ mean (type ให้รวมเข้าไปใน mean)
  - mean ออกแบบโดยใช้คลาส ArrayList <String> เพื่อเก็บความหมายของคำศัพท์ที่ซ้ำกัน
    - ถ้าเจอคำศัพท์ซ้ำกัน ให้ add เฉพาะความหมายเพิ่มใน ArrayList (ก่อน add จะต้องค้นว่าความหมายมีอยู่ใน ArrayList แล้วหรือยัง .contains(mean))
  - นับจำนวนคำศัพท์ที่มีอยู่ (45921)
  - นับจำนวนความหมายที่มีอยู่ (73981)
  - แสดงผลคำศัพท์ที่มีความหมายมากที่สุด (get off = 35 ตัว)

```
public class Bnode implements Comparable<Bnode> {
        String word; // คำศัพท์
        ArrayList<String> mean;  // ความหมายให้เป็น  array list
        public Bnode() {
            word = "";
            mean = new ArrayList<String>(); // สร้างโหนดใหม่จะต้องจอง  List ใหม่ด้วย
        }
```

- Test Case วนรอบอ่านคำค้นจากคีย์บอร์ด แล้วแสดงผลข้อมูล (นับลำดับ 1,2,3,..)

Total Read 74233 records. // แสดงผลข้อมูลทั้งหมดที่อ่านได้
Total word size 45921 words. // แสดงผลจำนวนข้อมูลที่เหลือ
Total meaning size 73981 words.
get off have 35 meaning.
..... แสดงข้อมูล get off อีก 35 บรรทัด ..... // แสดงผลโดยมีหมายเลข 1, 2, 3, ... นำหน้า
ใช้ Test Case เดิมเหมือนกับ Assignment 6

คำแนะนำ Assignment 7

- ทดลองใช้คลาส **TreeSet** ที่มีอยู่ใน **JAVA** ซึ่งจัดการโครงสร้างแบบ **Binary Tree**
- คลาส **TreeSet** //import java.util.TreeSet
  - ใช้ดำเนินงานเกี่ยวกับกลุ่มข้อมูล โดยจัดเก็บข้อมูลแบบ **Binary Tree**
  - ต้องสร้าง **Comparable** ให้กับข้อมูลที่จะใช้เป็นคีย์ด้วย // public int compareTo(...) {...}
  - ไม่สามารถใส่ **object** ที่มีคีย์ที่ซ้ำได้ (ถ้าคีย์ซ้ำจะทับตัวเดิม)
    - ใช้วิธีสร้าง **object** โหนดเดียว แล้วสร้างอาร์เรย์ลิสต์สำหรับเก็บคำแปลแต่ละตัว
  - ใช้ **Iterator (For Each)** ในการเข้าถึงข้อมูลทีละตัวตามลำดับ

    for (Bnode itr : dict) { ในแต่ละรอบจะได้ itr เป็นโหนดข้อมูลที่เรียงลำดับแล้วทีละตัวจนครบ **}**
  - การค้นข้อมูลใช้ตัวค้น 2 ตัวคือค่า **min** และ **max** ซึ่งจะได้ข้อมูลออกมาทั้งกลุ่มที่มีค่าเรียงลำดับ
- ตัวอย่างคำสั่งที่เกี่ยวข้องกับคลาส **TreeSet**
  - การจองตัวแปรในคลาส **TreeSet**
    - **TreeSet<Bnode> dict = new TreeSet<Bnode> ();**
  - การเพิ่มข้อมูล **dict.add(x);**
  - การลบข้อมูล **dict.remove(x);**
  - การตรวจสอบสมาชิกว่ามีอยู่หรือไม่ **dict.contains(x)**
  - นับจำนวนข้อมูล **size()** **i = dict.size();**
  - การอ้างข้อมูลน้อยสุด และมากสุด **x = dict.first(); x = dict.last();** // ตัวแรก , ตัวสุดท้าย
  - การอ้างถึงข้อมูล(คัน) บางส่วน(sub set) ตั้งแต่ x ถึง y ต้องใช้ **TreeSet z** มารับค่าที่ได้

    TreeSet<Bnode> z = (TreeSet<Bnode>)dict.subSet(x, true, y, true);

    // จะได้ข้อมูลมาตั้งแต่ z.first() จนถึง z.last()

    // y, true หมายถึงรวมตัว y ถ้า y,false จะไม่รวม y ถ้า (x,true,y,false) จะใช้เป็น (x,y)

    for (String x : z.first().mean) // z.first คือ ตัวแรกของ subset z

คำแนะนำ Assignment 7

- สร้าง/ดัดแปลง **Assignment 6** ใหม่เปลี่ยนเป็นใช้คลาส **TreeSet**
  - ออกแบบโหนดให้มีฟิลด์ **word** และ **mean** **(type ให้รวมเข้าไปใน mean)**
  - **String word;** ใช้เป็นคำค้น(ทำ **Comparable**)
  - **mean** ออกแบบโดยใช้คลาส **ArrayList <String>** เพื่อเก็บความหมายของคำศัพท์ที่ซ้ำกัน

```
public class Bnode implements Comparable<Bnode> {
        String word; // คำศัพท์
        ArrayList<String> mean;  // ความหมายให้เป็น  array list
        public Bnode() {// สำหรับโหนดที่ไม่มีข้อมูล
            word = "";
            mean = new ArrayList<String>(); // สร้างโหนดใหม่จะต้องจอง  List ใหม่ด้วย
        }
```

  - สร้าง **constructor** ที่รับข้อมูลสตริง มาแบ่งเป็น 2 ส่วน โดยใช้คำสั่ง **split** แล้วนำไปเก็บในตัวแปรของคลาส //เพื่อใช้ในกรณีที่อ่านข้อมูลมา 1 ชุด แล้วส่งมาสร้าง object

```
public Bnode(String buff) {
    buff = buff.trim().replaceAll("\\s+", " ");
    String[] str = buff.split(",");
    word = str[0]; //ใส่คำศัพท์ลงใน word
    String meaning = str[1] + "(" + str[2] + ")";
    mean = new ArrayList<String>(); // โหนดใหม่จะต้องจอง  List ใหม่ด้วย
    mean.add(meaning); //ใส่ความหมายลงใน meaning
}
```

  - สร้างเมธอดชื่อ **compareTo** เพื่อเปรียบเทียบค่าคีย์แล้ว **return** ตัวเลข **<0, 0 ,>0**
  - อาจสร้างเมธอด **toString** เพื่อใช้พิมพ์ข้อมูลในโหนด

คำแนะนำ Assignment 7

- สร้างคลาส **dictTree** สำหรับรัน นำโครงสร้าง **Bnode** มาสร้าง **TreeSet** เพื่อนำไปใช้
  - กำหนดตัวแปร **dict** สำหรับเก็บข้อมูลทั้งหมดเป็น ชนิด **TreeSet**

    TreeSet<Bnode> dict = new TreeSet<Bnode>();
  - สร้างเมธอดสำหรับอ่านข้อมูลจากไฟล์ทีละ **1** บรรทัด
    - ส่งบรรทัดที่อ่านได้ไปสร้างโหนด // Bnode x = new Bnode(buff);
    - ตรวจสอบดูว่ามีคำซ้ำอยู่ใน **dict** หรือไม่ **(**ต้องมี **CompareTo)** // if (dict.contains(x))
      - ถ้าไม่มีให้เพิ่มโหนดเข้าไปใน **dict** //dict.add(x);
      - แต่ถ้ามีอยู่แล้วให้สร้าง นำโหนดนั้น มาเพิ่มความหมาย // ค้นเจอที่ z.first

        TreeSet<Bnode> z = (TreeSet<Bnode>) data.subSet(x, true, x, true);
      - ตรวจสอบโหนดที่ดึงออกมาว่ามีความหมายซ้ำอยู่ก่อนหรือไม่

        //ความหมายเก็บอยู่ใน x.mean.get(0) นำไปค้นใน z.first.mean

        if (!z.first().mean.contains(x.mean.get(0))) //กรณีค้นไม่เจอ

        ถ้าไม่มีให้เพิ่มเฉพาะคำแปลเข้าไปใน **mean** // z.first().mean.add()

        z.first().mean.add(x.mean.get(0));
  - สร้างเมธอดสำหรับค้นหาและแสดงผล
    - สร้างโหนดที่มีคีย์เป็นคำค้น ไม่ต้องสนใจส่วนที่เป็นคำแปล

      Bnode key = new Bnode();

      key.word = keystr;
    - นำโหนดไปค้นใน **dict** //ค้นเช่นเดียวกับตอนสร้างโหนด
      - ถ้าเจอให้นำโหนดนั้น มาแสดงผลความหมายทั้งหมด

      for (int i = 0; i < z.first().mean.size(); i++)

      System.out.println(i + 1 + ") " + z.first().mean.get(i));