DISCUSS:

HOW TO CREATE A LIST WHICH EACH ELEMENT JUST OCCUR ONE TIME ?

SET

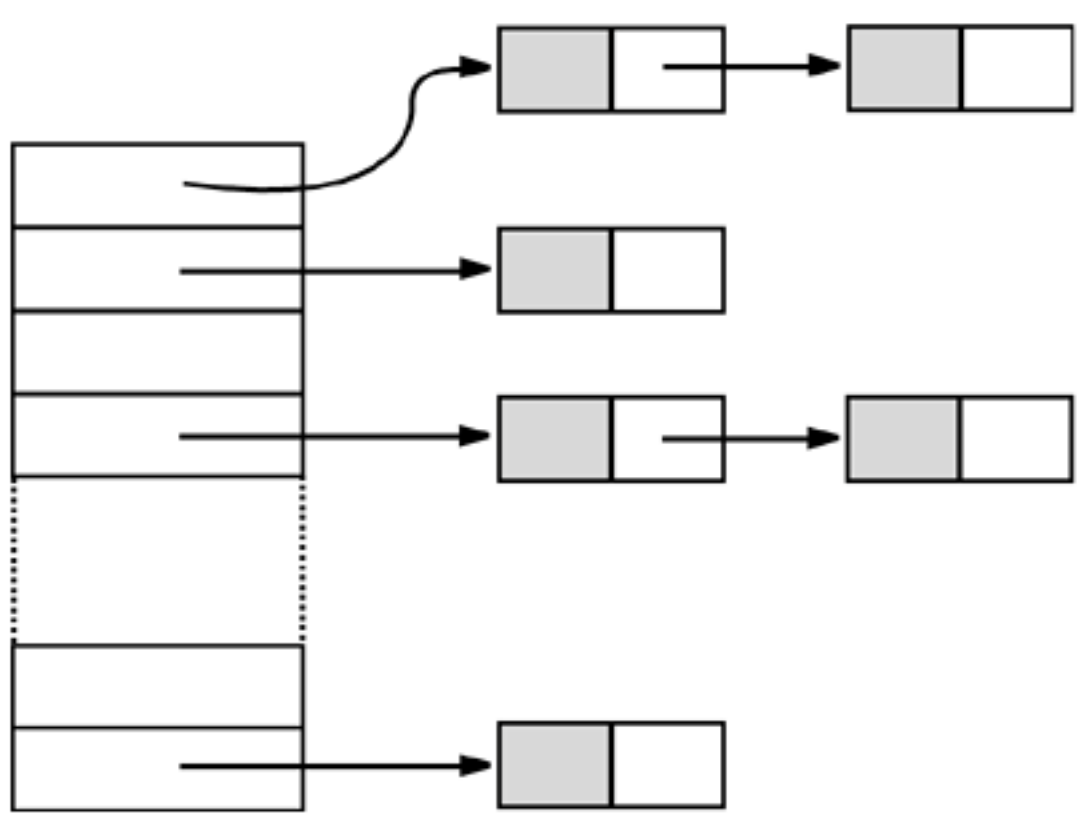
DEFINITION SET

- *A Set is a Collection that cannot contain duplicate elements*
- *EXAMPLE:*
  - *{“PINE”,“APPLE”,”PEN”,”PINE”} => LIST*
  - *{“PINE”,”APPLE”,”PEN”} => SET*
  - *{1,2,3,4,5,2,3} => LIST*
  - *{1,2,3,4,5} => SET*

CLASSIFY OF SET

1. *HASH SET*
2. *TREE SET*
3. *LINKED HASH SET*

HASH SET



HASH SET

- HashSet(): constructs an empty hash set.
- HashSet(Collection<? extends E> elements): constructs a hash set and adds all elements from a collection.
- HashSet(int initialCapacity) constructs an empty hash set with the specified capacity.
- HashSet(int initialCapacity, float loadFactor) constructs an empty hash set with the specified capacity and load factor (a number between 0.0 and 1.0 that determines at what percentage of fullness the hash table will be rehashed into a larger one).

TEST FOR HASHSET

```
public class SetTest {
public static void main(String[] args)
{
Set<String> words = new HashSet<String>(); // HashSet implements Set
long totalTime = 0;
Scanner in = new Scanner(System.in);
while (in.hasNext())
{
String word = in.next();
long callTime = System.currentTimeMillis();
words.add(word);
callTime = System.currentTimeMillis() - callTime;
totalTime += callTime;
System.out.println(words);
}
}
}
```

TEST FOR LINKEDHASHSET

```
public class SetTest {
public static void main(String[] args)
{
Set<String> words = new LinkedHashSet<String>(); // HashSet implements Set
long totalTime = 0;
Scanner in = new Scanner(System.in);
while (in.hasNext())
{
String word = in.next();
long callTime = System.currentTimeMillis();
words.add(word);
callTime = System.currentTimeMillis() - callTime;
totalTime += callTime;
System.out.println(words);
}
}
}
```

TREE SET

- *The TreeSet class is similar to the hash set, with one added improvement. A tree set is a sorted collection*
- *You insert elements into the collection in any order. When you iterate through the collection, the values are automatically presented in sorted order*

TREE SET

```
SortedSet<String> sorter = new TreeSet <String>();
sorter.add("Bob");
sorter.add("Amy");
sorter.add("Carl");
for (String s : sorter) System.println(s);
```

Results :

Amy

Bob

Carl

CONSTRUCTOR OF TREESET

- *TreeSet( )*
- *TreeSet(Collection c)*
- *TreeSet(Comparator cmp)*
- *TreeSet(SortedSet s)*

TEST FOR TREESET

```
public class SetTest {
public static void main(String[] args)
 {
Set<String> words = new TreeSet<String>(); // HashSet implements Set
long totalTime = 0;
Scanner in = new Scanner(System.in);
while (in.hasNext())
{
String word = in.next();
long callTime = System.currentTimeMillis();
words.add(word);
callTime = System.currentTimeMillis() - callTime;
totalTime += callTime;
System.out.println(words);
}
}
}
```

HOW DOES THE TREESET KNOW HOW YOU WANT THE ELEMENTS SORTED?

COMPARABLE

```
public interface Comparable<T> {
int compareTo(T other);
}
```

- The call a.compareTo(b) must return 0 if a and b are equal, a negative integer if a comes before b in the sort order, and a positive integer if a comes after b. The exact value does not matter; only its sign (>0, 0, or < 0) matters.
- < 0 : a comes before b a b …
- = 0 : a and b are equals a b … hay b a …
- > 0 : a comes after b b a …
- Note : dựa vào kết quả a – b ta sẽ biết thứ tự là a b hay b a

COMPARABLE

- For example, here is how you can sort Item objects by part number. Des or Asc ???

```
class Item implements Comparable<Item> {
    public int compareTo(Item other) {
    if (partNumber > other.partNumber)
    return 1;
    else if (partNumber < other.partNumber) return -1;
    else
    return 0;
}
. . .
}
```

COMPARATOR

```
public interface Comparator<T> {
int compare(T a, T b);
}
```

int **compare(** Object a, Object b)

< 0 : a comes before b -> a b

> 0 : a comes after b -> b a

= 0 : a and b are equals

COMPARATOR

```
class ItemComparator implements Comparator<Item>{
public int compare(Item a, Item b){
String descrA = a.getDescription();
String descrB = b.getDescription();
return descrA.compareTo(descrB);
}
}
```

SORTEDSET

```
SortedSet<Item> sortByDescription = new TreeSet<Item>
(new Comparator<Item>()
{
public int compare(Item a, Item b)
{
String descrA = a.getDescription();
String descrB = b.getDescription();
return descrA.compareTo(descrB);
}
});
```

EXERCISE

- Tạo một lớp OrderItem có các thuộc tính sau:
- orderItemId, itemId, itemName, unitPrice, quantity
- Lớp OrderItem hiện thực lớp Comparator hoặc Comparable.
- Tùy theo cách chọn lớp Mhang hiện thực Comparator hay Comparable mà ta viết hàm compareTo hay compare.

```
import java.util.*;
public class OrderItem implements Comparable<MatHang>{
String orderItemId;
String itemId; String itemName;
int unitPrice;
int quantity;
public OrderItem(String orderItemId,String orderItemId,int itemName,int unitPrice, int quantity) {
// TODO Auto-generated constructor stub
}
@Override
public int compareTo(MatHang mh) {
return this.maMH.compareTo(mh.maMH);
}
@Override
public String toString(){ //TODO}
```

EXERCISE

- *Tạo lớp Order có thuộc tính gio là một TreeSet hoặc HashSet*

```
public class Order {
TreeSet<OrderItem> bag = new TreeSet<MatHang>();
public Order() {
}
```

- *Viết các phương thức add(OrderItem oi), remove(OrderItem oi) để hỗ trợ cho thao tác thêm hàng vào giỏ và xóa hàng ra khỏi giỏ.*
- Viết hàm tìm kiếm thông tin mặt hàng dựa trên tên mặt hàng.
- Viết hàm tìm kiếm trả về những mặt hàng nào có bắt đầu bằng chữ cái người dùng nhận vào.
- Viết hàm tìm kiếm trả về những mặt hàng có số lượng lớn hơn số lượng người dùng nhận vào.

EXERCISE

- Viết hàm AddSpecial(MatHang mh) trả về void. Phương thức này thêm mặt hàng vào trong giỏ, nếu mặt hàng đã có rồi thì sẽ cập nhật phần số lượng của mặt hàng = số lượng cũ của mặt hàng + số lượng mới thêm.
- Viết hàm RemoveSpecial(MatHang mh) trả về void. Phương thức này xóa mặt hàng trong giỏ nếu mặt hàng cần xóa có cùng tất cả các thông tin về mặt hàng(cùng mahang, soluong…), nếu số lượng mặt hàng trong giỏ có số lượng lớn hơn thì sẽ cập nhật phần số lượng của mặt hàng = số lượng cũ của mặt hàng - số lượng lấy ra, nếu số lượng mặt hàng trong giỏ có số lượng bé hơn thì báo lỗi.

EXERCISE

- Viết hàm tính giá trị của giỏ hàng. Biết rằng trị giá giỏ hàng = tổng trị giá của các mặt hàng trong giỏ. Trị giá của từng mặt hàng trong giỏ = đơn giá * số lượng.
- Viết hàm in hóa đơn của giỏ hàng thể hiện chi tiết thông tin về mặt hàng trong giỏ, thành tiền của từng mặt hàng. Tổng trị giá giỏ hàng.