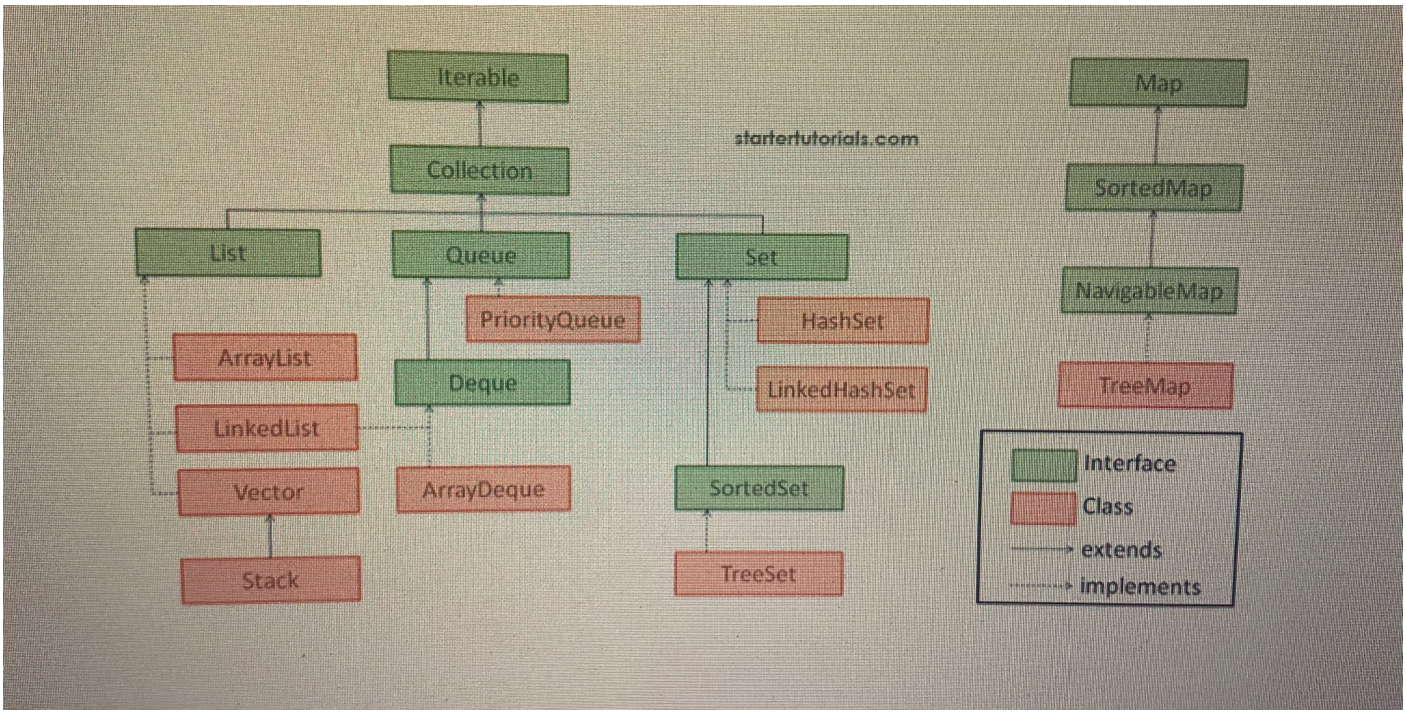
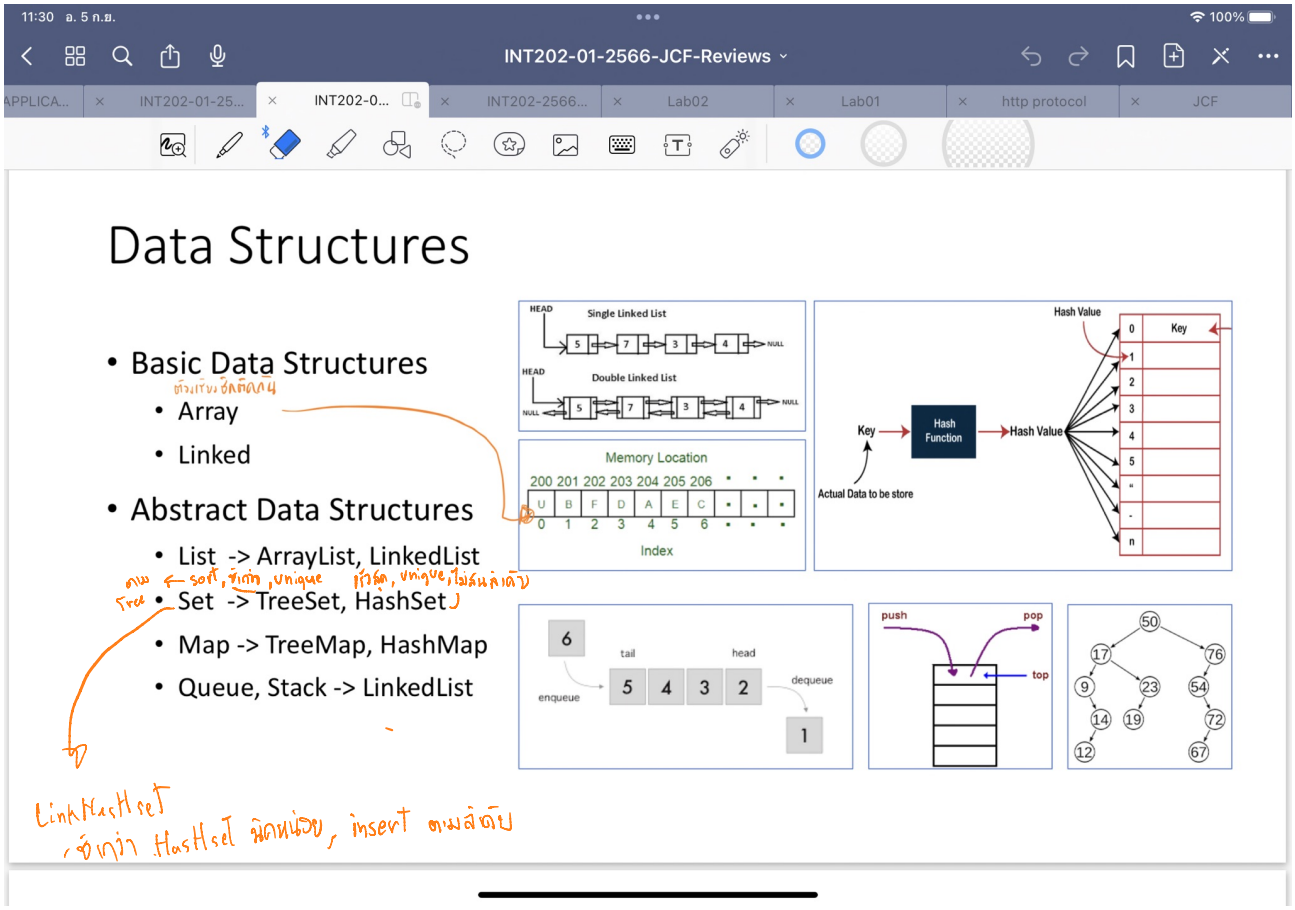

Iterable
Collection
List
Queue
Set
PriorityQueue
HashSet
ArrayList
Deque
LinkedHashSet
LinkedList
Vector
ArrayDeque
SortedSet
Stack
TreeSet
Map
SortedMap
NavigableMap
TreeMap
startertutorials.com
Interface
Class
extends
implements


11:30 อ. 5 ก.ย.
100%
INT202-01-2566-JCF-Reviews
APPLICA...
INT202-01-25...
INT202-0...
INT202-2566...
Lab02
Lab01
http protocol
JCF
Data Structures
Basic Data Structures
Array
Linked
Abstract Data Structures
List -> ArrayList, LinkedList
Set -> TreeSet, HashSet
Map -> TreeMap, HashMap
Queue, Stack -> LinkedList
Single Linked List
Double Linked List
HEAD
NULL
Memory Location
200 201 202 203 204 205 206
U B F D A E C
0 1 2 3 4 5 6
Index
Hash Value
Key
Hash Function
Actual Data to be store
0
1
2
3
4
5
n
6
tail
head
enqueue
dequeue
5 4 3 2
1
push
pop
top
50
17
76
9
23
54
14
19
72
12
67

Link

Array List

Link List เป็น Node

1 Node มีค่า
- next (node)
- previous (node)
- data

ข้อดี Insert / ลบ ง่าย

ข้อเสีย ค้นพื้นที่

Array List

ถ้าไม่ระบุจะมี size = 10

= ต้องจองพื้นใน memory ขนาดใหญ่เพราะข้อมูลต้องยัดกลับ

สูตร resize formular (size * 3) / 2 + 1

↳ 50 % increase

ไปลอง modifie สูตรได้ ถ้าอยากทำก็สร้าง

class ใหม่ แล้ว implement เองนะ

Hash

(Index)

input(key) + HashFunction = HashValue

- HashValue should unique

* implemet HashCode, equals ()

		Ascii						
Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Jan	J	74	a	97	n	110	281	6
Ada	A	65	d	100	a	97	262	9
Leo	L	76	e	101	o	111	288	2

เอา Ascii มาบวก

HashTable like Array

jan goes in at six
Eider at nine Leo it

Hash function

Index number = *sum ASCII codes* Mod *size of array*

ASCII codes modulo the size of the array
in this case 11

หาค่าที่ต้องการ โดยเอาค่านั้นแปลงเป็น index แล้วไปเอาค่า O(1)

Find Ada 262 Mod 11 = 9

Ada myData = Array(9)

calculated index number to perform a
fast array look up rather than just

Hashing Algorithm → Hashing

- Calculation applied to a key to transform it into an address → index
- For numeric keys, divide the key by the number of available addresses, *n*, and take the remainder

address = *key* Mod *n*

- For alphanumeric keys, divide the sum of ASCII codes in a key by the number of available addresses, *n*, and take the remainder
- Folding method divides key into equal parts then adds the parts together
  - The telephone number 01452 8345654, becomes 01 + 45 + 28 + 34 + 56 + 54 = 218
  - Depending on size of table, may then divide by some constant and take remainder

we'll divide this by some constant and take the remainder there

Collisions

เพราะ ไม่ Real work แม้ key จะต่าง ก็มีโอกาส ได้ index ที่ซ้ำกัน

index number for both but both items can't go in the same place this is known

open address เพราะ ถ้า Load Factor น้อย

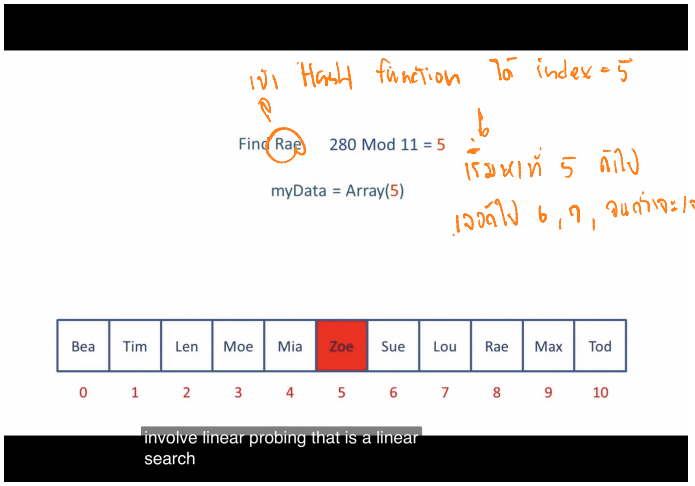
Mia	M	77	i	105	a	97	279	4
Tim	T	84	i	105	m	109	298	1
Bea	B	66	e	101	a	97	264	0
Zoe	Z	90	o	111	e	101	302	5
Sue	S	83	u	117	e	101	301	4
Len	L	76	e	101	n	110	287	1
Moe	M	77	o	111	e	101	289	3
Lou	L	76	o	111	u	117	304	7
Rae	R	82	a	97	e	101	280	5
Max	M	77	a	97	x	120	294	8
Tod	T	84	o	111	d	100	295	9

ถ้า 4 ไม่ว่าง ก็ไปช่องถัดไป จนถึงตัวสุดท้าย แล้วถ้ายังไม่มี ที่ลงก็จะวนไปที่ตัวแรก

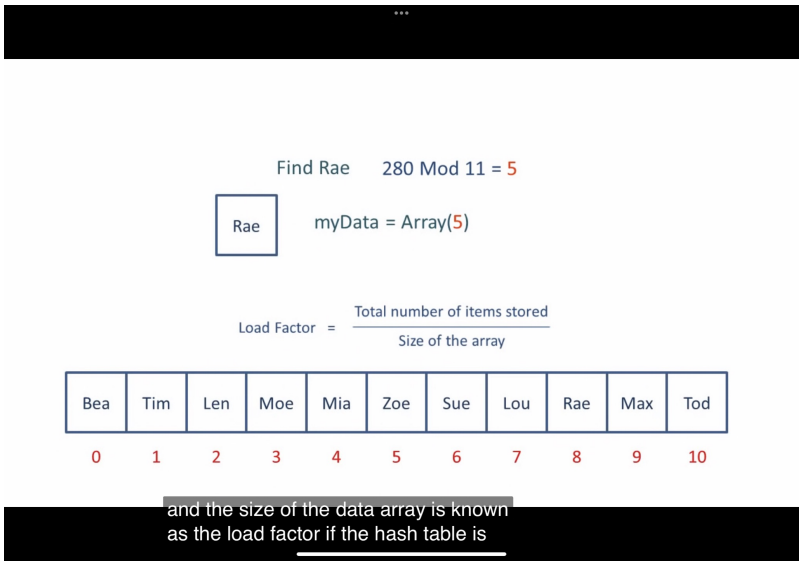
Bea	Tim	Len	Moe	Mia	Zoe	Sue	Lou	Rae	Max	Tod
0	1	2	3	4	5	6	7	8	9	10

it's calculated address is called open addressing because every

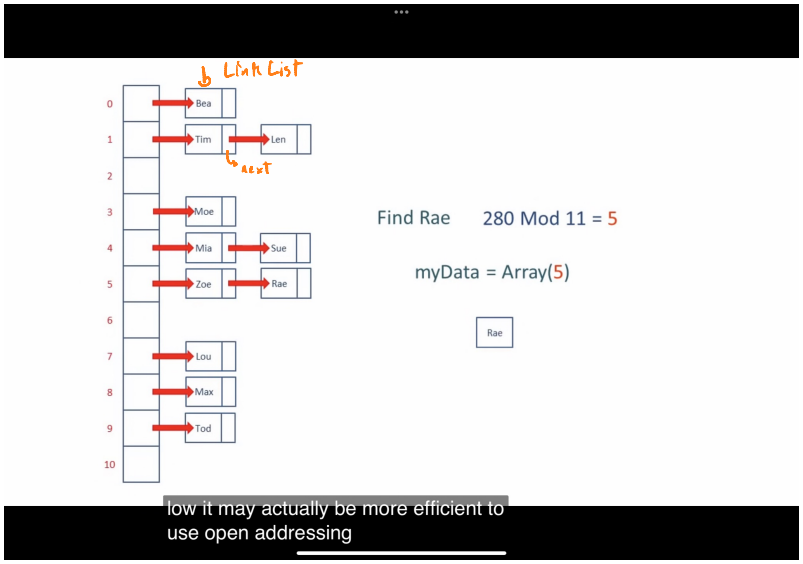
ใน search ใน table



ถูกแก้ Collision ด้วย Load Factor



Chaing / close addressing

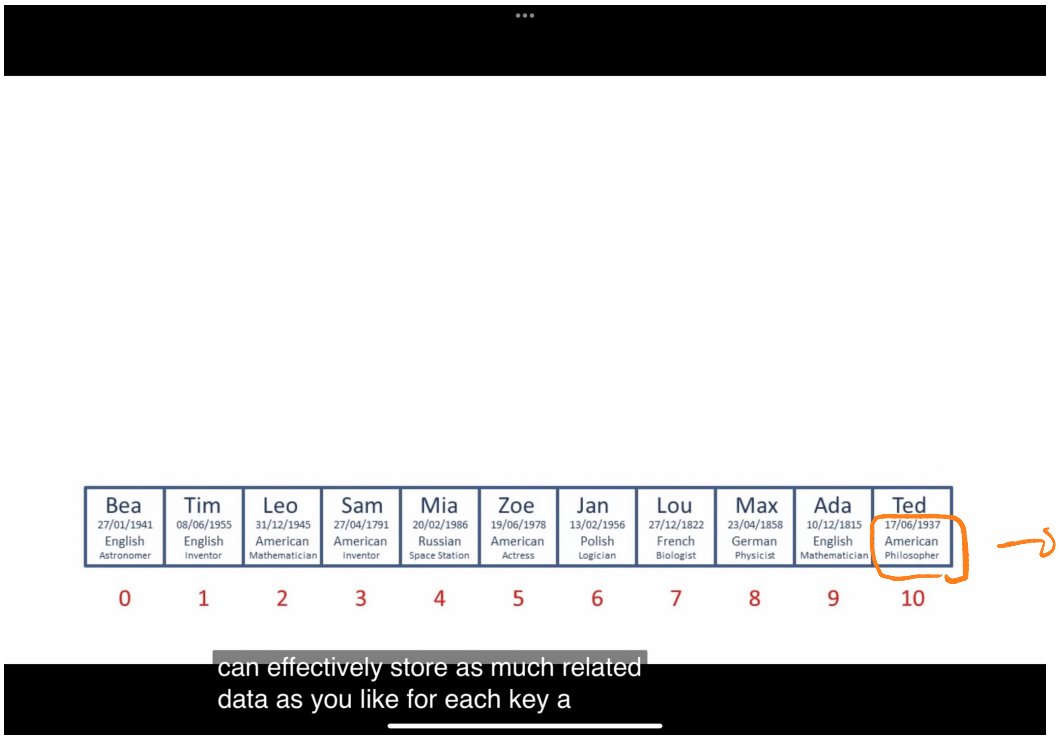


Hash Table store "key = value"

like map and use key for calcu

late the index

In real word



→ obj มีหลาย properties

HashMap (HashTable + Map)

- key can be null
- insert key that already exist will replace the old value

- put()
- get()

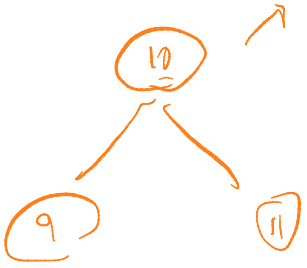
TreeMap sort ตาม Natural order

ค่า key เป็น shadow class of Primitive

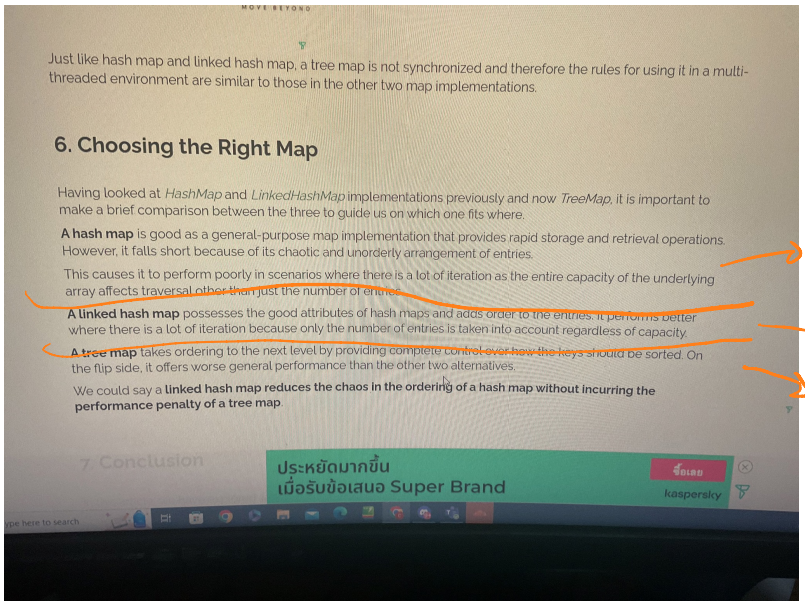
ถ้าเป็น class สร้างขึ้นมาเอง ต้อง Implement compareTo()

red-black Tree → balance binary search Tree

↗ ขึ้นอยู่กับการ implement compare ()



ข้อแตกต่างของ Map ต่างๆ



MOVE BEYOND

Just like hash map and linked hash map, a tree map is not synchronized and therefore the rules for using it in a multi-threaded environment are similar to those in the other two map implementations.

6. Choosing the Right Map

Having looked at *HashMap* and *LinkedHashMap* implementations previously and now *TreeMap*, it is important to make a brief comparison between the three to guide us on which one fits where.

A hash map is good as a general-purpose map implementation that provides rapid storage and retrieval operations. However, it falls short because of its chaotic and unorderly arrangement of entries.

This causes it to perform poorly in scenarios where there is a lot of iteration as the entire capacity of the underlying array affects traversal other than just the number of entries.

A linked hash map possesses the good attributes of hash maps and adds order to the entries. It performs better where there is a lot of iteration because only the number of entries is taken into account regardless of capacity.

A tree map takes ordering to the next level by providing complete control over how the keys should be sorted. On the flip side, it offers worse general performance than the other two alternatives.

We could say a **linked hash map reduces the chaos in the ordering of a hash map without incurring the performance penalty of a tree map**.

7 Conclusion

Hash Map แย่เมื่อต้อง iterate เพราะมันต้องมี Space ว่างไว้ (load Factor)

Cache แค่ element ไม่ต้องเตรียมที่ว่างไว้

Insert ใช้เวลานานกว่าเพราะต้อง คอย maintain order


Set

Tree Set

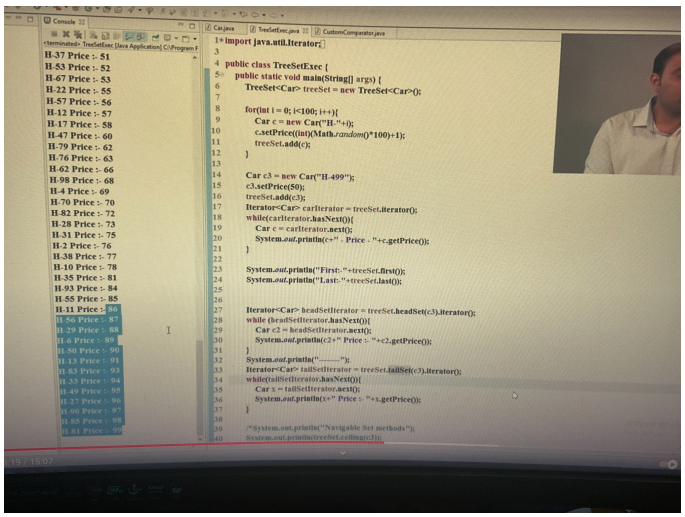
- มันจะ sort ตาม Compareto()

หมายกรณีที่ไม่มี Compareto ใน class

เราสามารถ new TreeSet(iterator)

แต่ดีๆ Compare to อยู่ต่อเราใส่ Comparator

มันก็ใช้ Comparator

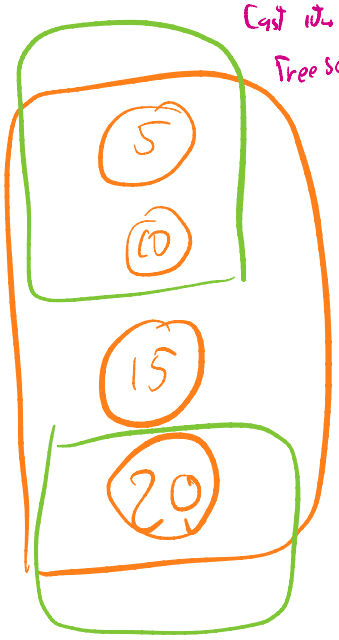


Sort แล้ว

return NavigableSet<E>

↓ ต่อ

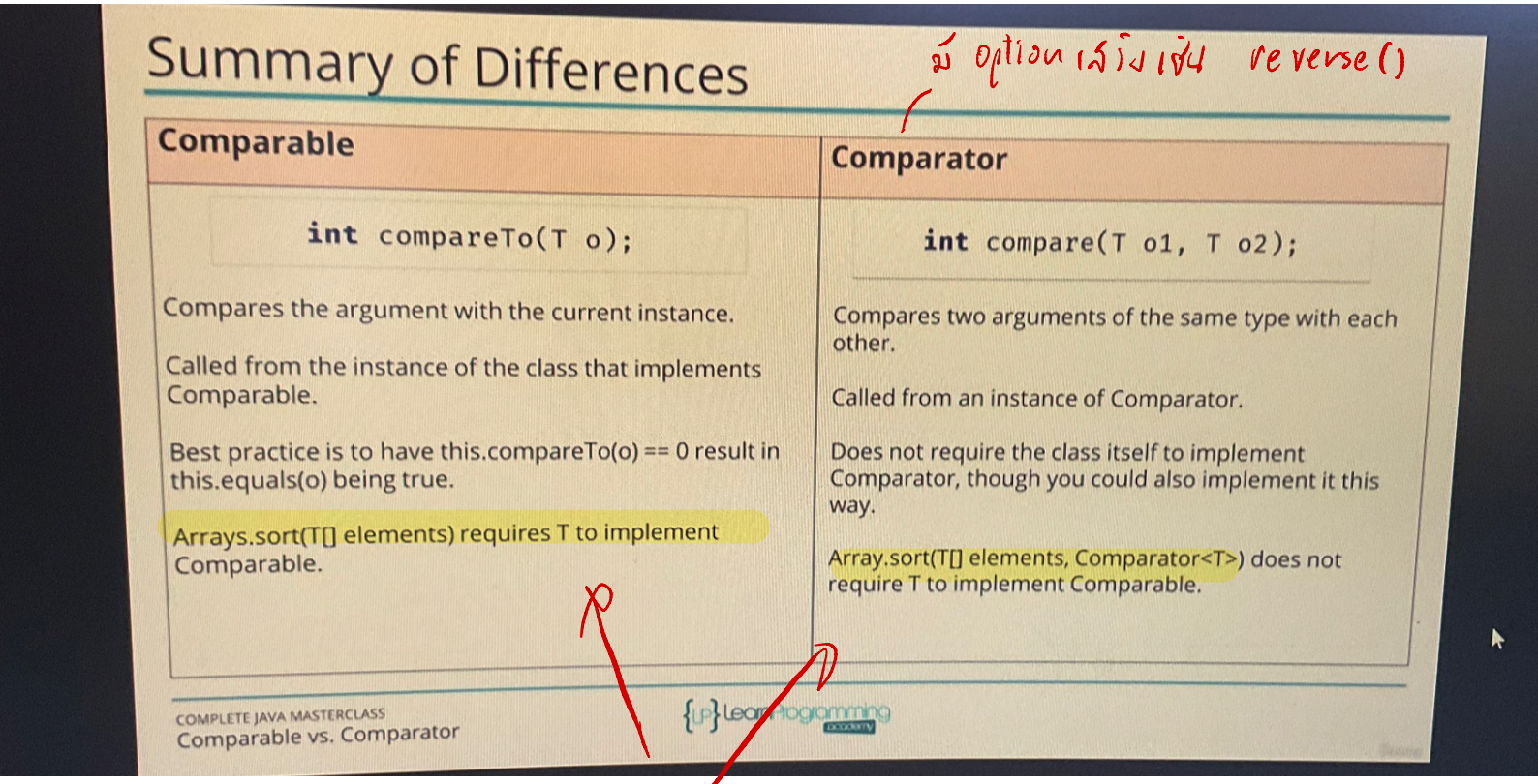
headSet(15) Cast เป็น TreeSet



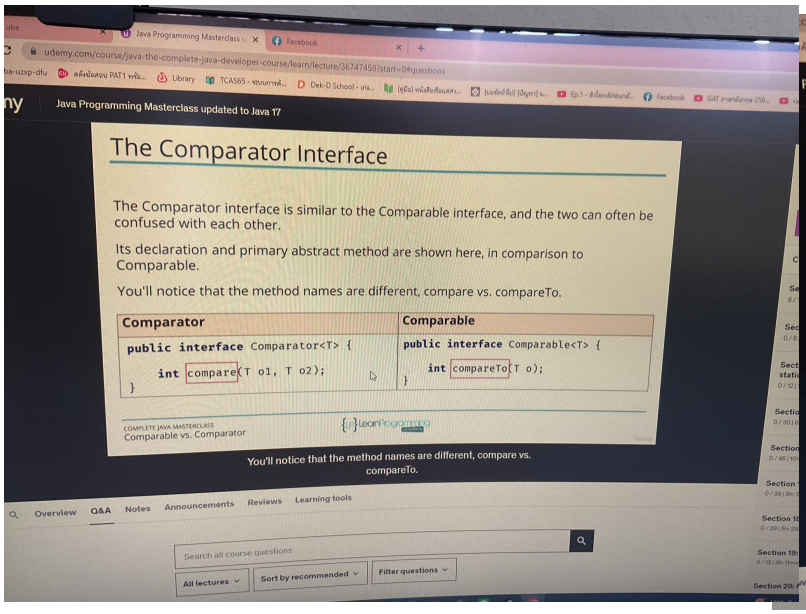
TailSet(15)

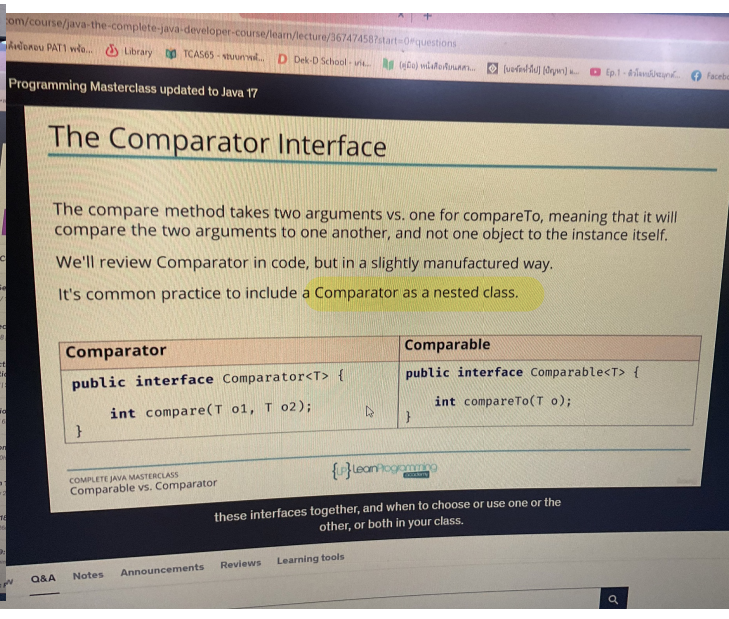
ตัวที่อยู่ตั้งแต่ 15 ขึ้นไป

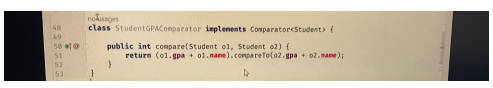
Comparator/ble

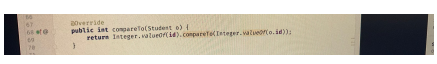


สำคัญ เพราะ ช่วยจะใช้งานร่วมกัน









? class implement Comparable<Students>

Change the comparator to:

```
public int compare(Object o1, Object o2) {
    PlayerStats p1 = (PlayerStats) o1;
    PlayerStats p2 = (PlayerStats) o2;
    int res =  p1.getPlayerLastName().compareToIgnoreCase(p2.getPlayerLastName());
    if (res != 0)
        return res;
    return p1.getPlayerFirstName().compareToIgnoreCase(p2.getPlayerFirstName())
}
```

Share Improve this answer Follow

edited Jan 20, 2022 at 17:16 — John Kugelman — 346k • 67 • 523 • 571

answered Jul 16, 2011 at 21:11 — Petar Ivanov — 90.9k • 11 • 81 • 95

Also, get rid of the nested loops. Collections.sort does all the work for you – dfb Jul 16, 2011 at 21:12

Add a comment

he best answer to your technical question, help

Sign up with email | Sign up with Google | Sign up with Git

Compare first name then last name

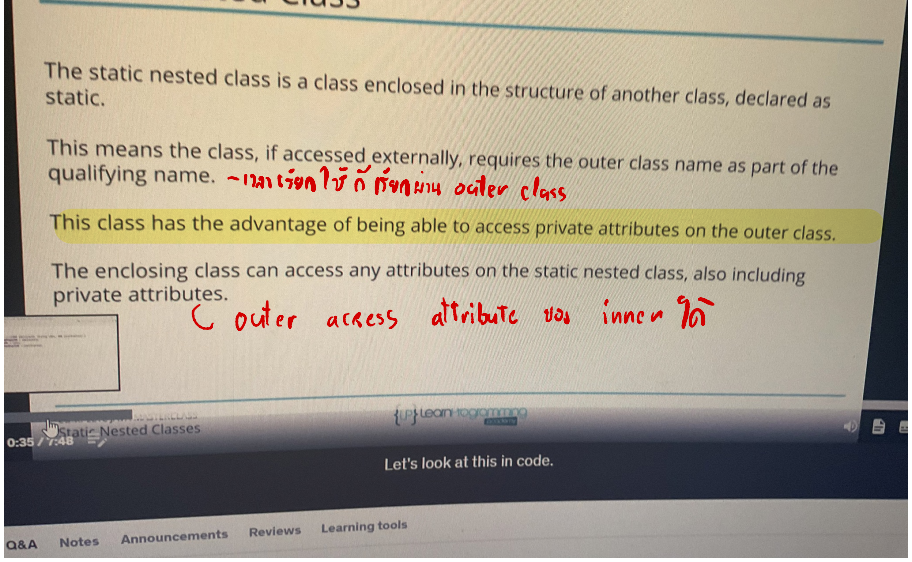
เปรียบเทียบกันไง Comparator

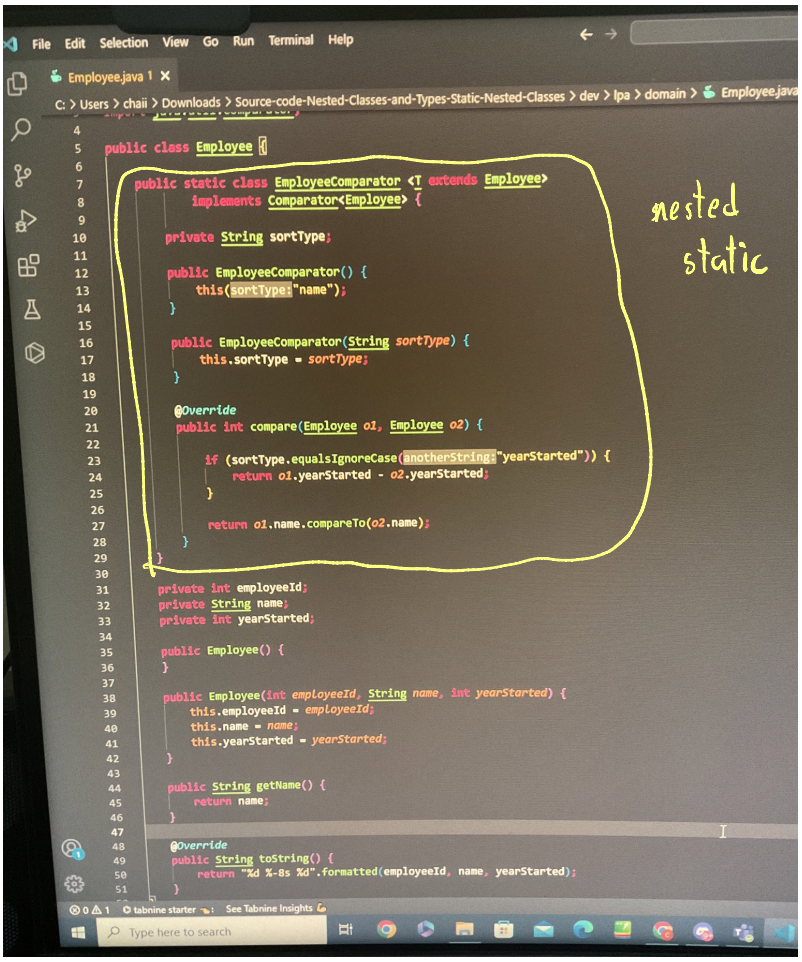
compare (ตัวแรก, ตัวสอง)

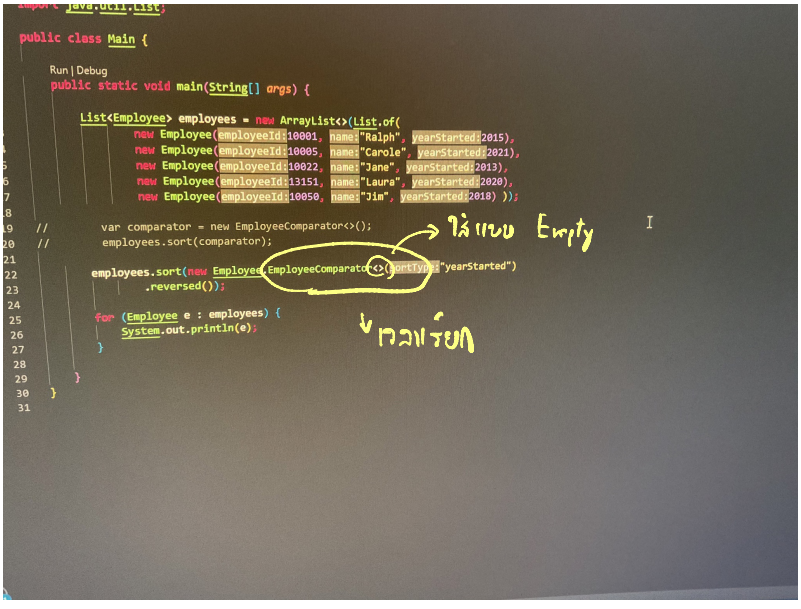
x = ตัวแรก

y = ตัวสอง

- 0: if (x==y)
- -1: if (x < y)
- 1: if (x > y)


The static nested class is a class enclosed in the structure of another class, declared as static.
This means the class, if accessed externally, requires the outer class name as part of the qualifying name.
This class has the advantage of being able to access private attributes on the outer class.
The enclosing class can access any attributes on the static nested class, also including private attributes.
Static Nested Classes
0:35 / 7:48
Let's look at this in code.
Q&A
Notes
Announcements
Reviews
Learning tools


File Edit Selection View Go Run Terminal Help
Employee.java 1
C: > Users > chaii > Downloads > Source-code-Nested-Classes-and-Types-Static-Nested-Classes > dev > lpa > domain > Employee.java
public class Employee {
public static class EmployeeComparator <T extends Employee>
implements Comparator<Employee> {
private String sortType;
public EmployeeComparator() {
this(sortType:"name");
}
public EmployeeComparator(String sortType) {
this.sortType = sortType;
}
@Override
public int compare(Employee o1, Employee o2) {
if (sortType.equalsIgnoreCase(anotherString:"yearStarted")) {
return o1.yearStarted - o2.yearStarted;
}
return o1.name.compareTo(o2.name);
}
}
private int employeeId;
private String name;
private int yearStarted;
public Employee() {
}
public Employee(int employeeId, String name, int yearStarted) {
this.employeeId = employeeId;
this.name = name;
this.yearStarted = yearStarted;
}
public String getName() {
return name;
}
@Override
public String toString() {
return "%d %-8s %d".formatted(employeeId, name, yearStarted);
}
nested
static
Type here to search


public class Main {
Run | Debug
public static void main(String[] args) {
List<Employee> employees = new ArrayList<>(List.of(
new Employee(employeeId:10001, name:"Ralph", yearStarted:2015),
new Employee(employeeId:10005, name:"Carole", yearStarted:2021),
new Employee(employeeId:10022, name:"Jane", yearStarted:2013),
new Employee(employeeId:13151, name:"Laura", yearStarted:2020),
new Employee(employeeId:10050, name:"Jim", yearStarted:2018)));
// var comparator = new EmployeeComparator<>();
// employees.sort(comparator);
employees.sort(new Employee.EmployeeComparator<>(sortType:"yearStarted")
.reversed());
for (Employee e : employees) {
System.out.println(e);
}
ใส่แบบ Empty
ย้อนกลับ