Java 8 features:

Lambda

Stream api

Marker interface:

An interface that does not have any abstract method.

Functional interface:

An interface that contains only 1 abstract method is a functional interface.

Ex:

Comparable

Comparator

Runnable

Predicate

What is lambda?

Is used to provide quick implementation of a functional interface

Anonymous function

To shorten the length of the code

Ex:

**import** java.util.Arrays;

**import** java.util.Comparator;

**class** DescComparator **implements** Comparator<Integer>

{

**public** **int** compare(Integer a, Integer b)

{

**return** b-a;

}

}

**public** **class** App {

**public** **static** **void** main(String[] args) {

Integer arr[]= {50,20,60,10,70,40,30};

// Arrays.sort(arr, new DescComparator());

Arrays.*sort*(arr,(a,b)->b-a);

System.***out***.println(Arrays.*toString*(arr));

}

}

Stream API:

What is stream API?

It is introduced in Java 8 onwards.

Stream API is used only with collections.

Stream API is read only. It works on a copy of the collection. Collection is not affected.

We can do

Filter

Sort

Group by

Order by

Example:

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.stream.Collector;

**import** java.util.stream.Collectors;

**import** java.util.stream.Stream;

**import** java.util.Arrays;

**import** java.util.Collections;

**public** **class** App1 {

**public** **static** **void** main(String[] args) {

List<String> countries=Arrays.*asList*("India","Australia","China","Bangladesh","Sri lanka","Pakistan","Zimbabwe","Denmark","England");

// Stream<String> s = countries.stream();

// s=s.sorted((a,b)->b.compareTo(a));

// s.forEach(System.out::println);

List<String> result = countries.stream()

.sorted((a,b)->b.compareTo(a))

// .forEach(System.out::println);

.collect(Collectors.*toList*());

System.***out***.println(countries);

System.***out***.println(result);

}

}

Task:

Create a list of Integers with random values.

Filter the collection with a condition >80

Display the result in ascending order. And verify if all values are above 80 and the result is in asc order.

Solution:

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.function.Predicate;

//class A implements Predicate<Integer>

//{

//

// @Override

// public boolean test(Integer t) {

// return t>80;

// }

//

//}

**public** **class** App2 {

**public** **static** **void** main(String[] args) {

List<Integer> marks=Arrays.*asList*(44,66,11,55,99,22,88,77,33);

marks.stream()

// .filter(new A())

.filter((t)->t>80)

.sorted()

.forEach(System.***out***::println);

}

}

Task:

Create a class Employee

id, firstName, lastName, salary

Create constructors, getters, setters and toString methods.

In App3.java, main method

Create an ArrayList<Employee> and add few employee objects.

1. Filter the employees based on salary>20000
2. Sort the employees based on their salary in desc order.
3. Display the result

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** App3 {

**public** **static** **void** main(String[] args) {

List<Employee> empList=**new** ArrayList<Employee>();

empList.add(**new** Employee(248774,"Neethu"," Nelliparambil Rathimohan",12345.0));

empList.add(**new** Employee(249547,"Bhagyalakshmi"," Suresh",23451.0));

empList.add(**new** Employee(248825,"Ajay"," C M",45123.0));

empList.add(**new** Employee(249532,"kalyan"," Jalaneela",34512.0));

empList.add(**new** Employee(249529,"AISWARYA"," P",19000.0));

empList.add(**new** Employee(248764,"Ajayakrishnan"," J",10000.0));

}

}

Method Reference

Runnable

run()

Comparable

compareTo

There is an interface and we implement that interface, what we do?

We override the abstract methods of that interface.

Suppose, we take example Runnable interface

**public** **class** MyWork {

**public** **void** work()

{

**for**(**int** i=0;i<10;i++)

{

System.***out***.println(i);

**try** {

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

}

**public** **class** App4 {

**public** **static** **void** main(String[] args) {

Thread t1=**new** Thread(**new** MyWork()::work);

t1.start();

}

}

Activity:

Identify various types of method references.

Static classname::method

Instance objectname::method

Constructor classname:new

Stream API Collectors:

In the EmpList we have, we want to find the sum of salary for every department:

We have added Department field to the Employee class.

**public** **class** Employee {

**private** Integer id;

**private** String firstName;

**private** String lastName;

**private** String department;

**private** Double salary;

**public** Employee() {

}

**public** Employee(Integer id, String firstName, String lastName, Double salary) {

**super**();

**this**.id = id;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.salary = salary;

}

**public** Employee(Integer id, String firstName, String lastName, String department, Double salary) {

**super**();

**this**.id = id;

**this**.firstName = firstName;

**this**.lastName = lastName;

**this**.department = department;

**this**.salary = salary;

}

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** Double getSalary() {

**return** salary;

}

**public** **void** setSalary(Double salary) {

**this**.salary = salary;

}

**public** String getDepartment() {

**return** department;

}

**public** **void** setDepartment(String department) {

**this**.department = department;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", department="

+ department + ", salary=" + salary + "]";

}

}

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** java.util.stream.Collectors;

**public** **class** App5 {

**public** **static** **void** main(String[] args) {

List<Employee> empList=**new** ArrayList<Employee>();

empList.add(**new** Employee(248774,"Neethu"," Nelliparambil Rathimohan","EEE",10.0));

empList.add(**new** Employee(249547,"Bhagyalakshmi"," Suresh","CSE",10.0));

empList.add(**new** Employee(248825,"Ajay"," C M","EEE",10.0));

empList.add(**new** Employee(249532,"kalyan"," Jalaneela","ECE",10.0));

empList.add(**new** Employee(249529,"AISWARYA"," P","EEE",10.0));

empList.add(**new** Employee(248764,"Ajayakrishnan"," J","CSE",10.0));

Map<String, Double> result = empList.stream()

.collect(Collectors.*groupingBy*(Employee::getDepartment,Collectors.*summingDouble*(Employee::getSalary)));

**for**(Entry<String, Double> e:result.entrySet())

System.***out***.println(e);

}

}

Task:

Use stream api to find

Count by Department

Map<String, Long> result = empList.stream()

.collect(Collectors.*groupingBy*(Employee::getDepartment, Collectors.*counting*()));

**for** (Entry<String, Long> e : result.entrySet())

System.***out***.println(e);

Min salary

Map<String, Optional<Employee>> result = empList.stream()

.collect(Collectors.*groupingBy*(Employee::getDepartment,Collectors.*minBy*(Comparator.*comparing*(Employee::getSalary)))) ;

**for**(Entry <String, Optional<Employee>> e:result.entrySet())

System.***out***.println(e);

Max salary

Map<String, Optional<Employee>> result = empList.stream()

.collect(Collectors.*groupingBy*(Employee::getDepartment,Collectors.*maxBy*(Comparator.*comparing*(Employee::getSalary)))) ;

**for**(Entry <String, Optional<Employee>> e:result.entrySet())

System.***out***.println(e);

Avg salary by department

Map<String, Double> result = empList.stream()

.collect(Collectors.*groupingBy*(Employee::getDepartment,Collectors.*averagingDouble*(Employee::getSalary)));

**for**(Entry<String, Double> e:result.entrySet())

System.***out***.println(e);

Reminder: map

Mapping

In Stream api, we work on collections.

Collection of objects. We can map an object to any type.

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** App6 {

**public** **static** **void** main(String[] args) {

List<Employee> empList=**new** ArrayList<Employee>();

empList.add(**new** Employee(248774,"Neethu"," Nelliparambil Rathimohan","EEE",120.0));

empList.add(**new** Employee(249547,"Bhagyalakshmi"," Suresh","CSE",102.0));

empList.add(**new** Employee(248825,"Ajay"," C M","EEE",210.0));

empList.add(**new** Employee(249532,"kalyan"," Jalaneela","ECE",122.0));

empList.add(**new** Employee(249529,"AISWARYA"," P","EEE",112.0));

empList.add(**new** Employee(248764,"Ajayakrishnan"," J","CSE",212.0));

empList.stream()

.filter((a)->a.getFirstName().length()>10)

.map((e)->e.getFirstName().toUpperCase())

.forEach(System.***out***::println);

}

}

Task:

Input: ( is a String)

This sentence is typed by me. You are going to work on this.

Output:

t 5

h 2

i 4

s 4