**Lambda – flat map**

alright so let's now look at the *flat map method*now we've already seen map which can transform one object so that takes a function as an argument which takes one object and returns a value so we may want to map a single object more than one object and we can use a flat map method to actually achieve that the method accepts a function that returns a stream value so we can pass an object as the function argument and returns a string containing several objects which means that were effectively mapping one object too many in fact that's the intended use of the flat map method let's take a look at an example of using that.

|  |  |
| --- | --- |
| **package** com.company; **import** java.util.ArrayList; **import** java.util.Arrays; **import** java.util.List; **import** java.util.stream.Collectors; **import** java.util.stream.Stream; **public class** Main {   **public static void** main(String[] args) {  List<String> strings = Arrays.*asList*(  **"a01"**, **"a02"**,  **"b01"**, **"b04"**, **"b02"**, **"B00"**,  **"c01"**, **"c02"**,  **"d01"**, **"d02"**, **"d03"**, **"d04"** );   Employee snow = **new** Employee(**"Jon snow"**, 10);  Employee tygerrian = **new** Employee(**"Tygerrian Denerriys"**, 5);  Employee ned = **new** Employee(**"Ned Stark"**, 25);  Employee lenisters = **new** Employee(**"Kingslayer Drogo"**, 35);   Depertment hr = **new** Depertment(**"Depertment"**);  hr.addEmployee(snow);  hr.addEmployee(tygerrian);  hr.addEmployee(lenisters);   Depertment accounting = **new** Depertment(**"Accounting"**);  accounting.addEmployee(ned);   List<Depertment> mixlist = **new** ArrayList<>();  mixlist.add(hr);  mixlist.add(accounting);  System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**); **for** (Depertment d : mixlist){  System.***out***.println(d.getemployee());  }  System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);  *//flatmao method* List<Employee> emps = mixlist  .stream()  .flatMap(depertment -> depertment.getemployee().stream())  .collect(Collectors.*toList*());  **for** (Employee x : emps)  System.***out***.println(x);  System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);  *//INSTADE OF THIS FOREACH SHIT WE CAN FUCK THE BUSTERD USING COLLECTION  /\*  strings  .stream()  .map(String::toUpperCase) //s -> s.toUpperCase()  .filter(s -> s.startsWith("B"))  .sorted()  .forEach(System.out :: println);   \*/* List<String> shortedstrings = strings  .stream()  .map(s -> s.toUpperCase()) *//String::toUpperCase* .filter(s -> s.startsWith(**"B"**))  .sorted()  .collect(Collectors.*toList*());   **for** (String s : shortedstrings)  System.***out***.println(s);  } } | **DEPERTMENT**  **package** com.company;  **import** java.util.ArrayList; **import** java.util.List;  **public class** Depertment {  String **name**;  List<Employee> **employees**;   **public** Depertment(String name) {  **this**.**name** = name;  **this**.**employees** = **new** ArrayList<>();  }   **public void** addEmployee(Employee emps){  **employees**.add(emps);  }   **public** List<Employee> getemployee(){  **return employees**;  } }  **EMPLOYEE**  **package** com.company;  **class** Employee{  String **name**;  **int experience**;   *//constructor* **public** Employee(String name, **int** experience) {  **this**.**name** = name;  **this**.**experience** = experience;  }  *//getter and setter* **public** String getName() {  **return name**;  }  **public void** setName(String name) {  **this**.**name** = name;  }  **public int** getExperience() {  **return experience**;  }  **public void** setExperience(**int** experience) {  **this**.**experience** = experience;  }   @Override  **public** String toString() {  **return name**;  } } |
| **OUTPUT** | |
| **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **[Jon snow, Tygerrian Denerriys, Kingslayer Drogo]**  **[Ned Stark]**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **Jon snow**  **Tygerrian Denerriys**  **Kingslayer Drogo**  **Ned Stark**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***  **B00**  **B01**  **B02**  **B04** | |

**Interfaces of COLLECT:**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.ArrayList; **import** java.util.Arrays; **import** java.util.List; **import** java.util.stream.Collectors; **import** java.util.stream.Stream;  **public class** Main {   **public static void** main(String[] args) {  List<String> strings = Arrays.*asList*(  **"a01"**, **"a02"**,  **"b01"**, **"b04"**, **"b02"**, **"B00"**,  **"c01"**, **"c02"**,  **"d01"**, **"d02"**, **"d03"**, **"d04"** );List<String> shortedstrings = strings  .stream()  .map(s -> s.toUpperCase()) *//String::toUpperCase* .filter(s -> s.startsWith(**"B"**))  .sorted()  *//first argument to collect is the supplier  // a supplier create objects now we want an array list so we  //passed the arraylist :: new as the supplier which will construct a new  //arraylist  //passing method references we use the name of the class arraylist in this case  // :: new when we want to pass the constructor  //  //add --> is accumulator add method that's how  //we'll add the items to the arraylist and finally the add all method is the  //combiner so the accumulator is used when the runtime needs to add a single  //version or a single item into the list* .collect(ArrayList::**new**, ArrayList::add, ArrayList::addAll);    **for** (String s : shortedstrings)  System.***out***.println(s);  } } | B00  B01  B02  B04 |

**Interfaces of COLLECT: groupby**

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.ArrayList; **import** java.util.Arrays; **import** java.util.List; **import** java.util.Map; **import** java.util.stream.Collectors; **import** java.util.stream.Stream;  **public class** Main {   **public static void** main(String[] args) {  List<String> strings = Arrays.*asList*(  **"a01"**, **"a02"**,  **"b01"**, **"b04"**, **"b02"**, **"B00"**,  **"c01"**, **"c02"**,  **"d01"**, **"d02"**, **"d03"**, **"d04"** );   Employee snow = **new** Employee(**"Jon snow"**, 10);  Employee tygerrian = **new** Employee(**"Tygerrian Denerriys"**, 5);  Employee ned = **new** Employee(**"Ned Stark"**, 25);  Employee lenisters = **new** Employee(**"Kingslayer Drogo"**, 35);   Depertment hr = **new** Depertment(**"Depertment"**);  hr.addEmployee(snow);  hr.addEmployee(tygerrian);  hr.addEmployee(lenisters);   Depertment accounting = **new** Depertment(**"Accounting"**);  accounting.addEmployee(ned);   List<Depertment> mixlist = **new** ArrayList<>();  mixlist.add(hr);  mixlist.add(accounting);   System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);  **for** (Depertment d : mixlist){  System.***out***.println(d.getemployee());  }  System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);  *//flatmao method* List<Employee> emps = mixlist  .stream()  .flatMap(depertment -> depertment.getemployee().stream())  .collect(Collectors.*toList*());  **for** (Employee x : emps)  System.***out***.println(x);  System.***out***.println(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**);  Map<Integer, List<Employee>> groupbyage =  mixlist  .stream()  .flatMap(depertment -> depertment.getemployee().stream())  .collect(Collectors.*groupingBy*(employess -> employess.getExperience()));   *//finding the youngest employee* mixlist.stream()  .flatMap(depertment -> depertment.getemployee().stream())  .reduce((e1, e2) -> e1.getExperience() < e2.getExperience() ? e1 : e2)  .ifPresent(System.***out***::println);  } } | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  [Jon snow, Tygerrian Denerriys, Kingslayer Drogo]  [Ned Stark]  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Jon snow  Tygerrian Denerriys  Kingslayer Drogo  Ned Stark  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Tygerrian Denerriys |

* Stream is a lazy operation : it means nothing happens until the TERMINAL OPERATION is evaluated

Without using TERMINAL OPERATION

|  |  |
| --- | --- |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.ArrayList; **import** java.util.Arrays; **import** java.util.List; **import** java.util.Map; **import** java.util.stream.Collectors; **import** java.util.stream.Stream;  **public class** Main {   **public static void** main(String[] args) {  Stream.*of*(**"xxx"**, **"xyy"**, **"yyy"**, **"yyx"**)  .filter(s -> {  System.***out***.println(s);  **return** s.length() == 3;  });  } } | <no output will come because we don’t have any TERMINAL OPERATION> |
| **Adding terminal operation** | |
| **package** com.company;  **import** java.lang.reflect.Array; **import** java.util.ArrayList; **import** java.util.Arrays; **import** java.util.List; **import** java.util.Map; **import** java.util.stream.Collectors; **import** java.util.stream.Stream;  **public class** Main {   **public static void** main(String[] args) {  Stream.*of*(**"xxx"**, **"xyy"**, **"yyy"**, **"yyx"**)  .filter(s -> {  System.***out***.println(s);  **return** s.length() == 3;  })  .count();  } } | xxx  xyy  yyy  yyx |



