Lecture 79:

File I.O Streams when we want to write data to file we will use i.o streams. Here the character data is converted to Binary data.

Java 1.8 Streams are entirely different. This is related to Collections. This is used to process objects from the Collections.

**package** com.durgaSoft.section7.lecture79;

**import** java.util.ArrayList;

/\*

\*To get filter a list of even numbers using Java 1.7 method.

\*

\*/

**public** **class** Example1 {

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

ArrayList<Integer> list1 = **new** ArrayList<Integer>();

list1.add(0); list1.add(10); list1.add(20);

list1.add(5); list1.add(15); list1.add(25);

ArrayList<Integer> list2 = **new** ArrayList<Integer>();

**for** (Integer integer : list1) {

**if**((integer%2)==0) {

list2.add(integer);

}

}

System.***out***.println(list1);

System.***out***.println(list2);

}

}

Using Java8:

**package** com.durgaSoft.section7.lecture79;

**import** java.util.ArrayList;

**import** java.util.List;

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

**public** **class** Example2 {

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

ArrayList<Integer> list1 = **new** ArrayList<Integer>();

list1.add(0); list1.add(10); list1.add(20);

list1.add(5); list1.add(15); list1.add(25);

List<Integer> list2 = list1.stream().filter(number -> number%2==0).collect(Collectors.*toList*());

System.***out***.println(list1);

System.***out***.println(list2);

}

}

Lecture 80:

Another Example.:

Here we are trying to process all the elements, where we try to double all the elements in the collection.

**package** com.durgaSoft.section7.lecture79;

**import** java.util.ArrayList;

**public** **class** Example3 {

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

ArrayList<Integer> list1 = **new** ArrayList<Integer>();

list1.add(0); list1.add(10); list1.add(20);

list1.add(5); list1.add(15); list1.add(25);

ArrayList<Integer> list2 = **new** ArrayList<Integer>();

**for** (Integer integer : list1) {

list2.add(integer+integer);

}

System.***out***.println(list1);

System.***out***.println(list2);

}

}

Using Java8:

**package** com.durgaSoft.section7.lecture79;

**import** java.util.ArrayList;

**import** java.util.List;

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

**public** **class** Example4 {

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

ArrayList<Integer> list1 = **new** ArrayList<Integer>();

list1.add(0); list1.add(10); list1.add(20);

list1.add(5); list1.add(15); list1.add(25);

List<Integer> list2 = list1.stream().map(i-> i\*2).collect(Collectors.*toList*());

System.***out***.println(list1);

System.***out***.println(list2);

}

}

Lecture 81:

How to get Stream Object.

Stream s = c.stream();

* Where c is collection Object.
* stream() method is present inside Collection interface as default method.
* Stream is a an interface present in java.uti.Stream package.

Once we obtain the Stream object we can process it. Processing can be done in two steps:

* Configuration:

Configuration can be done in two ways.

* + - Filter:
      * Eg: To get only even numbers
      * Filtering: If we want to filter elements form the collection based on some Boolean condition, then we should go for filtering.
      * We can configure filter by using filter() method, present in Stream interface.

Stream<T> filter(Predicate<? super T> predicate);

Eg:

Stream s = c.stream().filter(i->i%2==0)

* + - Map
      * If we want to process every elements in the collection eg: to double all the element value within a collection.
      * If we want to create a separate new object for every object present in the collection based on some function then we should go for mapping mechanism
      * We can implement mapping by using map() method of Stream interface.

Map Method:

<R> Stream<R> map(Function<? super T, ? extends R> mapper);

Eg:

Stream s = c.stream().map(i->i\*2);

* Processing:
  + Processing by Collect method:
    - This method collects the elements from the stream and adding it to the specified collection.
    - Eg: ArrayList<String> arrayList = new ArrayList<String>();

arrayList.add(“Pavan”); arrayList.add(“Raviteja”); arrayList.add(“Chiranjivi”);

arrayList.add(“Venkatesh”); arrayList.add(“Nagarjuna”);

syso(arrayList);

o/p:

Pavan, Raviteja, Chiranjivi, Venkatesh, Nagarjuna

* + - Now the requirement is to print all the names whose length is greater than 9.

List<String> filteredName = arrayList.stream().filter(name -> name.length>=9).

.collect(Collectors.toList());

syso(filteredName);

o/p:

Chiranjivi, Venkatesh, Nagarjuna

* + Processing by using count () method:
    - This method returns the number of elements present in the Stream
    - Public long count();
    - Eg: To count the number of element whose length is greater than 9.

ArrayList<String> arrayList = new ArrayList<String>();

arrayList.add(“Pavan”); arrayList.add(“Raviteja”); arrayList.add(“Chiranjivi”);

arrayList.add(“Venkatesh”); arrayList.add(“Nagarjuna”);

syso(arrayList);

int result = arrayList.stream().filter(name -> name.length>=9).count()

.

o/p:

Pavan, Raviteja, Chiranjivi, Venkatesh, Nagarjuna

* + Processing by sorted() method
    - We can use sorted() method to sort elements inside stream
    - We can sort either based on default natural sorting order or based on our own Customized sorting order specified by Comparator Object.
      * sorted() -> for default natural sorting order.
      * sorted(Comparator c) -> for Customized sorting order.
  + Processing by min() and max() methods.
    - Min(comparator c)
      * Return minimum value according to specified comparator.
    - max(comparator c)
      * Return maximum value according to specified comparator.

Eg:

//calculate minimum value

Comparator<Integer> defaultNaturalSortingOrder = (number1, number2) -> number1.compareTo(number2);

Integer minimumValue = intList.stream().sorted().min(defaultNaturalSortingOrder).get();

System.out.println("Minimum value "+ minimumValue);

//calculating max value

Integer maximumValue = intList.stream().sorted().max(defaultNaturalSortingOrder).get();

System.out.println("Maximum Value "+maximumValue);

* + Processing by using ForEach() method.
    - This method wont return anything

This method can take lambda expression as argument and apply that lambda expression for every element present in Stream

ArrayList<Integer> intList = new ArrayList<Integer>();

intList.add(0); intList.add(10); intList.add(20);intList.add(5);intList.add(15);intList.add(25);

System.out.println(intList);

intList.stream().forEach(message -> System.out.println(message));

using method reference

//with methodReference

intList.stream().forEach(System.out:: println);

System.out.println();

* + Processing by using toArray() method.
    - We can use toArray method to copy elements present in the stream into specified array.
    - Eg:

ArrayList<Integer> intList = new ArrayList<Integer>();

intList.add(0); intList.add(10); intList.add(20);intList.add(5);intList.add(15);intList.add(25);

Integer[] numbers = intList.stream().toArray(Integer[]::new);

for (Integer integer : numbers) {

System.out.println(integer);

}

* + Stream of() method
    - We can also apply stream for group of values and for arrays
    - For Group of values

Stream<Integer> stream = Stream.of(9,99,999,9999,99999);

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

Stream can also be applied for arrays:

* Double[] numbers= {10.0, 10.1, 10.2, 10.3, 10.4};

Stream<Double> doubleStream = Stream.of(numbers);

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

:

Eg: To filtered names whose length is greater than 9.

Using java 7 version.

**package** com.durgaSoft.section7.lecture81;

**import** java.util.ArrayList;

**public** **class** Example1 {

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

ArrayList<String> namesList = **new** ArrayList<String>();

namesList.add("Pavan"); namesList.add("Raviteja");

namesList.add("Chiranjivi"); namesList.add("Venkatesh");

namesList.add("Nagarjuna");

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

ArrayList<String> filteredName = **new** ArrayList<String>();

**for** (String name : namesList) {

**if**(name.length() >=9) {

filteredName.add(name);

}

}

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

}

}

Using java8:

**public** **class** Example2 {

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

ArrayList<String> namesList = **new** ArrayList<String>();

namesList.add("Pavan"); namesList.add("Raviteja");

namesList.add("Chiranjivi"); namesList.add("Venkatesh");

namesList.add("Nagarjuna");

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

List<String> filteredName = namesList.stream().filter(name -> name.length()>=9).collect(Collectors.*toList*());

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

}

}

Another Example: To convert all the string in an array to Upper case.

**package** com.durgaSoft.section7.lecture81;

**import** java.util.ArrayList;

**import** java.util.List;

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

**public** **class** Example3 {

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

ArrayList<String> namesList = **new** ArrayList<String>();

namesList.add("Pavan"); namesList.add("Raviteja");

namesList.add("Chiranjivi"); namesList.add("Venkatesh");

namesList.add("Nagarjuna");

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

List<String> upperCase = namesList.stream().map(name -> name.toUpperCase()).collect(Collectors.*toList*());

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

}

}

Eg: count the number of elements whose namelenght is less than 9.

**package** com.durgaSoft.section7.lecture81;

**import** java.util.ArrayList;

**import** java.util.List;

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

**public** **class** Example4 {

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

ArrayList<String> namesList = **new** ArrayList<String>();

namesList.add("Pavan");

namesList.add("Raviteja");

namesList.add("Chiranjivi");

namesList.add("Venkatesh");

namesList.add("Nagarjuna");

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

**long** countElementsGreaterThan9 = namesList.stream().filter(name -> name.length() >= 9).count();

System.***out***.println(countElementsGreaterThan9);

}

}

Eg:

For sorted method:

Here we are sorting the ArrayList based on default sorting order and customized sorting order;

**package** com.durgaSoft.section7.lecture81;

**import** java.util.ArrayList;

**import** java.util.Comparator;

**import** java.util.List;

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

**public** **class** Example5 {

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

ArrayList<Integer> intList = **new** ArrayList<Integer>();

intList.add(0); intList.add(10); intList.add(20);intList.add(5);intList.add(15);intList.add(25);

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

//default sorting order

List<Integer> sortedIntList=intList.stream().sorted().collect(Collectors.*toList*());

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

//reverse sorting order

Comparator<Integer> reverseSorting = (number1, number2) -> -(number1.compareTo(number2));

List<Integer> custSortedIntList = intList.stream().sorted(reverseSorting).collect(Collectors.*toList*());

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

}

}

Calculating minimum and maximum values:

**package** com.durgaSoft.section7.lecture81;

**import** java.util.ArrayList;

**import** java.util.Comparator;

**import** java.util.List;

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

**public** **class** Example6 {

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

ArrayList<Integer> intList = **new** ArrayList<Integer>();

intList.add(0); intList.add(10); intList.add(20);intList.add(5);intList.add(15);intList.add(25);

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

//calculate minimum value

Comparator<Integer> defaultNaturalSortingOrder = (number1, number2) -> number1.compareTo(number2);

Integer minimumValue = intList.stream().sorted().min(defaultNaturalSortingOrder).get();

System.***out***.println("Minimum value "+ minimumValue);

//calculating max value

Integer maximumValue = intList.stream().sorted().max(defaultNaturalSortingOrder).get();

System.***out***.println("Maximum Value "+maximumValue);

}

}

Example of toArray:

**package** com.durgaSoft.section7.lecture81;

**import** java.util.ArrayList;

**public** **class** Example8 {

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

ArrayList<Integer> intList = **new** ArrayList<Integer>();

intList.add(0); intList.add(10); intList.add(20);intList.add(5);intList.add(15);intList.add(25);

Integer[] numbers = intList.stream().toArray(Integer[]::**new**);

**for** (Integer integer : numbers) {

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

}

}

}

Example Stream.of() method

**package** com.durgaSoft.section7.lecture81;

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

**public** **class** Example9 {

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

//for a group values

Stream<Integer> stream = Stream.*of*(9,99,999,9999,99999);

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

//for Arrays

Double[] numbers= {10.0, 10.1, 10.2, 10.3, 10.4};

Stream<Double> doubleStream = Stream.*of*(numbers);

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

}

}