**Java Streams Programs**

**Examples for stream filter:**

**Filtering Even Numbers from a List**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

List<Integer> evenNumbers = numbers.stream()

.filter(n -> n % 2 == 0)

.collect(Collectors.toList());

System.out.println(evenNumbers); // Output: [2, 4, 6, 8, 10]

}

}

**Filtering Strings with Length Greater Than 3**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "bat", "car", "banana", "kiwi");

List<String> longWords = words.stream()

.filter(word -> word.length() > 3)

.collect(Collectors.toList());

System.out.println(longWords); // Output: [apple, banana]

}

}

**Filtering with Multiple Conditions**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> strings = Arrays.asList("apple", "banana", "cherry", "date");

List<String> filtered = strings.stream()

.filter(s -> s.startsWith("a") || s.startsWith("b"))

.filter(s -> s.length() > 5)

.collect(Collectors.toList());

System.out.println(filtered); // Output: [banana]

}

}

**Filtering with Complex Conditions**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "elderberry");

List<String> filteredWords = words.stream()

.filter(word -> word.length() >= 5) // length >= 5

.filter(word -> word.contains("e")) // contains 'e'

.collect(Collectors.toList());

System.out.println(filteredWords); // Output: [apple, cherry, elderberry]

}

}

**Filtering Null Values from a List**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("Alice", "Bob", null, "Charlie", null, "Diana");

List<String> nonNullNames = names.stream()

.filter(name -> name != null)

.collect(Collectors.toList());

System.out.println(nonNullNames); // Output: [Alice, Bob, Charlie, Diana]

}

}

**Filtering Based on Dynamic Criteria**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

int threshold = 5;

List<Integer> filteredNumbers = numbers.stream()

.filter(n -> n > threshold)

.collect(Collectors.toList());

System.out.println(filteredNumbers); // Output: [6, 7, 8, 9, 10]

}

}

**Filtering Elements in a Range**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 15, 20, 35, 40, 50);

List<Integer> rangeFiltered = numbers.stream()

.filter(n -> n >= 20 && n <= 40) // numbers in range 20 to 40

.collect(Collectors.toList());

System.out.println(rangeFiltered); // Output: [20, 35, 40]

}

}

**Filtering with Regex Patterns**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> emails = Arrays.asList("alice@example.com", "bob@domain", "carol@website.com", "dave@company");

List<String> validEmails = emails.stream()

.filter(email -> email.matches("^[\\w.+-]+@\\w+\\.\\w+$")) // simple email regex

.collect(Collectors.toList());

System.out.println(validEmails); // Output: [alice@example.com, carol@website.com]

}

}

**Filtering Prime Numbers**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(2, 3, 4, 5, 6, 7, 8, 9, 10);

List<Integer> primeNumbers = numbers.stream()

.filter(FilterExample::isPrime)

.collect(Collectors.toList());

System.out.println(primeNumbers); // Output: [2, 3, 5, 7]

}

private static boolean isPrime(int number) {

if (number <= 1) return false;

if (number == 2) return true;

if (number % 2 == 0) return false;

for (int i = 3; i \* i <= number; i += 2) {

if (number % i == 0) return false;

}

return true;

}

}

**Filtering Based on Collection Membership**

import java.util.Arrays;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> allItems = Arrays.asList("apple", "banana", "orange", "grape");

Set<String> availableItems = new HashSet<>(Arrays.asList("banana", "grape"));

List<String> inStockItems = allItems.stream()

.filter(availableItems::contains)

.collect(Collectors.toList());

System.out.println(inStockItems); // Output: [banana, grape]

}

}

**Examples for stream map:**

**Transform a list of strings to uppercase.**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class MapExample {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "cherry");

List<String> uppercaseWords = words.stream()

.map(String::toUpperCase)

.collect(Collectors.toList());

System.out.println(uppercaseWords); // Output: [APPLE, BANANA, CHERRY]

}

}

**Filtering and Transforming Data**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FilterExample {

public static void main(String[] args) {

List<String> names = Arrays.asList("alice", "bob", "charlie", "david", "eve");

List<String> uppercaseNames = names.stream()

.filter(name -> name.length() > 3) // names with length > 3

.map(String::toUpperCase) // convert to uppercase

.collect(Collectors.toList());

System.out.println(uppercaseNames); // Output: [ALICE, CHARLIE, DAVID]

}

}

**Mapping Integers to Their Squares**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class MapExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> squaredNumbers = numbers.stream()

.map(n -> n \* n)

.collect(Collectors.toList());

System.out.println(squaredNumbers); // Output: [1, 4, 9, 16, 25]

}

}

**Mapping with Conditional Logic**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class MapExample {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<String> numberDescriptions = numbers.stream()

.map(n -> {

if (n % 2 == 0) {

return n + " is even";

} else {

return n + " is odd";

}

})

.collect(Collectors.toList());

System.out.println(numberDescriptions); // Output: [1 is odd, 2 is even, 3 is odd, 4 is even, 5 is odd]

}

}

**Mapping Dates to Formatted Strings**

import java.text.SimpleDateFormat;

import java.util.Arrays;

import java.util.Date;

import java.util.List;

import java.util.stream.Collectors;

public class MapExample {

public static void main(String[] args) {

List<Date> dates = Arrays.asList(

new Date(121, 0, 15), // January 15, 2021

new Date(121, 1, 22), // February 22, 2021

new Date(121, 2, 30) // March 30, 2021

);

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd");

List<String> formattedDates = dates.stream()

.map(formatter::format)

.collect(Collectors.toList());

System.out.println(formattedDates); // Output: [2021-01-15, 2021-02-22, 2021-03-30]

}

}

**Mapping Strings to Their Lengths**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class MapExample {

public static void main(String[] args) {

List<String> words = Arrays.asList("hello", "world", "java", "streams");

List<Integer> lengths = words.stream()

.map(String::length)

.collect(Collectors.toList());

System.out.println(lengths); // Output: [5, 5, 4, 7]

}

}

**Examples for stream flatmap:**

**Flattening a Stream of Strings into Words**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FlatMapExample2 {

public static void main(String[] args) {

List<String> sentences = Arrays.asList(

"Hello world",

"Java Streams are powerful",

"FlatMap is useful"

);

List<String> words = sentences.stream()

.flatMap(sentence -> Arrays.stream(sentence.split(" "))) // Split each sentence into words and flatten the streams

.collect(Collectors.toList()); // Collect into a single list

System.out.println(words); // Output: [Hello, world, Java, Streams, are, powerful, FlatMap, is, useful]

}

}

**Flattening a List of Arrays**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class FlatMapExample5 {

public static void main(String[] args) {

List<int[]> listOfArrays = Arrays.asList(

new int[]{1, 2, 3},

new int[]{4, 5, 6},

new int[]{7, 8, 9}

);

List<Integer> flatList = listOfArrays.stream()

.flatMapToInt(Arrays::stream) // Flatten each array into an IntStream

.boxed() // Convert IntStream to Stream<Integer>

.collect(Collectors.toList()); // Collect into a single list

System.out.println(flatList); // Output: [1, 2, 3, 4, 5, 6, 7, 8, 9]

}

}

**Flattening a Map of Lists**

import java.util.Arrays;

import java.util.List;

import java.util.Map;

import java.util.stream.Collectors;

public class FlatMapExample6 {

public static void main(String[] args) {

Map<String, List<String>> map = Map.of(

"fruits", Arrays.asList("apple", "banana", "cherry"),

"vegetables", Arrays.asList("carrot", "potato", "broccoli")

);

List<String> allValues = map.values().stream()

.flatMap(List::stream) // Flatten each list of values

.collect(Collectors.toList()); // Collect into a single list

System.out.println(allValues); // Output: [apple, banana, cherry, carrot, potato, broccoli]

}

}

**Examples for stream distinct:**

**Removing Duplicates from a List of Integers**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class DistinctExample1 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 2, 3, 4, 4, 5);

List<Integer> distinctNumbers = numbers.stream()

.distinct() // Remove duplicates

.collect(Collectors.toList()); // Collect into a new list

System.out.println(distinctNumbers); // Output: [1, 2, 3, 4, 5]

}

}

**Removing Duplicates from a List of Strings**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class DistinctExample2 {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "apple", "orange", "banana", "pear");

List<String> uniqueWords = words.stream()

.distinct() // Remove duplicates

.collect(Collectors.toList()); // Collect into a new list

System.out.println(uniqueWords); // Output: [apple, banana, orange, pear]

}

}

**Examples for stream sorted:**

**Sorting a List of Integers in Ascending Order**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class SortedExample1 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(5, 3, 8, 1, 2, 9);

List<Integer> sortedNumbers = numbers.stream()

.sorted() // Sort in natural order (ascending)

.collect(Collectors.toList());

System.out.println(sortedNumbers); // Output: [1, 2, 3, 5, 8, 9]

}

}

**Sorting a List of Strings in Alphabetical Order**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class SortedExample2 {

public static void main(String[] args) {

List<String> words = Arrays.asList("banana", "apple", "cherry", "date");

List<String> sortedWords = words.stream()

.sorted() // Sort in natural order (alphabetical)

.collect(Collectors.toList());

System.out.println(sortedWords); // Output: [apple, banana, cherry, date]

}

}

**Sorting in Descending Order**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class SortedExample4 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(5, 3, 8, 1, 2, 9);

List<Integer> sortedDescending = numbers.stream()

.sorted((a, b) -> b.compareTo(a)) // Sort in descending order

.collect(Collectors.toList());

System.out.println(sortedDescending); // Output: [9, 8, 5, 3, 2, 1]

}

}

**Sorting a List of Strings by Length**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class SortedExample5 {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "kiwi", "banana", "pear");

List<String> sortedByLength = words.stream()

.sorted((s1, s2) -> Integer.compare(s1.length(), s2.length())) // Sort by length

.collect(Collectors.toList());

System.out.println(sortedByLength); // Output: [kiwi, pear, apple, banana]

}

}

**Sorting with Custom Comparator Using Method Reference**

import java.util.Arrays;

import java.util.Comparator;

import java.util.List;

import java.util.stream.Collectors;

public class SortedExample7 {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "cherry", "date");

List<String> sortedByLength = words.stream()

.sorted(Comparator.comparingInt(String::length)) // Sort by length using method reference

.collect(Collectors.toList());

System.out.println(sortedByLength); // Output: [date, apple, banana, cherry]

}

}

**Examples for stream peek:**

**Simple Debugging**

import java.util.Arrays;

import java.util.List;

public class PeekExample1 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);

List<Integer> result = numbers.stream()

.peek(n -> System.out.println("Before filtering: " + n)) // Print each element before filtering

.filter(n -> n % 2 == 0) // Filter even numbers

.peek(n -> System.out.println("After filtering: " + n)) // Print each element after filtering

.map(n -> n \* n) // Square each number

.collect(Collectors.toList()); // Collect to list

System.out.println(result); // Output: [4, 16]

}

}

**Tracking Elements through Multiple Stages**

import java.util.Arrays;

import java.util.List;

public class PeekExample2 {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "cherry", "date");

List<String> result = words.stream()

.peek(word -> System.out.println("Original: " + word)) // Print the original words

.map(String::toUpperCase) // Convert words to uppercase

.peek(word -> System.out.println("Uppercase: " + word)) // Print words after conversion

.sorted() // Sort words

.peek(word -> System.out.println("Sorted: " + word)) // Print words after sorting

.collect(Collectors.toList()); // Collect to list

System.out.println(result); // Output: [APPLE, BANANA, CHERRY, DATE]

}

}

**Tracking Elements through Multiple Stages**

import java.util.Arrays;

import java.util.List;

public class PeekExample2 {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "cherry", "date");

List<String> result = words.stream()

.peek(word -> System.out.println("Original: " + word)) // Print the original words

.map(String::toUpperCase) // Convert words to uppercase

.peek(word -> System.out.println("Uppercase: " + word)) // Print words after conversion

.sorted() // Sort words

.peek(word -> System.out.println("Sorted: " + word)) // Print words after sorting

.collect(Collectors.toList()); // Collect to list

System.out.println(result); // Output: [APPLE, BANANA, CHERRY, DATE]

}

}

**Using peek for Debugging Complex Pipelines**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class PeekExample3 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(10, 20, 30, 40, 50);

List<Integer> result = numbers.stream()

.filter(n -> n > 20)

.peek(n -> System.out.println("Filtered: " + n)) // Print elements after filtering

.map(n -> n / 10)

.peek(n -> System.out.println("Mapped: " + n)) // Print elements after mapping

.collect(Collectors.toList()); // Collect to list

System.out.println(result); // Output: [3, 4, 5]

}

}

**Examples for stream limit:**

**Limiting the Number of Elements in a List**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class LimitExample1 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9);

List<Integer> limitedNumbers = numbers.stream()

.limit(3) // Limit to the first 3 elements

.collect(Collectors.toList());

System.out.println(limitedNumbers); // Output: [1, 2, 3]

}

}

**Limiting the Number of Results from a Stream of Strings**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class LimitExample2 {

public static void main(String[] args) {

List<String> words = Arrays.asList("apple", "banana", "cherry", "date", "fig", "grape");

List<String> limitedWords = words.stream()

.limit(4) // Limit to the first 4 elements

.collect(Collectors.toList());

System.out.println(limitedWords); // Output: [apple, banana, cherry, date]

}

}

**Limiting Results in a Sorted Stream**

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class LimitExample3 {

public static void main(String[] args) {

List<Integer> numbers = Arrays.asList(10, 3, 5, 8, 1, 7, 6);

List<Integer> topThreeNumbers = numbers.stream()

.sorted() // Sort in natural order

.limit(3) // Limit to the first 3 elements after sorting

.collect(Collectors.toList());

System.out.println(topThreeNumbers); // Output: [1, 3, 5]

}

}