**Java 8 programs**

**Iteration of map:**

|  |
| --- |
| **// 1. Iterate map object using forEach**  Map<String, Integer>items = **new** HashMap<>();  items.put("A", 10);  items.put("B", 20);  items.put("C", 30);  items.put("D", 40);  items.put("E", 50);  items.put("F", 60);    //items.forEach((k,v) ->System.out.println(k +" : "+ v));  items.forEach((k,v) -> {  **if**(v> 40) {  System.***out***.println(k +" : "+ v);  }  });  **o/p:**  E : 50  F : 60  **// 2. Stream of skip and forEach**  IntStream.*range*(1, 10)  .skip(5)  .forEach(System.***out***::println);  **o/p:**  6  7  8  9  **// 3. Stream of sum of numbers**  **int**sum = IntStream.*range*(1, 10)  .sum();  System.***out***.println(sum);  **o/p:**  45  **// 4. Stream of sortd and findFirst**  Stream.*of*("ab", "nag", "scala", "java","mongodb")  .sorted()  .findFirst()  .ifPresent(System.***out***::println);  **// 5. Stream fromarray, filter,sorted and print**  String[] str = {"ab", "nag", "scala", "java","mongodb", "spring"};  Arrays.*stream*(str)  .filter( s ->s.startsWith("s"))  .sorted()  .forEach(System.***out***::println);  **// 6. Stream of mapaverage and ifPresent**  Arrays.*stream*(**newint**[] {2,4,6,8,10})  .map((e) ->e \* e)  .average()  .ifPresent(System.***out***::println);    **int**sum = Arrays.*stream*(**newint**[] {2,4,6,8,10})  .map(e ->e\*e)  .sum();  System.***out***.println("sum: "+ sum);  **// 7. Stream of sorted,filter and foreach**  List<String>list = Arrays.*asList*("ab", "nag", "scala", "java","mongodb", "spring");  list.stream().map(String::toUpperCase)  .sorted()  .filter(e ->e.length() > 3)  .forEach(System.***out***::println);  **// 8. Stream rowsfrom files, sorted, filter and print**  **Create demo.txt file**  nagendra  scala good programing language  java best programing  rabbitmq  mongodb  swagger  spring  cloud  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  filedata.sorted()  .filter(e ->e.length() > 15)  .forEach(System.***out***::println);  filedata.close();  **o/p:**  java best programing  scala good programing language  **// 9.Stream rows from files, filter and save into list**  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  List<String>list = filedata.sorted()  .filter( s ->s.contains("spring"))  .collect(Collectors.*toList*());  list.forEach(System.***out***::println);  filedata.close();  o/p  spring  **// 10. Stream rows from files map,split,filter,count**  A,1,2,3  B,4,5,6  C,7,8,9  D,10,15  E  F,16,17,18,19,20  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  **long**count = filedata.map( e ->e.split(","))  .filter(e ->e.length == 4)  .count();  System.***out***.println(count);  o/p:  3  **// 11. Stream rows from files map,split,filter,print**  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  filedata.map( e ->e.split(","))  .filter(e ->e.length == 4)  .filter(e ->Integer.*parseInt*(e[1]) > 6)  .forEach(x ->System.***out***.println(x[0] +" " + x[1] +" "+ x[2] + " "+ x[3]));  filedata.close();  o/p:  C 7 8 9  **// 12. Stream rows from files map,split,filters, save into map**  Stream<String>filedata = Files.*lines*(Paths.*get*("demo.txt"));  Map<String, Integer>map = **new** HashMap<>();  map = filedata.map( e ->e.split(","))  .filter(e ->e.length == 4)  .filter(e ->Integer.*parseInt*(e[1]) > 2)  .collect(Collectors.*toMap*( k ->k[0], v ->Integer.*parseInt*(v[1])));  filedata.close();  **for**(Map.Entry<String, Integer>m : map.entrySet()) {  System.***out***.println(m.getKey() +" : "+ m.getValue());  }  o/p:  B : 4  C : 7  **// 13. Stream of reduce**  **double**result = Stream.*of*(1.0,2.0,3.0,4.0,5.0).reduce(0.0, (Double a, Double b) ->a + b);  System.***out***.println(result);  o/p:  15.0  **// 13. Stream of summarystatistics**  IntSummaryStatisticssummary = IntStream.*of*(1,2,3,4,5)  .summaryStatistics();  System.***out***.println(summary);  o/p  IntSummaryStatistics{count=5, sum=15, min=1, average=3.000000, max=5}  **// 14 iterate elements from stream and applyoptional class**  List<String>list = Arrays.*asList*("ab", "db", "c");  Optional<String>opt = list.stream()  .filter(e ->e.startsWith("ab"))  .findAny();  **if** (opt.isPresent()) {  System.***out***.println(opt.get()); // if value present get the value from optional  } **else** {  System.***out***.println("no value found");  }  opt.ifPresent(e ->System.***out***.println(e)); // if value is not found, ifPresent statement executed  List<String>list2 = Arrays.*asList*("ab", "db", "c");  String str = list2.stream().  filter(e ->e.startsWith("dd"))  .findAny()  .orElse("no value found");    System.***out***.println(str);  Optional<String>nonEmptyGender = Optional.of("male");  Optional<String>emptyGender = Optional.empty();  **// 15 Stream from list of objects**  **publicclass** Student {  **private** String name;  **private** Set<String>book;  **publicvoid**addBook(String book) {  **if** (**this**.book == **null**) {  **this**.book = **new** HashSet<>();  }  **this**.book.add(book);  }  **public** String getName() {  **return**name;  }  **publicvoid**setName(String name) {  **this**.name = name;  }  **public** Set<String>getBook() {  **return**book;  }  **publicvoid**setBook(Set<String>book) {  **this**.book = book;  }  }  Studentobj1 = **new**Student();  obj1.setName("mkyong");  obj1.addBook("Java 8 in Action");  obj1.addBook("Spring Boot in Action");  obj1.addBook("Effective Java (2nd Edition)");  Studentobj2 = **new**Student();  obj2.setName("zilap");  obj2.addBook("Learning Python, 5th Edition");  obj2.addBook("Effective Java (2nd Edition)");  List<Student>list = **new**ArrayList<>();  list.add(obj1);  list.add(obj2);  /\* List<String>listresult = list.stream().map( e ->e.getBook())  .flatMap(e ->e.stream())  .filter(e ->e.startsWith("Spring"))  .collect(Collectors.toList());\*/  Optional<String>opnResults= list.stream().map( e ->e.getBook())  .flatMap(e ->e.stream())  .filter(e ->e.startsWith("Spring"))  .findAny();  opnResults.ifPresent(System.***out***::println);  **// 16 Stream from list of objects**  **publicstatic** List<Student>getStudents(){  List<Student>list = **new**ArrayList<>();  list.add(**new** Student(11, 28, "Lucy"));  list.add(**new** Student(28, 27, "Tim"));  list.add(**new** Student(32, 30, "Daniel"));  list.add(**new** Student(49, 27, "Steve"));  **return**list;  }    Predicate<Student> p1 = s ->s.stuName.startsWith("A");  Predicate<Student> p2 = s ->s.stuAge< 40;  Predicate<Student> p3 = s ->s.stuAge< 40 &&s.stuName.startsWith("P");  List<Student>list = Student.getStudents();  /\* allMatch() method returns true if all the elements of stream satisfy the  \* given predicate, else it returns false  \*/  /\* This will return false because all student names do not start with "A"  \*/  **boolean** b1 = list.stream().allMatch(p1);  System.out.println("list.stream().allMatch(p1): "+b1);  /\* This will return true because all students have age less than 40  \*/  **boolean** b2 = list.stream().allMatch(p2);  System.out.println("list.stream().allMatch(p2): "+b2);  /\* This will return false because all the students do not satisfy the predicate:  \* Age must be less than 40 and name starts with letter "P"  \*/  **boolean** b3 = list.stream().allMatch(p3);  System.out.println("list.stream().allMatch(p3): "+b3);  //17 Stream Collectors groupingBy and counting Example  List<String> names =  Arrays.asList("Jon", "Ajeet", "Steve",  "Ajeet", "Jon", "Ajeet");  Map<String, Long> map =  names.stream().collect(  Collectors.groupingBy(  Function.identity(), Collectors.counting()  )  );  System.out.println(map); |
|  |

**Programs**

**Program: Divide the letter and numbers in string**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  String digit="";  String letter="";  String str= "nagendra123";  **for**(**int** i =0; i<str.length();i++) {  **char** ch = str.charAt(i);  **if**(Character.*isDigit*(ch)) {  digit+=ch;  }**else** {  letter+=ch;  }  }  System.***out***.println(digit);  System.***out***.println(letter);  }  }  **o/p:**  123  nagendra |

**Program: count the space and character**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {  String str = "java is programing language";  **int** count = 0;  **for**(**int** i=0; i<str.length(); i++) {  **char** ch = str.charAt(i);  **if**(ch != ' ') count++;  }  System.***out***.println("no of characters: "+ count);  System.***out***.println("no of space: "+ (str.length() - count));  }  }  **o/p:**  no of characters: 24  no of space: 3 |

**Program: reverse method**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {    String str ="java";  String revStr = "";  **for**(**int** i=str.length()-1; i >= 0; i--) {  revStr+=str.charAt(i);  }  System.***out***.println(revStr);  }  } |

**Program: producer consumer design patteren**

|  |
| --- |
| **public** **class** ProducerConsumerDesignPattern {  **public** **static** **void** main(String args[]) {    BlockingQueue<String> queue = **new** LinkedBlockingDeque<>();  Thread producer = **new** Thread(**new** Producer(queue));  Thread consumer = **new** Thread(**new** Consumer(queue));    producer.start();  consumer.start();  }  }  **class** Producer **implements** Runnable {  **private** **final** BlockingQueue<String> blockingQueue;    **public** Producer(BlockingQueue<String> queue) {  **this**.blockingQueue = queue;  }  @Override  **public** **void** run() {  **try** {  **for** (**int** i = 0; i < 10; i++) {  String poducerMsg = "producer: " + i;  System.***out***.println("produced message : "+ poducerMsg);  blockingQueue.put(poducerMsg);  }  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  **class** Consumer **implements** Runnable {  **private** **final** BlockingQueue<String> blockingQueue;    **public** Consumer(BlockingQueue<String> queue) {  **this**.blockingQueue = queue;  }  @Override  **public** **void** run() {  **while**(**true**) {  **try** {  String consumeMsg = blockingQueue.take();  System.***out***.println("consumed message : "+ consumeMsg);  } **catch** (InterruptedException e) {  e.printStackTrace();  }  }  }  } |

**Program: producer consumer design pattern using wait and notify methods**

|  |
| --- |
|  |

**Program: find the duplicate elements**

|  |
| --- |
| **public** **static** **void** main(String args[]) **throws** IOException {    String str[] = {"ab","nag","java","scala","dig","ab","scala","java"};  **for**(**int** i=0; i<str.length; i++) {  **for**(**int** j=i+1; j<str.length; j++) {  **if**(str[i].equals(str[j])) {  System.***out***.println(str[i]);  }  }  }    }  } |