**Java 8-2 Functional Programming**

1. Write a lambda or method reference for each of the following tasks:
   1. Write a lambda expression that receives two double parameters a and b and returns their product.
   2. Write a no-argument lambda that implicitly returns the string "Here is a lambda String".
   3. Write a constructor reference for class ArrayList.
   4. Reimplement the following statement using a lambda as the event handler:

**1** slider.valueProperty().addListener(

**2** new ChangeListener<Number>() {

**3** @Override

**4** public void changed(ObservableValue<? extends Number> ov,

**5** Number oldValue, Number newValue) {

**6** System.out.printf("The slider's new value is %s%n", newValue);

**7** }

**8** }

**9** );

1. What’s wrong with the following stream pipeline assuming list is a correctly defined List<Integer>?

**1** list.stream()

**2** .filter(value -> value % 2 != 0)

**3** .sum()

1. Assuming that list is a List<Integer>, explain in detail the stream pipeline:

**1** list.stream()

**2** .filter(value -> value % 2 != 0)

**3** .reduce(0, Integer::sum)

1. Assuming that random is a SecureRandom object, explain in detail the stream pipeline:

**1** random.ints(1000000, 1, 3)

**2** .boxed()

**3** .collect(Collectors.groupingBy(Function.identity(),

**4** Collectors.counting()))

**5** .forEach((side, frequency) ->

**6** System.out.printf("%-6d%d%n", side, frequency));

1. .
2. ***(Manipulating a***Stream<Invoice>***)*** Create the class Invoice described with the information below and then create an array of Invoice objects. Use the sample data shown in [Fig. 1](https://learning.oreilly.com/library/view/java-how-to/9780134751962/xhtml/fileP700101433900000000000000000A6A3.xhtml#P700101433900000000000000000A75F). Class Invoice includes four instance variables—a partNumber (type String), a part-Description (type String), a quantity of the item being purchased (type int) and a pricePerItem (type double) and corresponding *get* methods. Perform the following queries on the array of Invoice objects and display the results:
   1. Use streams to sort the Invoice objects by partDescription, then display the results.
   2. Use streams to sort the Invoice objects by pricePerItem, then display the results.
   3. Use streams to map each Invoice to its partDescription and quantity, sort the results by quantity, then display the results.
   4. Use streams to map each Invoice to its partDescription and the value of the Invoice (i.e., quantity \* pricePerItem). Order the results by Invoice value.
   5. Modify Part (d) to select the Invoice values in the range $200 to $500.
   6. Find any one Invoice in which the partDescription contains the word "saw".

Fig. 1

| **Part number** | **Part description** | **Quantity** | **Price** |
| --- | --- | --- | --- |
| 83 | Electric sander | 7 | 57.98 |
| 24 | Power saw | 18 | 99.99 |
| 7 | Sledge hammer | 11 | 21.50 |
| 77 | Hammer | 76 | 11.99 |
| 39 | Lawn mower | 3 | 79.50 |
| 68 | Screwdriver | 106 | 6.99 |
| 56 | Jig saw | 21 | 11.00 |
| 3 | Wrench | 34 | 7.50 |

Sample data for [Exercise](https://learning.oreilly.com/library/view/java-how-to/9780134751962/xhtml/fileP700101433900000000000000000A6A3.xhtml#P700101433900000000000000000A730)

1. ***(Duplicate Word Removal)*** Write a program that inputs a sentence from the user (assume no punctuation), then determines and displays the unique words in alphabetical order. Treat uppercase and lowercase letters the same.
2. ***(Sorting Letters and Removing Duplicates)*** Write a program that inserts 30 random letters into a List<Character>. Perform the following operations and display your results:
   1. Sort the List in ascending order.
   2. Sort the List in descending order.
   3. Display the List in ascending order with duplicates removed.
3. ***(Mapping Integer Grades to Letter Grades)*** Create a program that reads integer grades and stores them in an ArrayList, then use stream processing to display each grade’s letter equivalent (A, B, C, D or F).