PPL Practice

1. Given a list of numbers, how would you return a list of the square of each number? For example, given [1, 2, 3, 4, 5] you should return [1, 4, 9, 16, 25].

Ans.

```
List <Integer> numbers = Arrays.asList (1, 2, 3, 4, 5);
List <Integer> squares = numbers     .stream ()
                                     .map (n -> n * n)
                                     .collect (toList ())
```

2. Given two lists of numbers, how would you return all pairs of numbers? For example, given a list [1, 2, 3] and a list [3, 4] you should return [(1, 3), (1, 4), (2, 3), (2, 4), (3, 3), (3, 4)]. For simplicity, you can represent a pair as an array with two elements.

Ans.

```
List <Integer> numbers1 = Arrays.asList (1, 2, 3);
List <Integer> numbers2 = Arrays.asList (3, 4);
List <int[]> pairs = numbers1        .stream ()
                                     .flatMap (i -> numbers2        .stream ()
                                                                    .map (j -> new int[] {i, j})
                                                 )
                                     .collect (toList ());
```

3. How would you extend the previous example to return only pairs whose sum is divisible by 3? For example, (2, 4) and (3, 3) are valid.

Ans.

```
List <Integer> numbers1 = Arrays.asList (1, 2, 3);
List <Integer> numbers2 = Arrays.asList (3, 4);
List <int[]> pairs = numbers1        .stream ()
                                     .flatMap (i -> numbers2        .stream ()
                                                                    .map (j -> new int[] {i, j})
                                                 )
                                     .filter (p -> (p[0] + p[1]) % 3 == 0)
                                     .collect (toList ());
```

Alternative:

```
List <Integer> numbers1 = Arrays.asList (1, 2, 3);
List <Integer> numbers2 = Arrays.asList (3, 4);
List <int[]> pairs = numbers1        .stream ()
                                     .flatMap (i -> numbers2        .stream ()
                                                                    .filter (j -> (i + j) % 3 == 0)
                                                                    .map (j -> new int[] {i, j})
                                                 )
                                     .collect (toList ());
```

Listing 5.1 Find all transactions in 2011 and sort by value (small to high)

Ans.

```
List <Transaction> result = transactions      .stream ()
                                              .filter (t -> t .getYear () == 2011)
                                              .sorted (comparing (Transaction :: getValue))
                                              .collect (toList ());
```

Listing 5.2 What are all the unique cities where the traders work?

Ans.

```
List <String> result = transactions    .stream ()
                                       .map (t -> t .getTrader () .getCity ())
                                       .distinct ()
                                       .collect (toList ());
```

Listing 5.3 Find all traders from Cambridge and sort them by name

Ans.

```
List <Traders> result = transactions  .stream ()
                                      .map (Transaction :: getTrader)
                                      .distinct ()
                                      .filter (trader -> trader .getCity () == "Cambridge")
                                      .sort (comparing(Trader :: getName))
                                      .collect (toList ());
```

Listing 5.4 Return a string of all traders' names sorted alphabetically

Ans.

```
String result = transactions    .stream ()
                                .map (transaction -> transaction .getTrader () .getName ())
                                .distinct ()
                                .sorted ()
                                .reduce ("", (a, b) -> a + b);
```

Listing 5.5 Are any traders based in Milan?

Ans.

```
bool result = transactions      .stream ()
                                .map (Transaction :: getTrader)
                                .anyMatch (trader -> trader .getCity () .equals ("Milan"));
```

Listing 5.6 Print all transactions' values from the traders living in Cambridge

Ans.

```
transactions    .stream ()
                .filter (t -> "Cambridge" .equals (t .getTrader () .getCity ()))
                .map (Transaction :: getValue)
                .forEach (System.out :: println)
```

Listing 5.7 What's the highest value of all the transactions?

Ans.

```
Optional <Integer> result = transactions      .stream ()
                                              .map (Transactions :: getValue)
                                              .reduce (Integer :: max)
```

Listing 5.8 Find the transaction with the smallest value

Ans.

```
Optional <Transaction> result = transactions      .stream ()
                                                  .reduce ((t1, t2) -> t1 .getValue () < t2 .getValue
() ? t1 : t2);
```

Alternative

```
Optional <Transaction> result = transactions      .stream ()
                                                  .min (comparing (Transaction :: getValue));
```

1. Traders execute transactions. The two class structures are as follows. A collection of transactions is given. 3+2+3 +3+4= 15

Trader

```
private final String name;
private final String city;

public Trader(String n, String c);

public String getName();

public String getCity();

public String toString();
```

Transactions

```
private final Trader trader;
private final int year;
private final int value;
public Transaction(Trader trader, int year, int value);
public Trader getTrader();
public int getYear();
public int getValue();
public String toString();
```

You're asked by your manager to find answers to the following queries.
(a) Group traders from different cities who performed high-value transactions.
(b) Find the transaction with the highest value for each year.
(c) Find the average transaction value for each year.
(d) What is the minimum transaction made by any trader from Mumbai?
(e) From a list of words, extract the list of words that ends with a number and then print the duplicates among them.

Ans.

(a)

```
int VAL = 1000; //criterion for high value transaction
Map <String, List <Trader>> result = transactions   .stream ()
                                                    .filter (t -> t .getValue () > VAL)
                                                    .map (Transactions :: getTrader)
                                                    .collect (groupingBy (Trader :: getCity));
```

(b)

```
Map <Integer, Transaction> result =
transactions    .stream ()
                .collect (groupingBy (Transactions :: getYear),
                        collectingAndThen (
                                maxBy (comparingInt (Transactions :: getValue)),
                                Optional :: get
                        )
                );
```

(c)

```
Map <Integer, Double> result =
transactions    .stream ()
                .collect (groupingBy (Transactions :: getYear),
                        averagingInt (Transactions :: getValue)
                );
```

(d)

```
Optional <Transaction> =
transactions    .stream ()
                .filter (t -> t .getTrader () .getCity () == “Mumbai”)
                .collect (maxBy (comparingInt (Transactions :: getValue)));
```

(e)

```
List <String> words = Arrays .asList (“hello1”, “abcd”, “hello1”, “matter56”);
Set <String> uniqueWords = new HashSet <> ();

words  .stream ()
        .filter (w -> Character .isDigit (w .charAt (w .length () - 1)))
        .filter (w -> ! uniqueWords .add (w))
        .forEach (System.out :: println);
```

2. *Artist class denotes an* individual or group who creates music having the following fields. +2=15

- *name*: The name of the artist (individual or group name)
- *members*: A set of other artists who comprise this group - this field might be empty
- *origin*: The primary location of origin of the group (e.g., "Kolkata").

(a) Convert the following code sample from using external iteration to internal iteration:

```
int totalMembers = 0;
for (Artist artist : artists)' {
if(artist.getOrigin()=="Kolkata") {
   Stream<Artist> members = artist.getMembers();
   totalMembers += members.count();}}
```

(b)Find the bands with most members using lambda expressions and/or streams API.

(c) max(), average() and count() using only reduce and lambda expressions. Can it handle empty list?

(d) Is this lambda expression side-effect free or does it mutate state?

```
x→x+1
```

Ans.

(a)

```
int totalMembers = artists      .stream ()
                                .filter (artist -> artist .getOrigin () == "Kolkata")
                                .flatMap (Artist :: getMembers)
                                .count ()
```

(b)

```
Optional <Artist> bandWithMostMembers = artists .stream ()
                                              .collect (maxBy (comparingInt (
                                                    artist -> artist .getMembers () .count ();
                                              )));
```

(c) Yes, max(), average() and count() can handle empty lists because

- max is a Java primitive stream such as **IntStream method** that returns an **OptionalInt** object. For empty streams, the OptionalInt object is empty.
- average() is also a primitive stream method. It returns 0 for an empty stream.
- count() also returns 0 in case the stream is empty.

(d)

The given lambda expression is **side-effect free** as it takes in an argument, and returns a different copy of the argument adding 1 to it. So the original argument remains unchanged.

3.	Given a text file, answer (a) and (b) (a) Identify and list the distinct letters; (b) Group it's words into three categories depending on word length-2-letter words, 3-letter words and more than 3 letter words. (c) Create a collection of *n* Tribonacci numbers using java streams API. The number series looks like `0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81,...`	4+6+5 =15

Ans.

(a)

```
try (Stream<String> lines = Files.lines(Paths.get("data.txt"), Charset.defaultCharset())) {
        List <String> distinctLetters = lines  .flatMap(line -> Arrays.stream(line.split("")));
                                               .distinct()
                                               .collect(toList());
} catch (IOException e) {

}
```

(b)

```
public enum WordLength { TWO, THREE, MORE }
try (Stream<String> lines = Files.lines(Path.get("data.txt"), Charset.defaultCharset())) {
        Map <WordLength, List<String>> result=lines       .flatMap(line ->
Arrays.toStream(line.split(" ")))
                                                          .collect (groupingBy (word -> {
                                                          if (word.length() == 2) return
WordLength.TWO;
                                               else if (word.length() == 3) return
WordLength.THREE;
                                                          else return WordLength.MORE;
                                                          }))
} catch (IOException e) {
}
```

(c)

```
List <Integer> tribonacci = Streams  .iterate(new int [] {0, 0, 1}, t -> new int[] {t[1], t[2], t[0] + t[1]
+ t[2]})
                                     .limit(n)
                                     .map(t -> t[0])
                                     .collect(toList());
```

4.	(a) Write code in Prolog to implement (i) maximum of 3 numbers, (ii) generating a list by replicating a number *n*, *x* times, (iii) finding the last element of a list, (iv) prepending an element to a list. (b) Write a program in Prolog that takes a list as input and sorts the list. Show how it works. (c) Given the following Prolog clauses: `ancestor(X, X).` `ancestor(X,Y) :- ancestor(Z, Y), parent(X, Z).` `parent(amy, bob).` Explain Prolog's response to the query `ancestor(amy, Y)` using a search tree of subgoals.	(3x3+1) +6+4= 20

Ans.
(a)

```
max(A, B, C, A) :- A >= B, A >= C, !.
max(A, B, C, B) :- B >= A, B >= C, !.
max(A, B, C, C).
```

(b)

```
insertionSort([], []).
insertionSort([X | T], L) :- insertionSort(T, U), insert(X, U, L).

insert(X, [], [X]).
insert(X, [H | T], [X, H | T]) :- H >= X, !.
insert(X, [H | T], [H | U]) :- insert(X, T, U).
```

(c)

(b)
```
if(roll_no%2!=0)
        return roll_no;
    else
        return "Even Number";
```

Represent above construct in lambda calculus. Derive any predicates, constructs and data types that you need. Do not use Y-combinator.

λroll. IF_THEN_ELSE (IS_ODD roll) roll EVENNUM

IF_THEN_ELSE = λcond.λthen.λelse. cond then else
IS_ODD = λn.n NOT FALSE
NOT = λp.p FALSE TRUE
TRUE = λxy.x
FALSE = λxy.y
EVENNUM = Lambda calculus predicate representing the string “Even Number”

(b)
```
int arr[]={1,2,3};
    if(arr[2]==0)
          return arr[2]+1;
    else
          return arr[2];
```

Represent above construct in lambda calculus. Derive any predicates, constructs and data types that you need. No need to define Church numerals, *pair*, '*true*', '*false*' and *if_then_else* construct.

Ans.

ARR = PAIR ONE (PAIR TWO (PAIR THREE (PAIR NIL)))

IF_THEN_ELSE (IS_ZERO (FIRST (TWO GET_TAIL ARR)))
(SUCC (FIRST (TWO GET_TAIL ARR)))
(FIRST (TWO GET_TAIL ARR))

Definitions:

PAIR = λxyf.fxy
ONE = λgs.gs
TWO = λgs.g(gs)
THREE = λgs.g(g(gs)))
NIL = λx.TRUE
IF_THEN_ELSE = λcond.λthen.λelse. cond then else
GET_TAIL = λp.p FALSE
SUCC = λnfx.f(nfx)
FALSE = λxy.y

8.	(a) Reduce the following lambda expression using both normal order and applicative order reduction. (λx . x x x) (λx . x x x) (b) ```while(i<10) { product*=a[i]; i++; } Average=product/10;``` Represent above construct in lambda calculus. Here, a[i] represents the collection of natural numbers. You can assume that *Church numerals, predecessor, and multiplication* predicates are in place. Justify your answer.	4+11= 15

Ans.

Y = λf.(λx.f(xx))(λx.f(xx))
T = λf.λi. IF_THEN_ELSE (LEQ i NINE)
(MULT (FIRST (i GET_TAIL A)) (f (SUCC i)))
ONE
AVERAGE = DIV (YT) TEN

Definitions:

IF_THEN_ELSE = λcond.λthen.λelse. cond then else
LEQ = λmn. IS_ZERO (SUB m n)
IS_ZERO = λn.n (λx.FALSE) TRUE
SUB = λmn. n PRED m
PRED = λn. FIRST(n Φ (PAIR 0 0))
Φ = λp. PAIR (SECOND p) (SUCC (SECOND p))
PAIR = λxyf.fxy
FIRST = λp. p TRUE
GET_TAIL = SECOND = λp. p FALSE
SUCC = λnfx.f(nfx)
ONE = λgs.gs
NINE = λgs.g(g(g(g(g(g(g(g(gs)))))))))
TEN = λgs.g(g(g(g(g(g(g(g(g(gs)))))))))
MULT = λmnfx.m(nf)x

```
T2 = λfab. IF_THEN_ELSE          (LT a b)
                                 ZERO
                                 (SUCC (f   (SUB a b)    b))
```

DIV = Y T2

LT = AND (LEQ a b) (NOT (EQUAL a b))
AND = λpq.pqp
NOT = λp. p FALSE TRUE
EQUAL = λmn. AND (LEQ m n) (LEQ n m)