Stream

■■ **E19.4** Write a program that reads all words from a file and, using a `Stream<String>`, prints all distinct words with at most four letters (in some order).

```
public static void printWords(String path){
    String[] words = ReadFile(path);
    Arrays.stream(words)
            .filter(w -> w.length() <= 4)
            .sorted()
            .forEach(System.out::println);
}
```

■■■ **E19.5** Write a method

```
public static <T> String toString(Stream<T> stream, int n)
```

that turns a `Stream<T>` into a comma-separated list of its first `n` elements.

```
public static <T> String toString(Stream<T> stream, int n) {
    Collector<T, StringJoiner, String> stringCollector =
            Collector.of(
                    () -> new StringJoiner(","),   // supplier
                    (j, s) -> j.add(String.valueOf(s)),     // accumulator
                    StringJoiner::merge,                    // combiner
                    StringJoiner::toString);                // finisher
    return stream.limit(n).collect(stringCollector);
}
```

■■ **E19.7** Write a lambda expression for a function that turns a string into a string made of the first letter, three periods, and the last letter, such as `"W...d"`. (Assume the string has at least two letters.) Then write a program that reads words into a stream, applies the lambda expression to each element, and prints the result. Filter out any words with fewer than two letters.

```
    public static String doMask(String originalStr) {
        return stringFilter(originalStr, (str, index) -> {
            if(str.length() <= 2){
                return false;
            }else return index == 0 || index == str.length() - 1;
        });
    }


    @FunctionalInterface
    public interface myFilter<T, R> {
        boolean filter(T originStr, R index);
    }


    public static String stringFilter(String originalStr, myFilter<String,
Integer> f) {
```

```
        StringBuilder filterStr = new StringBuilder();
        for (int i = 0; i < originalStr.length(); i++) {
            if (f.filter(originalStr, i)) {
                filterStr.append(originalStr.charAt(i));
            }
        }
        if(filterStr.length() == 2){
            return filterStr.charAt(0) + "..." + filterStr.charAt(1);
        }else{
            return "Word's length less than 2, filter!";
        }
    }
```

■■ **E19.10** Write a method

```
public static Optional<Integer> smallestProperDivisor(int n)
```

that returns the smallest proper divisor of `n` or, if `n` is a prime number, a value indicating that no result is present.

```
public static Optional<Integer> smallestProperDivisor(int n) {
    Optional<Integer> opt = Optional.empty();
    for (int i = n - 1; i >= 2; i--) {
        opt = Optional.of(i).filter(m -> n % m == 0);
    }
    return opt;
}
```

■■ **E19.13** Read all words from a file and print the one with the maximum number of vowels. Use a `Stream<String>` and the `max` method. Extra credit if you define the comparator with the `Comparator.comparing` method described in Special Topic 19.4.

```
public static void findMaxVowels(String filePath) {
    String[] words = ReadFile(filePath);
    Arrays.stream(words)
            .max(Comparator.comparingInt(E19_13::countVowels))
            .ifPresent(System.out::println);
}

public static int countVowels(String str){
    return str.chars()
    .filter(c -> c == 'a'|| c == 'e' || c == 'i' || c == 'o' || c == 'u')
    .sum();
}
```

■■ **E19.14** Read all words from a file into an `ArrayList<String>`, then turn it into a parallel stream. Use the dictionary file `words.txt` provided with the book's companion code. Use filters and the `findAny` method to find any palindrome that has at least five letters, then print the word. What happens when you run the program multiple times?

```java
public static void palindrome(String filePath) {
    ArrayList<String> words = ReadFile(filePath);
    words.parallelStream()
            .filter(s -> s.length() >= 5)
            .filter(s -> {
                String strReverse = new
StringBuffer(s).reverse().toString();
                return s.equals(strReverse);
            })
            .findAny()
            .ifPresent(System.out::println);
}
```

E19.16 Read all words in a file and group them by the first letter (in lowercase). Print the average word length for each initial letter. Use `collect` and `Collectors.groupingBy`.

```java
public static void averageWordLengthForEachInitialLetter(String filePath) {
    ArrayList<String> words = ReadFile(filePath);
    words.stream()
            .collect(Collectors.groupingBy(s ->
String.valueOf(s.charAt(0)).toLowerCase(),
Collectors.averagingInt(String::length)))
            .forEach((c, num) -> System.out.println(num));
}
```

E19.17 Assume that a `BankAccount` class has methods for yielding the account owner's name and the current balance. Write a function that, given a list of bank accounts, produces a map that associates owner names with the total balance in all their accounts. Use `collect` and `Collectors.groupingBy`.

```java
class BankAccount {
    private String name;

    public String getName() {
        return name;
    }

    public int getBalance() {
        return balance;
    }

    private int balance;

    public BankAccount(String name, int balance) {
        this.name = name;
        this.balance = balance;
    }

    public static Map<String, Integer> getAllBalance(List<BankAccount>
accountList) {
        return accountList
                .stream()
                .collect(
                        Collectors.groupingBy(BankAccount::getName,
                        Collectors.summingInt(BankAccount::getBalance)
                ));
```

```
    }

}
```

E19.18 Write a program that reads a `Stream<Country>` from a file that contains country names and numbers for the population and area. Print the most densely populated country.

```
class Country {

    private String name;
    private int population;
    private double area;

    public Country(String name, int population, int area) {
        this.name = name;
        this.population = population;
        this.area = area;
    }

    public String getName() {
        return name;
    }

    public int getPopulation() {
        return population;
    }

    public double getArea() {
        return area;
    }

    public static void mostDenselyPopulatedCountry(Stream<Country>
countryStream) {
        countryStream
                .max((c1, c2) -> (int) (c1.getPopulation() / c1.getArea() -
c2.getPopulation() / c2.getArea()))
                .ifPresent(c -> System.out.println(c.getName()));
    }
}

```

P19.2 Write a program that generates an infinite stream of integers that are perfect squares and then displays the first *n* of them that are palindromes (that is, their decimal representation equals its reverse). Extra credit if you use `BigInteger` so that you can find solutions of arbitrary length.

```
public static void generate(long n){
    Stream.iterate(BigInteger.ONE, i -> i.add(BigInteger.ONE))
            .map(i -> i.multiply(i))
            .filter(i-> i.toString().equals(new
StringBuilder(i.toString()).reverse().toString()))
            .limit(n)
            .forEach(System.out::println);
}
```

■■■ **P19.10** Write a program to determine how many actors there are in the data set in Worked Example 19.2. Note that many actors are in multiple movies. The challenge in this assignment is that each movie has a list of actors, not a single actor, and there is no ready-made collector to form the union of these lists. However, there is another `collect` method that has three parameters:

- A function to generate an instance of the target
- A function to add an element to a target
- A function to merge two targets into one

For example,

```
stream.collect(
   () -> 0,
   (t, e) -> t + e,
   (t, u) -> t + u)
```

computes the sum of elements in a `Stream<Integer>`. Note that the last function is only needed for parallel streams.

Define methods for generating a set, adding a list of actors into one, and for combining two sets.

```
1  public static HashSet<String> getActorSet(String filePath) {
2      ArrayList<String> test = getActorList(filePath);
3      HashSet<String> nameSet = test.stream()
4              .collect(() -> new HashSet<>(),
5                      (set, name) -> set.add(name),
6                      (set1, set2) -> set1.addAll(set2));
7      return nameSet;
8  }
```

■■■ **P19.11** Write a program to determine the 100 actors with the most movies, and the number of movies in which they appear. For each movie, produce a map whose keys are the actors, all with value 1. Merge those maps as in Exercise P19.10. Then extract the top 100 actors from a stream of actors.

```
1  public static void getTopHundredActors(String filePath) {
2      Map<String, Integer> actorMovieMap = getMovieMap(filePath);
3      ArrayList<String> actor = new ArrayList<>(getActorSet(filePath));
4      actor.stream()
5              .sorted((a1, a2) -> actorMovieMap.get(a2) -
   actorMovieMap.get(a1))
6              .limit(100)
7              .forEach(System.out::println);
8  }
```

Regex

5.1.26 *Challenging REs.* Construct an RE that specifies each of the following languages of binary strings over the alphabet `{0, 1}`.

a. All binary strings except `11` or `111`

b. Binary strings with `1` in every odd-number bit position

c. Binary strings with at least two `0`s and at most one `1`

d. Binary strings with no two consecutive `1`s

```
1  String REa = "(?!^11$|^111$).[01]*";
2  String REb = "(1[01])+|(1[01])*1";
3  String REc = "(?=^.*0.*0)(?!^.*1.*1).[01]*";
4  String REd = "(?!.*11.*).[01]*";
```

Basic

```java
String str = "";
System.out.println(str.length());
int[] arr = new int[4];
System.out.println(arr.length);
ArrayList<Integer> arrayList = new ArrayList<>();
System.out.println(arrayList.size());
```