1) D .
This program terminates normally without printing any output in the console
Since there is no terminal operation provided (such as count , forEach , reduce, or collect), this pipeline is not evaluated and hence the peek does not print any output to the console.

2) B .
This program prints: " aaaeaa“ Because the Consonants::removeVowels returns true when there is a vowel passed, only those characters are retained in the stream by the filter method. Hence, this program prints “aaaeaa”.

3) F .
This program prints: true The predicate str -> str.length() > 5 returns false for all the elements because the length of each string is 2. Hence, the filter() method results in an empty stream and the peek() method does not print anything.
The allMatch() method returns true for an empty stream and does not evaluate the given predicate. Hence this program prints true

4) A .
Compiler error: Cannot find symbol “sum” in interface Stream<Integer> Data and calculation methods such as sum() and average() are not available in the Stream<T> interface; they are available only in the primitive type versions IntStream, LongStream, and DoubleStream. To create an IntStream , one solution is to use mapToInt() method instead of map() method in this program. If mapToInt() were used, this program would compile without errors, and when executed, it will print 6 to the console.

5) D.
is the correct answer as this program compiles without errors, and when run, it prints 100 in console. Why other options are wrong: A. An interface can be defined inside a class B. The signature of the equals method matches that of the equal method in Object class; hence it is not counted as an abstract method in the functional interface C. It is acceptable to omit the parameter type when there is only one parameter and the parameter and return type are inferred from the LambdaFunction abstract method declaration int apply(int j)

6)
C . This program prints: mo miny meeny eeny This is a proper definition of a lambda expression. Since the second argument of Collections.sort() method takes the functional interface Comparator and a matching lambda expression is passed in this code. Note that second argument is compared with the first argument in the lambda expression (str1, str2) -> str2.compareTo(str1) . For this reason, the comparison is performed in descending order.

7)
D. This code segment does not print anything on the console The limit() method is an intermediate operation and not a terminal operation. Since there is no terminal operation in

this code segment, elements are not processed in the stream and hence it does not print anything on the console.

8) C.
The program prints the following: Brazil China India Russia. For the sort() method, null value is passed as the second argument, which indicates that the elements' "natural ordering" should be used. In this case, natural ordering for Strings results in the strings sorted in ascending order. Note that passing null to the sort() method does not result in a NullPointerException.

9) C
This program prints: 11 This program compiles without any errors.
The variable words point to a stream of Strings.
The call mapToInt(String::length) results in a stream of Integers with the length of the strings.
One of the overloaded versions of reduce() method takes two arguments: T reduce(T identity, BinaryOperator<T> accumulator); The first argument is the identity value, which is given as the value 0 here. The second operand is a BinaryOperator match for the lambda expression (len1, len2) -> len1 + len2. The reduce() method thus adds the length of all the three strings in the stream, which results in the value 11.

10) B
This program prints: [1, 4, 9, 16, 25] The replaceAll() method (added in Java 8 to the List interface) takes an UnaryOperator as the argument. In this case, the unary operator squares the integer values. Hence, the program prints [1, 4, 9, 16, 25].