**Section 17.1 Introduction**

• Prior to Java SE 8, Java supported three programming paradigms—procedural programming, object-oriented programming and generic programming. Java SE 8 adds functional programming.

• The new language and library capabilities that support functional programming were added to Java as part of Project Lambda.

#### Section 17.2 Functional Programming Technologies Overview

• Prior to functional programming, you typically determined what you wanted to accomplish, then specified the precise steps to accomplish that task.

• Using a loop to iterate over a collection of elements is known as external iteration (p. [731](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec1_html#page_731)) and requires accessing the elements sequentially. Such iteration also requires mutable variables.

• In functional programming (p. [732](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec2_html#page_732)), you specify what you want to accomplish in a task, but not how to accomplish it.

• Letting the library determine how to iterate over a collection of elements is known as internal iteration (p.[732](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec2_html#page_732)). Internal iteration is easier to parallelize.

• Functional programming focuses on immutability (p.[732](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec2_html#page_732))—not modifying the data source being processed or any other program state.

#### Section 17.2.1 Functional Interfaces

• Functional interfaces are also known as single abstract method (SAM) interfaces.

• Package java.util.function contains six basic functional interfaces BinaryOperator, Consumer,Function, Predicate, Supplier and UnaryOperator.

• There are many specialized versions of the six basic functional interfaces for use with int, long and doubleprimitive values. There are also generic customizations of Consumer, Function and Predicate for binary operations—that is, methods that take two arguments.

#### Section 17.2.2 Lambda Expressions

• A lambda expression (p. [733](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec3_html#page_733)) represents an anonymous method—a shorthand notation for implementing a functional interface.

• The type of a lambda is the type of the functional interface that the lambda implements.

• Lambda expressions can be used anywhere functional interfaces are expected.

• A lambda consists of a parameter list followed by the arrow token (->, p. [733](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec3_html#page_733)) and a body, as in:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec38_html#p765pro00a)

(*parameterList*) -> {*statements*}

For example, the following lambda receives two ints and returns their sum:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p765pro00aa)

(int x, int y) -> {return x + y;}

This lambda’s body is a statement block that may contain one or more statements enclosed in curly braces.

• A lambda’s parameter types may be omitted, as in:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0765pro01a)

(x, y) -> {return x + y;}

in which case, the parameter and return types are determined by the lambda’s context.

• A lambda with a one-expression body can be written as:

(x, y) -> x + y

In this case, the expression’s value is implicitly returned.

• When the parameter list contains only one parameter, the parentheses may be omitted, as in:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0766pro01a)

value -> System.out.printf("%d ", value)

• A lambda with an empty parameter list is defined with() to the left of the arrow token (->), as in:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0766pro02a)

() -> System.out.println("Welcome to lambdas!")

• There are also specialized shorthand forms of lambdas that are known as method references.

#### Section 17.2.3 Streams

• Streams (p. [734](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec4_html#page_734)) are objects that implement interfaceStream (from the package java.util.stream) and enable you to perform functional programming tasks. There are also specialized stream interfaces for processing int, long or double values.

• Streams move elements through a sequence of processing steps—known as a stream pipeline—that begins with a data source, performs various intermediate operations on the data source’s elements and ends with a terminal operation. A stream pipeline is formed by chaining method calls.

• Unlike collections, streams do not have their own storage—once a stream is processed, it cannot be reused, because it does not maintain a copy of the original data source.

• An intermediate operation (p. [734](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec4_html#page_734)) specifies tasks to perform on the stream’s elements and always results in a new stream.

• Intermediate operations are lazy (p. [734](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec4_html#page_734))—they aren’t performed until a terminal operation is invoked. This allows library developers to optimize stream-processing performance.

• A terminal operation (p. [734](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec4_html#page_734)) initiates processing of a stream pipeline’s intermediate operations and produces a result. Terminal operations are eager (p. [734](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec4_html#page_734))—they perform the requested operation when they are called.

#### Section 17.3 IntStream Operations

• An IntStream (package java.util.stream) is a specialized stream for manipulating int values.

#### Section 17.3.1 Creating an IntStream and Displaying Its Values with the forEach Terminal Operation

• IntStream static method of (p. [738](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec6_html#page_738)) receives an intarray as an argument and returns an IntStream for processing the array’s values.

• IntStream method forEach (a terminal operation; p.[738](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec6_html#page_738)) receives as its argument an object that implements the IntConsumer functional interface (package java.util.function). This interface’s acceptmethod receives one int value and performs a task with it.

• The Java compiler can infer the types of a lambda’s parameters and the type returned by a lambda from the context in which the lambda is used. This is determined by the lambda’s target type (p. [738](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec6_html#page_738))—the functional interface type that is expected where the lambda appears in the code.

• Lambdas may use final local variables or effectivelyfinal (p. [738](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec6_html#page_738)) local variables.

• A lambda that refers to a local variable in the enclosing lexical scope is known as a capturing lambda.

• A lambda can use the outer class’s this reference without qualifying it with the outer class’s name.

• The parameter names and variable names that you use in lambdas cannot be the same as any other local variables in the lambda’s lexical scope; otherwise, a compilation error occurs.

#### Section 17.3.2 Terminal Operations count, min,max, sum and average

• Class IntStream provides terminal operations for common stream reductions—count (p. [739](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec6_html#page_739)) returns the number of elements, min returns the smallest int, maxreturns the largest int, sum returns the sum of all theints and average returns an OptionalDouble (packagejava.util) containing the average of the ints as a value of type double.

• Class OptionalDouble’s getAsDouble method returns the double in the object or throws aNoSuchElementException. To prevent this exception, you can call method orElse, which returns theOptionalDouble’s value if there is one, or the value you pass to orElse, otherwise.

• IntStream method summaryStatistics performs thecount, min, max, sum and average operations in one pass of an IntStream’s elements and returns the results as an IntSummaryStatistics object (packagejava.util).

#### Section 17.3.3 Terminal Operation reduce

• You can define your own reductions for an IntStreamby calling its reduce method. The first argument is a value that helps you begin the reduction operation and the second argument is an object that implements theIntBinaryOperator (p. [740](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec8_html#page_740)) functional interface.

• Method reduce’s first argument is formally called an identity value (p. [740](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec8_html#page_740))—a value that, when combined with any stream element using the IntBinaryOperatorproduces that element’s original value.

#### Section 17.3.4 Intermediate Operations: Filtering and Sorting IntStream Values

• You filter elements to produce a stream of intermediate results that match a predicate. IntStream methodfilter (p. [741](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec8_html#page_741)) receives an object that implements theIntPredicate functional interface (packagejava.util.function).

• IntStream method sorted (a lazy operation) orders the elements of the stream into ascending order (by default). All prior intermediate operations in the stream pipeline must be complete so that method sortedknows which elements to sort.

• Method filter a stateless intermediate operation—it does not require any information about other elements in the stream in order to test whether the current element satisfies the predicate.

• Method sorted is a stateful intermediate operation that requires information about all of the other elements in the stream in order to sort them.

• Interface IntPredicate’s default method and (p. [741](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec8_html#page_741)) performs a logical AND operation with short-circuit evaluation between the IntPredicate on which it’s called and its IntPredicate argument.

• Interface IntPredicate’s default method negate (p.[741](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec8_html#page_741)) reverses the boolean value of the IntPredicateon which it’s called.

• Interface IntPredicate default method or (p. [741](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec8_html#page_741)) performs a logical OR operation with short-circuit evaluation between the IntPredicate on which it’s called and its IntPredicate argument.

• You can use the interface IntPredicate defaultmethods to compose more complex conditions.

#### Section 17.3.5 Intermediate Operation: Mapping

• Mapping is an intermediate operation that transforms a stream’s elements to new values and produces a stream containing the resulting (possibly different type) elements.

• IntStream method map (a stateless intermediate operation; p. [742](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec9_html#page_742)) receives an object that implements the IntUnaryOperator functional interface (packagejava.util.function).

#### Section 17.3.6 Creating Streams of ints withIntStream Methods range and rangeClosed

• IntStream methods range (p. [743](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec10_html#page_743)) and rangeClosedeach produce an ordered sequence of int values. Both methods take two int arguments representing the range of values. Method range produces a sequence of values from its first argument up to, but not including, its second argument. Method rangeClosed produces a sequence of values including both of its arguments.

#### Section 17.4 Stream<Integer> Manipulations

• Class Array’s stream method is used to create aStream from an array of objects.

#### Section 17.4.1 Creating a Stream<Integer>

• Interface Stream (package java.util.stream; p. [744](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec4_html#page_744)) is a generic interface for performing stream operations on objects. The types of objects that are processed are determined by the Stream’s source.

• Class Arrays provides overloaded stream methods for creating IntStreams, LongStreams and DoubleStreams from int, long and double arrays or from ranges of elements in the arrays.

#### Section 17.4.2 Sorting a Stream and Collecting the Results

• Stream method sorted (p. [745](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec13_html#page_745)) sorts a stream’s elements into ascending order by default.

• To create a collection containing a stream pipeline’s results, you can use Stream method collect (a terminal operation). As the stream pipeline is processed, method collect performs a mutable reduction operation that places the results into an object, such as a List, Map or Set.

• Method collect with one argument receives an object that implements interface Collector (packagejava.util.stream), which specifies how to perform the mutable reduction.

• Class Collectors (package java.util.stream) provides static methods that return predefinedCollector implementations.

• Collectors method toList transforms a Stream<T>into a List<T> collection.

#### Section 17.4.3 Filtering a Stream and Storing the Results for Later Use

• Stream method filter (p. [745](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec13_html#page_745)) receives a Predicateand results in a stream of objects that match thePredicate. Predicate method test returns a booleanindicating whether the argument satisfies a condition. Interface Predicate also has methods and, negate andor.

#### Section 17.4.5 Sorting Previously Collected Results

• Once you place the results of a stream pipeline into a collection, you can create a new stream from the collection for performing additional stream operations on the prior results.

#### Section 17.5.1 Mapping Strings to Uppercase Using a Method Reference

• Stream method map (p. [747](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec17_html#page_747)) maps each element to a new value and produces a new stream with the same number of elements as the original stream.

• A method reference (p. [747](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec17_html#page_747)) is a shorthand notation for a lambda expression.

• ClassName::instanceMethodName represents a method reference for an instance method of a class. Creates a one-parameter lambda that invokes the instance method on the lambda’s argument and returns the method’s result.

• objectName::instanceMethodName represents a method reference for an instance method that should be called on a specific object. Creates a one-parameter lambda that invokes the instance method on the specified object—passing the lambda’s argument to the instance method—and returns the method’s result.

• ClassName::staticMethodName represents a method reference for a static method of a class. Creates a one-parameter lambda in which the lambda’s argument is passed to the specified a static method and the lambda returns the method’s result.

• ClassName::new represents a constructor reference. Creates a lambda that invokes the no-argument constructor of the specified class to create and initialize a new object of that class.

#### Section 17.5.2 Filtering Strings Then Sorting Them in Case-Insensitive Ascending Order

• Stream method sorted (p. [748](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec18_html#page_748)) can receive aComparator as an argument to specify how to compare stream elements for sorting.

• By default, method sorted uses the natural order for the stream’s element type.

• For Strings, the natural order is case sensitive, which means that "Z" is less than "a". Passing the predefinedComparator String.CASE\_INSENSITIVE\_ORDERperforms a case-insensitive sort.

#### Section 17.5.3 Filtering Strings Then Sorting Them in Case-Insensitive Descending Order

• Functional interface Comparator’s default methodreversed (p. [748](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec18_html#page_748)) reverses an existing Comparator’s ordering.

#### Section 17.6.1 Creating and Displaying aList<Employee>

• When the instance method referenceSystem.out::println is passed to Stream methodforEach, it’s converted by the compiler into an object that implements the Consumer functional interface. This interface’s accept method receives one argument and returns void. In this case, the accept method passes the argument to the System.out object’s printlninstance method.

#### Section 17.6.2 Filtering Employees with Salaries in a Specified Range

• To reuse a lambda, you can assign it to a variable of the appropriate functional interface type.

• The Comparator interface’s static method comparing(p. [753](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec22_html#page_753)) receives a Function that’s used to extract a value from an object in the stream for use in comparisons and returns a Comparator object.

• A nice performance feature of lazy evaluation is the ability to perform short circuit evaluation—that is, to stop processing the stream pipeline as soon as the desired result is available.

• Stream method findFirst is a short-circuiting terminal operation that processes the stream pipeline and terminates processing as soon as the first object from the stream pipeline is found. The method returns anOptional containing the object that was found, if any.

#### Section 17.6.3 Sorting Employees By Multiple Fields

• To sort objects by two fields, you create a Comparatorthat uses two Functions. First you call Comparatormethod comparing to create a Comparator with the firstFunction. On the resulting Comparator, you call method thenComparing with the second Function. The resulting Comparator compares objects using the firstFunction then, for objects that are equal, compares them by the second Function.

#### Section 17.6.3 Sorting Employees By Multiple Fields

• To sort objects by two fields, you create a Comparatorthat uses two Functions. First you call Comparatormethod comparing to create a Comparator with the firstFunction. On the resulting Comparator, you call method thenComparing with the second Function. The resulting Comparator compares objects using the firstFunction then, for objects that are equal, compares them by the second Function.

#### Section 17.6.5 Grouping Employees By Department

• Collectors static method groupingBy (p. [755](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec24_html#page_755)) with one argument receives a Function that classifies objects in the stream—the values returned by this function are used as the keys in a Map. The corresponding values, by default, are Lists containing the stream elements in a given category.

• Map method forEach performs an operation on each key–value pair. The method receives an object that implements functional interface BiConsumer. This interface’s accept method has two parameters. ForMaps, the first represents the key and the second the corresponding value.

#### Section 17.6.6 Counting the Number of Employees in Each Department

• Collectors static method groupingBy with two arguments receives a Function that classifies the objects in the stream and another Collector (known as the downstream Collector; p. [756](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec24_html#page_756)).

• Collectors static method counting returns aCollector that counts the number of objects in a given classification, rather than collecting them into a List.

#### Section 17.6.7 Summing and Averaging EmployeeSalaries

• Stream method mapToDouble (p. [756](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec24_html#page_756)) maps objects todouble values and returns a DoubleStream. The method receives an object that implements the functional interface ToDoubleFunction (packagejava.util.function). This interface’s applyAsDoublemethod invokes an instance method on an object and returns a double value.

#### Section 17.7 Creating a Stream<String> from a File

• Files method lines creates a Stream<String> for reading the lines of text from a file.

• Stream method flatMap (p. [760](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec27_html#page_760)) receives a Functionthat maps an object into a stream—e.g., a line of text into words.

• Pattern method splitAsStream (p. [760](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec27_html#page_760)) uses a regular expression to tokenize a String.

• Collectors method groupingBy with three arguments receives a classifier, a Map factory and a downstreamCollector. The classifier is a Function that returns objects which are used as keys in the resulting Map. TheMap factory is an object that implements interfaceSupplier and returns a new Map collection. The downstream Collector determines how to collect each group’s elements.

• Map method entrySet returns a Set of Map.Entryobjects containing the Map’s key–value pairs.

• Set method stream returns a stream for processing theSet’s elements.

#### Section 17.8 Generating Streams of Random Values

• Class SecureRandom’s methods ints (p. [762](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec8_html#page_762)), longsand doubles (inherited from class Random) returnIntStream, LongStream and DoubleStream, respectively, for streams of random numbers.

• Method ints with no arguments creates an IntStreamfor an infinite stream (p. [762](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec8_html#page_762)) of random int values. An infinite stream is a stream with an unknown number of elements—you use a short-circuiting terminal operation to complete processing on an infinite stream.

• Method ints with a long argument creates anIntStream with the specified number of random intvalues.

• Method ints with two int arguments creates anIntStream for an infinite stream of random int values in the range starting with the first argument and up to, but not including, the second.

• Method ints with a long and two int arguments creates an IntStream with the specified number of random int values in the range starting with the first argument and up to, but not including, the second.

• To convert an IntStream to a Stream<Integer> callIntStream method boxed.

• Function static method identity creates aFunction that simply returns its argument.

#### Section 17.9 Lambda Event Handlers

• Some event-listener interfaces are functional interfaces (p. [763](http://proquest.safaribooksonline.com/9780133813036/ch17lev2sec32_html#page_763)). For such interfaces, you can implement event handlers with lambdas. For a simple event handler, a lambda significantly reduces the amount of code you need to write.

#### Section 17.10 Additional Notes on Java SE 8 Interfaces

• Functional interfaces must contain only one abstractmethod, but may also contain default methods andstatic methods that are fully implemented in the interface declarations.

• When a class implements an interface with defaultmethods and does not override them, the class inherits the default methods’ implementations. An interface’s designer can now evolve an interface by adding newdefault and static methods without breaking existing code that implements the interface.

• If one class inherits the same default method from two interfaces, the class must override that method; otherwise, the compiler will generate a compilation error.

• You can create your own functional interfaces by ensuring that each contains only one abstract method and zero or more default or static methods.

• You can declare that an interface is a functional interface by preceding it with the@FunctionalInterface annotation. The compiler will then ensure that the interface contains only oneabstract method; otherwise, it’ll generate a compilation error.

### Self-Review Exercises

[**17.1**](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec15_html#ch17ans1) Fill in the blanks in each of the following statements:

a) Lambda expressions implement \_\_\_\_\_\_\_\_\_.

b) Functional programs are easier to \_\_\_\_\_\_\_\_\_ (i.e., perform multiple operations simultaneously) so that your programs can take advantage of multi-core architectures to enhance performance.

c) With \_\_\_\_\_\_\_\_\_ iteration the library determines how to access all the elements in a collection to perform a task.

d) The functional interface \_\_\_\_\_\_\_\_\_ contains methodapply that takes two T arguments, performs an operation on them (such as a calculation) and returns a value of type T.

e) The functional interface \_\_\_\_\_\_\_\_\_ contains methodtest that takes a T argument and returns a boolean, and tests whether the T argument satisfies a condition.

f) A(n) \_\_\_\_\_\_\_\_\_ represents an anonymous method—a shorthand notation for implementing a functional interface.

g) Intermediate stream operations are \_\_\_\_\_\_\_\_\_ —they aren’t performed until a terminal operation is invoked.

h) The terminal stream operation \_\_\_\_\_\_\_\_\_ performs processing on every element in a stream.

i) \_\_\_\_\_\_\_\_\_ lambdas use local variables from the enclosing lexical scope.

j) A performance feature of lazy evaluation is the ability to perform \_\_\_\_\_\_\_\_\_ evaluation—that is, to stop processing the stream pipeline as soon as the desired result is available.

k) For Maps, a BiConsumer’s first parameter represents the \_\_\_\_\_\_\_\_\_ and its second represents the corresponding \_\_\_\_\_\_\_\_\_.

[**17.2**](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec15_html#ch17ans2) State whether each of the following is true or false. Iffalse, explain why.

a) Lambda expressions can be used anywhere functional interfaces are expected.

b) Terminal operations are lazy—they perform the requested operation when they are called.

c) Method reduce’s first argument is formally called an identity value—a value that, when combined with a stream element using the IntBinaryOperatorproduces the stream element’s original value. For example, when summing the elements, the identity value is 1 and when getting the product of the elements the identity value is 0.

d) Stream method findFirst is a short-circuiting terminal operation that processes the stream pipeline but terminates processing as soon as an object is found.

e) Stream method flatMap receives a Function that maps a stream into an object. For example, the object could be a String containing words and the result could be another intermediate Stream<String> for the individual words.

f) When a class implements an interface with defaultmethods and overrides them, the class inherits thedefault methods’ implementations. An interface’s designer can now evolve an interface by adding newdefault and static methods without breaking existing code that implements the interface.

[**17.3**](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec15_html#ch17ans3) Write a lambda or method reference for each of the following tasks:

a) Write a lambda that can be used in place of the following anonymous inner class:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0772pro01a)

new IntConsumer()  
{  
   public void accept(int value)  
   {  
      System.out.printf("%d ", value);  
   }  
}

b) Write a method reference that can be used in place of the following lambda:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0772pro02a)

(String s) -> {return s.toUpperCase();}

c) Write a no-argument lambda that implicitly returns the String "Welcome to lambdas!".

d) Write a method reference for Math method sqrt.

e) Create a one-parameter lambda that returns the cube of its argument.

### Answers to Self-Review Exercises

[**17.1**](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec14_html#ch17que1)

a) functional interfaces.

b) parallelize.

c) internal.

d) BinaryOperator<T>.

e) Predicate<T>.

f) lambda expression.

g) lazy.

h) forEach.

i) Capturing.

j) short circuit.

k) key, value.

[**17.2**](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec14_html#ch17que2)

a) True.

b) False. Terminal operations are eager—they perform the requested operation when they are called.

c) False. When summing the elements, the identity value is 0 and when getting the product of the elements the identity value is 1.

d) True.

e) False. Stream method flatMap receives a Functionthat maps an object into a stream.

f) False. Should say: “...does not override them, ...” instead of “overrides them.”

[**17.3**](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec14_html#ch17que3)

a) value -> System.out.printf("%d ", value)

b) String::toUpperCase

c) () -> "Welcome to lambdas!"

d) Math::sqrt

e) value -> value \* value \* value

### Exercises

**17.4** Fill in the blanks in each of the following statements:

a) Stream \_\_\_\_\_\_\_\_\_ are formed from stream sources, intermediate operations and terminal operations.

b) The following code uses the technique of \_\_\_\_\_\_\_\_\_ iteration:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0772pro03a)

int sum = 0;  
  
for (int counter = 0; counter < values.length; counter++)  
   sum += values[counter];

c) Functional programming capabilities focus on \_\_\_\_\_\_\_\_\_ —not modifying the data source being processed or any other program state.

d) The functional interface \_\_\_\_\_\_\_\_\_ contains methodaccept that takes a T argument and returns void;accept performs a task with its T argument, such as outputting the object, invoking a method of the object, etc.

e) The functional interface \_\_\_\_\_\_\_\_\_ contains methodget that takes no arguments and produces a value of type T—this is often used to create a collection object in which a stream operation’s results are placed.

f) Streams are objects that implement interface Streamand enable you to perform functional programming tasks on \_\_\_\_\_\_\_\_\_ of elements.

g) The intermediate stream operation \_\_\_\_\_\_\_\_\_ results in a stream containing only the elements that satisfy a condition.

h) \_\_\_\_\_\_\_\_\_ place the results of processing a stream pipeline into a collection such as a List, Set or Map.

i) Calls to filter and other intermediate streams are lazy—they aren’t evaluated until an eager \_\_\_\_\_\_\_\_\_ operation is performed.

j) Pattern method \_\_\_\_\_\_\_\_\_ (new in Java SE 8) uses a regular expression to tokenize a String.

k) Functional interfaces must contain only one \_\_\_\_\_\_\_\_\_ method, but may also contain \_\_\_\_\_\_\_\_\_ methods and static methods that are fully implemented in the interface declarations.

**17.5** State whether each of the following is true or false. Iffalse, explain why.

a) An intermediate operation specifies tasks to perform on the stream’s elements; this is efficient because it avoids creating a new stream.

b) Reduction operations take all values in the stream and turn them into a new stream.

c) If you need an ordered sequence of int values, you can create an IntStream containing such values withIntStream methods range and rangeClosed. Both methods take two int arguments representing the range of values. Method rangeClosed produces a sequence of values from its first argument up to, but not including, its second argument. Method rangeproduces a sequence of values including both of its arguments.

d) Class Files (package java.nio.file) is one of many classes throughout the Java APIs that have been enhanced to support Streams.

e) Interface Map does not contain any methods that return Streams.

f) The Function functional interface—which is used extensively in functional programming—has methodsapply (abstract), compose (abstract), andThen(default) and identity (static).

g) If one class inherits the same default method from two interfaces, the class must override that method; otherwise, the compiler does not know which method to use, so it generates a compilation error.

**17.6** Write a lambda or method reference for each of the following tasks:

a) Write a lambda expression that receives two doubleparameters a and b and returns their product. Use the lambda form that explicitly lists the type of each parameter.

b) Rewrite the lambda expression in Part (a) using the lambda form that does not list the type of each parameter.

c) Rewrite the lambda expression in Part (b) using the lambda form that implicitly returns the value of the lambda’s body expression.

d) Write a no-argument lambda that implicitly returns the string "Welcome to lambdas!".

e) Write a constructor reference for class ArrayList.

f) Reimplement the following statement using a lambda as the event handler:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0773pro01a)

button.addActionListener(  
   new ActionListener()  
   {  
      public voidactionPerformed(ActionEvent event)  
      {  
         JOptionPane.showMessageDialog(ParentFrame.this,  
            "JButton event handler");  
      }  
   }  
);

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

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0774pro01a)

list.stream()  
    .filter(value -> value % 2 !=  
    0) .sum()

**17.8** Assuming that random is a SecureRandom object, explain in detail the stream pipeline:

[**Click here to view code image**](http://proquest.safaribooksonline.com/9780133813036/app06_html#p0774pro02a)

random.ints(1000000, 1, 3)  
      .boxed()  
      .collect(Collectors.groupingBy(Function.identity(),  
         Collectors.counting()))  
      .forEach((side, frequency) ->  
         System.out.printf("%-6d%d%n", side, frequency));

**17.9 (Summarizing the Characters in a File)** Modify the program of [Fig. 17.17](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec7_html#ch17fig17) to summarize the number of occurrences of every character in the file.

**17.10****(Summarizing the File Types in a Directory)**[Section 15.3](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec3_html#ch15lev1sec3) demonstrated how to get information about files and directories on disk. In addition, you used aDirectoryStream to display the contents of a directory. Interface DirectoryStream now contains default methodentries, which returns a Stream. Use the techniques from[Section 15.3](http://proquest.safaribooksonline.com/9780133813036/ch15lev1sec3_html#ch15lev1sec3), DirectoryStream method entries, lambdas and streams to summarize the types of files in a specified directory.

**17.11 (Manipulating a** ***Stream<Invoice>*)** Use the classInvoice provided in the exercises folder with this chapter’s examples to create an array of Invoice objects. Use the sample data shown in [Fig. 17.20](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec16_html#ch17fig20). Class Invoice includes four properties—a PartNumber (type int), a PartDescription(type String), a Quantity of the item being purchased (typeint) and a Price (type double). Perform the following queries on the array of Invoice objects and display the results:

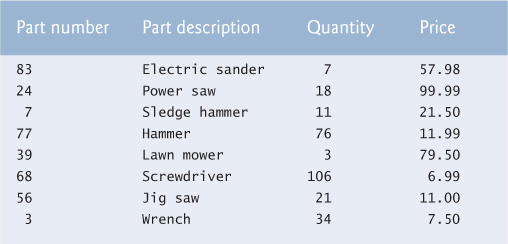
a) Use lambdas and streams to sort the Invoice objects by PartDescription, then display the results.

b) Use lambdas and streams to sort the Invoice objects by Price, then display the results.

c) Use lambdas and streams to map each Invoice to itsPartDescription and Quantity, sort the results byQuantity, then display the results.

d) Use lambdas and streams to map each Invoice to itsPartDescription and the value of the Invoice (i.e.,Quantity \* Price). Order the results by Invoicevalue.

e) Modify Part (d) to select the Invoice values in the range $200 to $500.



**Fig. 17.20** | Sample data for [Exercise 17.11](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec16_html#ch17que11).

**17.12 (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.

**17.13****(Sorting Letters and Removing Duplicates)** Write a program that inserts 30 random letters into aList<Character>. Perform the following operations and display your results:

a) Sort the List in ascending order.

b) Sort the List in descending order.

c) Display the List in ascending order with duplicates removed.

**17.14 (Mapping Then Reducing An** ***IntStream*** **for Parallelization)** The lambda you pass to a stream’s reducemethod should be associative—that is, regardless of the order in which its subexpressions are evaluated, the result should be the same. The lambda expression in lines 34–36 of[Fig. 17.5](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec3_html#ch17fig05) is not associative. If you were to use parallel streams ([Chapter 23](http://proquest.safaribooksonline.com/9780133813036/ch23_html#ch23), [Concurrency](http://proquest.safaribooksonline.com/9780133813036/ch23_html#ch23)) with that lambda, you might get incorrect results for the sum of the squares, depending on the order in which the subexpressions are evaluated. The proper way to implement lines 34–36 would be first to map each int value to the square of that value,then to reduce the stream to the sum of the squares. Modify[Fig. 17.5](http://proquest.safaribooksonline.com/9780133813036/ch17lev1sec3_html#ch17fig05) to implement lines 34–36 in this manner.