# Lambda expressions

<https://www.tutorialspoint.com/java8/java8_lambda_expressions.htm>

The important characteristics of a lambda expression:

* Optional type declaration − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
* Optional parenthesis around parameter − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
* Optional curly braces − No need to use curly braces in expression body if the body contains a single statement.
* Optional return keyword − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Using lambda expression, you can refer to any final variable or effectively final variable (which is assigned only once). Lambda expression throws a compilation error, if a variable is assigned a value the second time.

# Method references

Help to point to methods by their names. A method reference is described using "::" symbol. A method reference can be used to point the following types of methods −

* Static methods
* Instance methods
* Constructors using new operator

# Stream

Stream represents a sequence of objects from a source, which supports aggregate operations

The characteristics of a Stream:

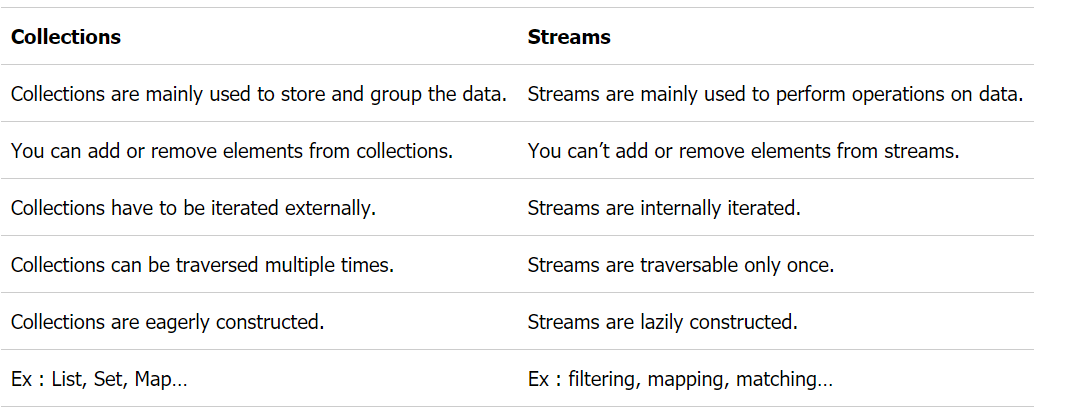
* Sequence of elements − A stream provides a set of elements of specific type in a sequential manner. A stream gets/computes elements on demand. It never stores the elements.
* Source − Stream takes Collections, Arrays, or I/O resources as input source.
* Aggregate operations − Stream supports aggregate operations like filter, map, limit, reduce, find, match, and so on.
* Pipelining − Most of the stream operations return stream itself so that their result can be pipelined.
* Automatic iterations

## Generating Streams

* stream() − Returns a sequential stream considering collection as its source.
* parallelStream() − Returns a parallel Stream considering collection as its source.
  + They should be used when the output of the operation is not needed to be dependent on the order of elements present in source collection (i.e. on which the stream is created)
  + Parallel Streams can be used in case of aggregate functions
  + Parallel Streams quickly iterate over the large-sized collections
  + Parallel Streams can be used if developers have performance implications with the Sequential Streams
  + If the environment is not multi-threaded, then Parallel Stream creates thread and can affect the new requests coming in

## Collections and Streams

Both are conceptually two different things which are used for two different purposes. If the collections are used to store the data then the streams are used to perform operations on that data



# QA

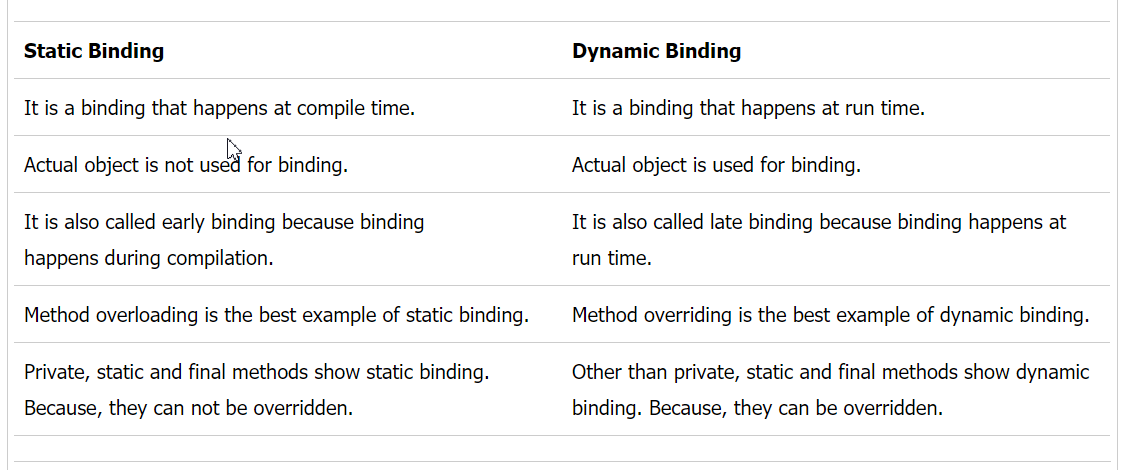
## Describe and compare fail-fast and fail-safe iterators.

* Fail-fast iterators operate directly on the collection itself. During iteration, fail-fast iterators fail as soon as they realize that the collection has been modified (i.e., upon realizing that a member has been added, modified, or removed) and will throw a ConcurrentModificationException. Some examples include ArrayList, HashSet, and HashMap (most JDK1.4 collections are implemented to be fail-fast).
* Fail-safe iterates operate on a cloned copy of the collection and therefore do not throw an exception if the collection is modified during iteration. Examples would include iterators returned by ConcurrentHashMap or CopyOnWriteArrayList.

## Do arguments in Java get passed by reference or by value?

* In Java, for primitive types, parameters are pass-by-value; For object types, object reference is pass-by-value, however, Java is allowed to modify object’s fields via object reference.

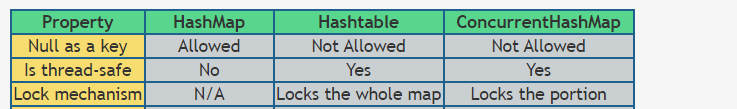
## Differences Between Static Binding And Dynamic Binding In Java?



## What is ForkJoinPool

The fork/join framework provides support for parallel programming by splitting up a task into smaller tasks to process them using the available CPU cores.

Difference between ConcurrentHashMap and Collections.synchronizedMap



## What is difference between CountDownLatch and CyclicBarrier in Java?

Both CyclicBarrier and CountDownLatch are used to implement a scenario where one Thread waits for one or more Thread to complete their job before starts processing. The differences are:

1) CyclicBarrier is resulable, CountDownLatch is not.

2) Both CyclicBarrier and CountDownLatch wait for fixed number of threads.

3) CountDownLatch is advanceable but CyclicBarrier is not.

## Annotations

* are used to provide supplement information about a program.
* Annotations start with ‘**@**’.
* Annotations do not change action of a compiled program.
* Annotations help to associate *metadata* (information) to the program elements i.e. instance variables, constructors, methods, classes,

## What are different types of spring auto-wiring modes

no: Default, no auto wiring, set it manually via “ref” attribute

byName: Auto wiring by property name. If the name of a bean is same as the name of other bean property, auto wire it.

byType: Auto wiring by property data type. If data type of a bean is compatible with the data type of other bean property, auto wire it.

constructor: byType mode in constructor argument.

autodetect: If a default constructor is found, use “autowired by constructor”; Otherwise, use “autowire by type”.

## Map() VS FlatMap()

when you use [map()](http://java67.blogspot.com/2015/01/java-8-map-function-examples.html), it applies a function on each element of stream and stores the value returned by the function into a new Stream.

FlatMap()  returns a Stream of values