1. Difference between Parallel Stream & Sequential Stream.

Parallel Stream:

↳ Parallel Stream divides the given task into many tasks & run them in different threads.

↳ Runs multiple iterations simultaneously in different available cores.

↳ If number of tasks are more than cores at a given time, the remaining tasks are queued to another task to finish.

↳ Fork & Join framework is used in background to create multiple threads

↳ Gives the flexibility to iterate over the list in a parallel pattern.

↳ process the subtasks & combines the results.

Sequential Stream:

↳ Sequential Stream works like a for loop which runs the tasks one after another

↳ Runs single Stevation at one and perform tasks sequentially

:) Each iteration waits another one to finish

↳ The output for using Sequential Stream will be in ordered sequence.

↳ Objects are piplined into a single Streams on the same Processing System.

↳ Returns a Sequential Stream containing a single element & ordered stream whose elements are Specified values.

2. Difference between Map & flatMap?

map:

- map operation produces one output value for each input value
- takes Stream and transform it to another Stream.map() wraps the underlying sequence in a Stream

flatmap:

- flatmap operation, for each input it will produces arbitrary (0 or) more) number of values
- flat the input in Stream of values into a new array.

3. Functional Interfaces in JAVA:

- Interface which contains exactly one abstract method is known as functional Interface.
- Can have number of default, static methods also known as Sigle Abstract method Interface.

Functional Interfaces :

"Consumer<T>" :

↳ It represents an operation that accept a single argument and returns no results

↳ With 2 arguments known as "BiConsumer<T, U>".

"Function<T,R>" :

↳ It represents an operation that accept a single arguments and returns a result.

↳ With 2 arguments and results is known as "BiFunction<T, U, R>".

"Predicate<T>" :

↳ It represents predicate (boolean) with one argument

↳ With 2 arguments and predicate (boolean-valued function) is "BiPredicate<T, U>".

"Binary Operator <T>":

↳ Represents an operation upon two operands of same data type which returns a result of the same type as the operands.

"Unary Operator <T>":

↳ Represents a operation on single operand results same type as its operand.

"Supplier <T>":

↳ Represent Supplier of results

↳ Represents with Boolean valued results a Supplier is "Boolean Supplier".

4. Date and Time API :

- Local Date :
  - Represents data in ISO Format (YYYY-MM-DD) without time.
  - Local Date representing a specific day, month, year.
  - Can be obtained using parse method or, of method.

Eg: Local Date date = Local Date.now().
Eg: Local Date.of(2017, 02, 28)

- Local Time :
  - Represents Time without Date.

Local Time t = Local Time.now()

- Local Date Time :
  - Both Date & Time.

Local Date Time.now();

- Zoned Date Time :
  - Zone Specific date & time.

~~Zone Id tz = Zone Id.of("");~~

  - used to represent different zones. Date & time.

Zoned Date Time z = Zoned Date Time.parse("2020-10-05 T05:20:10+05:30[Asia/Kolkata]");

Period:

↳ Represent quantity of time in terms of years, months & days. in base of date

Duration:

↳ Represent amount of Time in the base of time.

Compatibility of Alithe Date and Calendar:

↳ to Instant(), which helps to Convert existing Date & Calendar instance to new DateTime API.

5. Methods in Streams:

map(): map() used to returns a Stream Consisting of the result. produce one output value for each input value.

flatMap(): flattens the input into a new array

filter():

↳ used to select elements as per the Predicate passed on arguments.

↳ Based on Condition we provide.

Sorted: to Sort the Stream of data.

Collect: to return the result of operation performed on Stream.

reduce:

↳ to reduce the elements of Stream to be a single value.

peek:

↳ Returns a new Stream consisting of all the elements of original Stream applying method argument action.

5. String Joiners:

↳ String joiners used to Construct a sequence of characters seperated by a delimiter

↳ Can also pass prefix & suffix to Char sequence.

```
String Joiner join = new StringJoiner();
join.add("Sri");
join.add("Harsha");
Sop(join) => Sri Harsha.
```

7. Parallel Sort :

↳ Uses a parallel-Sort merge Sorting algorithm.

↳ Breaks array into sub arrays that are sorted themselves & merged.

Arrays. ParallelSort():

↳ Sort the array of objects & primitives.

↳ Similar to Sort(), it has 2 variants to sort full array & partial array.

8. Base 64 :

↳ Base 64 is used for encoding.

↳ Used ~~when~~ ~~Base~~ to encode Binary data. used in storing complex data in XML

↳ encodes 3 bytes of data 4 bytes of encoded data.

String BasicBase64 = Base64.getEncoder().
encodeToString("actual String".
getBytes());

9. Nashron JS:

- Directly compiles code in the memory and passed bytecode to JVM.
- Helps in improve the performance.
- JDK 8 introduces command line tool jjs to execute javascript codes at console for Nashron engine.
- used to execute JS code dynamically at JVM.

10. Collection API & Stream API.

Collection API:

- Storing finite Number of elements in a Data Structure
- used to storing date into different date structures
- Iteration & Consumption of elements can be done multiple times

Stream API :

- We can handle Streams of data that contains infinite Number of elements
- used for computation of date into different data structures.
- Creates objects in lazy manner
- iterate or consume elements only once.