Java 8 :

Purpose of Java 8

1) Concise & minimal code
2) Functional programming in Object Oriented programming
3) To enable parallel programming.

Features

1) Lambda exp -

Similar to methods, but they do not need a name and they can be implemented right in the body of a method.

$(x, y) \rightarrow x+y$

2) Stream API. → API denotes that this ~~concept~~ provides a set of classes, interfaces & methods for working

- For bulk data like lists, array
- Operation on collections "

3) Date & Time API
- new date-time API under package java.time.

4) Base 64 encode & decode
- base 64 encoding in java 8 has built-in encode & decode functions

5) Method reference & constructor reference
:: operator

6) Default methods in Interfaces.
→ Before interfaces used to only have public abstract methods. now they have default methods

7) Static methods.

8) Functional interface. (single abstract method)
A f.i is an interface that has exactly one abstract method. To designate an interface as a Functional interface, we don't need to use the @FunctionalInterface annotatio

9) Optional.

10) Improvements in Java I/p & o/p

11) Collection API improvement

Lambda

- Anonymous fn which
  1. does not have any name
  2. not have any return type
  3. not having any modifier

- Remove modifier
- Remove return type
- Remove method name
- Place arrow

~~Using compiler/ inference~~

```
private void add(int a, int b){
    Sout(a+b);
}
   ↓
converted to
   ↓
(int a, int b) -> { System.out.print(a+b)}
   ↓  compiler will guess the situation or content
(a,b) -> System.out.println(a+b)
```

Type Inference

```
~~private int getStringLen(String str){~~
    return str.leng();
}
   ↓
(String str) - { ~~return~~ str.length(); }
   ↓
~~(str)~~ → str.length()
   ↓
str → str.length()
```

Benifits -
1. To enable functional programming
2. To make code more readable, maintainable and consise
3. To enable parallel process
4. JAR file size reduction
5. Elimination of shadow variables.

Functional Interface

- Interface with one single abstract method but can have any number of defaults and static methods. We can invoke lambda Expression by using functional interface.

Before 1.8, not public abstract methods (with body) were allowed in interfaces
in 1.8 → default & static method
but interface can have (concrete methods)

You can use @FunctionalInterface so that no body can add new methods to the interface.

Default methods inside interface

Before 1.8, i.e until 1.7 only public abstract methods were allowed whether we declare by writing or not. (meaning no implementation)

→ ~~Stability~~
→ Since java 8, we can have concrete methods as well inside interface i.e implementation in interface.
→ methods with body in interface :) default methods.

S

▶. Static methods in interface are those methods which are defined in the interface with Keyword static.

• (Cannot be overriden) or changed in the implementation class.

• Static methods contain the complete definition of the fn. (same as default method, we can write defn)

See github.

▶ Use of lambda functions → to give definition for functional interface. instead of creating an implementation class.

interface Employee () {
void sayhello();
}

① Employee employee = () → "Software" → Implementation of the fn in Functional interface (sayHello())

functional interface Employee. with ONE fn = lambda expres instead of overriding the ONE func in a different file

Above is same as "int a = 3;"

② Create thread using lambda exp.

→ Runnable = } both are functional interface.

③ → Comparator

▶ Anonymous ~~function~~ inner class
- lambda functions can be used when ~~any~~ the interface has one method
- When it has more than one method, use anonymous inner class. to define the methods in the class

▶ Predicate (functional interface) - It is introduced in Java 8. (Condition check)
represent
It is a boolean valued function

↳ one method ~~signat~~
↳ n no. of static methods
↳ n no. of default methods

Imp!

Predicate stores a condition just like if else.

Predicate has ① negate
② or
③ and
④ isEqual.

Predicate used a lot in Stream (ex. filter())

▶ Function (functional interface) - return by doing little work or operations

Function < ~~I~~ T, R > = Function <Input, Output).

Has an function apply => R apply (T);

Function chaining is available.

Function has ① compose
② and Then - chaining, return Function
③ identity - returns same

▶ Consumer (functional interface).
- as it consumes (in the name) it returns nothing (void).
- has - andthen.

▶ Supplier (functional interface).

▶ Stream.

Any Collection $\xrightarrow{\text{convert}}$ Stream
(we can use declarative / functional programming).

if we use Predicate, Function, Consumer, Supplier

↳ meaning we will use few ~~[illegible]~~ methods
like map (Function)
filter (Predicate)
reduce

Stream is a sequence of elements from collection, ~~[illegible]~~ which we can use for functional programming.

① Readability
② Parrallelism.