1. concise code
2. Enabling functional programming
3. Lamda expression
4. Functional Interface
5. Default methods and static methods
6. Predefined functional interface

Predicate,Function,Consumer,Supplier etc

1. Double colon operator

Method reference, Constructor reference

1. Streams
2. Date and Time api
3. Optional classes
4. NaShorn javascript engine
5. Why Lamadas?

* provides implementation of functional interface.
* Readable and concise code.
* Easier to use APIs and libraries.
* Enable support for parallel processing.

1. Lambda expression syntax

->It is an anonymous function

-> For calling lamada expression functional interface must be required.

(argument-list) -> {body}

1. Functional programming vs Object Oriented Programming

* In oops, Everything is object and all code blocks are associated with classes and objects.
* But in functional programming, We are directly assigning the method to interface variable.
* Interface can contain only one method.
* Access modifier, returntype, method name ,parenthesis for one parameter,curly braces for one statement and return statement is not required.
* We need a (->) after () parenthesis.
* E.g. Greeting greeting=() -> System.out.println("Hello World!");

1. Type Inference
2. Functional Interface

* It contain only one abstract method.
* We have to use annotation @*FunctionalInterface.*
* Lambdas expression provides implementation of functional interface.

1. Predicate

* Functional interface that is used to return a value either true or false.
* test(),and(),or(),isEqual(),negate()

1. Consumer

* Take one input and returns no results.method is accept() and andThen().

1. BiConsumer

* Take two input and produces a result.

1. Function

* Used to convert one type to another type.e.g: employee list -> employee name list.
* Apply(),andThen(),compose(),identity()

1. Supplier

* Supplier represents a supplier of results.
* Get()

1. Stream

* Requirement is to get maximum salary without doing any computation at developer end.
* Using collection, JD has to use lops and do repeated check. So it causes efficiency problem.
* It will take the collection of object and will do aggregate operations.
* E.g: SELECT max(salary), employee\_id, employee\_name FROM Employee
* Collection interface has two methods to generate a Stream. stream() and parallelStream().

1. Method Reference and Constructor Reference

* A method reference is described using "::" symbol. A method reference can be used to point the following types of methods.
* Static method, Instance method, Constructors using new operator.(Treeset :: new)

1. Default Method and static method

* If we want to add any method in interface without affecting implementation classes then we have to use default method.
* If we have to define utility method then go for static method in interface.
* We can write default or static method in interface.
* possibility that a class is implementing two interfaces with same default methods and will get ambiguity. So resolving this issue either we have to override that method or call that method with interface name.super.
* if static method will be there then call with interface name.

1. Date and Time API

* Java Date Time classes are not defined consistently, java.util.Date contains both date and time, whereas java.sql.Date contains only date. Having this in java.sql package doesn’t make sense. Also both the classes have same name, that is a very bad design itself.
* java.text.DateFormat abstract class for parsing and formatting need. Usually SimpleDateFormatclass is used for parsing and formatting.
* Date classes are mutable, so they are **not thread safe**. there is no timezone support.
* So java.util.Calendarand java.util.TimeZone classes were introduced. But still above problem was there.
  1. So Joda time came in java8.that is
* immutable,
* have separate classes for Date,Time, DateTime, TimeZone, TimeStamp etc.
* Methods are clearly defined.
* Contains method for plus, minus, format, parsing etc.
* Work on ISO-8601 calendar system. Also it works with non ISO.

1. **Optional** : This is a class that contains multiple utility methods to handle values as “available” or “not available” instead of checking null values.