**Java 8 Lambda Expressions:**

An interface which has only one abstract method is called a functional interface. **@FunctionalInterface**

**(Argument-list) -> {body}**

* Argument list can be empty.
* Body contains expressions and statements.

e.g.: () -> {}, (p1) -> {}, (p1, p2) -> {}

For 1 Argument, bracket of argument list is not needed

**Lambda Expression can be used to iterate loops using for each and do operations on them.**

nameList.forEach(

(n) -> {

System.out.println(n);

System.out.println(" looping");

}

);

**Lambda expression can also be used for threading since runnable interface has only one method run ().**

Runnable r2 = () -> {

System.***out***.println("Thread 2 running");

};

Thread t2 = **new** Thread(r2);

t2.start();

**Sorting using Lambda Expressions:**

Collections.sort(list,Comparator). This comparator interface has only one method compare. So lambda expression can be used here.

Collections.*sort*(

productList,

(p1,p2) ->{

**return** p1.name.compareTo(p2.name);

}

);

**Filter using Lambda:**

Stream is used to filter the list based on criteria like price > 40.

Stream<Product> filteredData = productList.stream().filter(p -> p.price > 40);

**Method References:**

* Reference to Static method == **ContainingClass :: StaticMethodName**
* Reference to instance method == **ContainingObject :: InstanceMethodName**
* Reference to constructor == **ClassName :: new**

**Java Functional Interface:**

* A functional interface can have exactly one abstract method.
* It can have any number of default or static methods or other methods of object class as well.
* A functional interface can only extend another interface if that interface does not have an abstract method.

**Java 8 Stream (**java.util.stream and java.util.stream.Collectors**)**

* Stream is not for storage. It conveys elements from a source (data structure/I-O) through a pipeline of operations.
* It is functional. Operations performed does not modify source. E.g.: filtering a stream does not change the source data structure.
* Stream is lazy and fast.
* Stream is one time use.

**Filter-map-collect**

products.stream()

.filter(p -> p.price > 30000) // filters based on given predicate price > 30000

.map(p -> p.price) // fetch price

.collect(Collectors.*toList*());// collects to another list

**Iterate a stream**

Stream.*iterate*(1, e->e+1)//initial val = 1, increment by 1

.filter(e->e%5==0)//only multiple of 5

.limit(5)//max size = 5

.forEach(System.***out***::println);//print

**We can also convert a list to a set or a map using streams**

products.stream().map(p->p.price).collect(Collectors.*toSet*());//Store price in a set

products.stream().collect(Collectors.*toMap*(p->p.id, p->p.name)); //store id and name in a map with id as key

**Base 64 encoder and decoder**

* Basic encoding and decoding ( no line separator )
* URL encoding and decoding ( no line separator )
* MIME encoding and decoding \r\n as line separator. No line separator at the end

**Default methods**

Methods tagged as default inside interface are non-abstract default methods. We can also have static methods inside interface. Thus an interface is different from an abstract class in the sense that an abstract class can have constructors.

**StringJoiner**

They are used to create a string separated by delimiters

StringJoiner joiner = **new** StringJoiner(":", "[", "]");

joiner.add("Bua");

joiner.add("Titir"); //[Bua:Titir]