Java 1.8

<http://www.javagenerics.com/java-8-streams-filter-examples/>

<http://www.mkyong.com/tutorials/java-8-tutorials/>

public class TestDate {

public static void main(String[] args) {

int[] number = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

//Java 8

boolean result = IntStream.of(number).anyMatch(x -> x == 4);

if (result) {

System.out.println("Hello 4");

} else {

System.out.println("Where is number 4?");

}

long[] lNumber = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

boolean result2 = LongStream.of(lNumber).anyMatch(x -> x == 10);

if (result2) {

System.out.println("Hello 10");

} else {

System.out.println("Where is number 10?");

}

}

package com.journaldev.java8.stream;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Stream;

public class StreamExample {

public static void main(String[] args) {

List<Integer> myList = new ArrayList<>();

for(int i=0; i<100; i++) myList.add(i);

//sequential stream

Stream<Integer> sequentialStream = myList.stream();

//parallel stream

Stream<Integer> parallelStream = myList.parallelStream();

//using lambda with Stream API, filter example

Stream<Integer> highNums = parallelStream.filter(p -> p > 90);

//using lambda in forEach

highNums.forEach(p -> System.out.println("High Nums parallel="+p));

Stream<Integer> highNumsSeq = sequentialStream.filter(p -> p > 90);

highNumsSeq.forEach(p -> System.out.println("High Nums sequential="+p));

}}

we have an example of a stateful lambda function used in a parallel streams. This example is taken from the java JDK API and shows a simplified distinct() implementation.  
Set seen = Collections.synchronizedSet(new HashSet());  
stream.parallel().map(e -> { if (seen.add(e)) return 0; else return e; })...

Real time ex: (userdetailprocess-FIP)

errorParameter.parallelStream().forEachOrdered(**val** -> {

bizErrorList.add(((Map<?, ?>) val).get(AutoCallCommonConstant.**API\_PARAM\_MESSAGE\_ID**).toString(),

**new** String[] {((Map<?, ?>) val).get(AutoCallCommonConstant.**API\_PARAM\_PARAMETER**).toString()});

**getScheme():**

* **public java.lang.String getScheme():** Returns the name of the scheme used to make this request, for example, http, https, or ftp. Different schemes have different rules for constructing URLs, as noted in RFC 1738.

The method **getScheme()**returns a string, the protocol used in ACTION attribute of FORM tag used by the client to call the servlet.

long startTime = System.nanoTime();

//code

long endTime = System.nanoTime();

System.out.println("Took "+(endTime - startTime) + " ns");

**import** java.util.ArrayList;

**import** java.util.Date;

**import** java.util.List;

**import** java.util.concurrent.Callable;

**import** java.util.concurrent.ExecutionException;

**import** java.util.concurrent.ExecutorService;

**import** java.util.concurrent.Executors;

**import** java.util.concurrent.Future;

// Executor framework to execute 100 tasks in parallel and use Java Future to get the result of the submitted tasks.

//Future obj hold the result of excute submit

**public** **class** MyCallable **implements** Callable<String> {

@Override

**public** String call() **throws** Exception {

Thread.**sleep**(1000);

//return the thread name executing this callable task

**return** Thread.**currentThread**().getName();

}

**public** **static** **void** main(String args[]){

//Get ExecutorService from Executors utility class, thread pool size is 10

ExecutorService **executor** = Executors.**newFixedThreadPool**(2);

//create a list to hold the Future object associated with Callable

List<Future<String>> **list** = **new** ArrayList<Future<String>>();

//Create MyCallable instance

Callable<String> **callable** = **new** MyCallable();

**for**(**int** **i**=0; i< 100; i++){

//submit Callable tasks to be executed by thread pool

Future<String> **future** = executor.submit(callable);

//add Future to the list, we can get return value using Future

list.add(future);

}

**for**(Future<String> **fut** : list){

**try** {

//print the return value of Future, notice the output delay in console

// because Future.get() waits for task to get completed

System.**out**.println(**new** Date()+ "::"+fut.get());

}

**catch** (InterruptedException | ExecutionException **e**) {

e.printStackTrace();

}

}

//shut down the executor service now

executor.shutdown();

}

}

List –Filter - http://www.mkyong.com/java8/java-8-streams-filter-examples/

package com.mkyong.java8;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class NowJava8 {

public static void main(String[] args) {

List<String> lines = Arrays.asList("spring", "node", "mkyong");

List<String> result = lines.stream() // convert list to stream

.filter(line -> !"mkyong".equals(line)) // we dont like mkyong

.collect(Collectors.toList()); // collect the output and convert streams to a List

result.forEach(System.out::println); //output : spring, node

}

}

The equivalent example in Java 8, use stream.filter() to filter a List, and .findAny().orElse (null) to return an object conditional.

NowJava8.java

package com.mkyong.java8;

import java.util.Arrays;

import java.util.List;

public class NowJava8 {

public static void main(String[] args) {

List<Person> persons = Arrays.asList(

new Person("mkyong", 30),

new Person("jack", 20),

new Person("lawrence", 40)

);

Person result1 = persons.stream() // Convert to steam

.filter(x -> "jack".equals(x.getName())) // we want "jack" only

.findAny() // If 'findAny' then return found

.orElse(null); // If not found, return null

System.out.println(result1);

Person result2 = persons.stream()

.filter(x -> "ahmook".equals(x.getName()))

.findAny()

.orElse(null);

System.out.println(result2);

}

}

Multiple conditions:

List<Person> persons = Arrays.asList(

new Person("mkyong", 30),

new Person("jack", 20),

new Person("lawrence", 40)

);

Person result1 = persons.stream()

.filter((p) -> "jack".equals(p.getName()) && 20 == p.getAge())

.findAny()

.orElse(null);

System.out.println("result 1 :" + result1);

Stream with map:

// Java 8

List<String> collect = alpha.stream().map(String::toUpperCase).collect(Collectors.toList());

System.out.println(collect); //[A, B, C, D]

// Extra, streams apply to any data type.

List<Integer> num = Arrays.asList(1,2,3,4,5);

List<Integer> collect1 = num.stream().map(n -> n \* 2).collect(Collectors.toList());

System.out.println(collect1); //[2, 4, 6, 8, 10]

<http://www.mkyong.com/java8/java-8-filter-a-map-examples/>

Collectors.joining() with List of String

https://www.concretepage.com/java/jdk-8/java-8-collectors-joining-example

**JoiningExampleWithListOfString.java**

package com.concretepage;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class JoiningExampleWithListOfString {

public static void main(String[] args) {

List<String> list = Arrays.asList("Ram","Shyam","Shiv","Mahesh");

String result= list.stream().collect(Collectors.joining());

System.out.println(result);

result= list.stream().collect(Collectors.joining(","));

System.out.println(result);

result= list.stream().collect(Collectors.joining("-","[","]"));

System.out.println(result);

}

}

**Output**

RamShyamShivMahesh

Ram,Shyam,Shiv,Mahesh

[Ram-Shyam-Shiv-Mahesh]

<https://www.concretepage.com/java/jdk-8/java-8-collectors-joining-example>

<http://www.mkyong.com/java8/java-8-optional-in-depth/>

List<Staff> staff = Arrays.asList(

new Staff("mkyong", 30, new BigDecimal(10000)),

new Staff("jack", 27, new BigDecimal(20000)),

new Staff("lawrence", 33, new BigDecimal(30000))

);

//Java 8

List<String> collect = staff.stream().map(x -> x.getName()).collect(Collectors.toList());

System.out.println(collect); //[mkyong, jack, lawrence]

StringJoiner sj = new StringJoiner("/", "prefix-", "-suffix");

sj.add("2016");

sj.add("02");

sj.add("26");

String result = sj.toString(); //prefix-2016/02/26-suffix

// String -> Date

SimpleDateFormat.parse(String);

// Date -> String

SimpleDateFormat.format(date);

<http://www.mkyong.com/java/how-to-convert-string-to-date-java/>

RT Example :

--------------------

SimpleDateFormat **converStringToDate** = **new** SimpleDateFormat(AutoCallCommonConstant.**DEFAULT\_DATE\_FORMAT\_YYYYMMDD**);

Date **fromDate1**;

Date **toDate1**;

**try** {

fromDate1 = converStringToDate.parse(fromDate);

toDate1 = converStringToDate.parse(toDate);

**if** (fromDate1.before(toDate1) || fromDate1.equals(toDate1)) {

**return** **true**;

}

} **catch** (ParseException **e**) {

e.printStackTrace();

}

<http://www.mkyong.com/java8/java-8-temporaladjusters-examples/>

for date n specifiec 1st n last day of a month

Joining Array :

String[] s1 = new String[]{"a", "b", "c"};

String[] s2 = new String[]{"d", "e", "f"};

String[] s3 = new String[]{"g", "h", "i"};

//join object type array

String[] result = Stream.of(s1, s2, s3).flatMap(Stream::of).toArray(String[]::new);

System.out.println(Arrays.toString(result));

<https://www.mkyong.com/java8/java-8-flatmap-example/>