package com.logicbig.example;

import java.time.LocalTime;

import java.util.Arrays;

import java.util.stream.Stream;

public class SequentialParallelComparison {

public static void main (String[] args) {

String[] strings = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};

System.out.println("-------\nRunning sequential\n-------");

run(Arrays.stream(strings).sequential());

System.out.println("-------\nRunning parallel\n-------");

run(Arrays.stream(strings).parallel());

}

public static void run (Stream<String> stream) {

stream.forEach(s -> {

System.out.println(LocalTime.now() + " - value: " + s +

" - thread: " + Thread.currentThread().getName());

try {

Thread.sleep(200);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

import static com.logicbig.example.LogUtil.log;

public class LazyExample {

public static void main (String[] args) {

IntStream stream = IntStream.range(1, 5);

stream = stream.peek(i -> log("starting", i))

.filter(i -> {

log("filtering", i);

return i % 2 == 0;

})

.peek(i -> log("post filtering", i));

log("Invoking terminal method count.");

log("The count is", stream.count());

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

import static com.logicbig.example.LogUtil.log;

public class LazyParallelExample {

public static void main (String[] args) {

IntStream stream = IntStream.range(1, 5).parallel();

stream = stream.peek(i -> log("starting", i))

.filter(i -> {

log("filtering", i);

return i % 2 == 0;

})

.peek(i -> log("post filtering", i));

log("Invoking terminal method count.");

log("The count is", stream.count());

}

}

package com.logicbig.example;

import java.time.LocalTime;

public class LogUtil {

public static void log (Object... objects) {

String s = LocalTime.now().toString();

for (Object object : objects) {

s += " - " + object.toString();

}

System.out.println(s);

// just putting a little delay so that we can see a clear difference

// with parallel stream

try {

Thread.sleep(1);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class AllMatchExample {

public static void main (String[] args) {

Stream<String> stream = Stream.of("one", "two", "Three", "four");

boolean match = stream.allMatch(s -> s.length() > 0 &&

Character.isLowerCase(s.charAt(0)));

System.out.println(match);

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class AnyMatchExample {

public static void main(String[] args) {

Stream<String> stream = Stream.of("one", "two", "three", "four");

boolean match = stream.anyMatch(s -> s.contains("our"));

System.out.println(match);

}

}

package com.logicbig.example;

import java.util.OptionalInt;

import java.util.stream.IntStream;

public class FindAnyExample {

public static void main (String[] args) {

IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

.parallel();

stream = stream.filter(i -> i % 2 == 0);

OptionalInt opt = stream.findAny();

if (opt.isPresent()) {

System.out.println(opt.getAsInt());

}

}

}

package com.logicbig.example;

import java.util.OptionalInt;

import java.util.stream.IntStream;

public class FindFirstExample {

public static void main (String[] args) {

IntStream stream = IntStream.of(1, 2, 3, 4, 5, 6);

stream = stream.filter(i -> i % 3 == 0);

OptionalInt opt = stream.findFirst();

if (opt.isPresent()) {

System.out.println(opt.getAsInt());

}

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.stream.IntStream;

public class LimitExample {

public static void main (String[] args) {

int[] ints = {1, 2, 3, 4, 5, 6};

System.out.printf("Source: %s%n", Arrays.toString(ints));

System.out.println("Finding even numbers.");

runWithoutLimit(Arrays.stream(ints));

//Note: creating and passing new stream because it

// cannot be reused after a terminal operation is called.

runWithLimit(Arrays.stream(ints));

}

private static void runWithoutLimit (IntStream stream) {

System.out.println("Running without limit()");

//filter even numbers

stream.filter(i -> i % 2 == 0)

.forEach(System.out::println);

}

private static void runWithLimit (IntStream stream) {

System.out.println("Running with limit(2)");

//filter even numbers

stream.filter(i -> i % 2 == 0)

.limit(2)

.forEach(System.out::println);

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class LimitingInfiniteStream {

public static void main (String[] args) {

Stream<Integer> stream = Stream.iterate(1, i -> i + 1);

stream.filter(i -> i % 2 == 0)

.limit(5)

.forEach(System.out::println);

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class LimitingInfiniteStream2 {

public static void main (String[] args) {

Stream<Integer> stream = Stream.iterate(1, i -> i + 1);

stream.limit(5)

.filter(i -> i % 2 == 0)

.forEach(System.out::println);

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class NoneMatchExample {

public static void main (String[] args) {

Stream<String> stream = Stream.of("one", "two", "three", "four");

boolean match = stream.noneMatch(s -> s.length() > 0 &&

Character.isUpperCase(s.charAt(0)));

System.out.println(match);

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class DistinctExample {

public static void main (String[] args) {

PerformanceTestUtil.runTest("unordered stream", () -> {

IntStream stream = IntStream.range(0, 1000000);

stream.unordered().parallel().distinct().count();

});

PerformanceTestUtil.runTest("ordered stream", () -> {

IntStream stream = IntStream.range(0, 1000000);

stream.parallel().distinct().count();

});

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.stream.Stream;

public class DistinctStabilityExample {

public static void main (String[] args) {

Object[] myObjects = createStream().parallel().distinct().toArray();

System.out.printf("ordered distinct result 1: %s%n",

Arrays.toString(myObjects));

MyObject.c = 0;

myObjects = createStream().parallel().distinct().toArray();

System.out.printf("ordered distinct result 2: %s%n",

Arrays.toString(myObjects));

MyObject.c = 0;

myObjects = createStream().unordered().parallel().distinct().toArray();

System.out.printf("unordered distinct result 1: %s%n",

Arrays.toString(myObjects));

MyObject.c = 0;

myObjects = createStream().unordered().parallel().distinct().toArray();

System.out.printf("unordered distinct result 2: %s%n",

Arrays.toString(myObjects));

}

private static Stream<MyObject> createStream () {

return Stream.of(new MyObject("a"), new MyObject("b"),

new MyObject("c"), new MyObject("b"),

new MyObject("c"), new MyObject("c"),

new MyObject("a"));

}

private static class MyObject {

private static int c = 0;

private int id = ++c;

private String str;

public MyObject (String str) {

this.str = str;

}

@Override

public boolean equals (Object o) {

if (this == o)

return true;

if (o == null || getClass() != o.getClass())

return false;

MyObject myObject = (MyObject) o;

return str != null ? str.equals(myObject.str) : myObject.str == null;

}

@Override

public int hashCode () {

return str != null ? str.hashCode() : 0;

}

@Override

public String toString () {

return "MyObject{id=" + id + ", str='" + str + "\'}";

}

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.stream.IntStream;

public class LimitExample {

public static void main (String[] args) {

int[] ints = {1, 2, 3, 4, 5, 6};

System.out.printf("Source: %s%n", Arrays.toString(ints));

System.out.println("Finding even numbers.");

runWithoutLimit(Arrays.stream(ints));

//Note: creating and passing new stream because it

// cannot be reused after a terminal operation is called.

runWithLimit(Arrays.stream(ints));

}

private static void runWithoutLimit (IntStream stream) {

System.out.println("Running without limit()");

//filter even numbers

stream.filter(i -> i % 2 == 0)

.forEach(System.out::println);

}

private static void runWithLimit (IntStream stream) {

System.out.println("Running with limit(2)");

//filter even numbers

stream.filter(i -> i % 2 == 0)

.limit(2)

.forEach(System.out::println);

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.stream.DoubleStream;

import java.util.stream.IntStream;

import java.util.stream.LongStream;

public class ForEachExample {

public static void main (String[] args) {

final int[] ints = IntStream.range(0, 5).toArray();

PerformanceTestUtil.runTest("forEach() method", () -> {

Arrays.stream(ints).parallel().forEach(i -> doSomething(i));

});

PerformanceTestUtil.runTest("forEachOrdered() method", () -> {

Arrays.stream(ints).parallel().forEachOrdered(i -> doSomething(i));

});

}

private static void doSomething (int i) {

try {

Thread.sleep(10);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.printf("%s, ", i);

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class PeekExample {

public static void main (String[] args) {

IntStream.range(0, 5).parallel().peek(System.out::println).

count();

}

}

package com.logicbig.example;

import java.io.IOException;

import java.io.OutputStream;

import java.io.PrintStream;

import java.util.Optional;

import java.util.concurrent.TimeUnit;

import java.util.stream.Stream;

import static java.util.concurrent.TimeUnit.\*;

public class PerformanceTestUtil {

public static void runTest (String msg, Runnable testRunner) {

setPrintStreamDisabled(true);

//run test quietly first time to avoid cold start false-positive result

testRunner.run();

setPrintStreamDisabled(false);

long startTime = getTimeElapsed(0);

testRunner.run();

System.out.printf("%s time taken: %s%n", msg, timeToString(getTimeElapsed(startTime)));

}

private static String timeToString (long nanos) {

Optional<TimeUnit> first = Stream.of(DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS,

MICROSECONDS).filter(u -> u.convert(nanos, NANOSECONDS) > 0)

.findFirst();

TimeUnit unit = first.isPresent() ? first.get() : NANOSECONDS;

double value = (double) nanos / NANOSECONDS.convert(1, unit);

return String.format("%.4g %s", value, unit.name().toLowerCase());

}

private static long getTimeElapsed (long startTime) {

return System.nanoTime() - startTime;

}

private static void setPrintStreamDisabled (boolean b) {

if (b) {

System.setOut(blankPrintStream);

} else {

System.setOut(originalPrintStream);

}

}

private static final PrintStream originalPrintStream = System.out;

private static PrintStream blankPrintStream = new PrintStream(new OutputStream() {

public void write (int b) {

}

@Override

public void write (byte[] b, int off, int len) throws IOException {

}

});

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class SkipExample {

public static void main (String[] args) {

PerformanceTestUtil.runTest("unordered parallel skip", () -> {

IntStream intStream = IntStream.range(1, 100000000);

intStream.unordered().parallel().skip(1000).toArray();

});

PerformanceTestUtil.runTest("ordered parallel skip", () -> {

IntStream intStream = IntStream.range(1, 100000000);

intStream.parallel().skip(1000).toArray();

});

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.HashSet;

import java.util.Set;

import java.util.stream.IntStream;

public class SortedExample {

public static void main (String[] args) {

Set<Integer> list = new HashSet<>(Arrays.asList(2, 1, 3));

Object[] objects = list.stream().sorted().toArray();

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

}

}

package com.logicbig.example;

import java.util.HashSet;

import java.util.Set;

import java.util.stream.IntStream;

public class StatefulExample {

public static void main (String[] args) {

for (int i = 0; i < 10; i++) {

Set<Integer> seen = new HashSet<>();

IntStream stream = IntStream.of(1, 2, 1, 2, 3, 4, 4, 5);

int sum = stream.parallel().map(

//stateful behavioral parameter.

e -> {

try {//making it bit slow for more thread

//interference changes

Thread.sleep(10);

} catch (InterruptedException e1) {

e1.printStackTrace();

}

if (seen.add(e))

return e;

else

return 0;

}).sum();

System.out.println(sum);

}

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class StatefulExample2 {

private static int count = 0;

public static void main (String[] args) {

for (int i = 0; i < 5; i++) {

process();

}

}

private static void process () {

count = 0;

IntStream stream = IntStream.range(1, 1000);

//finding the sum of even numbers

int sum = stream.parallel()

.filter(i -> {

boolean b = i % 2 == 0;

if (b) {

count++;//updating count hence making it stateful.

}

return b;

})

.sum();

System.out.printf("sum :%d count:%d%n", sum, count);

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class StatefulFixExample {

public static void main (String[] args) {

for (int i = 0; i < 10; i++) {

IntStream stream = IntStream.of(1, 2, 1, 2, 3, 4, 4, 5);

int sum = stream.parallel().distinct().sum();

System.out.println(sum);

}

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class StatefulFixExample2 {

public static void main (String[] args) {

for (int i = 0; i < 5; i++) {

process();

}

}

private static void process () {

IntStream stream = IntStream.range(1, 1000);

//finding the even numbers

int[] even = stream.parallel()

.filter(i -> i % 2 == 0)

.toArray();

//finding sum

int sum = IntStream.of(even).parallel().sum();

System.out.printf("sum :%d count:%d%n", sum, even.length);

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class SideEffectWithPeek {

public static void main (String[] args) {

IntStream.range(0, 5)

.unordered()

.parallel()

.map(x -> x \* 2)

.peek(System.out::println)

.count();

}

}

package com.logicbig.example;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.IntStream;

public class SideEffectWrongUse {

public static void main (String[] args) {

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

IntStream.range(0, 150)

.parallel()

.filter(s -> s % 2 == 0)

.forEach(s -> results.add(s));//stateful side effect

//not thread safe

System.out.println(results);

}

}

package com.logicbig.example;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.stream.Stream;

/\*\*

\* Even though this is thread safe, but the result is non-deterministic

\*/

public class SideEffectWrongUse2 {

public static void main (String[] args) {

List<Integer> lengths = Collections.synchronizedList(new ArrayList<>());

Stream.of("Banana", "Pear", "Apple")

.peek(SideEffectWrongUse2::longTask)//applying side effect

.parallel()

.mapToInt(s -> s.length())

.forEach(lengths::add);//collecting via side effect

// updating state

System.out.println(lengths);

}

private static void longTask (String s) {

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.stream.Stream;

/\*\*

\* Even though this is thread safe, but the result is non-deterministic

\*/

public class SideEffectWrongUse2Fix {

public static void main(String[] args) {

int[] lengths = Stream.of("Banana", "Pear", "Apple")

.peek(SideEffectWrongUse2Fix::longTask)//applying side effect

.parallel()

.mapToInt(s -> s.length())

.toArray();

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

}

private static void longTask(String s) {

try {//some stateless task simulation. e.g. sending email

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

package com.logicbig.example;

import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.IntStream;

public class SideEffectWrongUseFix {

public static void main (String[] args) {

IntStream stream = IntStream.range(0, 1000);

List<Integer> list = stream.parallel()

.filter(s -> s % 2 == 0)

.boxed()

.collect(Collectors.toList());

System.out.println(list);

}

}

package com.logicbig.example;

import java.util.stream.LongStream;

public class AverageExample {

public static void main (String[] args) {

double v = LongStream.range(1, 10).average().orElse(-1);

System.out.println(v);

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class CountExample {

public static void main (String[] args) {

runCount();

runEquivalentReduce();

runEquivalentSum();

}

private static void runCount () {

long c = Stream.of("banana", "pie", "apple").count();

System.out.println(c);

}

private static void runEquivalentReduce () {

long sum = Stream.of("banana", "pie", "apple")

.mapToLong(s -> 1L)

.reduce(0, Long::sum);

System.out.println(sum);

}

private static void runEquivalentSum () {

long sum = Stream.of("banana", "pie", "apple")

.mapToLong(s -> 1L).sum();

System.out.println(sum);

}

}

package com.logicbig.example;

import java.util.Optional;

import java.util.stream.Stream;

public class MaxExample {

public static void main (String[] args) {

runMax();

runEquivalentReduce();

}

private static void runMax () {

String s = Stream.of("banana", "pie", "apple")

.max(String::compareTo) //dictionary order

.orElse("None");

System.out.println(s);

}

private static void runEquivalentReduce () {

Optional<String> reduce = Stream.of("apple", "banana", "pie")

.reduce((s, s2) -> s.compareTo(s2) > 0 ? s : s2);

System.out.println(reduce.get());

}

}

package com.logicbig.example;

import java.util.Optional;

import java.util.stream.Stream;

public class MinExample {

public static void main (String[] args) {

runMin();

runEquivalentReduce();

}

private static void runMin () {

String s = Stream.of("banana", "pie", "apple")

.min(String::compareTo) //dictionary order

.orElse("None");

System.out.println(s);

}

private static void runEquivalentReduce () {

Optional<String> reduce = Stream.of("apple", "banana", "pie")

.reduce((s, s2) -> s.compareTo(s2) <= 0 ? s : s2);

System.out.println(reduce.get());

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class ReduceExample1 {

public static void main (String[] args) {

int i = IntStream.range(1, 6)

.reduce((a, b) -> a \* b)

.orElse(-1);

System.out.println(i);

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class ReduceExample2 {

public static void main (String[] args) {

int i = IntStream.range(1, 6)

.parallel()

.reduce(1, (a, b) -> a \* b);

System.out.println(i);

}

}

package com.logicbig.example;

import java.util.stream.Stream;

public class ReduceExample3 {

public static void main (String[] args) {

int i = Stream.of("2", "3", "4", "5")

.parallel()

.reduce(0, (integer, s) -> Integer.sum(integer, Integer.parseInt(s)),

(integer, integer2) -> Integer.sum(integer, integer2));

System.out.println(i);

}

}

package com.logicbig.example;

import java.util.stream.DoubleStream;

public class SumExample {

public static void main (String[] args) {

runSum();

runEquivalentReduce();

}

private static void runSum () {

double sum = DoubleStream.of(1.1, 1.5, 2.5, 5.4).sum();

System.out.println(sum);

}

private static void runEquivalentReduce () {

double sum = DoubleStream.of(1.1, 1.5, 2.5, 5.4)

.reduce(0, Double::sum);

System.out.println(sum);

}

}

package com.logicbig.example;

import java.util.IntSummaryStatistics;

import java.util.stream.IntStream;

public class SummaryStatisticsExample {

public static void main (String[] args) {

IntSummaryStatistics s = IntStream.rangeClosed(1, 10)

.summaryStatistics();

System.out.println(s);

}

}

package com.logicbig.example;

import java.util.Collections;

import java.util.Set;

import java.util.function.BiConsumer;

import java.util.function.BinaryOperator;

import java.util.function.Function;

import java.util.function.Supplier;

import java.util.stream.Collector;

import java.util.stream.Stream;

public class CollectorExample {

public static void main (String[] args) {

String s = Stream.of("Mike", "Nicki", "John").collect(new

MyCollector());

System.out.println(s);

}

private static class MyCollector implements

Collector<String, StringBuilder, String> {

@Override

public Supplier<StringBuilder> supplier () {

return StringBuilder::new;

}

@Override

public BiConsumer<StringBuilder, String> accumulator () {

return (sb, s) -> sb.append(" ").append(s);

}

@Override

public BinaryOperator<StringBuilder> combiner () {

return (sb1, sb2) -> sb1.append(sb2);

}

@Override

public Function<StringBuilder, String> finisher () {

return stringBuilder -> stringBuilder.toString();

}

@Override

public Set<Characteristics> characteristics () {

return Collections.emptySet();

}

}

}

package com.logicbig.example;

import java.util.EnumSet;

import java.util.Set;

import java.util.function.BiConsumer;

import java.util.function.BinaryOperator;

import java.util.function.Function;

import java.util.function.Supplier;

import java.util.stream.Collector;

import java.util.stream.Stream;

public class CollectorExample2 {

public static void main (String[] args) {

String s = Stream.of("Mike", "Nicki", "John")

.parallel()

.unordered()

.collect(new MyCollector());

System.out.println(s);

}

private static class MyCollector implements

Collector<String, StringBuffer, String> {

@Override

public Supplier<StringBuffer> supplier () {

return () -> {

System.out.println("supplier call");

return new StringBuffer();

};

}

@Override

public BiConsumer<StringBuffer, String> accumulator () {

return (sb, s) -> {

System.out.println("accumulator function call,"

+ " accumulator container: "

+ System.identityHashCode(sb)

+ " thread: "

+ Thread.currentThread().getName()

+ ", processing: " + s);

sb.append(" ").append(s);

};

}

@Override

public BinaryOperator<StringBuffer> combiner () {

return (stringBuilder, s) -> {

System.out.println("combiner function call");

return stringBuilder.append(s);

};

}

@Override

public Function<StringBuffer, String> finisher () {

return stringBuilder -> stringBuilder.toString();

}

@Override

public Set<Characteristics> characteristics () {

// return Collections.emptySet();

return EnumSet.of(Characteristics.CONCURRENT);

}

}

}

package com.logicbig.example;

import java.util.EnumSet;

import java.util.Set;

import java.util.function.BiConsumer;

import java.util.function.BinaryOperator;

import java.util.function.Function;

import java.util.function.Supplier;

import java.util.stream.Collector;

import java.util.stream.Stream;

public class CollectorExample3 {

public static void main (String[] args) {

String s = Stream.of("Mike", "Nicki", "John")

.parallel()

.collect(new MyCollector());

System.out.println(s);

}

private static class MyCollector implements

Collector<String, StringBuffer, String> {

@Override

public Supplier<StringBuffer> supplier () {

return () -> {

System.out.println("supplier call");

return new StringBuffer();

};

}

@Override

public BiConsumer<StringBuffer, String> accumulator () {

return (sb, s) -> {

System.out.println("accumulator function call,"

+ " accumulator container: "

+ System.identityHashCode(sb)

+ " thread: "

+ Thread.currentThread().getName()

+ ", processing: " + s);

sb.append(" ").append(s);

};

}

@Override

public BinaryOperator<StringBuffer> combiner () {

return (stringBuilder, s) -> {

System.out.println("combiner function call");

return stringBuilder.append(s);

};

}

@Override

public Function<StringBuffer, String> finisher () {

return stringBuilder -> stringBuilder.toString();

}

@Override

public Set<Characteristics> characteristics () {

// return Collections.emptySet();

return EnumSet.of(Characteristics.CONCURRENT

, Characteristics.UNORDERED);

}

}

}

package com.logicbig.example;

import java.util.ArrayList;

import java.util.EnumSet;

import java.util.List;

import java.util.Set;

import java.util.function.BiConsumer;

import java.util.function.BinaryOperator;

import java.util.function.Function;

import java.util.function.Supplier;

import java.util.stream.Collector;

import java.util.stream.Stream;

public class CollectorExample4 {

public static void main (String[] args) {

List<String> s = Stream.of("Mike", "Nicki", "John")

.parallel()

.collect(new MyCollector());

System.out.println(s);

}

private static class MyCollector implements

Collector<String, List<String>, List<String>> {

@Override

public Supplier<List<String>> supplier () {

return ArrayList::new;

}

@Override

public BiConsumer<List<String>, String> accumulator () {

return List::add;

}

@Override

public BinaryOperator<List<String>> combiner () {

return (list, list2) -> {

list.addAll(list2);

return list;

};

}

@Override

public Function<List<String>, List<String>> finisher () {

return null;

}

@Override

public Set<Characteristics> characteristics () {

// return Collections.emptySet();

return EnumSet.of(Characteristics.IDENTITY\_FINISH);

}

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.List;

public class MutableReductionExample {

public static void main (String[] args) {

List<String> list = Arrays.asList("Mike", "Nicki", "John");

runMutableCollect(list);

runImmutableReduce(list);

}

private static void runMutableCollect (List<String> list) {

String s = list.stream().collect(StringBuilder::new,

(sb, s1) -> sb.append(" ").append(s1),

(sb1, sb2) -> sb1.append(sb2.toString())).toString();

System.out.println(s);

}

private static void runImmutableReduce (List<String> list) {

String s = list.stream().reduce("", (s1, s2) -> s1 + " " + s2);

System.out.println(s);

}

}

package com.logicbig.example;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.IntStream;

public class MutableReductionExample2 {

public static void main (String[] args) {

IntStream stream = IntStream.range(1, 100);

List<Integer> list = stream.parallel()

.filter(i -> i % 10 == 0)

.collect(ArrayList::new, ArrayList::add

, ArrayList::addAll);

System.out.println(list);

}

}

package com.logicbig.example;

import java.util.Arrays;

import java.util.stream.DoubleStream;

import java.util.stream.IntStream;

import java.util.stream.LongStream;

public class ForEachExample {

public static void main (String[] args) {

final int[] ints = IntStream.range(0, 5).toArray();

PerformanceTestUtil.runTest("forEach() method", () -> {

Arrays.stream(ints).parallel().forEach(i -> doSomething(i));

});

PerformanceTestUtil.runTest("forEachOrdered() method", () -> {

Arrays.stream(ints).parallel().forEachOrdered(i -> doSomething(i));

});

}

private static void doSomething (int i) {

try {

Thread.sleep(10);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.printf("%s, ", i);

}

}

package com.logicbig.example;

import java.util.HashSet;

import java.util.Set;

import java.util.stream.IntStream;

public class StatefulExample {

public static void main (String[] args) {

for (int i = 0; i < 10; i++) {

Set<Integer> seen = new HashSet<>();

IntStream stream = IntStream.of(1, 2, 1, 2, 3, 4, 4, 5);

int sum = stream.parallel().map(

//stateful behavioral parameter.

e -> {

try {//making it bit slow for more thread

//interference changes

Thread.sleep(10);

} catch (InterruptedException e1) {

e1.printStackTrace();

}

if (seen.add(e))

return e;

else

return 0;

}).sum();

System.out.println(sum);

}

}

}

package com.logicbig.example;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.stream.Stream;

/\*\*

\* Even though this is thread safe, but the result is non-deterministic

\*/

public class SideEffectWrongUse2 {

public static void main (String[] args) {

List<Integer> lengths = Collections.synchronizedList(new ArrayList<>());

Stream.of("Banana", "Pear", "Apple")

.peek(SideEffectWrongUse2::longTask)//applying side effect

.parallel()

.mapToInt(s -> s.length())

.forEach(lengths::add);//collecting via side effect

// updating state

System.out.println(lengths);

}

private static void longTask (String s) {

try {

Thread.sleep(100);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

package com.logicbig.example;

import java.util.stream.IntStream;

public class ReduceExample1 {

public static void main (String[] args) {

int i = IntStream.range(1, 6)

.reduce((a, b) -> a \* b)

.orElse(-1);

System.out.println(i);

}

}

package com.logicbig.example;

import java.util.Collections;

import java.util.Set;

import java.util.function.BiConsumer;

import java.util.function.BinaryOperator;

import java.util.function.Function;

import java.util.function.Supplier;

import java.util.stream.Collector;

import java.util.stream.Stream;

public class CollectorExample {

public static void main (String[] args) {

String s = Stream.of("Mike", "Nicki", "John").collect(new

MyCollector());

System.out.println(s);

}

private static class MyCollector implements

Collector<String, StringBuilder, String> {

@Override

public Supplier<StringBuilder> supplier () {

return StringBuilder::new;

}

@Override

public BiConsumer<StringBuilder, String> accumulator () {

return (sb, s) -> sb.append(" ").append(s);

}

@Override

public BinaryOperator<StringBuilder> combiner () {

return (sb1, sb2) -> sb1.append(sb2);

}

@Override

public Function<StringBuilder, String> finisher () {

return stringBuilder -> stringBuilder.toString();

}

@Override

public Set<Characteristics> characteristics () {

return Collections.emptySet();

}

}

}