Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Tutorials

By *mkyong* | Last updated: March 11, 2020
Viewed: 0 | +0 pv/w

< Previous

Next >

A series of Java 8 tips and examples, hope you like it.

FAQs

Some common asked questions.

- *Java 8 forEach examples*
- *Java 8 Convert List to Map*
- *Java 8 Lambda : Comparator example*
- *Java 8 method references, double colon (::) operator*
- *Java 8 Streams filter examples*
- *Java 8 Streams map() examples*

1. Functional Interface

Java 8 introduced `@FunctionalInterface`, an interface that has exactly one abstract method.

The compiler will treat any interfaces meeting the *definition of a functional interface* as a functional interface; it means the `@FunctionalInterface` annotation is optional.

Let us see the six basic function interfaces.

Interface	Signature	Examples
`UnaryOperator<T>`	`T apply(T t)`	`String::toLowerCase`, `Math::tan`
`BinaryOperator<T>`	`T apply(T t1, T t2)`	`BigInteger::add`, `Math::pow`
`Function<T, R>`	`R apply(T t)`	`Arrays::asList`, `Integer::toBinaryString`
`Predicate<T, U>`	`boolean test(T t, U u)`	`String::isEmpty`, `Character::isDigit`
`Supplier<T>`	`T get()`	`LocalDate::now`, `Instant::now`

Interface	Signature	Examples
`Consumer<T>`	`void accept(T t)`	`System.out::println`, `Error::printStackTrace`

- *Java 8 Function examples*
- *Java 8 BiFunction examples*
- *Java 8 BinaryOperator examples*
- *Java 8 UnaryOperator Examples*
- *Java 8 Predicate examples*
- *Java 8 BiPredicate examples*
- *Java 8 Consumer examples*
- *Java 8 BiConsumer examples*
- *Java 8 Supplier Examples*

2. Lambda Expressions & Method References

Java 8 introduced lambda expressions to provide the implementation of the abstract method of a functional interface.

Further Reading >>> *Java 8 Lambda : Comparator example*

Review the JDK `Iterable` class, it has a `default` method `forEach()` to accept a function interface `Consumer`

Iterable.java

```
public interface Iterable<T> {

    default void forEach(Consumer<? super T> action) {
        Objects.requireNonNull(action);
        for (T t : this) {
            action.accept(t);
        }
    }

   //...
}
```

First, we can provide an anonymous class as the `forEach` implementation.

```
List<String> list = Arrays.asList("node", "java", "python", "ruby");
list.forEach(new Consumer<String>() {       // anonymous class
    @Override
    public void accept(String str) {
        System.out.println(str);
    }
});
```

Alternatively, we can use a lambda expression to shorten the code like this:

```
List<String> list = Arrays.asList("node", "java", "python", "ruby");
list.forEach(str -> System.out.println(str)); // lambda expressions
```

To gain better readability, we can replace lambda expression with method reference.

```
List<String> list = Arrays.asList("node", "java", "python", "ruby");
list.forEach(System.out::println);            // method references
```

Further Reading >>> *Java 8 method references, double colon (::) operator*

> **Note**
> Both lambda expression or method reference does nothing but just another way call to an existing method. With method reference, it gains better readability.

3. Streams

- *Java 8 Streams filter examples*
- *Java 8 Streams map() examples*
- *Java 8 flatMap example*
- *Java 8 Parallel Streams Examples*
- *Java 8 Stream.iterate examples*
- *Java 8 Stream Collectors groupingBy examples*
- *Java 8 Filter a null value from a Stream*
- *Java 8 Convert a Stream to List*
- *Java 8 Stream findFirst() and findAny()*
- *Java 8 Stream.reduce() examples*
- *Java 8 Convert a Stream to List*
- *Java 8 How to Sum BigDecimal using Stream?*
- *Java 8 Stream – Read a file line by line*
- *Java 8 Stream – Convert List<List<String>> to List<String>*
- *Java 8 Stream – The peek() is not working with count()?*
- *Java 8 Should we close the Stream after use?*
- *Java 8 Convert a Stream to Array*
- *Java 8 How to convert IntStream to Integer array*
- *Java 8 How to convert IntStream to int or int array*
- *Java 8 How to sort list with stream.sorted()*
- *Java – How to sum all the stream integers*
- *Java – How to convert a primitive Array to List*
- *Java – How to convert Array to Stream*
- *Java – Stream has already been operated upon or closed*

4. New Date Time APIs

In the old days, we use `Date` and `Calendar` APIs to represent and manipulate date.

- `java.util.Date` – date and time, print with default time-zone.
- `java.util.Calendar` – date and time, more methods to manipulate date.
- `java.text.SimpleDateFormat` – formatting (date -> text), parsing (text -> date) for date and calendar.

Java 8 created a series of new date and time APIs in `java.time` package. (*JSR310* and inspired by Joda-time).

- `java.time.LocalDate` – date without time, no time-zone.
- `java.time.LocalTime` – time without date, no time-zone.

- `java.time.LocalDateTime` – date and time, no time-zone.
- `java.time.ZonedDateTime` – date and time, with time-zone.
- `java.time.DateTimeFormatter` – formatting (date -> text), parsing (text -> date) for java.time.
- `java.time.Instant` – date and time for machine, seconds passed since the Unix epoch time (midnight of January 1, 1970 UTC)
- `java.time.Duration` – Measures time in seconds and nanoseconds.
- `java.time.Period` – Measures time in years, months and days.
- `java.time.TemporalAdjuster` – Adjust date.
- `java.time.OffsetDateTime` – {update me}

Examples…

- *Java – How to get current date time*
- *Java – How to get current timestamp*
- *Java – How to convert String to a Date*
- *Java 8 – Duration and Period examples*
- *Java 8 – How to convert String to LocalDate*
- *Java 8 – How to format LocalDateTime*
- *Java 8 – Convert Instant to LocalDateTime*
- *Java 8 – Convert Instant to ZoneDateTime*
- *Java 8 – Convert Date to LocalDate and LocalDateTime*
- *Java 8 – ZonedDateTime examples*
- *Java – Convert date and time between timezone*
- *Java – How to add days to current date*
- *Java 8 – TemporalAdjusters examples*
- *Java 8 – Convert Epoch time milliseconds to LocalDate or LocalDateTime*
- *Java 8 – Difference between two LocalDate or LocalDateTime*
- *Java 8 – How to calculate days between two dates?*
- *Java 8 – How to parse date with "dd MMM" (02 Jan), without year?*
- *Java 8 – Convert LocalDate and LocalDateTime to Date*
- *Java 8 – Unable to obtain LocalDateTime from TemporalAccessor*
- *Java 8 – Convert ZonedDateTime to Timestamp*
- *Java – Display all ZoneId and its UTC offset*
- *Java 8 – Convert LocalDateTime to Timestamp*
- *Java – How to change date format in a String*
- *Check if the date is older than 6 months*
- *Java – How to compare dates*
- *Java – How to calculate elapsed time*
- *Java 8 – MinguoDate examples (Taiwan calendar)*
- *Java 8 – HijrahDate, How to calculate the Ramadan date (Islamic calendar)*
- *Java Date Time Tutorials*

5. Java 8 Tips

- *Java 8 Optional In Depth*
- *Java 8 How to sort a Map*
- *Java 8 Convert List to Map*
- *Java 8 Filter a Map examples*
- *Java 8 Convert Map to List*
- *Java 8 StringJoiner example*

- *Java 8 Math Exact examples*
- *Java 8 forEach print with Index*
- *Java 8 Convert Optional<String> to String*
- *Java – How to print a Pyramid*
- *Java – Check if Array contains a certain value?*
- *Java – How to join Arrays*
- *Java – Generate random integers in a range*
- *Java – How to print a name 10 times?*
- *Java – How to search a string in a List?*
- *Java – How to get keys and values from Map*
- *Java – Convert File to String*
- *Java – Convert Array to ArrayList*
- *Java – How to check if a String is numeric*
- *Java – How to join List String with commas*
- *Java – Convert comma-separated String to a List*
- *Java Prime Numbers examples*
- *How to tell Maven to use Java 8*
- *java.lang.UnsupportedClassVersionError*
- *Java Fibonacci examples*
- *How to loop a Map in Java*
- *Java Regular Expression Examples*
- *How to read file in Java – BufferedReader*

Installation

- *How to install Oracle JDK 8 on CentOS*
- *How to Install Oracle JDK 8 On Debian*
- *How to install Java on Mac OS*

References

- *What's New in JDK 8*
- *Why do we need a new date and time library?*
- *JSR 310: Date and Time API*
- *Stream JavaDoc*

< Previous Next >

Related Articles

- *Java Date Time Tutorials*
- *How to read file in Java - BufferedReader*
- *How to write a UTF-8 file in Java*
- *How to read a UTF-8 file in Java*
- *How to compare dates in Java*

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Function Examples

By *mkyong* | Last updated: February 26, 2020

In Java 8, *Function* is a functional interface; it takes an argument (object of type T) and returns an object (object of type R). The argument and output can be a different type.

Function.java

```
@FunctionalInterface
public interface Function<T, R> {

      R apply(T t);

}
```

- T – Type of the input to the function.
- R – Type of the result of the function.

> **Further Reading**
> *Java 8 BiFunction Examples*

1. Function<T, R>

1.1 This example takes a `<T> String` and returns the length of the string as `<R> Integer`.

Java8Function1.java

```
package com.mkyong;

import java.util.function.Function;

public class JavaMoney {

    public static void main(String[] args) {

        Function<String, Integer> func = x -> x.length();

        Integer apply = func.apply("mkyong");   // 6

        System.out.println(apply);

    }

}
```

Output

```
6
```

2. Chain Function<T, R>

2.1 This example chains the `Function` with `andThen()`.

Java8Function2.java

```
package com.mkyong;

import java.util.function.Function;

public class Java8Function2 {

    public static void main(String[] args) {

        Function<String, Integer> func = x -> x.length();

        Function<Integer, Integer> func2 = x -> x * 2;

        Integer result = func.andThen(func2).apply("mkyong");   // 12

        System.out.println(result);

    }

}
```

Output

```
12
```

3. List -> Map

3.1 This example accepts `Function` as an argument, convert a `List` into a `Map`.

Java8Function3.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Function;

public class Java8Function3 {

    public static void main(String[] args) {

        Java8Function3 obj = new Java8Function3();

        List<String> list = Arrays.asList("node", "c++", "java", "javascript");

        // lambda
        Map<String, Integer> map = obj.convertListToMap(list, x -> x.length());

        System.out.println(map);    // {node=4, c++=3, java=4, javascript=10}

        // method reference
        Map<String, Integer> map2 = obj.convertListToMap(list, obj::getLength);

        System.out.println(map2);
    }

    public <T, R> Map<T, R> convertListToMap(List<T> list, Function<T, R> func)
{

        Map<T, R> result = new HashMap<>();
        for (T t : list) {
            result.put(t, func.apply(t));
        }
        return result;

    }

    public Integer getLength(String str) {
        return str.length();
    }

}
```

Output

```
{node=4, c++=3, java=4, javascript=10}
{node=4, c++=3, java=4, javascript=10}
```

4. List -> List

4.1 This example accepts `Function` as an argument, convert a `List` of String into another `List` of String, which was hashed with SHA256.

```
Java8Function4
```

```
package com.mkyong;

import org.apache.commons.codec.digest.DigestUtils;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Function;

public class Java8Function4 {

    public static void main(String[] args) {

        Java8Function4 obj = new Java8Function4();

        List<String> list = Arrays.asList("node", "c++", "java", "javascript");

        // lambda
        //List<String> result = obj.map(list, x -> obj.sha256(x));

        // method reference
        List<String> result = obj.map(list, obj::sha256);

        result.forEach(System.out::println);

    }

    public <T, R> List<R> map(List<T> list, Function<T, R> func) {

        List<R> result = new ArrayList<>();
        for (T t : list) {
            result.add(func.apply(t));
        }
        return result;

    }

    // sha256 a string
    public String sha256(String str) {
        return DigestUtils.sha256Hex(str);
    }

}
```

Output

```
545ea538461003efdc8c81c244531b003f6f26cfccf6c0073b3239fdedf49446
cedb1bac7efcd7db47e9f2f2250a7c832aba83b410dd85766e2aea6ec9321e51
38a0963a6364b09ad867aa9a66c6d009673c21e182015461da236ec361877f77
eda71746c01c3f465ffd02b6da15a6518e6fbc8f06f1ac525be193be5507069d
```

P.S The `DigestUtils.sha256Hex` is from the `commons-codec`.

pom.xml

```
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.14</version>
</dependency>
```

References

- *Function JavaDoc*
- *Java 8 Tutorials*
- *Java 8 Predicate Examples*
- *Java 8 BiPredicate Examples*
- *Java 8 Consumer Examples*
- *Java 8 BiConsumer Examples*

Related Articles

- *Java 8 BiFunction Examples*
- *Java 8 UnaryOperator Examples*
- *Java 8 method references, double colon (::) operat*
- *Java SHA-256 and SHA3-256 Hashing Example*
- *Java - Convert String to int*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Join the discussion

3000

5 COMMENTS Most Voted

Sudharson 8 months ago

When i try the andThen method that you have given in your example, I get the below error. Kindly help me understand what is the issue here. The Function example is clear, however the Chain Function example gives me the below error.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 BiFunction Examples

By *mkyong* | Last updated: February 26, 2020

In Java 8, *BiFunction* is a functional interface; it takes two arguments and returns an object.

BiFunction.java

```
@FunctionalInterface
public interface BiFunction<T, U, R> {

      R apply(T t, U u);

}
```

- T – Type of the first argument to the function.
- U – Type of the second argument to the function.
- R – Type of the result of the function.

1. BiFunction<T, U, R>

1.1 This example takes two `Integers` and returns an `Integer`, `Double` or `List`

Java8BiFunction1.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.List;
import java.util.function.BiFunction;

public class Java8BiFunction1 {

    public static void main(String[] args) {

        // takes two Integers and return an Integer
        BiFunction<Integer, Integer, Integer> func = (x1, x2) -> x1 + x2;

        Integer result = func.apply(2, 3);

        System.out.println(result); // 5

        // take two Integers and return an Double
        BiFunction<Integer, Integer, Double> func2 = (x1, x2) -> Math.pow(x1,
x2);

        Double result2 = func2.apply(2, 4);

        System.out.println(result2);    // 16.0

        // take two Integers and return a List<Integer>
        BiFunction<Integer, Integer, List<Integer>> func3 = (x1, x2) ->
Arrays.asList(x1 + x2);

        List<Integer> result3 = func3.apply(2, 3);

        System.out.println(result3);

    }

}
```

Output

```
5
16.0
[5]
```

2. BiFunction<T, U, R> + Function<T, R>

2.1 This `BiFunction` takes two `Integer` and returns a `Double`, and uses `andThen()` to chain it with a `Function` to convert the `Double` into a `String`.

```
Java8BiFunction2a.java
```

```
package com.mkyong;

import java.util.function.BiFunction;
import java.util.function.Function;

public class Java8BiFunction2a {

    public static void main(String[] args) {

        // Math.pow(a1, a2) returns Double
        BiFunction<Integer, Integer, Double> func1 = (a1, a2) -> Math.pow(a1,
a2);

        // takes Double, returns String
        Function<Double, String> func2 = (input) -> "Result : " +
String.valueOf(input);

        String result = func1.andThen(func2).apply(2, 4);

        System.out.println(result);

    }

}
```

Output

```
Result : 16.0
```

2.2 This example converts the above program into a method that accepts `BiFunction` and `Function` as arguments and chains it together.

```
Java8BiFunction2b.java
```

```
package com.mkyong;

import java.util.function.BiFunction;
import java.util.function.Function;

public class Java8BiFunction2b {

    public static void main(String[] args) {

        String result = powToString(2, 4,
                (a1, a2) -> Math.pow(a1, a2),
                (r) -> "Result : " + String.valueOf(r));

        System.out.println(result); // Result : 16.0

    }

    public static <R> R powToString(Integer a1, Integer a2,
                                    BiFunction<Integer, Integer, Double> func,
                                    Function<Double, R> func2) {

        return func.andThen(func2).apply(a1, a2);

    }

}
```

Output

```
Result : 16.0
```

2.3 This example converts the above method into a generic method:

```
public static <A1, A2, R1, R2> R2 convert(A1 a1, A2 a2,
                                          BiFunction<A1, A2, R1> func,
                                          Function<R1, R2> func2) {

    return func.andThen(func2).apply(a1, a2);

}
```

A lot of possibilities in this generic method, let see:

```
Java8BiFunction2c.java
```

```
package com.mkyong;

import java.util.function.BiFunction;
import java.util.function.Function;

public class Java8BiFunction2c {

    public static void main(String[] args) {

        // Take two Integers, pow it into a Double, convert Double into a
String.
        String result = convert(2, 4,
                (a1, a2) -> Math.pow(a1, a2),
                (r) -> "Pow : " + String.valueOf(r));

        System.out.println(result);     // Pow : 16.0

        // Take two Integers, multiply into an Integer, convert Integer into a
String.
        String result2 = convert(2, 4,
                (a1, a2) -> a1 * a1,
                (r) -> "Multiply : " + String.valueOf(r));

        System.out.println(result2);    // Multiply : 4

        // Take two Strings, join both, join "cde"
        String result3 = convert("a", "b",
                (a1, a2) -> a1 + a2,
                (r) -> r + "cde");      // abcde

        System.out.println(result3);

        // Take two Strings, join both, convert it into an Integer
        Integer result4 = convert("100", "200",
                (a1, a2) -> a1 + a2,
                (r) -> Integer.valueOf(r));

        System.out.println(result4);    // 100200

    }

    public static <A1, A2, R1, R2> R2 convert(A1 a1, A2 a2,
                                              BiFunction<A1, A2, R1> func,
                                              Function<R1, R2> func2) {

        return func.andThen(func2).apply(a1, a2);

    }

}
```

Output

```
Pow : 16.0
Multiply : 4
abcde
100200
```

3. Factory

3.1 This example uses `BiFunction` to create an object, acts as a factory pattern.

Java8BiFunction3.java

```
package com.mkyong;

import java.util.function.BiFunction;

public class Java8BiFunction3 {

    public static void main(String[] args) {

        GPS obj = factory("40.741895", "-73.989308", GPS::new);
        System.out.println(obj);

    }

    public static <R extends GPS> R factory(String Latitude, String Longitude,
                                            BiFunction<String, String, R> func)
{
        return func.apply(Latitude, Longitude);
    }

}

class GPS {

    String Latitude;
    String Longitude;

    public GPS(String latitude, String longitude) {
        Latitude = latitude;
        Longitude = longitude;
    }

    public String getLatitude() {
        return Latitude;
    }

    public void setLatitude(String latitude) {
        Latitude = latitude;
    }

    public String getLongitude() {
        return Longitude;
    }

    public void setLongitude(String longitude) {
        Longitude = longitude;
    }

    @Override
    public String toString() {
        return "GPS{" +
                "Latitude='" + Latitude + '\'' +
                ", Longitude='" + Longitude + '\'' +
                '}';
    }
}
```

Output

```
GPS{Latitude='40.741895', Longitude='-73.989308'}
```

The `GPS::new` calls the following constructor, which accepts two arguments and return an object (GPS), so it matches with the `BiFunction` signature.

```
  public GPS(String latitude, String longitude) {
        Latitude = latitude;
        Longitude = longitude;
    }
```

4. More

4.1 Filtering a `List` by some conditions.

```
Java8BiFunction4.java
```

```
package com.mkyong;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.BiFunction;

public class Java8BiFunction4 {

    public static void main(String[] args) {

        Java8BiFunction4 obj = new Java8BiFunction4();

        List<String> list = Arrays.asList("node", "c++", "java", "javascript");

        List<String> result = obj.filterList(list, 3, obj::filterByLength);

        System.out.println(result);   // [node, java, javascript]

        List<String> result1 = obj.filterList(list, 3, (l1, size) -> {
            if (l1.length() > size) {
                return l1;
            } else {
                return null;
            }
        });

        System.out.println(result1);  // [node, java, javascript]

        List<String> result2 = obj.filterList(list, "c", (l1, condition) -> {
            if (l1.startsWith(condition)) {
                return l1;
            } else {
                return null;
            }
        });

        System.out.println(result2);  // [c++]

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

        List<Integer> result3 = obj.filterList(number, 2, (l1, condition) -> {
            if (l1 % condition == 0) {
                return l1;
            } else {
                return null;
            }
        });

        System.out.println(result3);  // [2, 4]

    }

    public String filterByLength(String str, Integer size) {
        if (str.length() > size) {
            return str;
        } else {
            return null;
        }
    }

    public <T, U, R> List<R> filterList(List<T> list1, U condition,
                                        BiFunction<T, U, R> func) {
```

```
        List<R> result = new ArrayList<>();

        for (T t : list1) {
            R apply = func.apply(t, condition);
            if (apply != null) {
                result.add(apply);
            }
        }

        return result;

    }

}
```

Output

```
[node, java, javascript]
[node, java, javascript]
[c++]
[2, 4]
```

References

- *Function JavaDoc*
- *BiFunction JavaDoc*
- *Java 8 Function Examples*
- *Java 8 Tutorials*
- *Java 8 Predicate Examples*
- *Java 8 BiPredicate Examples*
- *Java 8 Consumer Examples*
- *Java 8 BiConsumer Examples*

Related Articles

- *Java 8 method references, double colon (::) operat*
- *Java 8 Function Examples*
- *Java 8 BinaryOperator Examples*
- *Java 8 UnaryOperator Examples*
- *Java 8 Supplier Examples*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Java 8 BinaryOperator Examples

By *mkyong* | Last updated: March 2, 2020

In Java 8, *BinaryOperator* is a functional interface and it extends `BiFunction`.

The `BinaryOperator` takes two arguments of the same type and returns a result of the same type of its arguments.

BinaryOperator.java

```
@FunctionalInterface
public interface BinaryOperator<T> extends BiFunction<T,T,T> {
}
```

The `BiFunction` takes two arguments of any type, and returns a result of any type.

BiFunction.java

```
@FunctionalInterface
public interface BiFunction<T, U, R> {
      R apply(T t, U u);
}
```

1. BinaryOperator

1.1 In this example, the `BiFunction<Integer, Integer, Integer>` which accepts and returns the same type, can be replaced with `BinaryOperator<Integer>`.

Java8BinaryOperator1.java

```
package com.mkyong;

import java.util.function.BiFunction;
import java.util.function.BinaryOperator;

public class Java8BinaryOperator1 {

    public static void main(String[] args) {

        // BiFunction
        BiFunction<Integer, Integer, Integer> func = (x1, x2) -> x1 + x2;

        Integer result = func.apply(2, 3);

        System.out.println(result); // 5

        // BinaryOperator
        BinaryOperator<Integer> func2 = (x1, x2) -> x1 + x2;

        Integer result2 = func.apply(2, 3);

        System.out.println(result2); // 5

    }

}
```

Output

```
5
5
```

2. BinaryOperator as argument

2.1 This example simulates a `stream.reduce()` to sum all the `Integer`.

```
Java8BinaryOperator2.java
```

```
package com.mkyong;

import java.util.Arrays;
import java.util.List;
import java.util.function.BinaryOperator;

public class Java8BinaryOperator2 {

    public static void main(String[] args) {

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

        Integer result = math(Arrays.asList(numbers), 0, (a, b) -> a + b);

        System.out.println(result); // 55

        Integer result2 = math(Arrays.asList(numbers), 0, Integer::sum);

        System.out.println(result2); // 55

    }

    public static <T> T math(List<T> list, T init, BinaryOperator<T>
accumulator) {
        T result = init;
        for (T t : list) {
            result = accumulator.apply(result, t);
        }
        return result;
    }

}
```

Output

```
55
55
```

3. IntBinaryOperator

3.1 If the math operations involve primitive types like `int`, change to `IntBinaryOperator` for better performance.

```
Java8BinaryOperator3.java
```

```
package com.mkyong;

import java.util.function.IntBinaryOperator;
import java.util.stream.IntStream;

public class Java8BinaryOperator3 {

    public static void main(String[] args) {

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

        int result = math((numbers), 0, (a, b) -> a + b);

        System.out.println(result); // 55

        int result2 = math((numbers), 0, Integer::sum);

        System.out.println(result2); // 55

        IntStream
    }

    public static int math(int[] list, int init, IntBinaryOperator accumulator)
{
        int result = init;
        for (int t : list) {
            result = accumulator.applyAsInt(result, t);
        }
        return result;
    }

}
```

Output

```
55
55
```

4. BinaryOperator.maxBy() and BinaryOperator.minBy()

4.1 This example uses `BinaryOperator` and a custom `Comparator` to find the highest and lowest pay developer from a list of developers.

```
Java8BinaryOperator4.java
```

```
package com.mkyong;

import java.math.BigDecimal;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.function.BinaryOperator;

public class Java8BinaryOperator4 {

    public static void main(String[] args) {

        Developer dev1 = new Developer("jordan", BigDecimal.valueOf(9999));
        Developer dev2 = new Developer("jack", BigDecimal.valueOf(8888));
        Developer dev3 = new Developer("jaden", BigDecimal.valueOf(10000));
        Developer dev4 = new Developer("ali", BigDecimal.valueOf(2000));
        Developer dev5 = new Developer("mkyong", BigDecimal.valueOf(1));

        List<Developer> list = Arrays.asList(dev1, dev2, dev3, dev4, dev5);

        // 1. Create a Comparator
        Comparator<Developer> comparing =
Comparator.comparing(Developer::getSalary);

        // 2. BinaryOperator with a custom Comparator
        BinaryOperator<Developer> bo = BinaryOperator.maxBy(comparing);

        Developer result = find(list, bo);

        System.out.println(result);     // Developer{name='jaden',
salary=10000}

        // one line

        // find developer with highest pay
        Developer developer = find(list,
BinaryOperator.maxBy(Comparator.comparing(Developer::getSalary)));
        System.out.println(developer);  // Developer{name='jaden',
salary=10000}

        // find developer with lowest pay
        Developer developer2 = find(list,
BinaryOperator.minBy(Comparator.comparing(Developer::getSalary)));
        System.out.println(developer2); // Developer{name='mkyong', salary=1}

    }

    public static Developer find(List<Developer> list,
BinaryOperator<Developer> accumulator) {
        Developer result = null;
        for (Developer t : list) {
            if (result == null) {
                result = t;
            } else {
                result = accumulator.apply(result, t);
            }
        }
        return result;
    }

}
```

Developer.java

Java 8 UnaryOperator Examples

By *mkyong* | Last updated: March 2, 2020

In Java 8, *UnaryOperator* is a functional interface and it extends `Function`.

The `UnaryOperator` takes one argument, and returns a result of the same type of its arguments.

UnaryOperator.java

```
@FunctionalInterface
public interface UnaryOperator<T> extends Function<T, T> {
}
```

The `Function` takes one argument of any type and returns a result of any type.

Function.java

```
@FunctionalInterface
public interface Function<T, R> {
    R apply(T t);
}
```

Further Read *Java 8 Function Examples*

1. UnaryOperator

1.1 In this example, the `Function<Integer, Integer>` which accepts and returns the same type, can be replaced with `UnaryOperator<Integer>`.

Java8UnaryOperator1.java

```
package com.mkyong;

import java.util.function.Function;
import java.util.function.UnaryOperator;

public class Java8UnaryOperator1 {

    public static void main(String[] args) {

        Function<Integer, Integer> func = x -> x * 2;

        Integer result = func.apply(2);

        System.out.println(result);         // 4

        UnaryOperator<Integer> func2 = x -> x * 2;

        Integer result2 = func2.apply(2);

        System.out.println(result2);        // 4

    }

}
```

Output

```
4
4
```

2. UnaryOperator as argument

Java8UnaryOperator2.java

```
package com.mkyong;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.UnaryOperator;

public class Java8UnaryOperator2 {

    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        List<Integer> result = math(list, x -> x * 2);

        System.out.println(result); // [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]

    }

    public static <T> List<T> math(List<T> list, UnaryOperator<T> uo) {
        List<T> result = new ArrayList<>();
        for (T t : list) {
            result.add(uo.apply(t));
        }
        return result;
    }

}
```

Output

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

3. Chain UnaryOperator

Java8UnaryOperator3.java

```
package com.mkyong;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.UnaryOperator;

public class Java8UnaryOperator3 {

    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        List<Integer> result = math(list,
                x -> x * 2,
                x -> x + 1);

        System.out.println(result); // [3, 5, 7, 9, 11, 13, 15, 17, 19, 21]

    }

    public static <T> List<T> math(List<T> list,
                                   UnaryOperator<T> uo, UnaryOperator<T> uo2) {
        List<T> result = new ArrayList<>();
        for (T t : list) {
            result.add(uo.andThen(uo2).apply(t));
        }
        return result;
    }

}
```

Output

```
[3, 5, 7, 9, 11, 13, 15, 17, 19, 21]
```

References

- *UnaryOperator JavaDoc*
- *Function JavaDoc*
- *Java 8 Function Examples*
- *Java 8 Tutorials*

Related Articles

- *Java 8 method references, double colon (::) operat*
- *Java 8 Predicate Examples*
- *Java 8 BiPredicate Examples*
- *Java 8 Consumer Examples*
- *Java 8 BiConsumer Examples*

Search...

Java Tutorials
Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Predicate Examples

By *mkyong* | Last updated: February 11, 2020

In Java 8, *Predicate* is a functional interface, which accepts an argument and returns a boolean. Usually, it used to apply in a filter for a collection of objects.

```
@FunctionalInterface
public interface Predicate<T> {
  boolean test(T t);
}
```

Further Reading

Java 8 BiPredicate Examples

1. Predicate in filter()

`filter()` accepts predicate as argument.

Java8Predicate.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Java8Predicate {

    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        List<Integer> collect = list.stream().filter(x -> x >
5).collect(Collectors.toList());

        System.out.println(collect); // [6, 7, 8, 9, 10]

    }

}
```

Output

```
[6, 7, 8, 9, 10]
```

Java8Predicate.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class Java8Predicate {

    public static void main(String[] args) {

        Predicate<Integer> noGreaterThan5 =  x -> x > 5;

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        List<Integer> collect = list.stream()
                .filter(noGreaterThan5)
                .collect(Collectors.toList());

        System.out.println(collect); // [6, 7, 8, 9, 10]

    }

}
```

Output

```
[6, 7, 8, 9, 10]
```

2. Predicate.and()

2.1 Multiple filters.

Java8Predicate2.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class Java8Predicate2 {

    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        // multiple filters
        List<Integer> collect = list.stream()
                .filter(x -> x > 5 && x < 8).collect(Collectors.toList());

        System.out.println(collect);

    }

}
```

Output

```
[6, 7]
```

2.1 Replace with `Predicate.and()`

Java8Predicate2.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class Java8Predicate2 {

    public static void main(String[] args) {

        Predicate<Integer> noGreaterThan5 = x -> x > 5;
        Predicate<Integer> noLessThan8 = x -> x < 8;

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        List<Integer> collect = list.stream()
                .filter(noGreaterThan5.and(noLessThan8))
                .collect(Collectors.toList());

        System.out.println(collect);

    }

}
```

Output

```
[6, 7]
```

3. Predicate.or()

Java8Predicate3.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class Java8Predicate3 {

    public static void main(String[] args) {

        Predicate<String> lengthIs3 = x -> x.length() == 3;
        Predicate<String> startWithA = x -> x.startsWith("A");

        List<String> list = Arrays.asList("A", "AA", "AAA", "B", "BB", "BBB");

        List<String> collect = list.stream()
                .filter(lengthIs3.or(startWithA))
                .collect(Collectors.toList());

        System.out.println(collect);

    }

}
```

Output

```
[A, AA, AAA, BBB]
```

4. Predicate.negate()

Find all elements not start with ‘A’.

```
Java8Predicate4.java
```

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class Java8Predicate4 {

    public static void main(String[] args) {

        Predicate<String> startWithA = x -> x.startsWith("A");

        List<String> list = Arrays.asList("A", "AA", "AAA", "B", "BB", "BBB");

        List<String> collect = list.stream()
                .filter(startWithA.negate())
                .collect(Collectors.toList());

        System.out.println(collect);

    }

}
```

Output

```
[B, BB, BBB]
```

5. Predicate.test() in function

Predicate in function.

```
Java8Predicate5.java
```

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class Java8Predicate5 {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("A", "AA", "AAA", "B", "BB", "BBB");

        System.out.println(StringProcessor.filter(
                list, x -> x.startsWith("A")));                    // [A, AA,
AAA]

        System.out.println(StringProcessor.filter(
                list, x -> x.startsWith("A") && x.length() == 3)); // [AAA]

    }

}

class StringProcessor {
    static List<String> filter(List<String> list, Predicate<String> predicate)
{
        return
list.stream().filter(predicate::test).collect(Collectors.toList());
    }
}
```

Output

```
[A, AA, AAA]
[AAA]
```

6. Predicate Chaining

We can chain predicates together.

Java8Predicate6.java

```
package com.mkyong.java8;

import java.util.function.Predicate;

public class Java8Predicate6 {

    public static void main(String[] args) {

        Predicate<String> startWithA = x -> x.startsWith("a");

        // start with "a" or "m"
        boolean result = startWithA.or(x -> x.startsWith("m")).test("mkyong");
        System.out.println(result);     // true

        // !(start with "a" and length is 3)
        boolean result2 = startWithA.and(x -> x.length() ==
3).negate().test("abc");
        System.out.println(result2);    // false

    }

}
```

Output

```
true
false
```

7. Predicate in Object

Hosting.java

```
package com.mkyong.java8;

public class Hosting {

    private int Id;
    private String name;
    private String url;

    public Hosting(int id, String name, String url) {
        Id = id;
        this.name = name;
        this.url = url;
    }

    //... getters and setters, toString()
}
```

HostingRespository.java

```
package com.mkyong.java8;

import java.util.List;
import java.util.function.Predicate;
import java.util.stream.Collectors;

public class HostingRespository {

    public static List<Hosting> filterHosting(List<Hosting> hosting,
                                              Predicate<Hosting> predicate) {
        return hosting.stream()
                .filter(predicate)
                .collect(Collectors.toList());
    }

}
```

Java8Predicate7.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Predicate;

public class Java8Predicate7 {

    public static void main(String[] args) {

        Hosting h1 = new Hosting(1, "amazon", "aws.amazon.com");
        Hosting h2 = new Hosting(2, "linode", "linode.com");
        Hosting h3 = new Hosting(3, "liquidweb", "liquidweb.com");
        Hosting h4 = new Hosting(4, "google", "google.com");

        List<Hosting> list = Arrays.asList(new Hosting[]{h1, h2, h3, h4});

        List<Hosting> result = HostingRespository.filterHosting(list, x ->
x.getName().startsWith("g"));
        System.out.println("result : " + result);  // google

        List<Hosting> result2 = HostingRespository.filterHosting(list,
isDeveloperFriendly());
        System.out.println("result2 : " + result2); // linode

    }

    public static Predicate<Hosting> isDeveloperFriendly() {
        return n -> n.getName().equals("linode");
    }
}
```

Output

```
result : [Hosting{Id=4, name='google', url='google.com'}]
result2 : [Hosting{Id=2, name='linode', url='linode.com'}]
```

Done.

Java 8 BiPredicate Examples

By *mkyong* | Last updated: February 11, 2020

In Java 8, *BiPredicate* is a functional interface, which accepts two arguments and returns a boolean, basically this `BiPredicate` is same with the `Predicate`, instead, it takes 2 arguments for the test.

```
@FunctionalInterface
public interface BiPredicate<T, U> {
    boolean test(T t, U u);
}
```

Further Reading

Java 8 Predicate Examples

1. BiPredicate Hello World.

If the String length matches the provided length?

JavaBiPredicate1.java

```
package com.mkyong.java8;

import java.util.function.BiPredicate;

public class JavaBiPredicate1 {

    public static void main(String[] args) {

        BiPredicate<String, Integer> filter = (x, y) -> {
            return x.length() == y;
        };

        boolean result = filter.test("mkyong", 6);
        System.out.println(result);  // true

        boolean result2 = filter.test("java", 10);
        System.out.println(result2); // false
    }

}
```

Output

```
true
false
```

2. BiPredicate as function argument.

This example uses `BiPredicate` to filter bad domains by the domain name or threat score.

JavaBiPredicate2.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.BiPredicate;
import java.util.stream.Collectors;

public class JavaBiPredicate2 {

    public static void main(String[] args) {

        List<Domain> domains = Arrays.asList(new Domain("google.com", 1),
                new Domain("i-am-spammer.com", 10),
                new Domain("mkyong.com", 0),
                new Domain("microsoft.com", 2));

        BiPredicate<String, Integer> bi = (domain, score) -> {
            return (domain.equalsIgnoreCase("google.com") || score == 0);
        };

        // if google or score == 0
        List<Domain> result = filterBadDomain(domains, bi);
        System.out.println(result); // google.com, mkyong.com

        //  if score == 0
        List<Domain> result2 = filterBadDomain(domains, (domain, score) ->
score == 0);
        System.out.println(result2); // mkyong.com, microsoft.com

        // if start with i or score > 5
        List<Domain> result3 = filterBadDomain(domains, (domain, score) ->
domain.startsWith("i") && score > 5);
        System.out.println(result3); // i-am-spammer.com

        // chaining with or
        List<Domain> result4 = filterBadDomain(domains, bi.or(
                (domain, x) -> domain.equalsIgnoreCase("microsoft.com"))
        );
        System.out.println(result4); // google.com, mkyong.com, microsoft.com


    }

    public static <T extends Domain> List<T> filterBadDomain(
            List<T> list, BiPredicate<String, Integer> biPredicate) {

        return list.stream()
                .filter(x -> biPredicate.test(x.getName(), x.getScore()))
                .collect(Collectors.toList());

    }
}

class Domain {

    String name;
    Integer score;

    public Domain(String name, Integer score) {
        this.name = name;
        this.score = score;
    }
    // getters , setters , toString
}
```

Output

```
[Domain{name='google.com', score=1}, Domain{name='mkyong.com', score=0}]
[Domain{name='mkyong.com', score=0}]
[Domain{name='i-am-spammer.com', score=10}]
[Domain{name='google.com', score=1}, Domain{name='mkyong.com', score=0},
Domain{name='microsoft.com', score=2}]
```

References

- *BiPredicate JavaDoc*
- *Java 8 Method Reference: How to Use it*
- *Java 8 Predicate Examples*

Related Articles

- *Java 8 Predicate Examples*
- *Java 8 Consumer Examples*
- *Java 8 BiConsumer Examples*
- *Java 8 BinaryOperator Examples*
- *Java 8 UnaryOperator Examples*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Consumer Examples

By *mkyong* | Last updated: February 12, 2020

In Java 8, *Consumer* is a functional interface; it takes an argument and returns nothing.

```
@FunctionalInterface
public interface Consumer<T> {
  void accept(T t);
}
```

1. Consumer

Java8Consumer1.java

```
package com.mkyong.java8;

import java.util.function.Consumer;

public class Java8Consumer1 {

    public static void main(String[] args) {

        Consumer<String> print = x -> System.out.println(x);
        print.accept("java");   // java

    }

}
```

Output

```
java
```

2. Higher Order Function

2.1 This example accepts `Consumer` as an argument, simulates a `forEach` to print each item from a list.

Java8Consumer2.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class Java8Consumer2 {

    public static void main(String[] args) {

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

        // implementation of the Consumer's accept methods.
        Consumer<Integer> consumer = (Integer x) -> System.out.println(x);
        forEach(list, consumer);

        // or call this directly
        forEach(list, (Integer x) -> System.out.println(x));

    }

    static <T> void forEach(List<T> list, Consumer<T> consumer) {
        for (T t : list) {
            consumer.accept(t);
        }
    }

}
```

Output

```
1
2
3
4
5
1
2
3
4
5
```

2.2 Same `forEach` method to accept `Consumer` as an argument; this time, we will print the length of the string.

```
Java8Consumer3.java
```

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class Java8Consumer3 {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("a", "ab", "abc");
        forEach(list, (String x) -> System.out.println(x.length()));

    }

    static <T> void forEach(List<T> list, Consumer<T> consumer) {
        for (T t : list) {
            consumer.accept(t);
        }
    }

}
```

Output

```
1
2
3
```

See the flexibility?

References

- *Consumer JavaDoc*
- *Java 8 Predicate Examples*

Related Articles

- *Java 8 forEach examples*
- *Java 8 BiConsumer Examples*
- *Java 8 Predicate Examples*
- *Java 8 BiPredicate Examples*
- *Java 8 forEach print with Index*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Java 8 BiConsumer Examples

By *mkyong* | Last updated: February 13, 2020

In Java 8, *BiConsumer* is a functional interface; it takes two arguments and returns nothing.

```
@FunctionalInterface
public interface BiConsumer<T, U> {
  void accept(T t, U u);
}
```

Further Reading – *Java 8 Consumer Examples*

1. BiConsumer

JavaBiConsumer1.java

```
package com.mkyong.java8;

import java.util.function.Consumer;

public class JavaBiConsumer1 {

    public static void main(String[] args) {

      BiConsumer<Integer, Integer> addTwo = (x, y) -> System.out.println(x +
y);
      addTwo.accept(1, 2);    // 3

    }

}
```

Output

```
3
```

2. Higher Order Function

2.1 This example accepts `BiConsumer` as an argument, create a generic `addTwo` to join two objects.

JavaBiConsumer2.java

```
package com.mkyong.java8;

import java.util.function.BiConsumer;

public class JavaBiConsumer2 {

    public static void main(String[] args) {

        addTwo(1, 2, (x, y) -> System.out.println(x + y));          // 3
        addTwo("Node", ".js", (x, y) -> System.out.println(x + y)); // Node.js

    }

    static <T> void addTwo(T a1, T a2, BiConsumer<T, T> c) {
        c.accept(a1, a2);
    }

}
```

Output

```
3
Node.js
```

2.2 More `BiConsumer` examples.

JavaBiConsumer3.java

```
package com.mkyong.java8;

import java.util.function.BiConsumer;

public class JavaBiConsumer3 {

    public static void main(String[] args) {

        math(1, 1, (x, y) -> System.out.println(x + y));   // 2
        math(1, 1, (x, y) -> System.out.println(x - y));   // 0
        math(1, 1, (x, y) -> System.out.println(x * y));   // 1
        math(1, 1, (x, y) -> System.out.println(x / y));   // 1

    }

    static <Integer> void math(Integer a1, Integer a2, BiConsumer<Integer,
Integer> c) {
        c.accept(a1, a2);
    }

}
```

Output

```
2
0
1
1
```

3. Map.forEach

In the JDK source code, `Map.forEach` accepts a `BiConsumer` as an argument.

Map.java

```
default void forEach(BiConsumer<? super K, ? super V> action) {
        Objects.requireNonNull(action);
        for (Map.Entry<K, V> entry : entrySet()) {
            K k;
            V v;
            try {
                k = entry.getKey();
                v = entry.getValue();
            } catch (IllegalStateException ise) {
                // this usually means the entry is no longer in the map.
                throw new ConcurrentModificationException(ise);
            }
            action.accept(k, v);
        }
    }
```

JavaMapBiConsumer.java

```
package com.mkyong.java8;

import java.util.LinkedHashMap;
import java.util.Map;

public class JavaMapBiConsumer {

    public static void main(String[] args) {

        Map<Integer, String> map = new LinkedHashMap<>();

        map.put(1, "Java");
        map.put(2, "C++");
        map.put(3, "Rust");
        map.put(4, "JavaScript");
        map.put(5, "Go");

        map.forEach((k, v) -> System.out.println(k + ":" + v));

    }

}
```

Output

```
1:Java
2:C++
3:Rust
4:JavaScript
5:Go
```

References

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Supplier Examples

By *mkyong* | Last updated: May 18, 2020
Viewed: 29,481 | +443 pv/w

In Java 8, *Supplier* is a functional interface; it takes no arguments and returns a result.

Supplier.java

```
@FunctionalInterface
public interface Supplier<T> {
    T get();
}
```

1. Supplier

1.1 This example uses `Supplier` to return a current date-time.

Java8Supplier1.java

```
package com.mkyong;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.function.Supplier;

public class Java8Supplier1 {

    private static final DateTimeFormatter dtf =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    public static void main(String[] args) {

        Supplier<LocalDateTime> s = () -> LocalDateTime.now();
        LocalDateTime time = s.get();

        System.out.println(time);

        Supplier<String> s1 = () -> dtf.format(LocalDateTime.now());
        String time2 = s1.get();

        System.out.println(time2);

    }

}
```

Output

```
2020-03-02T16:10:49.281223
2020-03-02 16:10:49
```

2. Returns a Supplier

Java8Supplier2.java

```
package com.mkyong;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Supplier;

public class Java8Supplier2<T> {

    public static void main(String[] args) {

        Java8Supplier2<String> obj = new Java8Supplier2();

        List<String> list = obj.supplier().get();

    }

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

        // lambda
        // return () -> new ArrayList<>();

        // constructor reference
        return ArrayList::new;

    }

}
```

3. Factory

3.1 A simple factory method to return a `Developer` object.

Java8Supplier3.java

```
package com.mkyong;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.util.function.Supplier;

public class Java8Supplier3 {

    public static void main(String[] args) {

        Developer obj = factory(Developer::new);
        System.out.println(obj);

        Developer obj2 = factory(() -> new Developer("mkyong"));
        System.out.println(obj2);

    }

    public static Developer factory(Supplier<? extends Developer> s) {

        Developer developer = s.get();
        if (developer.getName() == null || "".equals(developer.getName())) {
            developer.setName("default");
        }
        developer.setSalary(BigDecimal.ONE);
        developer.setStart(LocalDate.of(2017, 8, 8));

        return developer;

    }

}
```

Developer.java

```
package com.mkyong;

import java.math.BigDecimal;
import java.time.LocalDate;

public class Developer {

    String name;
    BigDecimal salary;
    LocalDate start;

    // for factory(Developer::new);
    public Developer() {
    }

    // for factory(() -> new Developer("mkyong"));
    public Developer(String name) {
        this.name = name;
    }

    // get, set, constructor, toString
    //...

}
```

Output

```
Developer{name='default', salary=1, start=2017-08-08}
Developer{name='mkyong', salary=1, start=2017-08-08}
```

References

- *Supplier*
- *Java 8 – How to format LocalDateTime*
- *DateTimeFormatter JavaDoc*
- *Java 8 Tutorials*

Related Articles

- *Java 8 BiFunction Examples*
- *Java 8 forEach examples*
- *Java 8 Predicate Examples*
- *Java 8 BiPredicate Examples*
- *Java 8 Consumer Examples*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Lambda : Comparator example

By *mkyong* | Last updated: August 5, 2015

In this example, we will show you how to use Java 8 Lambda expression to write a `Comparator` to sort a List.

1. Classic `Comparator` example.

```
Comparator<Developer> byName = new Comparator<Developer>() {
    @Override
    public int compare(Developer o1, Developer o2) {
        return o1.getName().compareTo(o2.getName());
    }
};
```

2. Lambda expression equivalent.

```
Comparator<Developer> byName =
    (Developer o1, Developer o2)->o1.getName().compareTo(o2.getName());
```

1. Sort without Lambda

Example to compare the `Developer` objects using their age. Normally, you use `Collections.sort` and pass an anonymous `Comparator` class like this :

TestSorting.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class TestSorting {

    public static void main(String[] args) {

        List<Developer> listDevs = getDevelopers();

        System.out.println("Before Sort");
        for (Developer developer : listDevs) {
            System.out.println(developer);
        }

        //sort by age
        Collections.sort(listDevs, new Comparator<Developer>() {
            @Override
            public int compare(Developer o1, Developer o2) {
                return o1.getAge() - o2.getAge();
            }
        });

        System.out.println("After Sort");
        for (Developer developer : listDevs) {
            System.out.println(developer);
        }

    }

    private static List<Developer> getDevelopers() {

        List<Developer> result = new ArrayList<Developer>();

        result.add(new Developer("mkyong", new BigDecimal("70000"), 33));
        result.add(new Developer("alvin", new BigDecimal("80000"), 20));
        result.add(new Developer("jason", new BigDecimal("100000"), 10));
        result.add(new Developer("iris", new BigDecimal("170000"), 55));

        return result;

    }

}
```

Output

```
Before Sort
Developer [name=mkyong, salary=70000, age=33]
Developer [name=alvin, salary=80000, age=20]
Developer [name=jason, salary=100000, age=10]
Developer [name=iris, salary=170000, age=55]

After Sort
Developer [name=jason, salary=100000, age=10]
Developer [name=alvin, salary=80000, age=20]
Developer [name=mkyong, salary=70000, age=33]
Developer [name=iris, salary=170000, age=55]
```

When the sorting requirement is changed, you just pass in another new anonymous `Comparator` class :

```
    //sort by age
    Collections.sort(listDevs, new Comparator<Developer>() {
        @Override
        public int compare(Developer o1, Developer o2) {
            return o1.getAge() - o2.getAge();
        }
    });

    //sort by name
    Collections.sort(listDevs, new Comparator<Developer>() {
        @Override
        public int compare(Developer o1, Developer o2) {
            return o1.getName().compareTo(o2.getName());
        }
    });

    //sort by salary
    Collections.sort(listDevs, new Comparator<Developer>() {
        @Override
        public int compare(Developer o1, Developer o2) {
            return o1.getSalary().compareTo(o2.getSalary());
        }
    });
```

It works, but, do you think it is a bit weird to create a class just because you want to change a single line of code?

2. Sort with Lambda

In Java 8, the `List` interface is supports the `sort` method directly, no need to use `Collections.sort` anymore.

```
        //List.sort() since Java 8
    listDevs.sort(new Comparator<Developer>() {
        @Override
        public int compare(Developer o1, Developer o2) {
            return o2.getAge() - o1.getAge();
        }
    });
```

Lambda expression example :

TestSorting.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.List;

public class TestSorting {

    public static void main(String[] args) {

        List<Developer> listDevs = getDevelopers();

        System.out.println("Before Sort");
        for (Developer developer : listDevs) {
            System.out.println(developer);
        }

        System.out.println("After Sort");

        //lambda here!
        listDevs.sort((Developer o1, Developer o2)->o1.getAge()-o2.getAge());

        //java 8 only, lambda also, to print the List
        listDevs.forEach((developer)->System.out.println(developer));
    }

    private static List<Developer> getDevelopers() {

        List<Developer> result = new ArrayList<Developer>();

        result.add(new Developer("mkyong", new BigDecimal("70000"), 33));
        result.add(new Developer("alvin", new BigDecimal("80000"), 20));
        result.add(new Developer("jason", new BigDecimal("100000"), 10));
        result.add(new Developer("iris", new BigDecimal("170000"), 55));

        return result;

    }

}
```

Output

```
Before Sort
Developer [name=mkyong, salary=70000, age=33]
Developer [name=alvin, salary=80000, age=20]
Developer [name=jason, salary=100000, age=10]
Developer [name=iris, salary=170000, age=55]

After Sort
Developer [name=jason, salary=100000, age=10]
Developer [name=alvin, salary=80000, age=20]
Developer [name=mkyong, salary=70000, age=33]
Developer [name=iris, salary=170000, age=55]
```

3. More Lambda Examples

3.1 Sort By age

```
//sort by age
Collections.sort(listDevs, new Comparator<Developer>() {
    @Override
    public int compare(Developer o1, Developer o2) {
        return o1.getAge() - o2.getAge();
    }
});

//lambda
listDevs.sort((Developer o1, Developer o2)->o1.getAge()-o2.getAge());

//lambda, valid, parameter type is optional
listDevs.sort((o1, o2)->o1.getAge()-o2.getAge());
```

3.2 Sort by name

```
//sort by name
Collections.sort(listDevs, new Comparator<Developer>() {
    @Override
    public int compare(Developer o1, Developer o2) {
        return o1.getName().compareTo(o2.getName());
    }
});

//lambda
listDevs.sort((Developer o1, Developer o2)-
>o1.getName().compareTo(o2.getName()));

//lambda
listDevs.sort((o1, o2)->o1.getName().compareTo(o2.getName()));
```

3.3 Sort by salary

```
//sort by salary
Collections.sort(listDevs, new Comparator<Developer>() {
    @Override
    public int compare(Developer o1, Developer o2) {
        return o1.getSalary().compareTo(o2.getSalary());
    }
});

//lambda
listDevs.sort((Developer o1, Developer o2)-
>o1.getSalary().compareTo(o2.getSalary()));

//lambda
listDevs.sort((o1, o2)->o1.getSalary().compareTo(o2.getSalary()));
```

3.4 Reversed sorting.

3.4.1 Lambda expression to sort a List using their salary.

```
Comparator<Developer> salaryComparator = (o1, o2)-
>o1.getSalary().compareTo(o2.getSalary());
listDevs.sort(salaryComparator);
```

Output

```
Developer [name=mkyong, salary=70000, age=33]
Developer [name=alvin, salary=80000, age=20]
Developer [name=jason, salary=100000, age=10]
Developer [name=iris, salary=170000, age=55]
```

3.4.2 Lambda expression to sort a List using their salary, reversed order.

```
    Comparator<Developer> salaryComparator = (o1, o2)-
>o1.getSalary().compareTo(o2.getSalary());
    listDevs.sort(salaryComparator.reversed());
```

Output

```
Developer [name=iris, salary=170000, age=55]
Developer [name=jason, salary=100000, age=10]
Developer [name=alvin, salary=80000, age=20]
Developer [name=mkyong, salary=70000, age=33]
```

References

1. *Start Using Java Lambda Expressions*
2. *Oracle : Lambda Expressions*
3. *Oracle : Comparator*

Related Articles

- *Java 8 - How to sort a Map*
- *Java - How to display all System properties*
- *How to compare dates in Java*
- *How to loop a Map in Java*
- *How to sort an ArrayList in java*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials
Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 method references, double colon (::) operator

By *mkyong* | Last updated: March 6, 2020

In Java 8, the double colon (::) operator is called method references. Refer to the following examples:

Anonymous class to print a list.

```
List<String> list = Arrays.asList("node", "java", "python", "ruby");
list.forEach(new Consumer<String>() {       // anonymous class
    @Override
    public void accept(String str) {
        System.out.println(str);
    }
});
```

Anonymous class -> Lambda expressions.

```
List<String> list = Arrays.asList("node", "java", "python", "ruby");
list.forEach(str -> System.out.println(str)); // lambda
```

Lambda expressions -> Method references.

```
List<String> list = Arrays.asList("node", "java", "python", "ruby");
list.forEach(System.out::println);          // method references
```

Anonymous Class -> Lambda expression -> Method Reference

> **Note**
> Both lambda expression or method reference does nothing but just another way call to an existing method. With method reference, it gains better readability.

There are four kinds of method references:

- Reference to a static method `ClassName::staticMethodName`
- Reference to an instance method of a particular object `Object::instanceMethodName`
- Reference to an instance method of an arbitrary object of a particular type `ContainingType::methodName`–
- Reference to a constructor `ClassName::new`

1. Static method

Lambda expression.

```
(args) -> ClassName.staticMethodName(args)
```

Method Reference.

```
ClassName::staticMethodName
```

1.1 This example prints a list of Strings, method reference to a static method `SimplePrinter::print`.

Java8MethodReference1a.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.List;
import java.util.function.Consumer;

public class Java8MethodReference1a {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("A", "B", "C");

        // anonymous class
        list.forEach(new Consumer<String>() {
            @Override
            public void accept(String x) {
                SimplePrinter.print(x);
            }
        });

        // lambda expression
        list.forEach(x -> SimplePrinter.print(x));

        // method reference
        list.forEach(SimplePrinter::print);

    }

}

class SimplePrinter {
    public static void print(String str) {
        System.out.println(str);
    }
}
```

1.2 This example converts a list of Strings into a list of Integers, method reference to a static method `Integer::parseInt`.

Integer.java

```
  public static int parseInt(String s) throws NumberFormatException {
        return parseInt(s,10);
  }
```

Java8MethodReference1b.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.List;
import java.util.function.Function;
import java.util.stream.Collectors;

public class Java8MethodReference1b {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("1", "2", "3");

        // anonymous class
        List<Integer> collect1 = list.stream()
                .map(new Function<String, Integer>() {
                    @Override
                    public Integer apply(String s) {
                        return Integer.parseInt(s);
                    }
                })
                .collect(Collectors.toList());

        // lambda expression
        List<Integer> collect2 = list.stream()
                .map(s -> Integer.parseInt(s))
                .collect(Collectors.toList());

        // method reference
        List<Integer> collect3 = list.stream()
                .map(Integer::parseInt)
                .collect(Collectors.toList());

    }

}
```

1.3 This example joins two `Integer` and returns a `String`. It passes a method reference static method `IntegerUtils::join` as an argument into another method that accepts a `BiFunction`.

Java8MethodReference1c.java

```
package com.mkyong;

import java.util.function.BiFunction;

public class Java8MethodReference1c {

    public static void main(String[] args) {

        // anonymous class
        String result1 = playTwoArgument(1, 2, new BiFunction<Integer, Integer,
String>() {
                    @Override
                    public String apply(Integer a, Integer b) {
                        return IntegerUtils.join(a, b);
                    }
                });
// 3

        // lambda
        String result1 = playTwoArgument(1, 2, (a, b) -> IntegerUtils.join(a,
b));  // 3

        // method reference
        String result2 = playTwoArgument(1, 2, IntegerUtils::join);
// 3

    }

    private static <R> R playTwoArgument(Integer i1, Integer i2,
        BiFunction<Integer, Integer, R> func) {
        return func.apply(i1, i2);
    }

}

class IntegerUtils{

    public static String join(Integer a, Integer b) {
        return String.valueOf(a + b);
    }

}
```

2. Reference to an instance method of a particular object

Lambda expression.

```
(args) -> object.instanceMethodName(args)
```

Method Reference.

```
object::instanceMethodName
```

2.1 This example sorts a list of `Employee` by salary. We can reference to an instance method `compareBySalary` of a particular object `ComparatorProvider`.

Java8MethodReference2

```
package com.mkyong;

import java.math.BigDecimal;
import java.util.Arrays;
import java.util.List;

public class Java8MethodReference2 {

    public static void main(String[] args) {

        List<Employee> list = Arrays.asList(
                new Employee("mkyong", 38, BigDecimal.valueOf(3800)),
                new Employee("zilap", 5, BigDecimal.valueOf(100)),
                new Employee("ali", 25, BigDecimal.valueOf(2500)),
                new Employee("unknown", 99, BigDecimal.valueOf(9999)));

        // anonymous class
        /*list.sort(new Comparator<Employee>() {
            @Override
            public int compare(Employee o1, Employee o2) {
                return provider.compareBySalary(o1, o2);
            }
        });*/

        ComparatorProvider provider = new ComparatorProvider();

        // lambda
        // list.sort((o1, o2) -> provider.compareBySalary(o1, o2));

        // method reference
        list.sort(provider::compareBySalary);

        list.forEach(x -> System.out.println(x));

    }

}

class ComparatorProvider {

    public int compareByAge(Employee o1, Employee o2) {
        return o1.getAge().compareTo(o2.getAge());
    }

    public int compareByName(Employee o1, Employee o2) {
        return o1.getName().compareTo(o2.getName());
    }

    public int compareBySalary(Employee o1, Employee o2) {
        return o1.getAge().compareTo(o2.getAge());
    }

}
```

Employee.java

```
package com.mkyong;

import java.math.BigDecimal;

public class Employee {

    String name;
    Integer age;
    BigDecimal salary;

    // generated by IDE, getters, setters, constructor, toString
}
```

Output

```
Employee{name='zilap', age=5, salary=100}
Employee{name='ali', age=25, salary=2500}
Employee{name='mkyong', age=38, salary=3800}
Employee{name='unknown', age=99, salary=9999}
```

3. Reference to an instance method of an arbitrary object of a particular type.

The statement is a bit confusing, need little explanation, see below samples:

Lambda expression.

```
// arg0 is the first argument
(arg0, rest_of_args) -> arg0.methodName(rest_of_args)

// example, assume a and b are String
(a, b) -> a.compareToIgnoreCase(b)
```

Method Reference.

```
// first argument type
arg0_Type::methodName

// arg0 is type of ClassName
ClassName::methodName

// example, a is type of String
String::compareToIgnoreCase
```

For `(String a, String b)`, where `a` and `b` are arbitrary names, and `String` is its arbitrary type. This example uses method reference to an instance method `compareToIgnoreCase` of an arbitrary object `a` (first argument) of a particular type `String`.

3.1 Review the official example in this *Method References*

```
String[] stringArray = { "Barbara", "James", "Mary", "John",
                         "Patricia", "Robert", "Michael", "Linda" };
Arrays.sort(stringArray, String::compareToIgnoreCase);
```

We passed a method reference `String::compareToIgnoreCase` as a comparator for `Arrays.sort`.

explanation

Review the `Arrays.sort` method signature:

```
public static <T> void sort(T[] a, Comparator<? super T> c) {
}
```

In the above example, `Arrays.sort` expects a `Comparator<String>`. The `Comparator` is a function interface, its abstract method `compare` matches `BiFunction<String, String, Integer>`, it takes two arguments of `String` and returns an `int`.

Comparator.java

```
@FunctionalInterface
public interface Comparator<T> {
    int compare(T o1, T o2);  // this matches BiFunction<String, String,
Integer>
}
```

Review the `BiFunction` method signature:

BiFunction.java

```
@FunctionalInterface
public interface BiFunction<T, U, R> {
      R apply(T t, U u);
}
```

Further Reading –*Java 8 BiFunction examples*

The below lambda provides implementation for `BiFunction<String,String,Integer>`, so the `Arrays.sort` accepts the below lambda expression as valid syntax.

```
(String a, String b) -> a.compareToIgnoreCase(b) // return int

// a is type of String
// method reference
String::compareToIgnoreCase
```

3.2 Let us see another example.

Java8MethodReference3a.java

```
package com.mkyong;

import java.util.function.BiPredicate;
import java.util.function.Function;

public class Java8MethodReference3a {

    public static void main(String[] args) {

        // lambda
        int result = playOneArgument("mkyong", x -> x.length());   // 6

        // method reference
        int result2 = playOneArgument("mkyong", String::length);   // 6

        // lambda
        Boolean result3 = playTwoArgument("mkyong", "y", (a, b) ->
a.contains(b)); // true

        // method reference
        Boolean result4 = playTwoArgument("mkyong", "y", String::contains);
// true

        // lambda
        Boolean result5 = playTwoArgument("mkyong", "1", (a, b) ->
a.startsWith(b)); // false

        // method reference
        Boolean result6 = playTwoArgument("mkyong", "y", String::startsWith);
// false

        System.out.println(result6);
    }

    static <R> R playOneArgument(String s1, Function<String, R> func) {
        return func.apply(s1);
    }

    static Boolean playTwoArgument(String s1, String s2, BiPredicate<String,
String> func) {
        return func.test(s1, s2);
    }

}
```

3.3 Let us see another example, custom object.

```
Java8MethodReference3b.java
```

```
package com.mkyong;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.function.BiFunction;

public class Java8MethodReference3b {

    public static void main(String[] args) {

        Invoice obj = new Invoice("A001", BigDecimal.valueOf(1.99), 3);

        InvoiceCalculator formula = new InvoiceCalculator();

        // lambda
        BigDecimal result = calculate(formula, obj, (f, o) -> f.normal(o));
// 5.97

        // method reference
        BigDecimal result2 = calculate(formula, obj,
InvoiceCalculator::normal);    // 5.97

        // lambda
        BigDecimal result3 = calculate(formula, obj, (f, o) -> f.promotion(o));
// 5.37

        // method reference
        BigDecimal result4 = calculate(formula, obj,
InvoiceCalculator::promotion); // 5.37

    }

    static BigDecimal calculate(InvoiceCalculator formula, Invoice s1,
                                BiFunction<InvoiceCalculator, Invoice,
BigDecimal> func) {
        return func.apply(formula, s1);
    }

}

class InvoiceCalculator {

    public BigDecimal normal(Invoice obj) {
        return obj.getUnitPrice().multiply(BigDecimal.valueOf(obj.qty));
    }

    public BigDecimal promotion(Invoice obj) {
        return obj.getUnitPrice()
                .multiply(BigDecimal.valueOf(obj.qty))
                .multiply(BigDecimal.valueOf(0.9))
                .setScale(2, RoundingMode.HALF_UP);
    }
}

class Invoice {

    String no;
    BigDecimal unitPrice;
    Integer qty;

    // generated by IDE, setters, gettes, constructor, toString
}
```

The first argument is a type of `InvoiceCalculator`. So, we can reference to an instance method (`normal or promotion`) of an arbitrary object (`f`) of a particular type `InvoiceCalculator`.

```
(f, o) -> f.normal(o))
(f, o) -> f.promotion(o))

InvoiceCalculator::normal
InvoiceCalculator::promotion
```

Got it? No more example ☺

4. Reference to a constructor.

Lambda expression.

```
(args) -> new ClassName(args)
```

Method Reference.

```
ClassName::new
```

4.1 Reference to a default constructor.

```
Java8MethodReference4a.java
```

```
package com.mkyong;

import java.math.BigDecimal;
import java.util.HashMap;
import java.util.Map;
import java.util.function.Supplier;

public class Java8MethodReference4a {

    public static void main(String[] args) {

        // lambda
        Supplier<Map> obj1 = () -> new HashMap();   // default HashMap()
constructor
        Map map1 = obj1.get();

        // method reference
        Supplier<Map> obj2 = HashMap::new;
        Map map2 = obj2.get();

        // lambda
        Supplier<Invoice> obj3 = () -> new Invoice(); // default Invoice()
constructor
        Invoice invoice1 = obj3.get();

        // method reference
        Supplier<Invoice> obj4 = Invoice::new;
        Invoice invoice2 = obj4.get();

    }

}

class Invoice {

    String no;
    BigDecimal unitPrice;
    Integer qty;

    public Invoice() {
    }

    //... generated by IDE
}
```

4.2 Reference to a constructor which accepts an argument – `Invoice(BigDecimal unitPrice)`

Java8MethodReference4b.java

```
package com.mkyong;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.function.Function;

public class Java8MethodReference4b {

    public static void main(String[] args) {

        List<BigDecimal> list = Arrays.asList(
                BigDecimal.valueOf(9.99),
                BigDecimal.valueOf(2.99),
                BigDecimal.valueOf(8.99));

        // lambda
        // List<Invoice> invoices = fakeInvoice(list, (price) -> new
Invoice(price));

        // method reference
        List<Invoice> invoices = fakeInvoice(list, Invoice::new);

        invoices.forEach(System.out::println);
    }

    static List<Invoice> fakeInvoice(List<BigDecimal> list,
Function<BigDecimal, Invoice> func) {
        List<Invoice> result = new ArrayList<>();

        for (BigDecimal amount : list) {
            result.add(func.apply(amount));
        }
        return result;
    }

}

class Invoice {

    String no;
    BigDecimal unitPrice;
    Integer qty;

    public Invoice(BigDecimal unitPrice) {
        this.unitPrice = unitPrice;
    }

    //... generated by IDE
}
```

Output

```
Invoice{no='null', unitPrice=9.99, qty=null}
Invoice{no='null', unitPrice=2.99, qty=null}
Invoice{no='null', unitPrice=8.99, qty=null}
```

Done.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Streams filter examples

By *mkyong* | Last updated: April 3, 2017

In this tutorial, we will show you few Java 8 examples to demonstrate the use of Streams `filter()`, `collect()`, `findAny()` and `orElse()`

1. Streams filter() and collect()

1.1 Before Java 8, filter a `List` like this :

BeforeJava8.java

```
package com.mkyong.java8;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class BeforeJava8 {

    public static void main(String[] args) {

        List<String> lines = Arrays.asList("spring", "node", "mkyong");
        List<String> result = getFilterOutput(lines, "mkyong");
        for (String temp : result) {
            System.out.println(temp);    //output : spring, node
        }

    }

    private static List<String> getFilterOutput(List<String> lines, String
filter) {
        List<String> result = new ArrayList<>();
        for (String line : lines) {
            if (!"mkyong".equals(line)) { // we dont like mkyong
                result.add(line);
            }
        }
        return result;
    }

}
```

Output

```
spring
node
```

1.2 The equivalent example in Java 8, `stream.filter()` to filter a `List`, and `collect()` to convert a stream into a `List`.

NowJava8.java

```
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

    }

}
```

Output

```
spring
node
```

2. Streams filter(), findAny() and orElse()

Person.java

```
package com.mkyong.java8;

public class Person {

    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    //gettersm setters, toString
}
```

2.1 Before Java 8, you get a `Person` by name like this :

BeforeJava8.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;

public class BeforeJava8 {

    public static void main(String[] args) {

        List<Person> persons = Arrays.asList(
                new Person("mkyong", 30),
                new Person("jack", 20),
                new Person("lawrence", 40)
        );

        Person result = getStudentByName(persons, "jack");
        System.out.println(result);

    }

    private static Person getStudentByName(List<Person> persons, String name) {

        Person result = null;
        for (Person temp : persons) {
            if (name.equals(temp.getName())) {
                result = temp;
            }
        }
        return result;
    }
}
```

Output

```
Person{name='jack', age=20}
```

2.2 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);

    }

}
```

Output

```
Person{name='jack', age=20}
null
```

2.3 For multiple condition.

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()
                .filter((p) -> "jack".equals(p.getName()) && 20 == p.getAge())
                .findAny()
                .orElse(null);

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

        //or like this
        Person result2 = persons.stream()
                .filter(p -> {
                    if ("jack".equals(p.getName()) && 20 == p.getAge()) {
                        return true;
                    }
                    return false;
                }).findAny()
                .orElse(null);

        System.out.println("result 2 :" + result2);

    }


}
```

Output

```
result 1 :Person{name='jack', age=20}
result 2 :Person{name='jack', age=20}
```

3. Streams filter() and map()

NowJava8.java

```
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<Person> persons = Arrays.asList(
                new Person("mkyong", 30),
                new Person("jack", 20),
                new Person("lawrence", 40)
        );

        String name = persons.stream()
                .filter(x -> "jack".equals(x.getName()))
                .map(Person::getName)                          //convert stream
to String
                .findAny()
                .orElse("");

        System.out.println("name : " + name);

        List<String> collect = persons.stream()
                .map(Person::getName)
                .collect(Collectors.toList());

        collect.forEach(System.out::println);

    }

}
```

Output

```
name : jack

mkyong
jack
lawrence
```

> **Note**
> Highly recommend this Streams tutorial – *Processing Data with Java SE 8 Streams*

References

1. *Java 8 cyclic inference in my case*
2. *Java 8 Explained: Using Filters, Maps, Streams and Foreach to apply Lambdas to Java Collections!*
3. *Java 8 forEach examples*
4. *Java 8 Streams: multiple filters vs. complex condition*
5. *Processing Data with Java SE 8 Streams*

Java 8 Streams map() examples

By *mkyong* | Last updated: April 3, 2017

In Java 8, `stream().map()` lets you convert an object to something else. Review the following examples :

1. A List of Strings to Uppercase

1.1 Simple Java example to convert a list of Strings to upper case.

TestJava8.java

```
package com.mkyong.java8;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class TestJava8 {

    public static void main(String[] args) {

        List<String> alpha = Arrays.asList("a", "b", "c", "d");

        //Before Java8
        List<String> alphaUpper = new ArrayList<>();
        for (String s : alpha) {
            alphaUpper.add(s.toUpperCase());
        }

        System.out.println(alpha); //[a, b, c, d]
        System.out.println(alphaUpper); //[A, B, C, D]

        // 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]

    }

}
```

2. List of objects -> List of String

2.1 Get all the `name` values from a list of the `staff` objects.

Staff.java

```
package com.mkyong.java8;

import java.math.BigDecimal;

public class Staff {

    private String name;
    private int age;
    private BigDecimal salary;
    //...
}
```

TestJava8.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class TestJava8 {

    public static void main(String[] args) {

        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))
        );

        //Before Java 8
        List<String> result = new ArrayList<>();
        for (Staff x : staff) {
            result.add(x.getName());
        }
        System.out.println(result); //[mkyong, jack, lawrence]

        //Java 8
        List<String> collect = staff.stream().map(x ->
x.getName()).collect(Collectors.toList());
        System.out.println(collect); //[mkyong, jack, lawrence]

    }

}
```

3. List of objects -> List of other objects

3.1 This example shows you how to convert a list of `staff` objects into a list of `StaffPublic` objects.

Staff.java

```
package com.mkyong.java8;

import java.math.BigDecimal;

public class Staff {

    private String name;
    private int age;
    private BigDecimal salary;
    //...
}
```

StaffPublic.java

```
package com.mkyong.java8;

public class StaffPublic {

    private String name;
    private int age;
    private String extra;
    //...
}
```

3.2 Before Java 8.

BeforeJava8.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class BeforeJava8 {

    public static void main(String[] args) {

        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))
        );

        List<StaffPublic> result = convertToStaffPublic(staff);
        System.out.println(result);

    }

    private static List<StaffPublic> convertToStaffPublic(List<Staff> staff) {

        List<StaffPublic> result = new ArrayList<>();

        for (Staff temp : staff) {

            StaffPublic obj = new StaffPublic();
            obj.setName(temp.getName());
            obj.setAge(temp.getAge());
            if ("mkyong".equals(temp.getName())) {
                obj.setExtra("this field is for mkyong only!");
            }

            result.add(obj);
        }

        return result;

    }

}
```

Output

```
[
    StaffPublic{name='mkyong', age=30, extra='this field is for mkyong only!'},
    StaffPublic{name='jack', age=27, extra='null'},
    StaffPublic{name='lawrence', age=33, extra='null'}
]
```

3.3 Java 8 example.

NowJava8.java

```
package com.mkyong.java8;

package com.hostingcompass.web.java8;

import java.math.BigDecimal;
import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class NowJava8 {

    public static void main(String[] args) {

        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))
        );

        // convert inside the map() method directly.
        List<StaffPublic> result = staff.stream().map(temp -> {
            StaffPublic obj = new StaffPublic();
            obj.setName(temp.getName());
            obj.setAge(temp.getAge());
            if ("mkyong".equals(temp.getName())) {
                obj.setExtra("this field is for mkyong only!");
            }
            return obj;
        }).collect(Collectors.toList());

        System.out.println(result);

    }

}
```

Output

```
[
    StaffPublic{name='mkyong', age=30, extra='this field is for mkyong only!'},
    StaffPublic{name='jack', age=27, extra='null'},
    StaffPublic{name='lawrence', age=33, extra='null'}
]
```

References

1. *Processing Data with Java SE 8 Streams, Part 1*
2. *Java 8 – Filter a Map examples*
3. *Java 8 flatMap example*
4. *Collectors JavaDoc*

Related Articles

- *Java 8 - Filter a Map examples*
- *Java 8 - How to sort a Map*

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 flatMap example

By *mkyong* | Last updated: December 16, 2020

This article explains the Java 8 *Stream.flatMap* and how to use it.

Topic

1. What is flatMap()?

1.1 Review the below structure. It consists of a 2 levels Stream or a 2d arrays.

```
# Stream<String[]>
# Stream<Stream<String>>
# String[][]

[
  [1, 2],
  [3, 4],
  [5, 6]
]
```

In Java 8, we can use the `flatMap` to convert the above 2 levels Stream into one Stream level or a 2d array into a 1d array.

```
# Stream<String>
# String[]

[1, 2, 3, 4, 5, 6]
```

2. Why flat a Stream?

2.1 It's challenging to process a Stream containing more than one level, like `Stream<String[]>` or `Stream<List<LineItem>>` or `Stream<Stream<String>>`. And we flat the 2 levels Stream into one level, like `Stream<String>` or `Stream<LineItem>`, so that we can easily loop the Stream and process it.

Review the below example, before and after applying `flatMap` on a Stream.

2.2 Below is a 2d array, and we can use `Arrays.stream` or `Stream.of` to convert it into a Stream, and it produces a Stream of `String[]` or `Stream<String[]>`.

```
String[][] array = new String[][]{{"a", "b"}, {"c", "d"}, {"e", "f"}};

// array to a stream
Stream<String[]> stream1 = Arrays.stream(array);

// same result
Stream<String[]> stream2 = Stream.of(array);
```

or like this.

```
[
  [a, b],
  [c, d],
  [e, f]
]
```

2.3 Here's the requirement, we want to filter out the `a` and print out all the characters.

First, we try the `Stream#filter` directly. However, the below program will print nothing, and it is because the `x` inside the `Stream#filter` is a `String[]`, not a `String`; the condition will always remain false, and the Stream will collect nothing.

```
String[][] array = new String[][]{{"a", "b"}, {"c", "d"}, {"e", "f"}};

// convert array to a stream
Stream<String[]> stream1 = Arrays.stream(array);

List<String[]> result = stream1
    .filter(x -> !x.equals("a"))      // x is a String[], not String!
    .collect(Collectors.toList());

System.out.println(result.size());    // 0

result.forEach(System.out::println);  // print nothing?
```

OK, this time, we refactor the filter method to deal with the `String[]`.

```
String[][] array = new String[][]{{"a", "b"}, {"c", "d"}, {"e", "f"}};

// array to a stream
Stream<String[]> stream1 = Arrays.stream(array);

// x is a String[]
List<String[]> result = stream1
        .filter(x -> {
            for(String s : x){      // really?
                if(s.equals("a")){
                    return false;
                }
            }
            return true;
        }).collect(Collectors.toList());

// print array
result.forEach(x -> System.out.println(Arrays.toString(x)));
```

Output

```
Terminal
```

```
[c, d]
[e, f]
```

In the above case, the `Stream#filter` will filter out the entire `[a, b]`, but we want to filter out only the character `a`

3.4 Below is the final version, and we combine the array first and follow by a filter later. In Java, to convert a 2d array into a 1d array, we can loop the 2d array and put all the elements into a new array; Or we can use the Java 8 `flatMap` to flatten the 2d array into a 1d array, or from `Stream<String[]>` to `Stream<String>`.

```
String[][] array = new String[][]{{"a", "b"}, {"c", "d"}, {"e", "f"}};

// Java 8
String[] result = Stream.of(array)  // Stream<String[]>
        .flatMap(Stream::of)        // Stream<String>
        .toArray(String[]::new);    // [a, b, c, d, e, f]

for (String s : result) {
    System.out.println(s);
}
```

Output

```
Terminal
```

```
a
b
c
d
e
f
```

Now, we can easily filter out the `a`; let see the final version.

```
String[][] array = new String[][]{{"a", "b"}, {"c", "d"}, {"e", "f"}};

List<String> collect = Stream.of(array)     // Stream<String[]>
        .flatMap(Stream::of)                // Stream<String>
        .filter(x -> !"a".equals(x))        // filter out the a
        .collect(Collectors.toList());      // return a List

collect.forEach(System.out::println);
```

Output

```
Terminal
```

```
b
c
d
e
f
```

I want to point out that dealing with more than one level of Stream is challenging, confusing, and error-prone, and we can use this `Stream#flatMap` to flatten the 2 levels Stream into one level Stream.

```
Stream<String[]>      -> flatMap ->     Stream<String>
Stream<Set<String>>   -> flatMap ->     Stream<String>
Stream<List<String>>  -> flatMap ->     Stream<String>
Stream<List<Object>>  -> flatMap ->     Stream<Object>
```

3. flatMap example – Find all books.

This example uses `.stream()` to convert a `List` into a stream of objects, and each object contains a set of books, and we can use `flatMap` to produces a stream containing all the book in all the objects.

In the end, we also filter out the book containing the word `python` and collect a `Set` to remove the duplicated book.

Developer.java

```
package com.mkyong.java8.stream.flatmap;

import java.util.HashSet;
import java.util.Set;

public class Developer {

    private Integer id;
    private String name;
    private Set<String> book;

    //getters, setters, toString

    public void addBook(String book) {
        if (this.book == null) {
            this.book = new HashSet<>();
        }
        this.book.add(book);
    }
}
```

FlatMapExample1.java

```
package com.mkyong.java8.stream.flatmap;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

public class FlatMapExample1 {

    public static void main(String[] args) {

        Developer o1 = new Developer();
        o1.setName("mkyong");
        o1.addBook("Java 8 in Action");
        o1.addBook("Spring Boot in Action");
        o1.addBook("Effective Java (3nd Edition)");

        Developer o2 = new Developer();
        o2.setName("zilap");
        o2.addBook("Learning Python, 5th Edition");
        o2.addBook("Effective Java (3nd Edition)");

        List<Developer> list = new ArrayList<>();
        list.add(o1);
        list.add(o2);

        // hmm....Set of Set...how to process?
        /*Set<Set<String>> collect = list.stream()
                .map(x -> x.getBook())
                .collect(Collectors.toSet());*/

        Set<String> collect =
                list.stream()
                        .map(x -> x.getBook())                          //
Stream<Set<String>>
                        .flatMap(x -> x.stream())                       //
Stream<String>
                        .filter(x -> !x.toLowerCase().contains("python"))   //
filter python book
                        .collect(Collectors.toSet());                   //
remove duplicated

        collect.forEach(System.out::println);

    }

}
```

Output

```
Terminal
```

```
Spring Boot in Action
Effective Java (3nd Edition)
Java 8 in Action
```

The `map` is optional.

```
    Set<String> collect2 = list.stream()
                    //.map(x -> x.getBook())
                    .flatMap(x -> x.getBook().stream())                 //
Stream<String>
                    .filter(x -> !x.toLowerCase().contains("python"))   //
filter python book
                    .collect(Collectors.toSet());
```

4. flatMap example – Order and LineItems

This example is similar to the official *flatMap JavaDoc* example.

The `orders` is a stream of purchase orders, and each purchase order contains a collection of line items, then we can use `flatMap` to produce a Stream or `Stream<LineItem>` containing all the line items in all the orders. Furthermore, we also add a `reduce` operation to sum the line items' total amount.

```
FlatMapExample2.java
```

```
package com.mkyong.java8.stream.flatmap;

import java.math.BigDecimal;
import java.util.Arrays;
import java.util.List;

public class FlatMapExample2 {

    public static void main(String[] args) {

        List<Order> orders = findAll();

        /*
            Stream<List<LineItem>> listStream = orders.stream()
                    .map(order -> order.getLineItems());

            Stream<LineItem> lineItemStream = orders.stream()
                    .flatMap(order -> order.getLineItems().stream());
        */

        // sum the line items' total amount
        BigDecimal sumOfLineItems = orders.stream()
                .flatMap(order -> order.getLineItems().stream())    //
Stream<LineItem>
                .map(line -> line.getTotal())                       //
Stream<BigDecimal>
                .reduce(BigDecimal.ZERO, BigDecimal::add);          //  reduce
to sum all

        // sum the order's total amount
        BigDecimal sumOfOrder = orders.stream()
                .map(order -> order.getTotal())                     //
Stream<BigDecimal>
                .reduce(BigDecimal.ZERO, BigDecimal::add);          //  reduce
to sum all

        System.out.println(sumOfLineItems);                         // 3194.20
        System.out.println(sumOfOrder);                             // 3194.20

        if (!sumOfOrder.equals(sumOfLineItems)) {
            System.out.println("The sumOfOrder is not equals to
sumOfLineItems!");
        }

    }

    // create dummy records
    private static List<Order> findAll() {

        LineItem item1 = new LineItem(1, "apple", 1, new BigDecimal("1.20"),
new BigDecimal("1.20"));
        LineItem item2 = new LineItem(2, "orange", 2, new BigDecimal(".50"),
new BigDecimal("1.00"));
        Order order1 = new Order(1, "A0000001", Arrays.asList(item1, item2),
new BigDecimal("2.20"));

        LineItem item3 = new LineItem(3, "monitor BenQ", 5, new
BigDecimal("99.00"), new BigDecimal("495.00"));
        LineItem item4 = new LineItem(4, "monitor LG", 10, new
BigDecimal("120.00"), new BigDecimal("1200.00"));
        Order order2 = new Order(2, "A0000002", Arrays.asList(item3, item4),
new BigDecimal("1695.00"));

        LineItem item5 = new LineItem(5, "One Plus 8T", 3, new
```

```
BigDecimal("499.00"), new BigDecimal("1497.00"));
        Order order3 = new Order(3, "A0000003", Arrays.asList(item5), new
BigDecimal("1497.00"));

        return Arrays.asList(order1, order2, order3);

    }
}
```

Order.java

```
public class Order {

    private Integer id;
    private String invoice;
    private List<LineItem> lineItems;
    private BigDecimal total;

    // getter, setters, constructor
}
```

LineItem.java

```
public class LineItem {

    private Integer id;
    private String item;
    private Integer qty;
    private BigDecimal price;
    private BigDecimal total;

    // getter, setters, constructor
}
```

Output

Terminal

```
3194.20
3194.20
```

5. flatMap example – Splits the line by spaces

This example read a text file, split the line by spaces, and displayed the total number of the words.

A text file.

```
c:\\test\\test.txt
```

```
hello world Java
hello world Python
hello world Node JS
hello world Rust
hello world Flutter
```

Read the comment for self-explanatory.

FlatMapExample3.java

```
package com.mkyong.java8.stream.flatmap;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.stream.Stream;

public class FlatMapExample3 {

    public static void main(String[] args) throws IOException {

        Path path = Paths.get("C:\\test\\test.txt");

        // read file into a stream of lines
        Stream<String> lines = Files.lines(path, StandardCharsets.UTF_8);

        // stream of array...hard to process.
        // Stream<String[]> words = lines.map(line -> line.split(" +"));

        // stream of stream of string....hmm...better flat to one level.
        // Stream<Stream<String>> words = lines.map(line ->
Stream.of(line.split(" +")));

        // result a stream of words, good!
        Stream<String> words = lines.flatMap(line -> Stream.of(line.split("
+")));

        // count the number of words.
        long noOfWords = words.count();

        System.out.println(noOfWords);  // 16
    }
}
```

Output

Terminal

```
16
```

6. flatMap and primitive type

For primitive types like `int`, `long`, `double`, etc. Java 8 Stream also provide related `flatMapTo{primative type}` to flat the Stream of primitive type; the concept is the same.

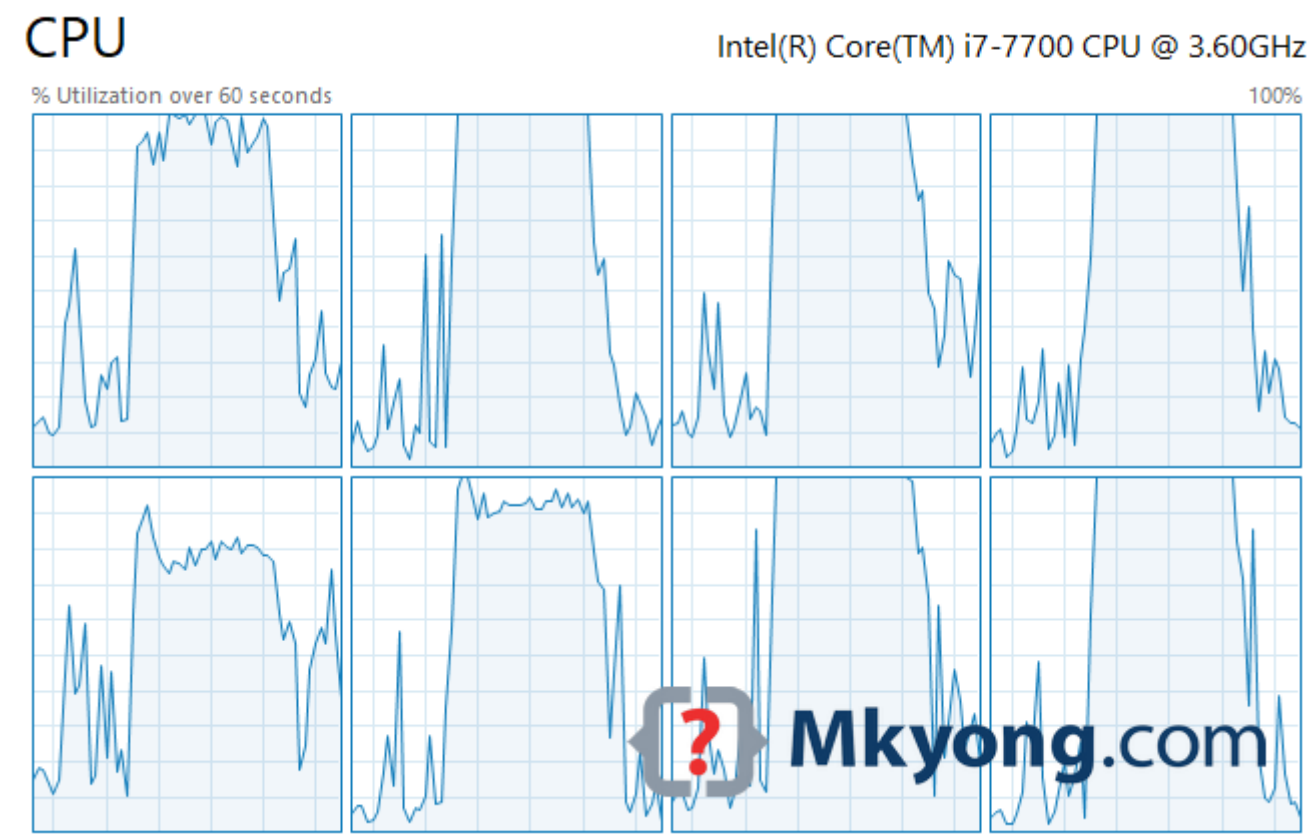
Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Parallel Streams Examples

By *mkyong* | Last updated: July 3, 2019



Few Java 8 examples to execute streams in parallel.

1. BaseStream.parallel()

A simple parallel example to print 1 to 10.

ParallelExample1.java

```
package com.mkyong.java8;

import java.util.stream.IntStream;

public class ParallelExample1 {

    public static void main(String[] args) {

        System.out.println("Normal...");

        IntStream range = IntStream.rangeClosed(1, 10);
        range.forEach(System.out::println);

        System.out.println("Parallel...");

        IntStream range2 = IntStream.rangeClosed(1, 10);
        range2.parallel().forEach(System.out::println);

    }

}
```

Output

```
Normal...
1
2
3
4
5
6
7
8
9
10

Parallel...
7
6
8
9
10
1
4
5
3
2
```

2. Collection.parallelStream()

Another simple parallel example to print `a` to `z`. For collection, we can use `parallelStream()`.

```
ParallelExample2.java
```

```
package com.mkyong.java8;

import java.util.ArrayList;
import java.util.List;

public class ParallelExample2 {

    public static void main(String[] args) {

        System.out.println("Normal...");

        List<String> alpha = getData();
        alpha.stream().forEach(System.out::println);

        System.out.println("Parallel...");

        List<String> alpha2 = getData();
        alpha2.parallelStream().forEach(System.out::println);

    }

    private static List<String> getData() {

        List<String> alpha = new ArrayList<>();

        int n = 97;  // 97 = a , 122 = z
        while (n <= 122) {
            char c = (char) n;
            alpha.add(String.valueOf(c));
            n++;
        }

        return alpha;

    }

}
```

Output

```
Normal...
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
Parallel...
q
s
r
o
x
h
l
p
d
i
g
t
u
n
z
v
j
k
w
f
m
c
a
e
b
y
```

3. Is Stream running in parallel mode?

3.1 We can test it with `isParallel()`

```
ParallelExample3a.java
```

```
package com.mkyong.java8;

import java.util.stream.IntStream;

public class ParallelExample3a {

    public static void main(String[] args) {

        System.out.println("Normal...");

        IntStream range = IntStream.rangeClosed(1, 10);
        System.out.println(range.isParallel());         // false
        range.forEach(System.out::println);

        System.out.println("Parallel...");

        IntStream range2 = IntStream.rangeClosed(1, 10);
        IntStream range2Parallel = range2.parallel();
        System.out.println(range2Parallel.isParallel()); // true
        range2Parallel.forEach(System.out::println);

    }

}
```

3.2 Or print the current thread name like this:

ParallelExample3b.java

```
package com.mkyong.java8;

import java.util.stream.IntStream;

public class ParallelExample3b {

    public static void main(String[] args) {

        System.out.println("Normal...");

        IntStream range = IntStream.rangeClosed(1, 10);
        range.forEach(x -> {
            System.out.println("Thread : " + Thread.currentThread().getName() +
", value: " + x);
        });

        System.out.println("Parallel...");

        IntStream range2 = IntStream.rangeClosed(1, 10);
        range2.parallel().forEach(x -> {
            System.out.println("Thread : " + Thread.currentThread().getName() +
", value: " + x);
        });

    }

}
```

Output

```
Normal...
Thread : main, value: 1
Thread : main, value: 2
Thread : main, value: 3
Thread : main, value: 4
Thread : main, value: 5
Thread : main, value: 6
Thread : main, value: 7
Thread : main, value: 8
Thread : main, value: 9
Thread : main, value: 10

Parallel...
Thread : main, value: 7
Thread : main, value: 6
Thread : ForkJoinPool.commonPool-worker-5, value: 3
Thread : ForkJoinPool.commonPool-worker-7, value: 8
Thread : ForkJoinPool.commonPool-worker-5, value: 5
Thread : ForkJoinPool.commonPool-worker-5, value: 4
Thread : ForkJoinPool.commonPool-worker-3, value: 9
Thread : ForkJoinPool.commonPool-worker-5, value: 1
Thread : ForkJoinPool.commonPool-worker-7, value: 2
Thread : ForkJoinPool.commonPool-worker-9, value: 10
```

P.S By default, parallel streams use `ForkJoinPool`

4. Calculation

4.1 Java 8 streams to print all prime numbers up to 1 million:

```
ParallelExample4.java
```

```
package com.mkyong.java8;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class ParallelExample4 {

    public static void main(String[] args) {

        long count = Stream.iterate(0, n -> n + 1)
                .limit(1_000_000)
                //.parallel()   with this 23s, without this 1m 10s
                .filter(ParallelExample4::isPrime)
                .peek(x -> System.out.format("%s\t", x))
                .count();

        System.out.println("\nTotal: " + count);

    }

    public static boolean isPrime(int number) {
        if (number <= 1) return false;
        return !IntStream.rangeClosed(2, number / 2).anyMatch(i -> number % i
== 0);
    }

}
```

Result:

- For normal streams, it takes 1 minute 10 seconds.
- For parallel streams, it takes 23 seconds.

P.S Tested with i7-7700, 16G RAM, WIndows 10

4.2 Yet another parallel stream example to find out the average age of a list of employees.

```
    List<Employee> employees = obj.generateEmployee(10000);

    double age = employees
            .parallelStream()
            .mapToInt(Employee::getAge)
            .average()
            .getAsDouble();

    System.out.println("Average age: " + age);
```

5. Case Study

5.1 Parallel streams to increase the performance of a time-consuming save file tasks.

This Java code will generate 10,000 random employees and save into 10,000 files, each employee save into a file.

- For normal stream, it takes 27-29 seconds.
- For parallel stream, it takes 7-8 seconds.

P.S Tested with i7-7700, 16G RAM, WIndows 10

```
ParallelExample5.java
```

```
package com.mkyong.java8;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class ParallelExample5 {

    private static final String DIR = System.getProperty("user.dir") +
"/test/";

    public static void main(String[] args) throws IOException {

        Files.createDirectories(Paths.get(DIR));

        ParallelExample5 obj = new ParallelExample5();

        List<Employee> employees = obj.generateEmployee(10000);

        // normal, sequential
        //employees.stream().forEach(ParallelExample5::save);           // 27s-
29s

        // parallel
        employees.parallelStream().forEach(ParallelExample5::save); // 7s-8s

    }

    private static void save(Employee input) {

        try (FileOutputStream fos = new FileOutputStream(new File(DIR +
input.getName() + ".txt"));
             ObjectOutputStream obs = new ObjectOutputStream(fos)) {
            obs.writeObject(input);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    private List<Employee> generateEmployee(int num) {

        return Stream.iterate(0, n -> n + 1)
                .limit(num)
                .map(x -> {
                    return new Employee(
                            generateRandomName(4),
                            generateRandomAge(15, 100),
                            generateRandomSalary(900.00, 200_000.00)
                    );
                })
                .collect(Collectors.toList());

    }
```

Search...

Java Tutorials
Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Stream.iterate examples

By *mkyong* | Last updated: November 28, 2018

In Java 8, we can use `Stream.iterate` to create stream values on demand, so called infinite stream.

1. Stream.iterate

1.1 Stream of 0 – 9

```
    //Stream.iterate(initial value, next value)
    Stream.iterate(0, n -> n + 1)
                .limit(10)
                .forEach(x -> System.out.println(x));
```

Output

```
0
1
2
3
4
5
6
7
8
9
```

1.2 Stream of odd numbers only.

```
    Stream.iterate(0, n -> n + 1)
            .filter(x -> x % 2 != 0) //odd
            .limit(10)
            .forEach(x -> System.out.println(x));
```

Output

```
1
3
5
7
9
11
13
15
17
19
```

1.3 A classic Fibonacci example.

```
Stream.iterate(new int[]{0, 1}, n -> new int[]{n[1], n[0] + n[1]})
    .limit(20)
    .map(n -> n[0])
    .forEach(x -> System.out.println(x));
```

Output

```
0
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
```

1.4 Sum all the Fibonacci values.

```
    int sum = Stream.iterate(new int[]{0, 1}, n -> new int[]{n[1], n[0] +
n[1]})
                .limit(10)
                .map(n -> n[0]) // Stream<Integer>
                .mapToInt(n -> n)
                .sum();

        System.out.println("Fibonacci 10 sum : " + sum);
```

Output

```
Fibonacci 10 sum : 88
```

2. Java 9

The `stream.iterate` was enhanced in Java 9. It supports a predicate (condition) as second argument, and the `stream.iterate` will stop if the predicate is false.

2.1 Stop the stream iteration if `n >= 20`

```
    Stream.iterate(1, n -> n < 20 , n -> n * 2)
           .forEach(x -> System.out.println(x));
```

Output

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – Stream Collectors groupingBy examples

By *mkyong* | Last updated: August 10, 2016

In this article, we will show you how to use Java 8 Stream `Collectors` to group by, count, sum and sort a `List`.

1. Group By, Count and Sort

1.1 Group by a `List` and display the total count of it.

Java8Example1.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

public class Java8Example1 {

    public static void main(String[] args) {

        //3 apple, 2 banana, others 1
        List<String> items =
                Arrays.asList("apple", "apple", "banana",
                        "apple", "orange", "banana", "papaya");

        Map<String, Long> result =
                items.stream().collect(
                        Collectors.groupingBy(
                                Function.identity(), Collectors.counting()
                        )
                );

        System.out.println(result);


    }
}
```

output

```
{papaya=1, orange=1, banana=2, apple=3}
```

1.2 Add sorting.

Java8Example2.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.function.Function;
import java.util.stream.Collectors;

public class Java8Example2 {

    public static void main(String[] args) {

        //3 apple, 2 banana, others 1
        List<String> items =
                Arrays.asList("apple", "apple", "banana",
                        "apple", "orange", "banana", "papaya");

        Map<String, Long> result =
                items.stream().collect(
                        Collectors.groupingBy(
                                Function.identity(), Collectors.counting()
                        )
                );

        Map<String, Long> finalMap = new LinkedHashMap<>();

        //Sort a map and add to finalMap
        result.entrySet().stream()
                .sorted(Map.Entry.<String, Long>comparingByValue()
                        .reversed()).forEachOrdered(e ->
  finalMap.put(e.getKey(), e.getValue()));

        System.out.println(finalMap);


    }
}
```

output

```
{       apple=3, banana=2, papaya=1, orange=1
}
```

2. List Objects

Examples to ‘group by’ a list of user defined Objects.

2.1 A Pojo.

```
Item.java
```

```
package com.mkyong.java8;

import java.math.BigDecimal;

public class Item {

    private String name;
    private int qty;
    private BigDecimal price;

    //constructors, getter/setters
}
```

2.2 Group by the name + Count or Sum the Qty.

Java8Examples3.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

public class Java8Examples3 {

    public static void main(String[] args) {

        //3 apple, 2 banana, others 1
        List<Item> items = Arrays.asList(
                new Item("apple", 10, new BigDecimal("9.99")),
                new Item("banana", 20, new BigDecimal("19.99")),
                new Item("orang", 10, new BigDecimal("29.99")),
                new Item("watermelon", 10, new BigDecimal("29.99")),
                new Item("papaya", 20, new BigDecimal("9.99")),
                new Item("apple", 10, new BigDecimal("9.99")),
                new Item("banana", 10, new BigDecimal("19.99")),
                new Item("apple", 20, new BigDecimal("9.99"))
        );

        Map<String, Long> counting = items.stream().collect(
                Collectors.groupingBy(Item::getName, Collectors.counting()));

        System.out.println(counting);

        Map<String, Integer> sum = items.stream().collect(
                Collectors.groupingBy(Item::getName,
  Collectors.summingInt(Item::getQty)));

        System.out.println(sum);

    }
}
```

output

```
//Group by + Count
{
        papaya=1, banana=2, apple=3, orang=1, watermelon=1
}

//Group by + Sum qty
{
        papaya=20, banana=30, apple=40, orang=10, watermelon=10
}
```

2.2 Group by Price – `Collectors.groupingBy` and `Collectors.mapping` example.

Java8Examples4.java

```
package com.mkyong.java8;

import java.math.BigDecimal;
import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

public class Java8Examples4 {

    public static void main(String[] args) {

        //3 apple, 2 banana, others 1
        List<Item> items = Arrays.asList(
                new Item("apple", 10, new BigDecimal("9.99")),
                new Item("banana", 20, new BigDecimal("19.99")),
                new Item("orang", 10, new BigDecimal("29.99")),
                new Item("watermelon", 10, new BigDecimal("29.99")),
                new Item("papaya", 20, new BigDecimal("9.99")),
                new Item("apple", 10, new BigDecimal("9.99")),
                new Item("banana", 10, new BigDecimal("19.99")),
                new Item("apple", 20, new BigDecimal("9.99"))
                );

        //group by price
        Map<BigDecimal, List<Item>> groupByPriceMap =
            items.stream().collect(Collectors.groupingBy(Item::getPrice));

        System.out.println(groupByPriceMap);

        // group by price, uses 'mapping' to convert List<Item> to Set<String>
        Map<BigDecimal, Set<String>> result =
                items.stream().collect(
                        Collectors.groupingBy(Item::getPrice,
                                Collectors.mapping(Item::getName,
Collectors.toSet())
                        )
                );

        System.out.println(result);

    }
}
```

output

```
{
        19.99=[
                        Item{name='banana', qty=20, price=19.99},
                        Item{name='banana', qty=10, price=19.99}
                ],
        29.99=[
                        Item{name='orang', qty=10, price=29.99},
                        Item{name='watermelon', qty=10, price=29.99}
                ],
        9.99=[
                        Item{name='apple', qty=10, price=9.99},
                        Item{name='papaya', qty=20, price=9.99},
                        Item{name='apple', qty=10, price=9.99},
                        Item{name='apple', qty=20, price=9.99}
                ]
}

//group by + mapping to Set
{
        19.99=[banana],
        29.99=[orang, watermelon],
        9.99=[papaya, apple]
}
```

References

1. *Java 8 Stream Collectors JavaDoc*
2. *Java – How to sort a Map*
3. *Stackoverflow – Sort a Map by values (Java)*

Related Articles

- *Java 8 Streams filter examples*
- *Java 8 - Find duplicate elements in a Stream*
- *Java - How to convert String to Char Array*
- *Java - How to join Arrays*
- *Java 8 Stream - Read a file line by line*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – Filter a null value from a Stream

By *mkyong* | Last updated: August 10, 2016

Review a `Stream` containing `null` values.

Java8Examples.java

```
package com.mkyong.java8;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java8Examples {

    public static void main(String[] args) {

        Stream<String> language = Stream.of("java", "python", "node", null,
"ruby", null, "php");

        List<String> result = language.collect(Collectors.toList());

        result.forEach(System.out::println);

    }
}
```

output

```
java
python
node
null   // <--- NULL
ruby
null   // <--- NULL
php
```

Solution

To solve it, uses `Stream.filter(x -> x!=null)`

Java8Examples.java

```
package com.mkyong.java8;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java8Examples {

    public static void main(String[] args) {

        Stream<String> language = Stream.of("java", "python", "node", null,
"ruby", null, "php");

        //List<String> result = language.collect(Collectors.toList());

        List<String> result = language.filter(x ->
x!=null).collect(Collectors.toList());

        result.forEach(System.out::println);


    }
}
```

output

```
java
python
node
ruby
php
```

Alternatively, filter with `Objects::nonNull`

```
import java.util.List;

    List<String> result =
language.filter(Objects::nonNull).collect(Collectors.toList());
```

References

1. *Objects::nonNull JavaDoc*
2. *Java 8 Streams filter examples*
3. *Java 8 Collectors JavaDoc*

Related Articles

- *Java 8 Streams filter examples*
- *Java - How to convert String to Char Array*
- *Java - How to join Arrays*
- *Java 8 Stream - Read a file line by line*
- *Java 8 - Filter a Map examples*

mkyong

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Stream findFirst() and findAny()

By *mkyong* | Last updated: February 7, 2020

In Java 8 Stream, the `findFirst()` returns the first element from a Stream, while `findAny()` returns any element from a Stream.

1. findFirst()

1.1 Find the first element from a Stream of Integers.

Java8FindFirstExample1.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

public class Java8FindFirstExample1 {

    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3, 2, 1);

        Optional<Integer> first = list.stream().findFirst();
        if (first.isPresent()) {
            Integer result = first.get();
            System.out.println(result);       // 1
        } else {
            System.out.println("no value?");
        }

        Optional<Integer> first2 = list
                .stream()
                .filter(x -> x > 1).findFirst();

        if (first2.isPresent()) {
            System.out.println(first2.get()); // 2
        } else {
            System.out.println("no value?");
        }
    }

}
```

Output

```
1
2
```

1.2 Find the first element from a Stream of String which is not equal to "node".

Java8FindFirstExample2.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

public class Java8FindFirstExample2 {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("node", "java", "python", "ruby");

        Optional<String> result = list.stream()
                .filter(x -> !x.equalsIgnoreCase("node"))
                .findFirst();

        if (result.isPresent()) {
            System.out.println(result.get()); // java
        } else {
            System.out.println("no value?");
        }

    }

}
```

Output

```
java
```

2. findAny()

2.1 Find any element from a Stream of Integers. If we run the below program, most of the time, the result is 2, it looks like `findAny()` always returns the first element? But, there is **no guaranteed** for this, `findAny()` may return any element from a Stream.

Java8FindAnyExample1.java

```
package com.mkyong.java8;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

public class Java8FindAnyExample1 {

    public static void main(String[] args) {

        List<Integer> list = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

        Optional<Integer> any = list.stream().filter(x -> x > 1).findAny();
        if (any.isPresent()) {
            Integer result = any.get();
            System.out.println(result);
        }

    }

}
```

Output

```
2 // no guaranteed
```

References

- *Java 8 Streams filter examples*
- *Java 8 Stream JavaDoc*
- *Stream.findFirst() JavaDoc*
- *Stream.findAny() JavaDoc*

Related Articles

- *Java 8 - Convert a Stream to List*
- *Java 8 Predicate Examples*
- *How to copy directory in Java*
- *How to delete directory in Java*
- *Java 8 Streams filter examples*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials
Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Stream.reduce() examples

By *mkyong* | Last updated: February 24, 2020

In Java 8, the `Stream.reduce()` combine elements of a stream and produces a single value.

A simple sum operation using a for loop.

```
int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int sum = 0;
for (int i : numbers) {
    sum += i;
}

System.out.println("sum : " + sum); // 55
```

The equivalent in `Stream.reduce()`

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

// 1st argument, init value = 0
int sum = Arrays.stream(numbers).reduce(0, (a, b) -> a + b);

System.out.println("sum : " + sum); // 55
```

or method reference with `Integer::sum`

```
int sum = Arrays.stream(numbers).reduce(0, Integer::sum); // 55
```

Integer.java

```
    /**
     * Adds two integers together as per the + operator.
     *
     * @param a the first operand
     * @param b the second operand
     * @return the sum of {@code a} and {@code b}
     * @see java.util.function.BinaryOperator
     * @since 1.8
     */
    public static int sum(int a, int b) {
        return a + b;
    }
```

1. Method Signature

1.1 Review the `Stream.reduce()` method signature:

Stream.java

```
T reduce(T identity, BinaryOperator<T> accumulator);
```

IntStream.java

```
int reduce(int identity, IntBinaryOperator op);
```

LongStream.java

```
long reduce(int identity, LongBinaryOperator op);
```

- identity = default or initial value.
- BinaryOperator = functional interface, take two values and produces a new value.

1.2 if the `identity` argument is missing, there is no default or initial value, and it returns an optional.

Stream.java

```
Optional<T> reduce(BinaryOperator<T> accumulator);
```

2. More Examples

2.1 Math operations.

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

int sum = Arrays.stream(numbers).reduce(0, (a, b) -> a + b);    // 55
int sum2 = Arrays.stream(numbers).reduce(0, Integer::sum);      // 55

int sum3 = Arrays.stream(numbers).reduce(0, (a, b) -> a - b);   // -55
int sum4 = Arrays.stream(numbers).reduce(0, (a, b) -> a * b);   // 0, initial
is 0, 0 * whatever = 0
int sum5 = Arrays.stream(numbers).reduce(0, (a, b) -> a / b);   // 0
```

2.2 Max and Min.

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

int max = Arrays.stream(numbers).reduce(0, (a, b) -> a > b ? a : b);  // 10
int max1 = Arrays.stream(numbers).reduce(0, Integer::max);            // 10

int min = Arrays.stream(numbers).reduce(0, (a, b) -> a < b ? a : b);  // 0
int min1 = Arrays.stream(numbers).reduce(0, Integer::min);            // 0
```

2.3 Join String.

```
String[] strings = {"a", "b", "c", "d", "e"};

// |a|b|c|d|e , the initial | join is not what we want
String reduce = Arrays.stream(strings).reduce("", (a, b) -> a + "|" + b);

// a|b|c|d|e, filter the initial "" empty string
String reduce2 = Arrays.stream(strings).reduce("", (a, b) -> {
    if (!"".equals(a)) {
        return a + "|" + b;
    } else {
        return b;
    }
});

// a|b|c|d|e , better uses the Java 8 String.join :)
String join = String.join("|", strings);
```

3. Map & Reduce

A simple map and reduce example to sum `BigDecimal` from a list of invoices.

```
JavaReduce.java
```

```
package com.mkyong;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Arrays;
import java.util.List;

public class JavaReduce {

    public static void main(String[] args) {

        List<Invoice> invoices = Arrays.asList(
                new Invoice("A01", BigDecimal.valueOf(9.99),
BigDecimal.valueOf(1)),
                new Invoice("A02", BigDecimal.valueOf(19.99),
BigDecimal.valueOf(1.5)),
                new Invoice("A03", BigDecimal.valueOf(4.99),
BigDecimal.valueOf(2))
        );

        BigDecimal sum = invoices.stream()
                .map(x -> x.getQty().multiply(x.getPrice()))    // map
                .reduce(BigDecimal.ZERO, BigDecimal::add);      // reduce

        System.out.println(sum);    // 49.955
        System.out.println(sum.setScale(2, RoundingMode.HALF_UP));  // 49.96

    }

}

class Invoice {

    String invoiceNo;
    BigDecimal price;
    BigDecimal qty;

    // getters, stters n constructor
}
```

Output

```
49.955
49.96
```

References

- *Reduction JavaDoc*
- *BinaryOperator JavaDoc*
- *Java – How to join List String with commas*
- *Java 8 – StringJoiner example*

Related Articles

- *Java 8 - How to Sum BigDecimal using Stream?*
- *Java 8 - Convert List to Map*

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – Convert a Stream to List

By *mkyong* | Last updated: March 18, 2019

A Java 8 example to show you how to convert a `Stream` to a `List` via `Collectors.toList`

Java8Example1.java

```
package com.mkyong.java8;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java8Example1 {

    public static void main(String[] args) {

        Stream<String> language = Stream.of("java", "python", "node");

        //Convert a Stream to List
        List<String> result = language.collect(Collectors.toList());

        result.forEach(System.out::println);

    }
}
```

output

```
java
python
node
```

Yet another example, filter a number 3 and convert it to a List.

Java8Example2.java

```
package com.mkyong.java8;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java8Example2 {

    public static void main(String[] args) {

        Stream<Integer> number = Stream.of(1, 2, 3, 4, 5);

        List<Integer> result2 = number.filter(x -> x !=
3).collect(Collectors.toList());

        result2.forEach(x -> System.out.println(x));


    }
}
```

output

```
1
2
4
5
```

References

- *Java 8 Collectors JavaDoc*
- *Java 8 Streams filter examples*
- *Java 8 – How to convert a Stream to Array*

Related Articles

- *Java 8 - Convert List to Map*
- *Java - How to search a string in a List?*
- *Java 8 Stream findFirst() and findAny()*
- *Java 8 Predicate Examples*
- *How to copy directory in Java*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – How to Sum BigDecimal using Stream?

By *mkyong* | Last updated: February 24, 2020

In Java 8, we can use the `Stream.reduce()` to sum a list of `BigDecimal`.

1. Stream.reduce()

Java example to sum a list of `BigDecimal` values, using a normal for loop and a `stream.reduce()`.

JavaBigDecimal.java

```
package com.mkyong;

import java.math.BigDecimal;
import java.util.LinkedList;
import java.util.List;

public class JavaBigDecimal {

    public static void main(String[] args) {

        List<BigDecimal> invoices = new LinkedList<>();
        invoices.add(BigDecimal.valueOf(9.9));
        invoices.add(BigDecimal.valueOf(1.0));
        invoices.add(BigDecimal.valueOf(19.99));
        invoices.add(BigDecimal.valueOf(0.2));
        invoices.add(BigDecimal.valueOf(5.5));

        // sum using a for loop
        BigDecimal sum = BigDecimal.ZERO;
        for (BigDecimal amt : invoices) {
            sum = sum.add(amt);
        }
        System.out.println("Sum = " + sum);

        // sum using stream
        BigDecimal sum2 = invoices.stream().reduce(BigDecimal.ZERO,
BigDecimal::add);
        System.out.println("Sum (Stream) = " + sum2);

    }

}
```

Output

```
Sum = 36.59
Sum (Stream) = 36.59
```

2. Map & Reduce

Sum all `BigDecimal` from a list of `Invoices`.

```
JavaBigDecimalObject.java
```

```
package com.mkyong;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Arrays;
import java.util.List;

public class JavaBigDecimalObject {

    public static void main(String[] args) {

        List<Invoice> invoices = Arrays.asList(
                new Invoice("I1001", BigDecimal.valueOf(9.99),
BigDecimal.valueOf(1)),
                new Invoice("I1002", BigDecimal.valueOf(19.99),
BigDecimal.valueOf(1.5)),
                new Invoice("I1003", BigDecimal.valueOf(4.888),
BigDecimal.valueOf(2)),
                new Invoice("I1004", BigDecimal.valueOf(4.99),
BigDecimal.valueOf(5)),
                new Invoice("I1005", BigDecimal.valueOf(.5),
BigDecimal.valueOf(2.3))
        );

        BigDecimal sum = invoices.stream()
                .map(x -> x.getQty().multiply(x.getPrice()))    // map
                .reduce(BigDecimal.ZERO, BigDecimal::add);      // reduce

        System.out.println(sum);    // 75.851
        System.out.println(sum.setScale(2, RoundingMode.HALF_UP));  // 75.85

    }

}

class Invoice {

    String invoiceNo;
    BigDecimal price;
    BigDecimal qty;

    public Invoice(String invoiceNo, BigDecimal price, BigDecimal qty) {
        this.invoiceNo = invoiceNo;
        this.price = price;
        this.qty = qty;
    }

    public String getInvoiceNo() {
        return invoiceNo;
    }

    public void setInvoiceNo(String invoiceNo) {
        this.invoiceNo = invoiceNo;
    }

    public BigDecimal getPrice() {
        return price;
    }

    public void setPrice(BigDecimal price) {
        this.price = price;
    }

    public BigDecimal getQty() {
```

```
            return qty;
        }

        public void setQty(BigDecimal qty) {
            this.qty = qty;
        }
    }
```

Output

```
75.851
75.85
```

References

- *Java 8 Stream.reduce() examples*
- *How to calculate monetary values in Java*

Related Articles

- *Java 8 Stream.reduce() examples*
- *Java 8 flatMap example*
- *Java 8 Stream.iterate examples*
- *Java 8 - Get the last element of a Stream?*
- *How to copy directory in Java*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Stream – Read a file line by line

By *mkyong* | Last updated: October 29, 2015

In Java 8, you can use `Files.lines` to read file as `Stream`.

c://lines.txt – A simple text file for testing

```
line1
line2
line3
line4
line5
```

1. Java 8 Read File + Stream

TestReadFile.java

```
package com.mkyong.java8;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Stream;

public class TestReadFile {

    public static void main(String args[]) {

        String fileName = "c://lines.txt";

        //read file into stream, try-with-resources
        try (Stream<String> stream = Files.lines(Paths.get(fileName))) {

            stream.forEach(System.out::println);

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

Output

```
line1
line2
line3
line4
line5
```

2. Java 8 Read File + Stream + Extra

This example shows you how to use `Stream` to filter content, convert the entire content to upper case and return it as a `List`.

TestReadFile2.java

```
package com.mkyong.java8;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class TestReadFile2 {

    public static void main(String args[]) {

        String fileName = "c://lines.txt";
        List<String> list = new ArrayList<>();

        try (Stream<String> stream = Files.lines(Paths.get(fileName))) {

            //1. filter line 3
            //2. convert all content to upper case
            //3. convert it into a List
            list = stream
                    .filter(line -> !line.startsWith("line3"))
                    .map(String::toUpperCase)
                    .collect(Collectors.toList());

        } catch (IOException e) {
            e.printStackTrace();
        }

        list.forEach(System.out::println);

    }

}
```

Output

```
LINE1
LINE2
LINE4
LINE5
```

3. BufferedReader + Stream

A new method `lines()` has been added since 1.8, it lets `BufferedReader` returns content as `Stream`.

TestReadFile3.java

```
package com.mkyong.java8;

import java.io.BufferedReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class TestReadFile3{

    public static void main(String args[]) {

        String fileName = "c://lines.txt";
        List<String> list = new ArrayList<>();

        try (BufferedReader br = Files.newBufferedReader(Paths.get(fileName)))
{

            //br returns as stream and convert it into a List
            list = br.lines().collect(Collectors.toList());

        } catch (IOException e) {
            e.printStackTrace();
        }

        list.forEach(System.out::println);

    }

}
```

Output

```
line1
line2
line3
line4
line5
```

4. Classic BufferedReader And Scanner

Enough of Java 8 and `Stream`, let revisit the classic `BufferedReader` (JDK1.1) and `Scanner` (JDK1.5) examples to read a file line by line, it is working still, just developers are moving toward `Stream`.

4.1 `BufferedReader` + try-with-resources example.

```
TestReadFile4.java
```

```
package com.mkyong.core;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class TestReadFile4{

    public static void main(String args[]) {

        String fileName = "c://lines.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(fileName)))
{

            String line;
            while ((line = br.readLine()) != null) {
                System.out.println(line);
            }

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

4.2 `Scanner` + try-with-resources example.

TestReadFile5.java

```
package com.mkyong.core;

import java.io.File;
import java.io.IOException;
import java.util.Scanner;

public class TestReadFile5 {

    public static void main(String args[]) {

        String fileName = "c://lines.txt";

        try (Scanner scanner = new Scanner(new File(fileName))) {

            while (scanner.hasNext()){
                System.out.println(scanner.nextLine());
            }

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

References

1. *Java 8 File.lines()*
2. *Java 8 Stream*
3. *Java BufferedReader*

Related Articles

- *Java - How to convert String to Char Array*
- *Java - How to join Arrays*
- *Java 8 Streams filter examples*
- *Java 8 - Filter a Map examples*
- *Java 8 – Stream Collectors groupingBy examples*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials
Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Stream – Convert List<List<String>> to List<String>

By *mkyong* | Last updated: July 18, 2019

As title, we can use `flatMap` to convert it.

Java9Example1.java

```
package com.mkyong.test;

import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.util.stream.Collectors;

public class Java9Example1 {

    public static void main(String[] args) {

        List<String> numbers = Arrays.asList("1", "2", "A", "B", "C1D2E3");

        List<List<String>> collect = numbers.stream()
                .map(x -> new Scanner(x).findAll("\\D+")
                        .map(m -> m.group())
                        .collect(Collectors.toList())
                )
                .collect(Collectors.toList());

        collect.forEach(x -> System.out.println(x));

    }

}
```

Output

```
[]
[]
[A]
[B]
[C, D, E]
```

Solution

Java9Example2.java

```
package com.mkyong.test;

import java.util.Arrays;
import java.util.List;
import java.util.Scanner;
import java.util.stream.Collectors;

public class Java9Example2 {

    public static void main(String[] args) {

        List<String> numbers = Arrays.asList("1", "2", "A", "B", "C1D2E3");

        List<String> collect = numbers.stream()
                .map(x -> new Scanner(x).findAll("\\D+")
                        .map(m -> m.group())
                        .collect(Collectors.toList())
                )
// List<List<String>>
                .flatMap(List::stream)                                  //
List<String>
                .collect(Collectors.toList());


        collect.forEach(x -> System.out.println(x));

    }

}
```

Output

```
A
B
C
D
E
```

References

- *Java Regular Expression Examples*
- *Scanner.findAll JavaDocs*

Related Articles

- *Java Regular Expression Examples*
- *Java 8 flatMap example*
- *Java 8 Stream.iterate examples*
- *Java 8 Stream - The peek() is not working with cou*
- *Java Prime Numbers examples*

mkyong

Search...

Java Tutorials
Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 Stream – The peek() is not working with count()?

By *mkyong* | Last updated: June 14, 2019

Many examples are using the `.count()` as the terminal operation for `.peek()`, for example:

Java 8

```
List<String> l = Arrays.asList("A", "B", "C", "D");

long count = l.stream().peek(System.out::println).count();

System.out.println(count); // 4
```

Output – It's working fine.

```
A
B
C
D
4
```

However, for Java 9 and above, the `peek()` may print nothing:

Java 9 and above

```
List<String> l = Arrays.asList("A", "B", "C", "D");

long count = l.stream().peek(System.out::println).count();

System.out.println(count); // 4
```

Output

```
4
```

Why peek() print nothing now?

Refer to the Java 9 *.count()* Java docs

```
An implementation may choose to not execute the stream pipeline (either
sequentially or in parallel)
if it is capable of computing the count directly from the stream source.
In such cases no source elements will be traversed and no intermediate
operations will be evaluated.
```

Since Java 9, if JDK compiler is able computing the `count` directly from the stream (optimization in Java 9), it didn't traverse the stream, so there is no need to run `peek()` at all.

```
List<String> l = Arrays.asList("A", "B", "C", "D");

// JDK compiler know the size of the stream via the variable l
long count = l.stream().peek(System.out::println).count();
```

To force the `peek()` to run, just alter some elements with `filter()` or switch to another terminal operation like `collect()`

filter()

```
List<String> l = Arrays.asList("A", "B", "C", "D");

long count = l.stream()
        .filter(x->!x.isEmpty())
        .peek(System.out::println)
        .count();

System.out.println(count); // 4
```

Output

```
A
B
C
D
4
```

collect()

```
List<String> l = Arrays.asList("A", "B", "C", "D");

List<String> result = l.stream()
        .peek(System.out::println)
        .collect(Collectors.toList());

System.out.println(result.size()); // 4
```

Output

```
A
B
C
D
4
```

Be careful of mixing `.peek()` with `.count()`, the `peek()` may not work as expected in Java 9 and above.

P.S Tested with Java 12

References

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – Should we close the Stream after use?

By *mkyong* | Last updated: May 3, 2019

Only Streams whose source are an IO channel like `Files.lines(Path, Charset)` need to be closed.

Read this *Stream JavaDocs*

> Streams have a BaseStream.close() method and implement AutoCloseable, but nearly all stream instances do not actually need to be closed after use. Generally, only streams whose source is an IO channel (such as those returned by Files.lines(Path, Charset)) will require closing. Most streams are backed by collections, arrays, or generating functions, which require no special resource management. (If a stream does require closing, it can be declared as a resource in a try-with-resources statement.)

1. For normal Stream like this, the stream instance does not need to be closed after use.

Java8StreamExample.java

```
package com.mkyong;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java8StreamExample {

    public static void main(String[] args) {

        Stream<String> stream = Stream.of("A", "B", "C");

        List<String> filter = stream.filter(x -> !x.equalsIgnoreCase("B"))
            .collect(Collectors.toList());

        // no need close the stream.
        //stream.close();

        System.out.println(filter); // [A, C]

    }

}
```

2. For Stream whose source are an IO channel, close it with `try-with-resources`

Java8StreamIO.java

```
package com.mkyong;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class Java8StreamIO {

    public static void main(String[] args) {

        String path = "c:\\projects\\app.log";

        // auto close
        try (Stream<String> lines = Files.lines(Paths.get(path))) {

            String content =
lines.collect(Collectors.joining(System.lineSeparator()));

        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

References

- *Stream JavaDocs*
- *Java 8 Stream – Read a file line by line*

Related Articles

- *How to copy directory in Java*
- *How to delete directory in Java*
- *Java 8 - Convert List to Map*
- *Java 8 flatMap example*
- *Java 8 - Convert a Stream to List*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – Convert a Stream to Array

By *mkyong* | Last updated: March 18, 2019

In Java 8, we can use `.toArray()` to convert a Stream into an Array.

1. Stream -> String[]

StreamString.java

```
package com.mkyong;

import java.util.Arrays;

public class StreamString {

    public static void main(String[] args) {

        String lines = "I Love Java 8 Stream!";

        // split by space, uppercase, and convert to Array
        String[] result = Arrays.stream(lines.split("\\s+"))
            .map(String::toUpperCase)
            .toArray(String[]::new);

        for (String s : result) {
            System.out.println(s);
        }

    }

}
```

Output

```
I
LOVE
JAVA
8
STREAM!
```

2. IntStream -> Integer[] or int[]

2.1 Stream to `Integer[]`

StreamInt1.java

```
package com.mkyong;

import java.util.Arrays;

public class StreamInt1 {

    public static void main(String[] args) {

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

        Integer[] result = Arrays.stream(num)
            .map(x -> x * 2)
            .boxed()
            .toArray(Integer[]::new);

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

    }

}
```

Output

```
[2, 4, 6, 8, 10]
```

2.2 Stream to `int[]`

StreamInt2.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class StreamInt2 {

    public static void main(String[] args) {

        // IntStream -> int[]
        int[] stream1 = IntStream.rangeClosed(1, 5).toArray();
        System.out.println(Arrays.toString(stream1));

        // Stream<Integer> -> int[]
        Stream<Integer> stream2 = Stream.of(1, 2, 3, 4, 5);
        int[] result = stream2.mapToInt(x -> x).toArray();

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

    }

}
```

Output

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
```

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – How to convert IntStream to Integer[]

By *mkyong* | Last updated: March 18, 2019

The key is `boxed()` the `IntStream` into a `Stream<Integer>`, then only convert to an Array.

StreamExample.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.stream.IntStream;
import java.util.stream.Stream;

public class StreamExample {

    public static void main(String[] args) {

        //int[] -> IntStream -> Stream<Integer> -> Integer[]
        int[] num = {3, 4, 5};

        //1. int[] -> IntStream
        IntStream stream = Arrays.stream(num);

        //2. IntStream -> Stream<Integer>
        Stream<Integer> boxed = stream.boxed();

        //3. Stream<Integer> -> Integer[]
        Integer[] result = boxed.toArray(Integer[]::new);

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

        // one line
        Integer[] oneLineResult =
Arrays.stream(num).boxed().toArray(Integer[]::new);
        System.out.println(Arrays.toString(oneLineResult));
    }

}
```

Output

```
[3, 4, 5]
[3, 4, 5]
```

References

- *Java – How to declare and initialize an Array*
- *Stream toArray()*
- *Stream boxed()*

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – How to convert IntStream to int or int[]

By *mkyong* | Last updated: March 2, 2020

Few Java 8 examples to get a primitive `int` from an `IntStream`.

1. IntStream -> int

Java8Example1.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.OptionalInt;
import java.util.stream.IntStream;

public class Java8Example1 {

    public static void main(String[] args) {

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

        //1. int[] -> IntStream
        IntStream stream = Arrays.stream(num);

        // 2. OptionalInt
        OptionalInt first = stream.findFirst();

        // 3. getAsInt()
        int result = first.getAsInt();

        System.out.println(result);                                        // 1

        // one line
        System.out.println(Arrays.stream(num).findFirst().getAsInt());  // 1

    }

}
```

Output

```
1
1
```

Java8Example2.java

```
package com.mkyong;

import java.util.Arrays;
import java.util.OptionalInt;
import java.util.stream.IntStream;

public class Java8Example2 {

    public static void main(String[] args) {

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

        //1. int[] -> IntStream
        IntStream stream = Arrays.stream(num);

        // 2. OptionalInt
        OptionalInt any = stream.filter(x -> x % 2 == 0).findAny();

        // 3. getAsInt()
        int result = any.getAsInt();

        System.out.println(result); // 2 or 4

    }

}
```

Output

```
2 or 4
```

2. IntStream -> int[] or Integer[]

```
Java8Example3.java
```

```
package com.mkyong;

import java.util.Arrays;
import java.util.stream.IntStream;

public class Java8Example3 {

    public static void main(String[] args) {

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

        IntStream stream = Arrays.stream(num);

        // IntStream -> int[]
        int[] ints = stream.toArray();

        IntStream stream2 = Arrays.stream(num);

        // IntStream -> Integer[]
        Integer[] integers = stream2.boxed().toArray(Integer[]::new);

    }

}
```

References

- *Java 8 – How to convert IntStream to Integer Array*
- *IntStream toArray()*

Related Articles

- *Java 8 - How to convert IntStream to Integer[]*
- *How to copy directory in Java*
- *How to delete directory in Java*
- *Java 8 - Convert List to Map*
- *Java 8 flatMap example*

mkyong

Founder of *Mkyong.com*, love Java and open source stuff. Follow him on *Twitter*. If you like my tutorials, consider make a donation to *these charities*.

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java 8 – How to sort list with stream.sorted()

By *mkyong* | Last updated: March 6, 2019

Few examples to show you how to sort a `List` with `stream.sorted()`

1. List

1.1 Sort a List with `Comparator.naturalOrder()`

```
package com.mkyong.sorted;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class StreamApplication {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("9", "A", "Z", "1", "B", "Y", "4",
"a", "c");

        /*
        List<String> sortedList = list.stream()
            .sorted(Comparator.naturalOrder())
            .collect(Collectors.toList());

        List<String> sortedList = list.stream()
            .sorted((o1,o2)-> o1.compareTo(o2))
            .collect(Collectors.toList());
        */

        List<String> sortedList =
list.stream().sorted().collect(Collectors.toList());

        sortedList.forEach(System.out::println);

    }
}
```

Output

```
1
4
9
A
B
Y
Z
a
c
```

1.2 Sort a List with `Comparator.reverseOrder()`

```
package com.mkyong.sorted;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class StreamApplication {

    public static void main(String[] args) {

        List<String> list = Arrays.asList("9", "A", "Z", "1", "B", "Y", "4",
"a", "c");

        /*
        List<String> sortedList = list.stream()
            .sorted((o1,o2)-> o2.compareTo(o1))
            .collect(Collectors.toList());
        */

        List<String> sortedList = list.stream()
            .sorted(Comparator.reverseOrder())
            .collect(Collectors.toList());

        sortedList.forEach(System.out::println);

    }
}
```

Output

```
c
a
Z
Y
B
A
9
4
1
```

2. List Objects

1.1 Sort by age, natural order.

```
package com.mkyong.sorted;

import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.stream.Collectors;

public class StreamApplication {

    static List<User> users = Arrays.asList(
            new User("C", 30),
            new User("D", 40),
            new User("A", 10),
            new User("B", 20),
            new User("E", 50));

    public static void main(String[] args) {

        /*List<User> sortedList = users.stream()
            .sorted((o1, o2) -> o1.getAge() - o2.getAge())
            .collect(Collectors.toList());*/

        List<User> sortedList = users.stream()
            .sorted(Comparator.comparingInt(User::getAge))
            .collect(Collectors.toList());

        sortedList.forEach(System.out::println);

    }

    static class User {

        private String name;
        private int age;

        public User(String name, int age) {
            this.name = name;
            this.age = age;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        public int getAge() {
            return age;
        }

        public void setAge(int age) {
            this.age = age;
        }

        @Override
        public String toString() {
            return "User{" +
                    "name='" + name + '\'' +
                    ", age=" + age +
                    '}';
        }
```

```
    }
}
```

Output

```
User{name='A', age=10}
User{name='B', age=20}
User{name='C', age=30}
User{name='D', age=40}
User{name='E', age=50}
```

1.2 reverse order.

```
    List<User> sortedList = users.stream()
        .sorted(Comparator.comparingInt(User::getAge)
        .reversed())
        .collect(Collectors.toList());

    sortedList.forEach(System.out::println);
```

Output

```
User{name='E', age=50}
User{name='D', age=40}
User{name='C', age=30}
User{name='B', age=20}
User{name='A', age=10}
```

1.3 Sort by name

```
    /*List<User> sortedList = users.stream()
        .sorted((o1, o2) -> o1.getName().compareTo(o2.getName()))
        .collect(Collectors.toList());*/

    List<User> sortedList = users.stream()
        .sorted(Comparator.comparing(User::getName))
        .collect(Collectors.toList());
```

References

- *Java 8 Lambda : Comparator example*
- *Java 8 – How to sort a Map*
- *Stream sorted docs*

Related Articles

- *Java 8 - Convert List to Map*
- *How to copy directory in Java*
- *How to delete directory in Java*
- *Java 8 flatMap example*
- *Java 8 - Convert a Stream to List*

Search...

Java Tutorials

Java 15
Java 14
Java 13
Java 12
Java 11 (LTS)
Java 8 (LTS)
Java IO / NIO
Java JDBC
Java JSON
Java CSV
Java XML
Spring Boot
JUnit 5
Maven
Misc

Java – How to sum all the stream integers

By *mkyong* | Last updated: November 28, 2018

The `sum()` method is available in the primitive int-value stream like `IntStream`, not `Stream<Integer>`. We can use `mapToInt()` to convert a stream integers into a `IntStream`.

```
int sum = integers.stream().mapToInt(Integer::intValue).sum();
int sum = integers.stream().mapToInt(x -> x).sum();
```

Full example.

Java8Stream.java

```
package com.mkyong.concurrency;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Stream;

public class Java8Stream {

    public static void main(String[] args) {

        List<Integer> integers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
        int sum = integers.stream().mapToInt(Integer::intValue).sum();
        System.out.println("Total : " + sum);

        Stream<Integer> integers2 = Stream.iterate(1, n -> n + 1).limit(10);
        IntStream intStream = integers2.mapToInt(x -> x);
        int sum1 = intStream.sum();
        System.out.println("Total : " + sum1);

    }

}
```

Output

```
Total : 55
Total : 55
```

References

1. *Oracle doc – IntStream.html*

Related Articles

- *Java 8 flatMap example*