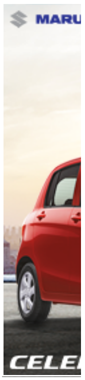
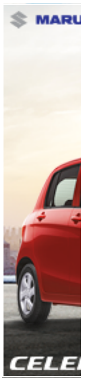
Java 8 Streams - Side Effects

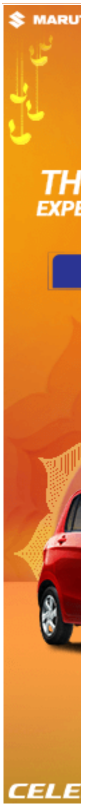
[Updated: Nov 17, 2016, Created: Nov 17, 2016]

A side effect is an action taken from a stream operation which changes something externally.

The change may be changing some variable values in the program or it can be sending a JMS message, sending email, printing with System.out.println or in a UI application changing the state of a button from enabled to disabled etc.







What operations should apply side-effects?

The pipeline operations `ForEach()`, `ForEachOrdered()` and `peek()` which returns void, are meant to produce side effects.

The intermediate operations with behavioral parameters which usually return a non-void value should entirely be avoided to apply side effects e.g. `filter()` , `map()` etc.

Intermediate operation peek() should be limited to side-effects like logging and debugging only.

Side-effects and stateful/stateless operations

Side effects may be applied via a stateful action (not recommended) i.e. side effects updating some mutable shared variable.

Side effects can also be applied via stateless actions e.g. logging, sending messages etc.

Stateful operations should entirely be avoided as we discussed in the last tutorial , even if we use them from operations like `forEach()`. Side effects which work in a stateless manner should be applied only from forEach(), forEachOrdered() and peek().

> Side-effects in behavioral parameters to stream operations are, in general, discouraged, as they can often lead to unwitting violations of the statelessness requirement, as well as other thread-safety hazards.

> If the behavioral parameters do have side-effects, unless explicitly stated, there are no guarantees as to the visibility of those side-effects to other threads, nor are there any guarantees that different operations on the "same" element within the same stream pipeline are executed in the same thread

Side-effects and parallelism

In parallel pipeline, the operation `peek()` and `forEach()` do not respect encounter order . They are also non-deterministic (producing different results on multiple executions). We should only apply side-effects from these operations if we don't care about the order. If we do care about the order then we have only one option: `forEachOrdered()`.

Attempts like doing some kind of locking or coordination between threads should entirely be avoided.

Examples

Applying side-effect via peek()

If we don't care about order, using peek for printing is ok:

```
public class SideEffectWithPeek {
   public static void main (String[] args) {
       IntStream.range(0, 5)
                .unordered()
                .parallel()
                .map(x -> x * 2)
                .peek(System.out::println)
                .count();
   }
}
```

Wrong use of side-effect:

Wrong way to collect pipeline result:

```
public class SideEffectWrongUse {

    public static void main (String[] args) {
        List<Integer> results = new ArrayList<>();
        IntStream.range(0, 150)
                .parallel()
                .filter(s -> s % 2 == 0)
                .forEach(s -> results.add(s));
        System.out.println(results);
    }
}
```

Running multiple times may result in this exception:

```
   Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
         at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
         at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)
         at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)
         at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
         at java.util.concurrent.ForkJoinTask.getThrowableException(ForkJoinTask.java:598)
         at java.util.concurrent.ForkJoinTask.reportException(ForkJoinTask.java:677)
         at java.util.concurrent.ForkJoinTask.invoke(ForkJoinTask.java:735)
         at java.util.stream.ForEachOps$ForEachOp.evaluateParallel(ForEachOps.java:160)
         at java.util.stream.ForEachOps$ForEachOp$OfInt.evaluateParallel(ForEachOps.java:189)
         at java.util.stream.AbstractPipeline.evaluate(AbstractPipeline.java:233)
         at java.util.stream.IntPipeline.forEach(IntPipeline.java:404)
         at com.logicbig.example.SideEffectWrongUse.main(SideEffectWrongUse.java:13)
         at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
         at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
         at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
```

Fixing above code:

```
public class SideEffectWrongUseFix {
    public static void main (String[] args) {
        IntStream stream = IntStream.range(0, 1000);
        List<Integer> list = stream.parallel()
                                  .filter(s -> s % 2 == 0)
                                  .boxed()
                                  .collect(Collectors.toList());
        System.out.println(list);
    }
}
```

Example Project

Dependencies and Technologies Used:

- JDK 1.8
- Maven 3.0.4

Side Effect Examples

streams-side-effect
- src
  - main
    - java
      - com
        - logicbig
          - example
            - SideEffectWithPeek.java
            - SideEffectWrongUse.java
            - SideEffectWrongUse2.jav
            - SideEffectWrongUse2Fix

```
import java.util.stream.Stream;

/**
 * Even though this is thread safe, but the result is non-deterministic
 */
public class SideEffectWrongUse2 {
    public static void main (String[] args) {

        List<Integer> lengths = Collections.synchronizedList(new ArrayList<>());
        Stream.of("Banana", "Pear", "Apple")
              .peek(SideEffectWrongUse2::longTask)//applying side effect
              .parallel()
              .mapToInt(s -> s.length())
              .forEach(lengths::add);//collecting via side effect
```

SideEffectWrongUseFix.j

pom.xml

```
                                                    // updating state
        System.out.println(lengths);
    }

    private static void longTask (String s) {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Project Structure

streams-side-effect
    src
        main
            java
                com
                    logicbig
                        example
                            SideEffectWithPeek.java
                            SideEffectWrongUse.java
                            SideEffectWrongUse2.java
                            SideEffectWrongUse2Fix.java
                            SideEffectWrongUseFix.java
    pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.logicbig.example</groupId>
    <artifactId>streams-side-effect</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.5.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

```java
package com.logicbig.example;

import java.util.stream.IntStream;

public class SideEffectWithPeek {
    public static void main (String[] args) {
        IntStream.range(0, 5)
                 .unordered()
                 .parallel()
                 .map(x -> x * 2)
                 .peek(System.out::println)
                 .count();
    }
}
```

```java
package com.logicbig.example;

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

public class SideEffectWrongUse {

    public static void main (String[] args) {
        List<Integer> results = new ArrayList<>();
        IntStream.range(0, 150)
                 .parallel()
                 .filter(s -> s % 2 == 0)
                 .forEach(s -> results.add(s));//stateful side effect
        //not thread safe
        System.out.println(results);
    }
}
```

```java
package com.logicbig.example;

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

/**
 * Even though this is thread safe, but the result is non-deterministic
 */
public class SideEffectWrongUse2 {
    public static void main (String[] args) {
        List<Integer> lengths = Collections.synchronizedList(new ArrayList<>());
        Stream.of("Banana", "Pear", "Apple")
              .peek(SideEffectWrongUse2::longTask)//applying side effect
              .parallel()
              .mapToInt(s -> s.length())
              .forEach(lengths::add);//collecting via side effect
                                     // updating state
        System.out.println(lengths);
    }

    private static void longTask (String s) {
        try {
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
package com.logicbig.example;

import java.util.Arrays;
import java.util.stream.Stream;

/**
 * Even though this is thread safe, but the result is non-deterministic
 */
public class SideEffectWrongUse2Fix {
    public static void main(String[] args) {

        int[] lengths = Stream.of("Banana", "Pear", "Apple")
                              .peek(SideEffectWrongUse2Fix::longTask)//applying side effect
                              .parallel()
                              .mapToInt(s -> s.length())
                              .toArray();
        System.out.println(Arrays.toString(lengths));
    }

    private static void longTask(String s) {
        try {//some stateless task simulation. e.g. sending email
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

```java
package com.logicbig.example;

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

public class SideEffectWrongUseFix {
    public static void main (String[] args) {
        IntStream stream = IntStream.range(0, 1000);
        List<Integer> list = stream.parallel()
                                   .filter(s -> s % 2 == 0)
                                   .boxed()
                                   .collect(Collectors.toList());
        System.out.println(list);
    }
}
```