Random, SecureRandom, ThreadLocalRandom and SplittableRandom - Different ways to create Random numbers in Java

[Last Updated: May 29, 2018]

Java Random | Java

In Java we can create random numbers using these classes: `Random`, `SecureRandom` `ThreadLocalRandom`, `SplittableRandom`. Let's see quick examples on each one of them then we will talk about their differences.

SplittableRandom

```
package com.logicbig.example;

import java.util.SplittableRandom;

public class SplittableRandomExample {
  public static void main(String[] args) {
      //creating a random int
      System.out.println("-- single random int --");
      int i = new SplittableRandom().nextInt(10, 100);
      System.out.println(i);
      //creating stream of ints
      System.out.println("-- stream  --");
      new SplittableRandom()
              .ints(5, 10, 100)
              .forEach(System.out::println);
  }
}
```

```
-- single random int --
49
-- stream  --
73
35
50
44
88
```

SecureRandom

```
package com.logicbig.example;

import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.concurrent.ThreadLocalRandom;

public class SecureRandomExample {
```

```
  public static void main(String[] args) throws NoSuchAlgorithmException {
      //creating a random int
      System.out.println("-- single random int --");
      int i = new SecureRandom().nextInt(100);
      System.out.println(i);
      //creating stream of ints
      System.out.println("-- stream --");
      new SecureRandom()
              .ints(5, 10, 100)
              .forEach(System.out::println);
  }
}
```

```
-- single random int --
13
-- stream --
80
73
92
30
92
```

ThreadLocalRandom

```
package com.logicbig.example;

import java.util.concurrent.ThreadLocalRandom;

public class ThreadLocalRandomExample {
  public static void main(String[] args) {
      //creating a random int
      System.out.println("-- single random int --");
      int i = ThreadLocalRandom.current().nextInt(10, 100);
      System.out.println(i);
      //creating stream of ints
      System.out.println("-- stream --");
      ThreadLocalRandom.current()
                       .ints(5, 10, 100)
                       .forEach(System.out::println);
  }
}
```

```
-- single random int --
40
-- stream --
23
41
95
44
33
```

Random

```
package com.logicbig.example;

import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;
```

```java
public class RandomExample {
  public static void main(String[] args) {
      //creating a random int
      System.out.println("-- single random int --");
      int i = new Random().nextInt(100);
      System.out.println(i);
      //creating stream of ints
      System.out.println("-- stream --");
      new Random().ints(5, 10, 100)
                  .forEach(System.out::println);
  }
}
```

```
-- single random int --
18
-- stream --
13
60
41
68
32
```


Differences

java.util.Random

It's thread safe. However, the concurrent use of the same java.util.Random instance across threads may encounter contention and consequent poor performance.
Since Java 1.0

java.security.SecureRandom

This class provides a cryptographically strong random number generator. We should use it in security-sensitive applications.
Since Java 1.1

java.util.concurrent.ThreadLocalRandom

A random number generator isolated to the current thread. A single instance is initialized for the current thread with an internally generated seed, which can be accessed by a single thread over and over again by method `ThreadLocalRandom.current()`. As compared to `java.util.Random`, this generator encounters much less overhead and contention.
Since Java 1.7

java.util.SplittableRandom

It is very high performance random number generator. A single Instance of SplittableRandom is not thread-safe. They are designed to be split, not shared, across threads. For example, a fork/join-style computation using random numbers might include a construction of the form new Subtask(aSplittableRandom.split()).fork().
They are good for parallel computation of Java 8 streams. They consistently produce same result every time when split tasks are joined, hence they are deterministic, ThreadLocalRandom does not have that characteristic.
Since Java 1.8

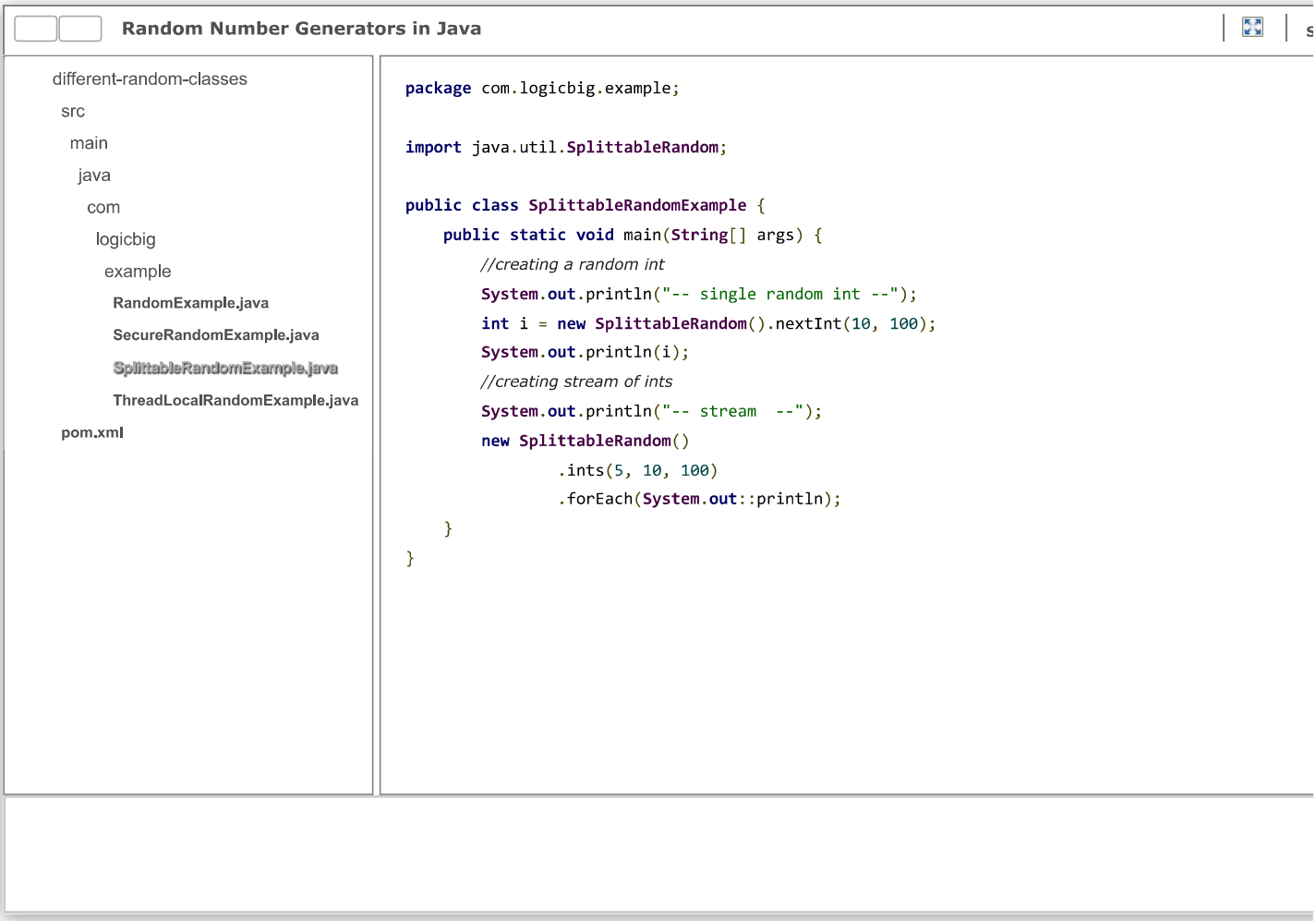
Each of above classes has various methods (more or less) for producing individual int, float, double, long, boolean, bytes[] etc.

Each of the above classes has various methods to produce IntStream, DoubleStream and LongStream.

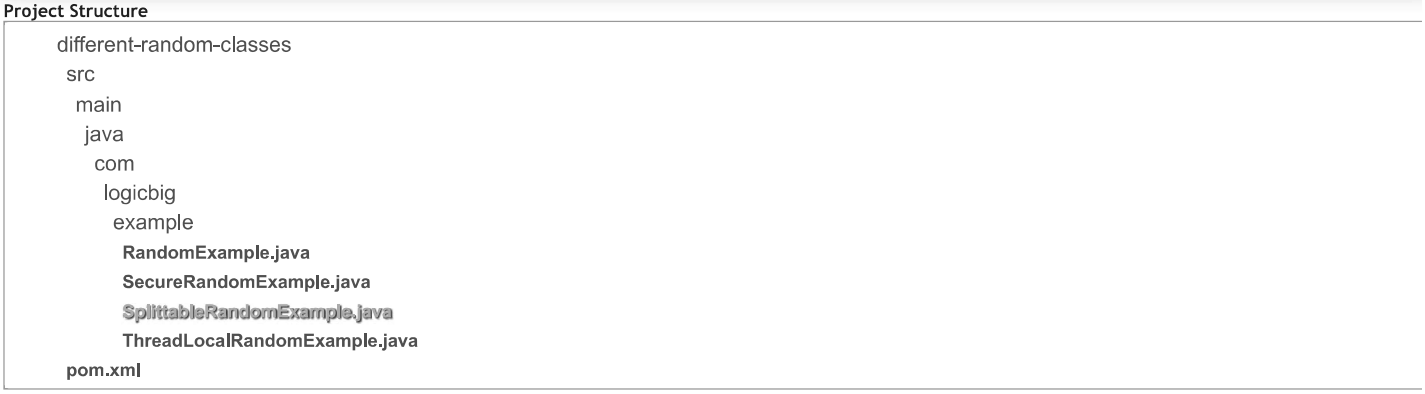
Example Project

Dependencies and Technologies Used:

- JDK 10
- Maven 3.3.9



Project Structure




```
<?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>different-random-classes</artifactId>
    <version>1.0-SNAPSHOT</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.7.0</version>
                <configuration>
                    <source>10</source>
                    <target>10</target>
                    <encoding>UTF-8</encoding>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

UP

```
package com.logicbig.example;

import java.util.Random;
import java.util.concurrent.ThreadLocalRandom;

public class RandomExample {
    public static void main(String[] args) {
        //creating a random int
        System.out.println("-- single random int --");
        int i = new Random().nextInt(100);
        System.out.println(i);
        //creating stream of ints
        System.out.println("-- stream --");
        new Random().ints(5, 10, 100)
                    .forEach(System.out::println);
    }
}
```

UP

```
package com.logicbig.example;

import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.util.concurrent.ThreadLocalRandom;

public class SecureRandomExample {
    public static void main(String[] args) throws NoSuchAlgorithmException {
        //creating a random int
        System.out.println("-- single random int --");
        int i = new SecureRandom().nextInt(100);
        System.out.println(i);
        //creating stream of ints
```

```
        System.out.println("-- stream --");
        new SecureRandom()
                .ints(5, 10, 100)
                .forEach(System.out::println);
    }
}
```

UP

```
package com.logicbig.example;

import java.util.SplittableRandom;

public class SplittableRandomExample {
    public static void main(String[] args) {
        //creating a random int
        System.out.println("-- single random int --");
        int i = new SplittableRandom().nextInt(10, 100);
        System.out.println(i);
        //creating stream of ints
        System.out.println("-- stream  --");
        new SplittableRandom()
                .ints(5, 10, 100)
                .forEach(System.out::println);
    }
}
```

UP

```
package com.logicbig.example;

import java.util.concurrent.ThreadLocalRandom;

public class ThreadLocalRandomExample {
    public static void main(String[] args) {
        //creating a random int
        System.out.println("-- single random int --");
        int i = ThreadLocalRandom.current().nextInt(10, 100);
        System.out.println(i);
        //creating stream of ints
        System.out.println("-- stream --");
        ThreadLocalRandom.current()
                         .ints(5, 10, 100)
                         .forEach(System.out::println);
    }
}
```

UP