Multithreading

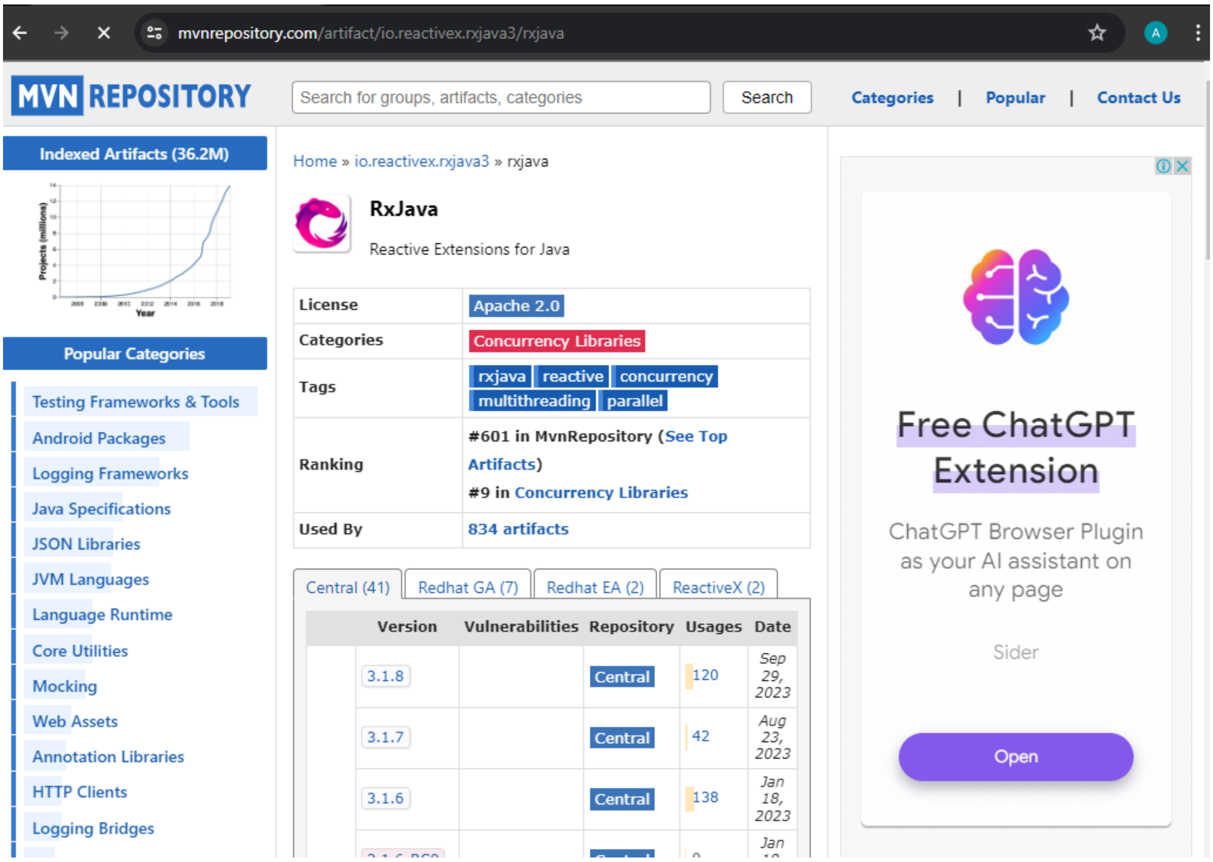
Q2. Write a Java program that uses reactive programming to read a file asynchronously. Use RxJava or another reactive library to handle the file reading and processing.

Steps:

1. Add RxJava dependency to your project.
2. Create a method to read a file asynchronously using RxJava.
3. Process the file contents using reactive streams.

Here is a complete example of how to achieve this:

Maven Dependency



mvnrepository.com/artifact/io.reactivex.rxjava3/rxjava

Home » io.reactivex.rxjava3 » rxjava

RxJava

Reactive Extensions for Java

License	Apache 2.0
Categories	Concurrency Libraries
Tags	rxjava reactive concurrency multithreading parallel
Ranking	#601 in MvnRepository (See Top Artifacts) #9 in Concurrency Libraries
Used By	834 artifacts

Version	Vulnerabilities	Repository	Usages	Date
3.1.8		Central	120	Sep 29, 2023
3.1.7		Central	42	Aug 23, 2023
3.1.6		Central	138	Jan 18, 2023

First, ensure that you have the RxJava dependency in your `pom.xml` if you're using Maven:

```
<!-- https://mvnrepository.com/artifact/io.reactivex.rxjava
3/rxjava -->
<dependency>
    <groupId>io.reactivex.rxjava3</groupId>
    <artifactId>rxjava</artifactId>
    <version>3.1.8</version>
</dependency>
```

```
javaCopy code
import io.reactivex.rxjava3.core.Observable;
import io.reactivex.rxjava3.schedulers.Schedulers;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReactiveFileReader {

    public static void main(String[] args) {
        String filePath = "example.txt"; // Replace with yo
ur file path

        readFileAsync(filePath)
                .subscribeOn(Schedulers.io())
                .observeOn(Schedulers.single())
                .subscribe(
                        line -> System.out.println("Read li
ne: " + line),
                        throwable -> System.err.println("Er
ror: " + throwable),
                        () -> System.out.println("Reading f
ile completed.")
                );

        // Sleep main thread to allow asynchronous processi
ng to complete
```

```
        try {
            Thread.sleep(3000); // Adjust time as necessary
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    public static Observable<String> readFileAsync(String filePath) {
        return Observable.create(emitter -> {
            try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {
                String line;
                while ((line = reader.readLine()) != null) {
                    if (emitter.isDisposed()) {
                        return;
                    }
                    emitter.onNext(line);
                }
                emitter.onComplete();
            } catch (IOException e) {
                if (!emitter.isDisposed()) {
                    emitter.onError(e);
                }
            }
        });
    }
}
```

Q4. Implement a program where two threads communicate with each other using wait() and notify() methods. One thread should print even numbers, and the other should print odd numbers in sequence.

- `wait()`:

    - This method causes the current thread to release the lock it holds on the object and enter a waiting state.
    - The thread remains in the waiting state until another thread calls `notify()` or `notifyAll()` on the same object.
    - `wait()` must be called within a synchronized block or method to ensure that the thread holds the lock on the object.
- **`notify()`**:
    - This method wakes up one of the threads that is waiting on the object's monitor.
    - If multiple threads are waiting, one of them is chosen (the choice is arbitrary and depends on the JVM implementation).
    - The awakened thread must re-acquire the lock on the object before it can proceed.
    - Like `wait()`, `notify()` must be called within a synchronized block or method

Q5. Implement a program that demonstrates the use of locks (e.g., ReentrantLock) for thread synchronization. Create a scenario where multiple threads access a shared resource, and use locks to ensure that only one thread can access the resource at a time.

```
import java.util.concurrent.locks.ReentrantLock;
```

- **`lock()`**: Acquires the lock. If the lock is not available, the current thread becomes disabled for thread scheduling purposes and lies dormant until the lock has been acquired.
- **`unlock()`**: Releases the lock. If other threads are waiting for the lock, one of them will acquire the lock.

Q8. Develop a Java program that analyzes real-time weather data using reactive programming. The program should fetch weather data from a weather API asynchronously and perform

analysis (e.g., temperature trends, rainfall predictions). Use a reactive approach to handle the asynchronous nature of weather data updates. Use reactive operators (e.g., map, filter) to process and analyze the weather data stream.

Steps:

1. Set up the project with the necessary dependencies.
2. Create a method to fetch weather data asynchronously.
3. Use reactive streams to process and analyze the data.

Dependencies:

First, make sure to add the Reactor library and a library for making HTTP requests (like `WebClient` from Spring WebFlux) to your `pom.xml` or `build.gradle` file.

For Maven:

```
xmlCopy code
<dependencies>
    <dependency>
        <groupId>io.projectreactor</groupId>
        <artifactId>reactor-core</artifactId>
        <version>3.4.12</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webflux</artifactId>
        <version>5.3.12</version>
    </dependency>
</dependencies>
```

For Gradle:

```
groovyCopy code
dependencies {
    implementation 'io.projectreactor:reactor-core:3.4.12'
    implementation 'org.springframework:spring-webflux:5.3.
12'
}
```

```
import org.springframework.web.reactive.function.client.Web
Client;
import reactor.core.publisher.Flux;
import reactor.core.publisher.Mono;

import java.time.Duration;
import java.util.List;

public class ReactiveWeatherAnalyzer {

    String WEATHER_API_URL = "https://api.weatherapi.com/v
1/current.json?key=YOUR_API_KEY&q=YOUR_LOCATION";
    WebClient webClient;

    public ReactiveWeatherAnalyzer() {
        this.webClient = WebClient.create();
    }

    // Method to fetch weather data asynchronously
    public Mono<WeatherData> fetchWeatherData() {
        return webClient.get()
                .uri(WEATHER_API_URL)
                .retrieve()
                .bodyToMono(WeatherData.class);
    }

    // Method to simulate real-time weather data fetching
    public Flux<WeatherData> getRealTimeWeatherData() {
        return Flux.interval(Duration.ofSeconds(10))
                .flatMap(tick -> fetchWeatherData());
    }

    // Method to analyze temperature trends
    public void analyzeTemperatureTrends() {
        getRealTimeWeatherData()
```

```
                .map(WeatherData::getTemperature)
                .buffer(6) // Collect temperature data for
every minute (assuming fetch every 10 seconds)
                .map(this::calculateTemperatureTrend)
                .subscribe(trend -> System.out.println("Tem
perature trend for the past minute: " + trend));
    }

    // Method to calculate temperature trend
    private String calculateTemperatureTrend(List<Double> t
emperatures) {
        double average = temperatures.stream().mapToDouble
(Double::doubleValue).average().orElse(0.0);
        return average > 25 ? "High" : average < 15 ? "Low"
: "Moderate";
    }

    public static void main(String[] args) {
        ReactiveWeatherAnalyzer analyzer = new ReactiveWeat
herAnalyzer();
        analyzer.analyzeTemperatureTrends();

        // Keep the application running to observe the real
-time data
        try {
            Thread.sleep(600000); // Run for 10 minutes
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}

// Sample WeatherData class to map the API response
class WeatherData {
    private Current current;

    public double getTemperature() {
        return current.temp_c;
    }

    public void setCurrent(Current current) {
        this.current = current;
    }

    static class Current {
        private double temp_c;

        public double getTemp_c() {
            return temp_c;
        }

        public void setTemp_c(double temp_c) {
            this.temp_c = temp_c;
        }
    }
}
```