Esercizio 1 – Confronto tra stream()e parallelStream()

```
class Filters {
    public static Earthquake getNearest(final Stream<Earthquake> quakes, final Coordinate loc) {
        return quakes
                .min(Comparator.comparing(q -> q.getPosition().distance(loc)))
                .get();
    }

    public static Earthquake getStrongest(final Stream<Earthquake> quakes) {
        return quakes
                .max(Comparator.comparing(q -> q.getMagnitude()))
                .get();
    }

    public static Earthquake getFurthest(final Stream<Earthquake> quakes, final Coordinate loc){
        return quakes
                .max(Comparator.comparing(q -> q.getPosition().distance(loc)))
                .get();
    }

    public static Long getQuakesCountAtLat(final Stream<Earthquake> quakes, final double lat){
        return quakes
                .filter(q -> (q.getPosition().getLat() >= lat &&
                              q.getPosition().getLat() < (lat + 1.0)))
                .count();
    }

    public static Long getQuakesCountAtLon(final Stream<Earthquake> quakes, final double lon){
        return quakes
                .filter(q -> (q.getPosition().getLon() >= lon &&
                              q.getPosition().getLon() < (lon + 1.0)))
                .count();
    }

    public static Map<Integer, Integer> getQuakesCountByDepth(final Stream<Earthquake> quakes){
        return quakes
                .collect(Collectors.groupingBy(q -> (int) q.getDepth() / 100))
                .entrySet().stream()
                .collect(Collectors.toMap(e -> e.getKey(), e -> e.getValue().size()));
    }

    public static Map<Integer, Integer> getQuakesCountByMag(final Stream<Earthquake> quakes) {
        return quakes
                .collect(Collectors.groupingBy(q -> (int) q.getMagnitude()))
                .entrySet().stream()
                .collect(Collectors.toMap(e -> e.getKey(), e -> e.getValue().size()));
    }

    public static List<Earthquake> getSpecialQuakes(final Stream<Earthquake> quakes, final Coordinate loc) {
        return quakes
                .filter(q -> q.getPosition().distance(loc) >= 2000 &&
                        q.getMagnitude() >= 4.0 &&
                        q.getMagnitude() <= 6.0)
                .sorted(Comparator.comparing(o -> o.getPosition().distance(loc)))
                .limit(10)
                .collect(Collectors.toList());
    }
}

public class S12Esercizio1 {

   [...]

    public static void main(final String[] args) {
        final String localFile = "Esercizi/serie11/2014-2015.csv";
        final Coordinate supsi = new Coordinate(46.0224644, 8.9181083);

        testSerial(localFile, supsi);
        testParallel(localFile, supsi);

    }

    static private void testSerial(final String localFile, final Coordinate supsi) {
        long startTime = System.currentTimeMillis();
        final List<Earthquake> quakes = loadEarthquakeDB(localFile, true);
        final long computeTime = System.currentTimeMillis();
        final Earthquake nearest = Filters.getNearest(quakes.stream(), supsi);
        final Earthquake strongest = Filters.getStrongest(quakes.stream());
        final Earthquake furthest = Filters.getFurthest(quakes.stream(), supsi);
        final List<Earthquake> n10_mag4_6_2000km = Filters.getSpecialQuakes(quakes.stream(), supsi);
        final Long lat46 =Filters. getQuakesCountAtLat(quakes.stream(), 46.0);
        final Long lon8 = Filters.getQuakesCountAtLon(quakes.stream(), 8.0);
final Map<Integer, Integer> quakeCountByDepthRange = Filters.getQuakesCountByDepth(quakes.stream());
final Map<Integer, Integer> quakeCountByStrength = Filters.getQuakesCountByMag(quakes.stream());
```

```
        final long endTime = System.currentTimeMillis();
        System.out.println("SimpleStream Completed in " + ((endTime - startTime)) + " ms"
                           + " (computation time=" + (endTime - computeTime) + " ms)");
        System.out.println("più forte: " + strongest);
        System.out.println("più vicino: " + nearest);
        System.out.println("più lontano: " + furthest);
        System.out.println("i 10 con mag. 4 e 6 più vicini, ma ad una distanza >= 2000 km:");
        for (final Earthquake quake : n10_mag4_6_2000km)
            System.out.println(" - (" + quake.getPosition().distance(supsi) + ") " + quake);
        System.out.println("nr. con lat 46 (46.0 <= lat < 47.0): " + lat46);
        System.out.println("nr. con longitudine 8 (8.0 <= longitudine < 9.0): " + lon8);
        System.out.println("nr. per fasce di profondità di 100 km:");
        quakeCountByDepthRange
            .entrySet().stream().sorted(Comparator.comparing(Map.Entry::getKey))
            .forEach(e -> System.out.println(
                       " - [" + e.getKey() + ", " + (e.getKey() + 1) + ") " + e.getValue()));
        System.out.println("nr. per fasce di intensità: ");
        quakeCountByStrength
            .entrySet().stream().sorted(Comparator.comparing(Map.Entry::getKey))
            .forEach(e -> System.out.println(
                       " - [" + e.getKey() + ", " + (e.getKey() + 1) + ") " + e.getValue()));
    }

    static private void testParallel(final String localFile, final Coordinate supsi) {
        long startTime = System.currentTimeMillis();
        final List<Earthquake> quakes = loadEarthquakeDB(localFile, true);
        final long computeTime = System.currentTimeMillis();
        final Earthquake nearest = Filters.getNearest(quakes.parallelStream(), supsi);
        final Earthquake strongest = Filters.getStrongest(quakes.parallelStream());
        final Earthquake furthest = Filters.getFurthest(quakes.parallelStream(), supsi);
final List<Earthquake> n10_mag4_6_2000km = Filters.getSpecialQuakes(quakes.parallelStream(), supsi);
        final Long lat46 = Filters.getQuakesCountAtLat(quakes.parallelStream(), 46.0);
        final Long lon8 = Filters.getQuakesCountAtLon(quakes.parallelStream(), 8.0);
        final Map<Integer, Integer> quakeCountByDepthRange = Filters.getQuakesCountByDepth(
                quakes.parallelStream());
        final Map<Integer, Integer> quakeCountByStrength = Filters. getQuakesCountByMag(
                quakes.parallelStream());
        final long endTime = System.currentTimeMillis();

        System.out.println("ParallelStream Completed in " + ((endTime - startTime)) + " ms"
                           + " (computation time=" + (endTime - computeTime) + " ms)");
        System.out.println("più forte: " + strongest);
        System.out.println("più vicino: " + nearest);
        System.out.println("più lontano: " + furthest);
        System.out.println("i 10 con mag 4 e 6 più vicini, ma ad una distanza >= 2000 km:");
        for (final Earthquake quake : n10_mag4_6_2000km)
            System.out.println(" - (" + quake.getPosition().distance(supsi) + ") " + quake);
        System.out.println("nr. con latitudine 46 (46.0 <= latitudine < 47.0): " + lat46);
        System.out.println("nr. con longitudine 8 (8.0 <= longitudine < 9.0): " + lon8);
        System.out.println("nr. per fasce di profondità di 100 km:");
        quakeCountByDepthRange
                .entrySet().stream().sorted(Comparator.comparing(Map.Entry::getKey))
                .forEach(e -> System.out.println(
                           " - [" + e.getKey() + ", " + (e.getKey() + 1) + ") " + e.getValue()));
        System.out.println("nr. per fasce di intensità: ");
        quakeCountByStrength.entrySet().stream().sorted(Comparator.comparing(Map.Entry::getKey))
                .forEach(e -> System.out.println(
                           " - [" + e.getKey() + ", " + (e.getKey() + 1) + ") " + e.getValue()));
}
```

Output:

SimpleStream Completed in 3600 ms (computation time=482 ms)
ParallelStream Completed in 2891 ms (computation time=160 ms)

Confrontando i tempi d'esecuzione totali non si riesce a notare una grande differenza. Il motivo è dovuto al tempo di lettura e parse del file che è molto più lungo rispetto al tempo delle operazioni di ricerca. Se però si confrontano i tempi di calcolo puro (computation time) si può osservare un miglioramento significativo con un fattore di speedup intorno al 3.

Esercizio 2 – Soluzione con le CompletableFutures

```
public class S12Esercizio2 {
    [...]
    public static void main(final String[] args) {
        final String localFile = "Esercizi/serie12/2014-2015.csv";
        final Coordinate supsi = new Coordinate(46.0224644, 8.9181083);
        long startTime = System.currentTimeMillis();
        final CompletableFuture<List<Earthquake>> cf_earthquakeDBLoader = CompletableFuture
                .supplyAsync(() -> loadEarthquakeDB(localFile, true));
        final CompletableFuture<Earthquake> cf_nearest = cf_earthquakeDBLoader
                .thenApplyAsync(quakes -> Filters.getNearest(quakes.stream(), supsi));
        final CompletableFuture<Earthquake> cf_strongest = cf_earthquakeDBLoader
                .thenApplyAsync(quakes -> Filters.getStrongest(quakes.stream()));
        final CompletableFuture<Earthquake> cf_furthest = cf_earthquakeDBLoader
                .thenApplyAsync(quakes -> Filters.getFurthest(quakes.stream(), supsi));
        final CompletableFuture<List<Earthquake>> cf_n10_mag4_6_2000km = cf_earthquakeDBLoader
                .thenApplyAsync(quakes -> Filters.getSpecialQuakes(quakes.stream(), supsi));
        final CompletableFuture<Long> cf_lat46 = cf_earthquakeDBLoader
                .thenApplyAsync(quakes -> Filters.getQuakesCountAtLat(quakes.stream(), 46.0));
        final CompletableFuture<Long> cf_lon8 = cf_earthquakeDBLoader
                .thenApplyAsync(quakes -> Filters.getQuakesCountAtLon(quakes.stream(), 8.0));
        final CompletableFuture<Map<Integer, Integer>> cf_quakeCountByDepthRange =
cf_earthquakeDBLoader.thenApplyAsync(quakes -> Filters.getQuakeCountsByDepth(quakes.stream()));
        final CompletableFuture<Map<Integer, Integer>> cf_quakeCountByStrength =
cf_earthquakeDBLoader.thenApplyAsync(quakes -> Filters.getQuakesCountByMag(quakes.stream()));
        System.out.println("Waiting for all results...");

        try {
            cf_earthquakeDBLoader.get();
            final long computeTime = System.currentTimeMillis();
            Earthquake strongest = cf_strongest.get();
            Earthquake nearest = cf_nearest.get();
            Earthquake furthest = cf_furthest.get();
            List<Earthquake> n10_mag4_6_2000km = cf_n10_mag4_6_2000km.get();
            Long lat46 = cf_lat46.get();
            Long lon8 = cf_lon8.get();
            Map<Integer, Integer> quakeCountByDepthRange = cf_quakeCountByDepthRange.get();
            Map<Integer, Integer> quakeCountByStrength = cf_quakeCountByStrength.get();

            final long endTime = System.currentTimeMillis();
            System.out.println("------------------------------------------------------------------------");
            System.out.println("CompletableFuture Completed in " +((endTime - startTime)) + " ms"
                                + " (computation time=" +(endTime - computeTime) + " ms)");
            System.out.println("------------------------------------------------------------------------");
            System.out.println("più forte: " + strongest);
            System.out.println("più vicino: " + nearest);
            System.out.println("più lontano: " + furthest);
            System.out.println("i 10 con mag tra 4 e 6 più vicini, ma ad una distanza >= 2000 km:");
            for (final Earthquake quake : n10_mag4_6_2000km)
                System.out.println(" - (" + quake.getPosition().distance(supsi) + ") " + quake);
            System.out.println("nr. con latitudine 46 (46.0 <= latitudine < 47.0): " + lat46);
            System.out.println("nr. con longitudine 8 (8.0 <= longitudine < 9.0): " + lon8);
            System.out.println("nr. per fasce di profondità di 100 km:");
            System.out.println("nr. per fasce di profondità di 100 km:");
            quakeCountByDepthRange
                .entrySet().stream().sorted(Comparator.comparing(Map.Entry::getKey))
                .forEach(e -> System.out.println(
                        " - [" + e.getKey() + ", " + (e.getKey() + 1) + ") " + e.getValue()));
            System.out.println("nr. per fasce di intensità: ");
            quakeCountByStrength.entrySet().stream().sorted(Comparator.comparing(Map.Entry::getKey))
                .forEach(e -> System.out.println(
                        " - [" + e.getKey() + ", " + (e.getKey() + 1) + ") " + e.getValue()));
        } catch (final InterruptedException | ExecutionException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

CompletableFuture Completed in 3399 ms (computation time=270 ms)

I tempi d'esecuzione per quest'ultima versione si collocano tra la versione con le stream semplici e quella con le parallelStream. La differenza delle due versioni parallele (la prima con le parallelStream e la seconda con le CompletableFuture) sta nel tipo di parallelismo ottenuto. Nel caso delle stream parallele, tutte le ricerche avvengono in modo sequenziale ed è l'operazione stessa ad essere eseguita in parallelo mentre, nel caso delle completable futures, sono le singole operazioni ad essere eseguite in parallelo.