```java
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.stream.Collectors;

public class ParallelUniverseExplorer {

    private static final ForkJoinPool pool = new ForkJoinPool();

    private static final List<String> sequentialTimeline = List.of(
        "Big Bang", "Formation of First Stars", "First Galaxies Form",
        "Solar System Formation", "Early Earth", "First Life Forms",
        "Cambrian Explosion", "Age of Dinosaurs", "Mass Extinction Event",
        "Age of Mammals", "Early Hominids", "Stone Age", "Bronze Age",
        "Iron Age", "Classical Antiquity", "Middle Ages", "Renaissance",
        "Industrial Revolution", "Digital Age", "Present Day"
    );

    private static final List<List<String>> parallelTimelines = List.of(
        List.of("Timeline Alpha: Big Bang", "Timeline Alpha: Alternate Evolution",
                "Timeline Alpha: Advanced Civilization", "Timeline Alpha: Cosmic Ascension"),
        List.of("Timeline Beta: Quantum Fluctuation", "Timeline Beta: Crystal Worlds",
                "Timeline Beta: Mechanical Life", "Timeline Beta: Singularity"),
        List.of("Timeline Gamma: Dark Matter Dominance", "Timeline Gamma: Energy Beings",
                "Timeline Gamma: Time Manipulation", "Timeline Gamma: Multiverse Travel"),
        List.of("Timeline Delta: Organic Technology", "Timeline Delta: Symbiotic Worlds",
                "Timeline Delta: Universal Harmony", "Timeline Delta: Transcendence"),
        List.of("Timeline Epsilon: Artificial Genesis", "Timeline Epsilon: Digital Reality",
                "Timeline Epsilon: Virtual Dimensions", "Timeline Epsilon: Code Universe")
    );

    public static void main(String[] args) {
        System.out.println("=== Parallel Universe Explorer ===");

        // Simulate sequential exploration
        exploreSequentialTimeline();

        // Simulate parallel exploration
        exploreParallelTimelines();

        // Compare performance
        compareSequentialVsParallel();

        // Advanced parallel operations
        advancedParallelOperations();
    }

    /**
     * Sequential exploration of the main timeline
     */
    public static void exploreSequentialTimeline() {
        System.out.println("\n--- Sequential Timeline Exploration ---");
        System.out.println("Starting sequential exploration...");

        long startTime = System.currentTimeMillis();

        sequentialTimeline.stream()
            .map(event -> {
                System.out.println(Thread.currentThread().getName() + " - Exploring: " + event);
                simulateExplorationDelay(200);
                return "Explored: " + event;
            })
            .collect(Collectors.toList());

        long endTime = System.currentTimeMillis();
        System.out.println("Sequential exploration completed in: " + (endTime - startTime) +
"ms");
    }

    /**
     * Parallel exploration of multiple timelines
     */
    public static void exploreParallelTimelines() {
        System.out.println("\n--- Parallel Timelines Exploration ---");
        System.out.println("Starting parallel exploration of " + parallelTimelines.size() + "
timelines...");

        long startTime = System.currentTimeMillis();

        parallelTimelines.parallelStream()
            .map(timeline -> {
                String timelineName = timeline.get(0).split(":")[0];
                System.out.println(Thread.currentThread().getName() + " - Starting exploration
of " + timelineName);

                timeline.forEach(event -> {
                    System.out.println(Thread.currentThread().getName() + " - Processing: " +
event);
                    simulateExplorationDelay(150);
                });

                return "Completed: " + timelineName;
            })
            .collect(Collectors.toList())
            .forEach(result -> System.out.println("Timeline result: " + result));

        long endTime = System.currentTimeMillis();
        System.out.println("Parallel exploration completed in: " + (endTime - startTime) +
"ms");
    }

    /**
     * Compare performance between sequential and parallel processing
     */
    public static void compareSequentialVsParallel() {
        System.out.println("\n--- Performance Comparison ---");

        // Sequential processing
        long seqStart = System.currentTimeMillis();
        List<String> sequentialResults = parallelTimelines.stream()
            .flatMap(List::stream)
            .map(event -> {
                simulateExplorationDelay(100);
                return "SEQ: " + event;
            })
            .collect(Collectors.toList());
        long seqEnd = System.currentTimeMillis();

        // Parallel processing
        long parStart = System.currentTimeMillis();
        List<String> parallelResults = parallelTimelines.parallelStream()
            .flatMap(List::stream)
            .map(event -> {
                simulateExplorationDelay(100);
                return "PAR: " + event;
            })
            .collect(Collectors.toList());
        long parEnd = System.currentTimeMillis();

        System.out.println("Sequential processing time: " + (seqEnd - seqStart) + "ms");
        System.out.println("Parallel processing time: " + (parEnd - parStart) + "ms");
        System.out.println("Performance improvement: " +
            ((seqEnd - seqStart) - (parEnd - parStart)) + "ms");
    }

    /**
     * Advanced parallel operations with custom thread pool
     */
    public static void advancedParallelOperations() {
        System.out.println("\n--- Advanced Parallel Operations ---");

        // Using custom ForkJoinPool for parallel processing
        System.out.println("Using custom ForkJoinPool with " + pool.getParallelism() + "
threads");

        try {
            pool.submit(() -> {
                parallelTimelines.parallelStream()
                    .forEach(timeline -> {
                        String timelineName = timeline.get(0).split(":")[0];
                        System.out.println(Thread.currentThread().getName() +
                                         " - Deep scanning: " + timelineName);

                        // Nested parallel processing within each timeline
                        timeline.parallelStream()
                            .forEach(event -> {
                                System.out.println(Thread.currentThread().getName() +
                                                 " - Analyzing: " + event);
                                simulateExplorationDelay(50);
                            });
                    });
            }).get(); // Wait for completion
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Parallel processing with filtering and mapping
        System.out.println("\n--- Filtered Parallel Processing ---");
        List<String> significantEvents = parallelTimelines.parallelStream()
            .flatMap(List::stream)
            .filter(event -> event.contains("Advanced") ||
                            event.contains("Singularity") ||
                            event.contains("Transcendence"))
            .map(String::toUpperCase)
            .collect(Collectors.toList());

        System.out.println("Significant events found: " + significantEvents);
    }

    /**
     * Process individual events in parallel across all timelines
     */
    public static void processAllEventsInParallel() {
        System.out.println("\n--- Processing All Events in Parallel ---");

        long eventCount = parallelTimelines.parallelStream()
```

```
            .flatMap(List::stream)
            .map(event -> {
                System.out.println(Thread.currentThread().getName() +
                                 " - Processing event: " + event);
                simulateExplorationDelay(80);
                return event + " [PROCESSED]";
            })
            .count();

        System.out.println("Total events processed in parallel: " + eventCount);
    }

    /**
     * Simulates exploration delay
     */
    private static void simulateExplorationDelay(int milliseconds) {
        try {
            Thread.sleep(milliseconds);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```