

import java.util.\*;

import java.util.concurrent.\*;

public class ConcurrentCollectionsDemo {

    // Shared collections

    private static ConcurrentHashMap<Integer, String> concurrentMap = new ConcurrentHashMap<>();

    private static HashMap<Integer, String> hashMap = new HashMap<>();

    private static ConcurrentLinkedQueue<Integer> concurrentQueue = new ConcurrentLinkedQueue<>();

    private static LinkedList<Integer> linkedList = new LinkedList<>();

    private static CopyOnWriteArrayList<String> concurrentList = new CopyOnWriteArrayList<>();

    private static ArrayList<String> arrayList = new ArrayList<>();

    private static final *int* NUM\_THREADS = 5;

    private static final *int* OPERATIONS\_PER\_THREAD = 10000;

    public static *void* main(String[] *args*) throws InterruptedException {

        System.out.println("Starting concurrent collections demo...\n");

        // Test ConcurrentHashMap vs HashMap with multiple threads

        System.out.println("Testing Map performance:");

        testMapPerformance();

        // Test ConcurrentLinkedQueue vs LinkedList

        System.out.println("\nTesting Queue performance:");

        testQueuePerformance();

        // Test CopyOnWriteArrayList vs ArrayList

        System.out.println("\nTesting List performance:");

        testListPerformance();

    }

    private static *void* testMapPerformance() throws InterruptedException {

        // Warmup HashMap (not thread-safe)

        hashMap.clear();

*long* startHashMap = System.currentTimeMillis();

        Thread[] threads = new Thread[NUM\_THREADS];

        for (*int* i = 0; i < NUM\_THREADS; i++) {

*int* threadId = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS\_PER\_THREAD; j++) {

                    // Not synchronized - may cause errors or exceptions

                    hashMap.put(threadId \* OPERATIONS\_PER\_THREAD + j, "Val" + j);

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* endHashMap = System.currentTimeMillis();

        System.out.println("HashMap (non-concurrent) time: " + (endHashMap - startHashMap) + " ms");

        System.out.println("HashMap size: " + hashMap.size());

        // ConcurrentHashMap

        concurrentMap.clear();

*long* startConcurrentMap = System.currentTimeMillis();

        for (*int* i = 0; i < NUM\_THREADS; i++) {

*int* threadId = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS\_PER\_THREAD; j++) {

                    concurrentMap.put(threadId \* OPERATIONS\_PER\_THREAD + j, "Val" + j);

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* endConcurrentMap = System.currentTimeMillis();

        System.out.println("ConcurrentHashMap time: " + (endConcurrentMap - startConcurrentMap) + " ms");

        System.out.println("ConcurrentHashMap size: " + concurrentMap.size());

    }

    private static *void* testQueuePerformance() throws InterruptedException {

        // LinkedList (not thread-safe)

        linkedList.clear();

        Thread[] threads = new Thread[NUM\_THREADS];

*long* startLinkedList = System.currentTimeMillis();

        for (*int* i = 0; i < NUM\_THREADS; i++) {

*int* threadId = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS\_PER\_THREAD; j++) {

                    synchronized (linkedList) {

                        linkedList.add(threadId \* OPERATIONS\_PER\_THREAD + j);

                    }

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* endLinkedList = System.currentTimeMillis();

        System.out.println("LinkedList with synchronization time: " + (endLinkedList - startLinkedList) + " ms");

        System.out.println("LinkedList size: " + linkedList.size());

        // ConcurrentLinkedQueue (thread-safe)

        concurrentQueue.clear();

*long* startConcurrentQueue = System.currentTimeMillis();

        for (*int* i = 0; i < NUM\_THREADS; i++) {

*int* threadId = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS\_PER\_THREAD; j++) {

                    concurrentQueue.add(threadId \* OPERATIONS\_PER\_THREAD + j);

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* endConcurrentQueue = System.currentTimeMillis();

        System.out.println("ConcurrentLinkedQueue time: " + (endConcurrentQueue - startConcurrentQueue) + " ms");

        System.out.println("ConcurrentLinkedQueue size: " + concurrentQueue.size());

    }

    private static *void* testListPerformance() throws InterruptedException {

        // ArrayList (not thread-safe)

        arrayList.clear();

        Thread[] threads = new Thread[NUM\_THREADS];

*long* startArrayList = System.currentTimeMillis();

        for (*int* i = 0; i < NUM\_THREADS; i++) {

*int* threadId = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS\_PER\_THREAD; j++) {

                    synchronized (arrayList) {

                        arrayList.add("Val" + (threadId \* OPERATIONS\_PER\_THREAD + j));

                    }

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* endArrayList = System.currentTimeMillis();

        System.out.println("ArrayList with synchronization time: " + (endArrayList - startArrayList) + " ms");

        System.out.println("ArrayList size: " + arrayList.size());

        // CopyOnWriteArrayList (thread-safe but costly on writes)

        concurrentList.clear();

*long* startCopyOnWrite = System.currentTimeMillis();

        for (*int* i = 0; i < NUM\_THREADS; i++) {

*int* threadId = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS\_PER\_THREAD; j++) {

                    concurrentList.add("Val" + (threadId \* OPERATIONS\_PER\_THREAD + j));

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

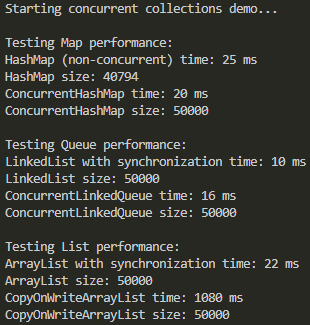
*long* endCopyOnWrite = System.currentTimeMillis();

        System.out.println("CopyOnWriteArrayList time: " + (endCopyOnWrite - startCopyOnWrite) + " ms");

        System.out.println("CopyOnWriteArrayList size: " + concurrentList.size());

    }

}



import java.util.\*;

import java.util.concurrent.\*;

public class ConcurrentCollectionsTest {

    // Collections to test

    private static ConcurrentHashMap<Integer, String> concurrentMap = new ConcurrentHashMap<>();

    private static HashMap<Integer, String> hashMap = new HashMap<>();

    private static ConcurrentLinkedQueue<Integer> concurrentQueue = new ConcurrentLinkedQueue<>();

    private static LinkedList<Integer> linkedList = new LinkedList<>();

    private static CopyOnWriteArrayList<String> concurrentList = new CopyOnWriteArrayList<>();

    private static ArrayList<String> arrayList = new ArrayList<>();

    private static final *int* THREADS = 4;

    private static final *int* OPERATIONS = 50000;

    public static *void* main(String[] *args*) throws InterruptedException {

        System.out.println("Concurrent Collections Performance Test\n");

        testMaps();

        testQueues();

        testLists();

    }

    private static *void* testMaps() throws InterruptedException {

        System.out.println("Testing Map:");

        // Non-concurrent HashMap (not thread-safe)

        hashMap.clear();

        Thread[] threads = new Thread[THREADS];

*long* start = System.currentTimeMillis();

        for (*int* i = 0; i < THREADS; i++) {

*int* tid = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS; j++) {

                    synchronized(hashMap) {

                        hashMap.put(tid \* OPERATIONS + j, "Val" + j);

                    }

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* end = System.currentTimeMillis();

        System.out.printf("HashMap with synchronization time: %d ms, size: %d%n", (end - start), hashMap.size());

        // ConcurrentHashMap

        concurrentMap.clear();

        for (*int* i = 0; i < THREADS; i++) {

*int* tid = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS; j++) {

                    concurrentMap.put(tid \* OPERATIONS + j, "Val" + j);

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

        end = System.currentTimeMillis();

        System.out.printf("ConcurrentHashMap time: %d ms, size: %d%n\n", (end - start), concurrentMap.size());

    }

    private static *void* testQueues() throws InterruptedException {

        System.out.println("Testing Queue:");

        // LinkedList with synchronization

        linkedList.clear();

        Thread[] threads = new Thread[THREADS];

*long* start = System.currentTimeMillis();

        for (*int* i = 0; i < THREADS; i++) {

*int* tid = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS; j++) {

                    synchronized(linkedList) {

                        linkedList.add(tid \* OPERATIONS + j);

                    }

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* end = System.currentTimeMillis();

        System.out.printf("LinkedList with synchronization time: %d ms, size: %d%n", (end - start), linkedList.size());

        // ConcurrentLinkedQueue

        concurrentQueue.clear();

        for (*int* i = 0; i < THREADS; i++) {

*int* tid = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS; j++) {

                    concurrentQueue.add(tid \* OPERATIONS + j);

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

        end = System.currentTimeMillis();

        System.out.printf("ConcurrentLinkedQueue time: %d ms, size: %d%n%n", (end - start), concurrentQueue.size());

    }

    private static *void* testLists() throws InterruptedException {

        System.out.println("Testing List:");

        // ArrayList with synchronization

        arrayList.clear();

        Thread[] threads = new Thread[THREADS];

*long* start = System.currentTimeMillis();

        for (*int* i = 0; i < THREADS; i++) {

*int* tid = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS; j++) {

                    synchronized(arrayList) {

                        arrayList.add("Val" + (tid \* OPERATIONS + j));

                    }

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

*long* end = System.currentTimeMillis();

        System.out.printf("ArrayList with synchronization time: %d ms, size: %d%n", (end - start), arrayList.size());

        // CopyOnWriteArrayList

        concurrentList.clear();

        for (*int* i = 0; i < THREADS; i++) {

*int* tid = i;

            threads[i] = new Thread(() *->* {

                for (*int* j = 0; j < OPERATIONS; j++) {

                    concurrentList.add("Val" + (tid \* OPERATIONS + j));

                }

            });

            threads[i].start();

        }

        for (Thread t : threads) t.join();

        end = System.currentTimeMillis();

        System.out.printf("CopyOnWriteArrayList time: %d ms, size: %d%n", (end - start), concurrentList.size());

    }

}

