```
package slashuscraper;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.concurrent.Callable;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentLinkedQueue;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import slashuscraper.analyze.AnalyzeComment;
import slashuscraper.analyze.AnalyzeUser;
import slashuscraper.object.Comment;
import slashuscraper.object.User;

public class ScraperManager {

	@SuppressWarnings("unused")
	private ConcurrentLinkedQueue<Comment> comments = new ConcurrentLinkedQueue<Comment>();

	public static List<User> scrapeUsers(ArrayList<String> usernames)
			throws InterruptedException {

		// Create a list of users
		ConcurrentHashMap<String, User> users = new ConcurrentHashMap<String, User>();

		// Create a queue of comments
		ConcurrentLinkedQueue<Comment> comments = new ConcurrentLinkedQueue<Comment>();

		/**************************************************************************/
		/* Scrape comments                                                        */
		/**************************************************************************/

		// Thread count
		final int NUMTHREADS = 8;
		// Create a fixed size pool of threads
		ExecutorService scrapeThreads = Executors.newFixedThreadPool(NUMTHREADS);

		System.out.println("There are: " + usernames.size() + " users to be scraped.");

		// ConcurrentLinkedQueue<Comment> comments = new ConcurrentLinkedQueue<Comment>();

		List<Future<ConcurrentLinkedQueue<Comment>>> toAnalyzeComments = new ArrayList<Future<ConcurrentLinkedQueue<Comment>>>();

		for (int i = 0; i < usernames.size(); i++) {
			String username = usernames.get(i);

			// Add user to users list
			users.put(usernames.get(i).trim().toLowerCase(), new User(usernames.get(i).trim().toLowerCase()));

			Callable<ConcurrentLinkedQueue<Comment>> scraper = new Scraper(username);
			Future<ConcurrentLinkedQueue<Comment>> addToComments = scrapeThreads.submit(scraper);

			toAnalyzeComments.add(addToComments);
		}

		scrapeThreads.shutdown();

		while (!scrapeThreads.isTerminated()) { ; }

		// Get the futures
		for (Future<ConcurrentLinkedQueue<Comment>> future : toAnalyzeComments) {
			try {
				ConcurrentLinkedQueue<Comment> scrapedUser = future.get();

				comments.addAll(scrapedUser);

				if (scrapedUser == null) {
					System.out.println("scrapedUser is null.");
				}
			} catch (ExecutionException e) {
				// Error
				System.out.println("FUTURE_ERROR: Could not 'get' comment from future");
			}
		}

		System.out.println("[COMPLETE] Done scraping");

		/**************************************************************************/
		/* Analyze comments                                                       */
		/**************************************************************************/

		// Set start time to calculate run time
		long startTime = System.currentTimeMillis();

		// Begin analyzing posts and comments
		// Use a cached thread pool to expand thread count dynamically
		ExecutorService cachedPool1 = Executors.newCachedThreadPool();
```

```
// Create a list of future objects
List<Future<Comment>> analyzedComments = new ArrayList<Future<Comment>>();

Comment toAnalyze = null;

while((toAnalyze = comments.poll()) != null) {
        Callable<Comment> analyzeComments = new AnalyzeComment(toAnalyze);
        Future<Comment> callableFuture = cachedPool1.submit(analyzeComments);
        analyzedComments.add(callableFuture);
}

// Shutdown the pool.
cachedPool1.shutdown();

// Wait until shutdown complete
while(!cachedPool1.isTerminated()) { ; }

// Key object for matching with user
String key = null;
// Object
Comment com;

// Testing output
System.out.println("[COMPLETE] Done processing comments");

// Match comments with user object
for(Future<Comment> f : analyzedComments) {
        // Clean up name to match up with key
        try {
                // Get comment from future
                com = f.get();
                // Get key from comment
                key = com.getAuthor().trim().toLowerCase();
                // Get user by key and add comment
                users.get(key).addComment(com);
        } catch (ExecutionException e) {
                // Error
                System.out.println("FUTURE_ERROR: Could not 'get' comment from future");
        }
}

// Testing output
System.out.println("[COMPLETE] Done matching comments with users");

/*************************************************************************/
/* Analyze User Data                                                     */
/*************************************************************************/

// Begin analyzing posts and comments by user
// Use a cached thread pool to expand thread count dynamically
ExecutorService cachedPool2 = Executors.newCachedThreadPool();

// Create a list of future objects
List<Future<User>> analyzedUsers = new ArrayList<Future<User>>();

// User entry to analyze
Map.Entry<String, User> entry = null;

// User hash map iterator
Iterator<Entry<String, User>> itr = users.entrySet().iterator();

while(itr.hasNext()) {
        entry = itr.next();
        Callable<User> analyzeUsers = new AnalyzeUser(entry.getValue());
        Future<User> callableFuture2 = cachedPool2.submit(analyzeUsers);
        analyzedUsers.add(callableFuture2);
}

// Shutdown the pool.
for(Future<User> f : analyzedUsers) {
        while(!f.isDone()) { ;; }
}

// Shutdown the pool.
cachedPool2.shutdown();

// Wait for termination
while(!cachedPool2.isTerminated()) { ; }

// Testing output
System.out.println("[COMPLETE] Done analyzing users");

// Get users from 'future' objects and append to list
ArrayList<User> processedUsers = new ArrayList<User>();

// Temporary user
User usr = null;

// Append
for(Future<User> f : analyzedUsers) {
        // Try to get user
        try {
                // Get user
                usr = f.get();
                // Get key
```

```
                                processedUsers.add(usr);
                        } catch(ExecutionException e) {
                                // Throw error
                                System.out.println("FUTURE_ERROR: Could not get user(s) for future object");
                                e.printStackTrace();
                        }
                }

                // Set end time and then calculate run time
                long endTime   = System.currentTimeMillis();
                long totalTime = endTime - startTime;
                // Disply run time
                System.out.println("[STATUS] Calculation time elapsed: " + totalTime +" milliseconds");

                // Return processed users
                return processedUsers;
        }
}
```