ACS Programming Assignment 1

Anders Holst, wlc376

2020-11-24

0 Questions for discussion on architecture

0.1 1. Short description of implementation and tests

- *(a) How does your implementation and tests address the all-or-nothing semantics?*

All-or-nothing semantics in implementation

Of those methods that I have implemented for the purposes of this assignment, only `CertainBookStore.rateBooks()` needs explicit handling of all-or-nothing semantics. However, this is simply covered by iterating the entire input `bookRatings` set, asserting that each `BookRating` is valid, failing immediately when a rating isn't, and *only then* registering the book ratings.

In the below snippet, lines 11-19 show how each `BookRating` is validated before ratings are registered in lines 22-26:

```
1  @Override
2  public synchronized void rateBooks(Set<BookRating> bookRatings) throws BookStoreException {
3
4    // validate all ratings in bookRatings.
5    for (BookRating bookRating : bookRatings) {
6      int isbn   = bookRating.getISBN();
7      int rating = bookRating.getRating();
8
9      // a BookRating is valid if its isbn is valid and corresponds to an existing book,
10     // and if its rating is a number in the range 0..5.
11     if (BookStoreUtility.isInvalidISBN(isbn)) {
12       throw new BookStoreException(BookStoreConstants.ISBN + isbn + BookStoreConstants.INVALID);
13     }
14     else if (!bookMap.containsKey(isbn)) {
15       throw new BookStoreException(BookStoreConstants.ISBN + isbn + BookStoreConstants.NOT_AVAILABLE);
16     }
17     else if (BookStoreUtility.isInvalidRating(rating)) {
18       throw new BookStoreException(BookStoreConstants.RATING + rating + BookStoreConstants.INVALID);
19     }
20   }
21
22   for (BookRating bookRating : bookRatings) {
23
24     BookStoreBook book = bookMap.get(bookRating.getISBN()); // the actual existence of a book with
25     book.addRating(bookRating.getRating());                 // the given ISBN was checked earlier.
26
27   }
28 }
```

All-or-nothing semantics in testing

Below snippet is of the test case asserting all-or-nothing semantics of `CertainBookStore.rateBooks()`. An invalid `BookRating` is created in line 16 and registered in line 20, and in line 26 I assert that no book has received any rating.

```
1  @Test
2  public void testRateBooksAllOrNothing() throws BookStoreException {
3
4    // add a couple more default books.
5    initializeMoreBooks();
6
7    // assert that no book has been rated yet.
8    storeManager.getBooks().forEach(book -> assertEquals(0, book.getTotalRating()));
9
10   Set<BookRating> ratings = new HashSet<>();
11
12   // create one valid and one invalid BookRating.
13   int valid_rating   = 3;
14   int invalid_rating = -42;
15   ratings.add(new BookRating(TEST_ISBN,  valid_rating));
16   ratings.add(new BookRating(TEST_ISBN2, invalid_rating));
17
18   // register the new ratings. this fails, since the second rating is invalid.
19   try {
20     client.rateBooks(ratings);
21   }
22     catch (BookStoreException ignored) {
23   }
24
25   // assert all-or-nothing semantics by verifying that all books still have rating 0.
26   storeManager.getBooks().forEach(book -> assertEquals(0, book.getTotalRating()));
27 }
```

- *(b) How did you test whether the service behaves according to the interface regardless of use of RPCs or local calls?*

I simply executed my tests both locally and non-locally by setting the `localTest` private field of each test class to `true` and `false`, respectively, before running my test suites.

0.2 2. Strong Modularity in the Architecture

- *(a) In which sense is the architecture strongly modular?*

The architecture is for example modular in the sense that it allows different types of clients (and stock managers), as long as these adhere to the BookStore interface. In addition, the fact that the bookstore is implemented as a simple HTTP server means that clients do not even have to be Java threads, as long as they adhere to the RPC message format.

- *(b) What kind of isolation and protection does the architecture provide between the two types of clients and the bookstore service?*

Blank.

- *(c) How is enforced modularity affected when we run clients and services locally in the same JVM, as possible through our test cases?*

When `localTest` is set to `true` in either test class, then the private fields `storeManager` and `client` are each set to the same instance of the `CertainBookStore` class. If either the client or store manager fails, then the other fails aswell.

0.3 3. Naming service in the architecture

- *(a) Is there a naming service in the architecture? If so, what is its functionality?*

I would say no - if I'm wrong, then either I can't find it or I'm unsure what is meant by "naming service".

- *(b) Describe the naming mechanism that allows clients to discover and communicate with services.*

Blank :)

0.4 4. RPC semantics in the architecture

Note: I should first say that I am not familiar with the behind-the-scenes of the various `jetty` *HTTP libraries we use.*

I would argue that the architecture implements *at-most-once* RPC semantics, since HTTP requests are never retried. This is a good idea for the book store, since a number of RPC's modify the book store in some way - we wouldn't want to accidentally too many copies of the same book, or to add duplicates of the same book to the store front.

0.5 5. Web proxies

- *(a) Is it safe to use web proxy servers with the architecture in Figure 1?*

Yes.

- *(b) If so, explain why this is safe and describe in between which components these proxy servers should be deployed. If not, why not?*

Proxy servers could be placed between clients and the book store server. This could alleviate incoming traffic to the book store server via eg. caching and traffic control.

I would argue that it is safe so long as the system is coherent. This would be a complex system to fault-secure, but would ultimately be a more secure, tolerant system than letting a single server receive all incoming requests in real-time.

0.6 6. Bottlenecks in the architecture

- *(a) Is/are there any scalability bottleneck/s in this architecture with respect to the number of clients?*

Yes!

- *If so, where is/are the bottleneck/s? If not, why can we infinitely scale the number of clients accessing this service?*

There is a bottleneck in that for the moment, only a single BookStoreHTTPServer handles all clients. Infinitely scaling the number of clients would mean huge queues of incoming HTTP requests and very plausibly dropped requests.

0.7 7. In the Event of CertainBookStore crashing

- *(a) Would clients experience failures differently if web proxies were used in the architecture?*

Definitely. The result of all RPCs which have no side effects can be stored in these proxies, such that they can be served to users even when the main bookstore server is temporarily down.

- *(b) Could caching at the web proxies be employed as a way to mask failures from clients?*

This depends on whether the proxy is able to cache requests and re-attempt forwarding to the CertainBookStore once it restarts. If we assume this is not possible, then caching would only be able to mask failure for those requests which only *read* from the book store, such as getting the selection of books or the editor picks, but would not be able to serve eg. purchase requests before the CertainBookStore was back up and running.

- *(c) How would the use of web caching affect the semantics offered by the bookstore service?*

Blank.
