PCPP Assignment 5

Group name: Thread Heresy
Real names : Nedas Surkus, Niclas Abelsen,
Github username: nesu, niab

November 2022

Contents

10.1

1

See *CASHistogram.java*.

Increment: The increment fetches old value from an AtomicLong and then transforms it into a new value by adding one. And uses CAS (Compere and Set) to set the new value. If it fails, that is returning false, the operation is retried.

Get bins: Get bins is thread safe as we return the value currently stored in a specific atomic long. Since the increment operation is optimistic and retries the operation if it failed. We can assume that get bins will output the correct value inside the bin.

Get span: Get span is thread safe because "counts" is declared as final. Thus it will not execute before it is initialized. It is initialized when the class is created and never changes. Thus simply retrieving span will always output the same value.

Get And clear: Get and clear uses the same principle as Increment. However, the new value that is being replace is 0. V will always get the current value inside the Atomic Long before its overridden. Thus V will output the last know value inside this specific bin.

2

See *TestHistogram.java*.

3

See *TestCASLockHistogram.java*.

```
HistogramCAS  1             8391435075.0 ns 40503574.52          2
HistogramLock  1            8457498005.0 ns 90187234.66          2
HistogramCAS  2             5441016130.0 ns 84122545.69          2
HistogramLock  2            5457223840.0 ns 73961885.79          2
HistogramCAS  3             4117844870.0 ns 62000996.86          2
HistogramLock  3            3877220940.0 ns 61352891.53          2
HistogramCAS  4             3211189635.0 ns 66964342.43          2
HistogramLock  4            3056710605.0 ns 25435293.00          2
HistogramCAS  5             2670320385.0 ns 28094657.29          2
HistogramLock  5            2569183825.0 ns 68735570.31          2
HistogramCAS  6             2388640045.0 ns 60233238.84          2
HistogramLock  6            2490018190.0 ns 44881974.75          2
HistogramCAS  7             2183299750.0 ns 44758310.77          2
HistogramLock  7            2273238525.0 ns 61314721.26          2
HistogramCAS  8             2056353840.0 ns 40823030.52          2
HistogramLock  8            2380426145.0 ns 79244631.33          2
HistogramCAS  9             1925693440.0 ns 31879131.85          2
HistogramLock  9            3002749250.0 ns 261846172.87          2
HistogramCAS 10             1811519610.0 ns 20171038.23          2
HistogramLock 10            2686373060.0 ns 35953886.11          2
HistogramCAS 11             1735546610.0 ns 25545367.43          2
HistogramLock 11            2507092615.0 ns 201825450.02          2
HistogramCAS 12             1662164320.0 ns 37337131.21          2
HistogramLock 12            2689391985.0 ns 553717393.19          2
HistogramCAS 13             1651494795.0 ns 24567298.00          2
HistogramLock 13            2723250220.0 ns 500249719.39          2
```

```
HistogramCAS 14              1614514040.0 ns 47899823.40          2
HistogramLock 14             2259881105.0 ns 63808159.10          2
HistogramCAS 15              1680497525.0 ns 28530557.32          2
HistogramLock 15             2246271690.0 ns 58357552.47          2
HistogramCAS 16              1578848030.0 ns 19034728.32          2
HistogramLock 16             2184661855.0 ns 66363229.29          2
HistogramCAS 17              1612846925.0 ns 27990805.96          2
HistogramLock 17             2181592455.0 ns 55472737.04          2
HistogramCAS 18              1618604335.0 ns 40207188.01          2
HistogramLock 18             2165984415.0 ns 56340213.26          2
HistogramCAS 19              1574959905.0 ns 32266507.34          2
HistogramLock 19             2282153235.0 ns 140634733.85          2
HistogramCAS 20              1572140690.0 ns 9169374.72          2
HistogramLock 20             2373805075.0 ns 53335297.86          2
HistogramCAS 21              1571300600.0 ns 11327841.30          2
HistogramLock 21             2369810820.0 ns 43453919.95          2
HistogramCAS 22              1529329040.0 ns 17719059.94          2
HistogramLock 22             2341183245.0 ns 17327611.75          2
HistogramCAS 23              1530909720.0 ns 19400224.68          2
HistogramLock 23             2376269805.0 ns 42722039.11          2
HistogramCAS 24              1555567045.0 ns 21882354.74          2
HistogramLock 24             2392827570.0 ns 29295978.15          2
HistogramCAS 25              1558897640.0 ns 24895928.48          2
HistogramLock 25             2375877640.0 ns 57386818.82          2
HistogramCAS 26              1525182965.0 ns 26927037.30          2
HistogramLock 26             2345174325.0 ns 22002258.33          2
HistogramCAS 27              1514125940.0 ns 9067280.34          2
HistogramLock 27             2326675365.0 ns 13575573.77          2
HistogramCAS 28              1512188775.0 ns 12510045.41          2
HistogramLock 28             2347957590.0 ns 42166318.90          2
HistogramCAS 29              1515386455.0 ns 25445720.23          2
HistogramLock 29             2354718115.0 ns 43246321.48          2
HistogramCAS 30              1506938255.0 ns 9428518.08          2
HistogramLock 30             2336772715.0 ns 30015742.41          2
HistogramCAS 31              1493542130.0 ns 10273534.42          2
HistogramLock 31             2365636110.0 ns 53139843.32          2
HistogramCAS 32              1491214575.0 ns 14935694.37          2
HistogramLock 32             2324805125.0 ns 22070193.28          2
```

10.2

1

See *ReadWriteCASLock.java*.

2

See *ReadWriteCASLock.java*.

3

See *ReadWriteCASLock.java*.

4

See *ReadWriteCASLock.java*.

5

See *TestLocks.java*.

6

See *TestLocks.java*.

11.1

1

The guardian handles a single message which is `startGuardian`. The responsibility of this is to spawn all initial actors for this program. That is two MobileApps, two banks and two accounts. One in each bank.

The guardian is initialized in the `Main.java` using the `ActorSystem` with an actor name `guardian_actor`. The kick-off message, which starts the guardian, is sent using 'fire-and-forge' with `guardian.tell(new Guardian.Start());`. The guardian is also terminated in this file using `guardian.terminate()` which terminates all the actors that has been created using this guardian and the possible children of the actors.

2

In account we have a state that contains two values. The account ID and the balance of the account. When we start the account, we pass some initial balance and an ID we will assign to this account. The account ID would help us keep track on which account is executing the transaction.

In account we do two behaviours. depositToAccount and readBalance.

Account has two messages and two response messages.

1. Deposit message contains the transaction id, to keep track on the transactions, value to be added which can be positive or negative, an actor reference. This actor reference will refer to the actor that sent this message. This allows us to reply to the actor with the results of the operation.

2. ReadBalance message contains transaction id and an actor to reply to. When a deposit message is sent to the account we execute depositToAccount behaviour that adds a specified value to the account and then replies to the actor that sends a message. When a ReadBalance message is sent, we execute readBalance behaviour and reply to the actor sending this message with a reply message containing the balance. This is done as Bank and Account actors are separate objects and cant access their values. In order for the mobile app or bank to know the balance, account must send a message back with the balance.

3

The Bank acts as a group manager for accounts. In its state it contains a Bank id and a duration time which functions as a timeout for the ask.

The Bank can get the messages: Transaction, ReadBalance and AdaptedMessage.
Transaction message contains a transaction id, a sender account reference and a receiver account reference and amount that needs to be transferred. Additionally it has a reference to the mobile application that sent this transaction so it can reply to it if the transaction was successful or not.

When the bank receives a transaction message, it will create 2 ask requests to the accounts provided. First it will reduce the amount of money from the sender and then he will add money to receiver account. In the real world, we would first check if the transaction can be completed by checking the senders balance and if it is not zero, we would reduce the senders account and only if that would be successful, we would add money to the receiver. Each of these transactions expect a future to be returned, once a future responds, we create create an Adapted message and send it the Bank itself to be printed out.

The Read Balance message contains a transaction id, bank actor reference and the account reference. This account reference will allow us to call ReadBalance on a specific account. The Adapted

message contains a string and will be used to format messages received from the Account.

When we received ReadBalance message, we create an ask request to the account we want to get the balance from. This request is also a future and once the account responds with a balance, we pass the response along to the mobile app that send ReadBalance message.

4

The mobile app's state contains a Mobile app id, transaction id that will be incremented for for each transaction that mobile app creates, a random and duration. The random and duration is used for generating the random asks requests.

The mobile application can get several messages from the guardian. TestTransaction, CreateTransaction and ReadBalance.

When the mobile app receives TestTransaction message, it will create two transactions and send it to the bank. One transaction will transfer 1000 from account A to B and the second transaction will transfer 1000 from B to A on a specified bank. if both transactions are accurate, if we print the balance account A and B should have 1000 in the end.

When the mobile app receives Create transaction, the mobile app will generate 100 transactions from account A to B using the specified bank. The amount that will be transferred is random between 1 to 1000.

When the mobile app receives ReadBalance it will send an ask ReadBalance message to the Specified Bank and add itself to the Actor Reference so it can get the reply message. Once the account sends its own balance to the Bank, the Bank replies with the provided balance to Mobile app which then receives this message and generates an AdaptedResponse reply. This will then print out the balance inside MobileApp.

5

See MobileApp.java TestTransactions method

6

See MobileApp.java CreateTransactions method

7

It depends when we are sending the request and to which bank. If we get the balance from the same bank, we will receive the balance at the end of the 100 transactions. If we try to get the balance using a different bank, we will get a stale balance as account actor executes the requests as they come.

8

If we try to update the account at the same time by two different banks. We can get stale data. While account making the transaction will not process 2 different requests at the same time, bank 1 might think accounts 1 balance is different than it is as bank 2 will do a transaction during the look up. Thus we cannot ensure that consistency if we execute a transaction on the same actor using two different actors. To fix that we need to assign specific accounts to a bank.