Hi from Diffbot!

At Diffbot we know that the traditional whiteboard interview is only one part of assessing real-world skills. We've chosen a take-home project for you that attempts emulates practical software engineering design decisions you will need to make at Diffbot. You can use any language and feel free to consult any resource, including the web.

This problem should typically take 45-75 minutes starting from having your programming environment ready. After 75 minutes, please send your code to recruiting@diffbot.com. A single source file which includes all code and tests is preferred to an archive or multiple files.

Problem descriptions

The programming exercise is designed to test general programming skills, data structures and optimization. It is language-agnostic and can be coded in any language.

1. Implement a simple in-memory key-value database on the lines of Redis.
2. All keys are Strings and all values are Integers
3. The database supports 2 types of commands:
    a. Basic: `GET, SET, INCR, DEL, DELVALUE`
    b. Transaction: `MULTI, EXEC, DISCARD`
4. Implement a simple shell that reads one command at a time from stdin.
5. Each command is entered one line at a time.

NOTE: Examples in this problem use a leading leading "`> `" prompt just for illustration, Implementing this prompt is optional.

What's expected

Provide a solution that is:

1. Correct
2. Clean, not over-engineered, minimally but clearly documented (aka, do not document the obvious)
3. Some unit tests that cover all cases
4. Solution and tests preferably submitted in a single source file
5. Programming solution must be in one of these languages: Java, Python, Scala, or C++/C
6. Make reasonable assumptions about any minor missing or ambiguous information in the problem statement and state those assumptions in code comments.

Commands

Basic commands

`SET key value`

Set `key` to hold the integer `value`. If key already holds a value, it is overwritten.
This command has no worse than `O(log n)` performance in the average case.

Example: `SET key1 12`

`INCR key`

Increments the number stored at `key` by one. If the key does not exist, it is set to 0 before performing the operation.
This command has no worse than `O(log n)` performance in the average case.

Example: `INCR key1`

`GET key`

Get the integer value corresponding to `key`. If the key does not exist the special value `<nil>` is printed. The value is printed to stdout.
This command has no worse than `O(log n)` performance in the average case.

Example: `GET key1`

`DEL key`

Removes the specified `key`. A key is ignored if it does not exist.
This command has no worse than `O(log n)` performance in the average case.

Example: `DEL key1`

`DELVALUE value`

Removes all keys which have the specified `value`. A value is ignored if it does not exist.
This command has no worse than `O(log n)` performance in the average case.

Example: `DELVALUE 12`

Transaction-specific commands

There are 3 transaction commands:

- MULTI: Starts a transaction
- DISCARD: Discard a transaction
- EXEC: Commit a transaction

Here's a simple transaction (the leading "`> `" prompt is just for illustration and is optional to implement):

```
> MULTI
> SET key1 12
> DISCARD
1
```

```
> INCR key1
> EXEC
1
> GET key1
1
```

MULTI value

Marks the start of a transaction block. Subsequent commands will be queued for atomic execution using `EXEC`. The `MULTI` command is followed by zero or more commands before `EXEC` or `DISCARD`.

Example: `MULTI`

EXEC value

Executes all previously queued commands in a transaction. Prints "`NOT IN TRANSACTION`" if no transaction was started otherwise prints the number of commands that were committed in the transaction.

Example: `DISCARD`

DISCARD value

Flushes (discards) all previously queued commands in a transaction without making any changes to the data. Prints "`NOT IN TRANSACTION`" if no transaction was started otherwise prints the number of commands that were discarded.

Example: `DISCARD`

Sample inputs/outputs

NOTE: the leading "`> `" prompt is just for illustration and is optional to implement.

Sample 1 (GET/SET/INCR/DEL):

```
> SET key 7
> GET key
7
> DEL key
> GET key
<nil>
> INCR key
> GET key
1
```

Sample 2 (DELVALUE):

```
> SET key1 7
> SET key2 7
> SET key3 8
> DELVALUE 7
> GET key1
<nil>
> GET key2
<nil>
> GET key3
8
```

Sample 3 (MULTI/EXEC/DISCARD):

```
> MULTI
> SET key1 12
> INCR key1
> GET key1
13
> DISCARD
2
> INCR key1
> INCR key1
> EXEC
2
> GET key1
2
```

NOTE: Do specifically note the behavior of `GET` command within the transaction.

Sample 4 (MULTI/EXEC/DISCARD):

```
> SET key1 12
> MULTI
> DEL key1
> DISCARD
1
> GET key1
12
```

Sample 5 (EXEC):

```
> EXEC
NOT IN TRANSACTION
```

Sample 6 (DISCARD):

```
> DISCARD
NOT IN TRANSACTION
```