CS 635 Advanced OO Design and Programming
Fall Semester, 2022
Assignment 3

San Diego State University -- This page was last updated on 11/8/22

Assignment 3
Reactive NoSQL
Due Nov 28

NoSQL databases are used for unstructured data. You are going to create a mini-in-memory database. NoSQL databases tend to act like hashtables. Each piece of data has a key. When you add data to the database, you provide the key and the data. To retrieve the data, you use the key. We will use strings as keys to keep things on the simple side. Our data will be Strings, Numbers, “Arrays” and “Objects.” “Arrays” are ordered lists of data (Strings, Numbers, “Arrays”, or “Objects”). A single “Array” can hold multiple types of data at the same time. An “Object” is a hashtable with a String as a key and any data as the value. You will find the JSON library useful here. The operations on your NoSQL database will be to add new data, get the data at a given key, replace data at a given key, and remove a key and its data. Modifying data at a given key is a bit more complex, so we will not support that operation directly. Modifying data can be done with a retrieval and an add. Your first task is to implement the mini-in-memory database.

Data Examples

Examples of numbers are: 1, 2.3, -234, -34.294, 1.8e2, 0.16e3, 1d

Examples of strings are: “”, “cat”, and “Rat”. We only need to support ASCII characters in our strings.

Examples of Arrays are:

```
[1, 2, 3]
[ ]
[2.3, “at”, 1.67e3, [1, “me”, {“a”:1}], “bat”]
```

Examples of Objects are:

```
{"name": "Roger", "age": 21}
{“account 12343”: {“name”:”Bill”, “address”:”123 main street”, “phones”:[“619-594-3535”],
“balance”: 1234.05”}}
```

Data Literals & Operations

Java has literals for numbers and strings. It does not have literals for the arrays and objects defined above. So we need to add support for these formats. It needs to be done in two ways. First, given a string presentation of an array, convert it to an array object. So we need to be able to take the string “[1, 2, 3]” and convert it to an array object. Similarly, we need to be able

to convert a string presentation of an Object like “{\"name\": \"Roger\", \"age\": 21}” into an object. The second way is to create an Array or Object instance and add data to it.

Array Operations

Your array needs at least the following methods.

put

> Has one argument that is any valid data type defined above. Adds the argument to the end of the array. The argument is not allowed to be null.

getX(int index)

> X is one of Int, Double, String, Array, or Object. Returns the item at the given index in the array. If “index” is out of bounds, the method throws an exception. If the item at the given index is not of type X, throw an exception. Returns the item as the type indicated by X. So getInt will return an int, and getString will return a String.

get(int index)

> Returns the item at the given index in the array. If “index” is out of bounds, the method throws an exception. The value is returned as a Java object.

length()

> Returns the number of elements in the array.

toString()

> Returns a string representation of the array.

remove(int index)

> Removes the top-level key and its associated value. Returns the associated value. It does nothing if the key does not exist and returns null.

class method: fromString

> Has one argument of type string. The input is a string representation of an array. Returns at array object created from the string. Throws an exception if the string does not represent a valid array object.

Object Operations

Your object needs at least the following operations.

put(String key, Y value)

Where Y is one of int, double, String, your array type, or object type. Adds the value at the key in the object. Returns the object that was the receiver. That is, if we have an object, say x, and then x.put(“a”, 1) will return the object x after adding 1 at the key “a”. The value is not allowed to be null.

getX(String key)

Where X is one of Int, Double, String, Array, or Object. Returns the item at the given key in the object. The method throws an exception if the key does not exist in the object. If the item at the given key is not of type X, throw an exception. Returns the item as the type indicated by X. So getInt will return an int, and getString will return a String.

get(String key)

Returns the item at the key in the object. The method throws an exception if the key does not exist in the object. The value is returned as a Java object.

length()

Returns the number of top-level keys in the object.

remove(String key)

Removes the top-level key and its associated value. Returns the associated value. It does nothing if the key does not exist and returns null.

toString()

Returns a string representation of the object.

class method: fromString

Has one argument of type string. The input is a string representation of an object. Returns at object created from the string. Throws an exception if the string does not represent a valid object.

DB Operations

Your database has to support the following basic operations.

put(String key, Y value)

Where Y is one of int, double, String, your array type, or object type. Adds the value at the key in the database. Returns the database. The value is not allowed to be null.

getX(String key)

Where X is one of Int, Double, String, Array, or Object. Returns the item at the given key in the database. The method throws an exception if the key does not exist in the database. If the item at the given key is not of type X, throw an exception. Returns the item as the type indicated by X. So getInt will return an int, and getString will return a String.

get(String key)

Returns the item at the key in the database. The method throws an exception if the key does not exist in the database. The value is returned as a Java object.

remove(String key)

Removes the top-level key and its associated value. Returns the associated value. It does nothing if the key does not exist and returns null.

More operations are added below.

Transactions

We need our database to support transactions. The database needs a transaction method that creates a transaction object and returns it. Each transaction on the database is independent of the other transactions on the database. We do not have to lock the Transaction object. Have the following basic methods:

put(String key, Y value)

getX(String key)

get(String key)

remove(String key)

The key in the getX, get, and remove can be any key in the database. All the methods act like the ones defined in the database.

When the above methods are called on a transaction, the operations need to be done on the database, as the current state of the database may be used to determine if the transaction will be aborted or committed.

In addition, the transaction has two other required methods.

commit()

Ends the transaction. All operations in the transaction remain executed.

abort()

Undoes all the operations that have been added to this transaction. Does not affect other transaction objects.

isActive()
Return true if the transaction has not been committed or aborted; otherwise, return false.

After a transaction has been aborted or committed, calling put, getX (i.e., getInt, getString,...), get, remove, commit, or abort on the transaction will throw an exception.

An example of these methods:

```
Transaction sample = db.transaction();
try {
    sample.put(“a”, 5);
    sample.remove(“cat”);
    int interestRate = sample.getInt(“interestRate”);
    if (interestRate > 0.09)
        sample.abort();
    else
        sample.commit();
} catch (Exception e) {
    sample.abort();
}
```

Persistence

The problem with in-memory data is that it is not persistent. So when the program stops running, the data is lost. For persistence, we will use two files.

The first file, "commands.txt”, will contain all the operations done on the database that change its state. The file will contain text, not binary. Each operation will be on a separate line. The operation needs to be saved to the file when the operation is being performed, preferably just before the operation is done.

The second file, "dbSnapshot.txt” will contain the snapshot of the database. When a snapshot of the database is taken, the current data in the database is saved to this file. The previous contents of this file are removed. Also, the contents of the file “commands.txt” are removed.

The database needs four methods related to snapshots:

snapshot()
snapshot(File commands, File snapshot)
Both methods perform the snapshot as defined above. The first method uses the default names for the files.

Class methods:

recover()
recover(File commands, File snapshot)
These methods recover the database from the two saved files. The first method uses the default names for the files. These methods first read the snapshot file to retrieve the last

snapshot of the database. Then they read the operations from the commands file and execute them in the order they are in the file.

Reactive

Since the database is in memory, it does not have to be separate from an application like SQLite. So we could get some data and use it in the program. If the data in the database changes, we want our program to reflect the current state of the database. For this, we will add more methods to the database class.

getCursor(String key)
: Where X is one of Int, Double, String, Array, or Object. Returns a Cursor object on the value stored at the key in the database. It throws an exception if the key is not in the database.

Cursor

A cursor holds a value from the database. The cursor knows the current state of the value. When the value changes in the database, the cursor knows the new state of the value. The cursor class needs the following methods:

getX()
: Where X is one of Int, Double, String, Array, or Object. Returns the item in the cursor. If the item at the given index is not of type X, throw an exception. Returns the item as the type indicated by X. So getInt will return an int, and getString will return a String.

get()
: Returns the item in the cursor. The value is returned as a Java object.

addObserver(Observer o)
: Adds an observer to the cursor. Each cursor holds a value from the database. When that value in the database changes, all the cursor's observers are notified.

removeObserver(Observer o)
: Removes the given observer from the cursor.

Some examples:

```
database.put(“bar”, 10);
Cursor data = database.getCursor(“bar”);
data.value()  // returns 10
database.put(“bar”, 20);
data.value()  // returns 20
```

```
database.put(“bar”, 10);
Cursor data = database.get(“bar”);
data.addObserver(foo);
database.put(“bar”, 20);
//So here, foo gets notified of the change.
```

Of course, the data at “bar” could be more complex. If the value at “bar” is changed in a transaction and the transaction has aborted, the cursor and its observers could receive multiple updates.

Grading

Item	Points
Working Code	10
Unit Tests	10
Patterns	40
Quality of Code	10

Turning in your Assignment

Turn in the hard copy in class.

Version History

1.1 March 15. Added due date, grading, and added observer to the Cursor.

2.0 April 2. Provided more detail. Changed due date.