Software Design and Architecture

Project 1:

In-Memory Database with Persistence

Date : Feb 27, 2019

In-Memory Database

An In-Memory Database (IMDB, also main memory database system or MMDB or memory resident database) is a database management system that primarily relies on main memory for computer data storage.

In-Memory Database

Each time you query a database or update data in a database, you only access the main memory.

The main memory is way faster than any disk.

A good example of such a database is Memcached.

Problems with In-Memory Database

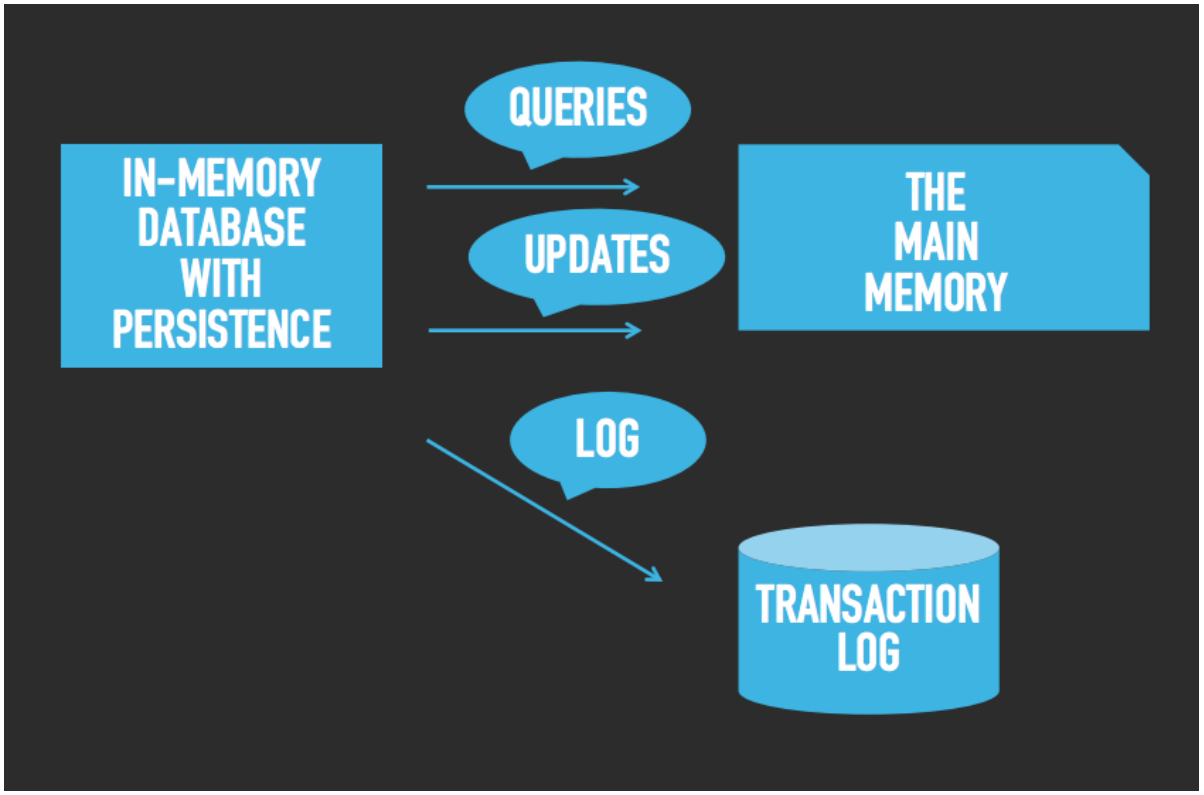
How would you recover your data after a machine with an in-memory database reboots or crashes?

Solution:

In-memory databases with persistence like Redis, Aerospike, Tarantool.

In-Memory Database with Persistence

We persist each operation on disk in a transaction log.



1. We will create an in-memory database for a book store inventory. The book store sells books. Each book has a name, price, unique id and a quantity. The store uses sequential integers for unique ids. We need to be able to
   - Add new books.
   - Sell a book in the inventory.
   - Add new copies of existing books
   - Change the price of a book
   - Find the price and/or quantity of a book by either name or id.

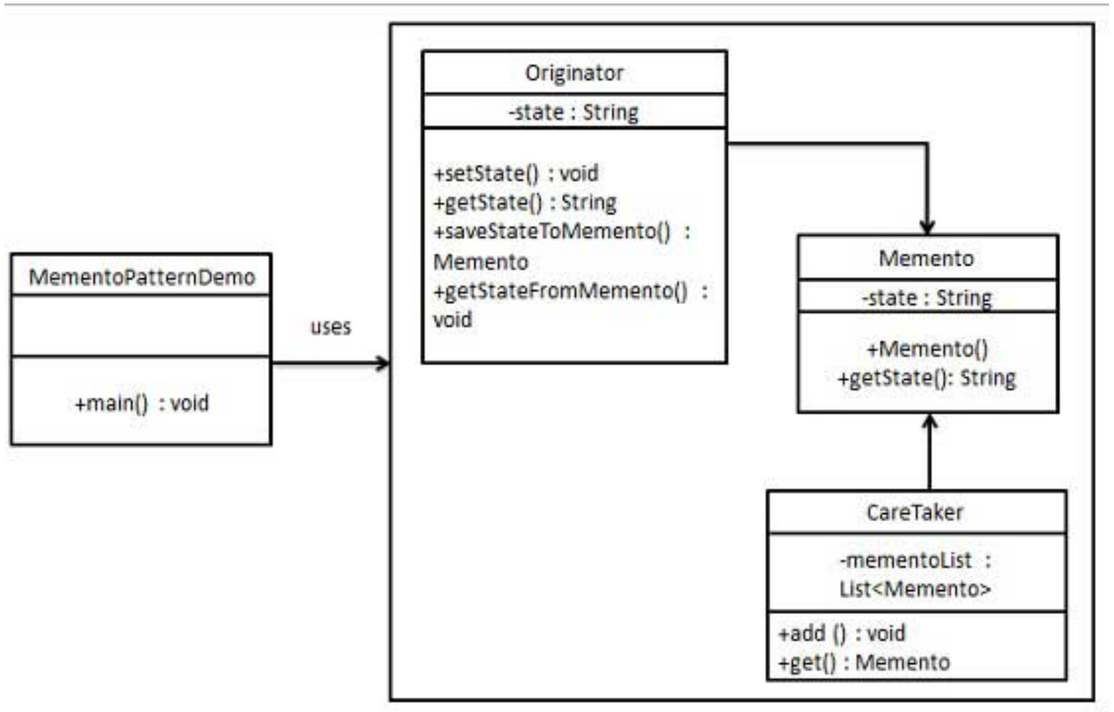
Create an Inventory class to keep track of the store inventory.

2. Use the **memento pattern** to copy the data in an Inventory object. Make the memento serializable so it can be saved in a file. Given an Inventory object and a memento you can restore the Inventory object to a previous state.

So now we can periodically create and save a memento of the Inventory object.

Memento Pattern

It captures and externalize an object's internal state so that the object can be restored to this state.



Originator
- creates and stores states in Memento objects

Caretaker
- restore object state from Memento

Memento
- contains state of an object to be restored

```
public class Originator {
  private String state;
  public void setState(String state){
    this.state = state;
  }
  public String getState(){
    return state;
  }

  public Memento saveStateToMemento(){
    return new Memento(state);
  }

  public void getStateFromMemento(Memento
memento){
    state = memento.getState();
  }
}
```

```
public class Memento {
  private String state;
  public Memento(String state){
    this.state = state;
  }
  public String getState(){
    return state;
  }
```

```
public class CareTaker {
  private List<Memento> mementoList = new
ArrayList<Memento>();

  public void add(Memento state){
    mementoList.add(state);
  }

  public Memento get(int index){
    return mementoList.get(index);
  }
}
```

```
public class MementoPatternDemo {
  public static void main(String[] args) {
    Originator originator = new Originator();
    CareTaker careTaker = new CareTaker();
    originator.setState("State #1");
    originator.setState("State #2");
    careTaker.add(originator.saveStateToMemento());
    originator.setState("State #3");
    careTaker.add(originator.saveStateToMemento());
    originator.setState("State #4");
    System.out.println("Current State: " +
originator.getState());
    originator.getStateFromMemento(careTaker.get(0));
    System.out.println("First saved State: " +
originator.getState());
    originator.getStateFromMemento(careTaker.get(1));
    System.out.println("Second saved State: " +
originator.getState());
  }
}
```

3. For each operation that changes the state of the Inventory object create a command. Make the commands serializable. Now every time we perform an operation on an Inventory object, we can create a command, perform the command and save the command to disk. This way we will have a history of all the operations. If our program were to crash we can recover the last state by first loading the last memento and then replaying all the commands done since the last memento was created.

4. Create a decorator for Inventory objects. For every operation that changes the Inventory object's state the decorator will create the command, perform the command and save the command to a file.

Grading Criteria

Presentation	25%
Demonstration	25%
Design	25%
Overall	25%