COMP-6231

Distributed Room Reservation System (DRRS) using Java IDL (CORBA)

Version – 1.0

by

Basant Singh
Manohar Gunturu
Pallavi Kumar
Vinita Singh

TABLE OF CONTENTS

1. Software Detailed Design

Objective:

To build and develop software as CORBA Distributed Room Reservation System (DRRS) to manage the room slot booking across three campus of the university . The main objective is to build the a software system which should be capable to tolerate a single software (non-malicious Byzantine) failure and be highly available under a single process crash failure using active replication.

- UDP Reliability is achieved through sequence numbers by sending negative acknowledgements(**NACKS**).
- In order to achieve concurrency and better performance thread pools are used.
- Locks are used while a thread is entering the critical section to ensure data consistency without any data race situations.
- Server crash is identified by FE by using timeout at 500ms, then FE says about this to RM, and RM rechecks whether replica is crashed or not by sending a ping message and then restarts the the replica instead of blindly restarting it, because we can get 500ms delay in udp due to the network congestion.
- Test cases are written to check UDP reliability, total ordering and fault tolerance.

1.1. Database Design

1.1.1. Physical Data Model

Implementation
Image 2.1.1.1

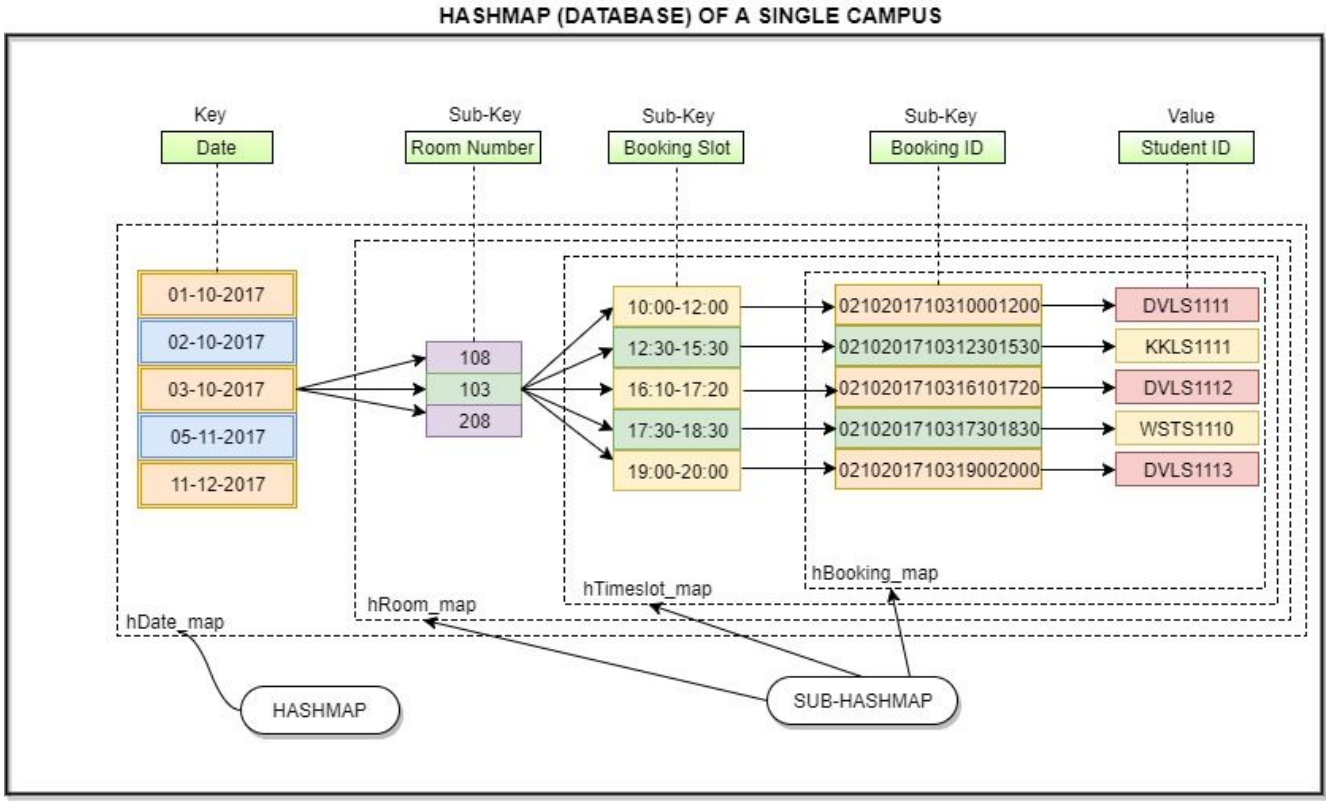
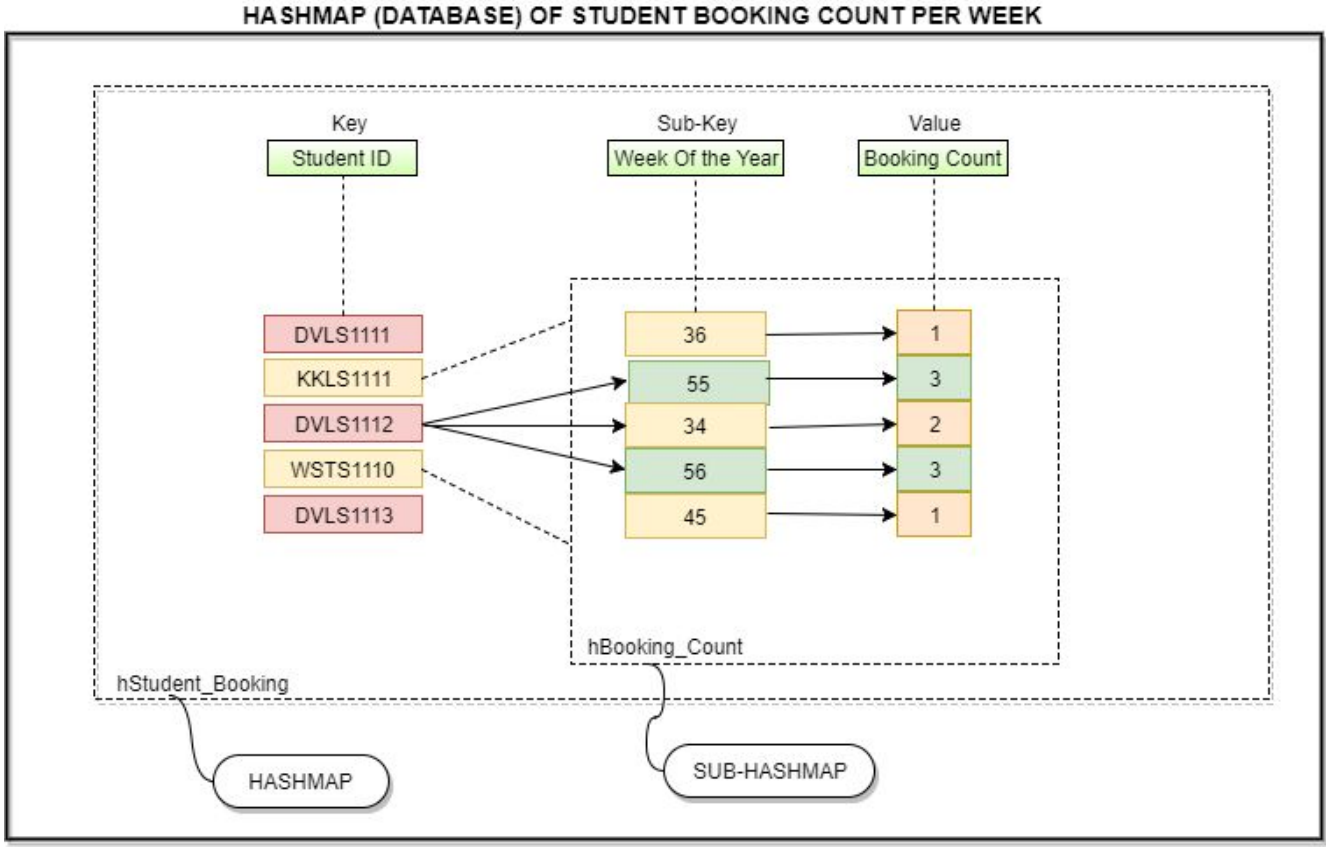


Image 2.1.1.2



As shown in above image , two nested hashmap database has been created

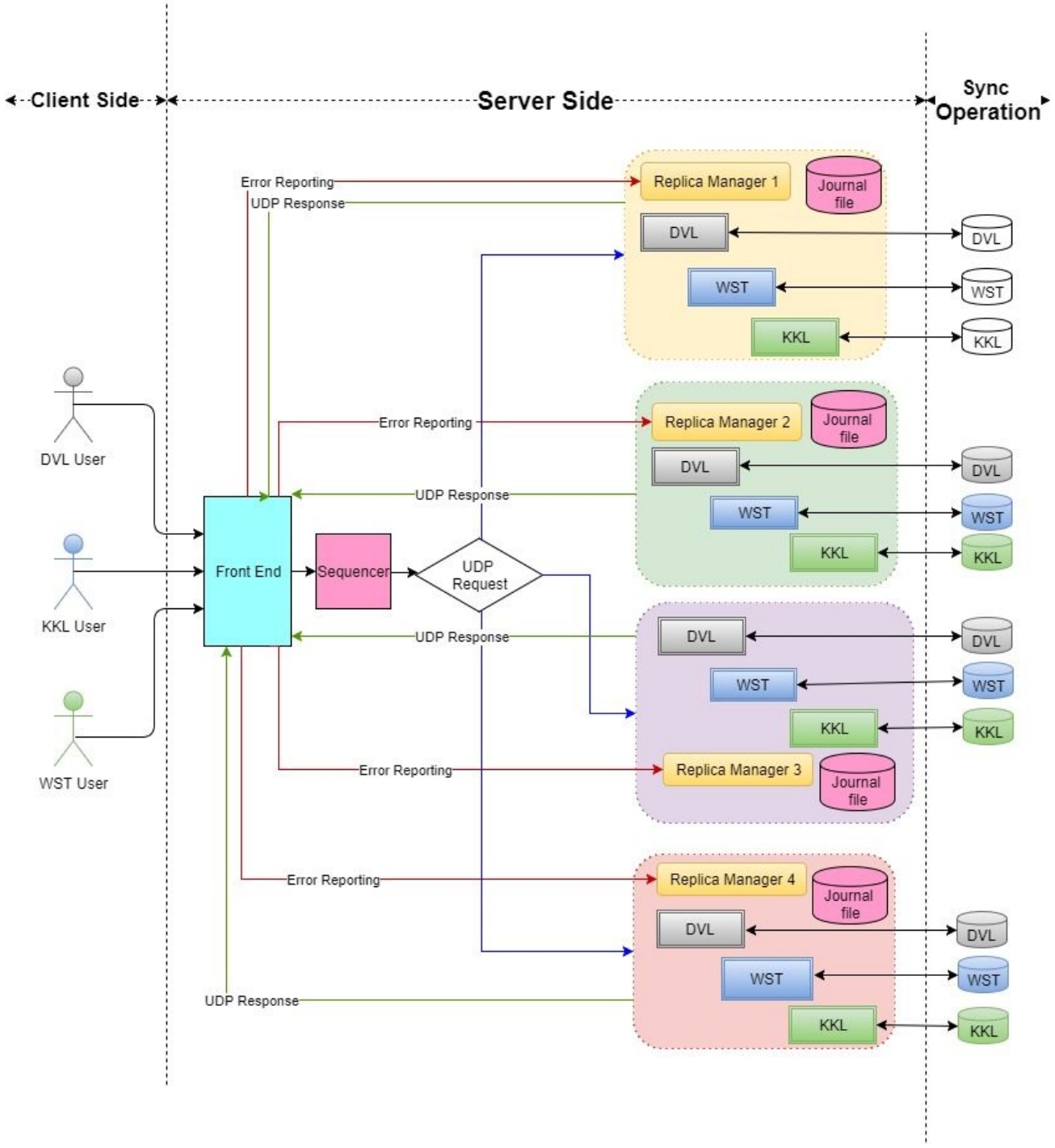
1. Hashmap for the room record for a campus:
    - Hashmap Name: **hDate_map(image 2.1.1.1)**, it is the nested hashmap with 3 sub-hashmap as mentioned below.
        - Key: Date , it is the booking date for which the room record is created
        - Value: hRoom_map , a nested map whose Room Number is the key
    - Sub-Hashmap Name : **hRoom_map**
        - Key: Room Number, is the room number for which room record is created
        - Value: hTimeslot_map , a nested hashmap whose booking timeslot is the key
    - Sub-Hashmap Name: **hTimeslot_map**
        - Key: Timeslot , is the time slot for which room record is created
        - Value: hBooking_map, a hashmap for which BookingID is the key
    - Sub-Hashmap Name: **hBooking_map**
        - Key: BookingID, a unique identifier which can be served as booking ID.
        - Value: StudentID, the ID of the students who books the room time slot.
2. Hashmap for the booking count for the student per week

    - Hashmap Name: **hStudent_Booking(image 2.1.1.2),** it is nested hashmap with one sub-hashmap as mentioned below
        - Key: Student ID, it is the student's ID using which students logs in to book room.
        - Value: **hBooking_Count**, a sub-hashmap records booking count per week per student.
    - Sub-Hashmap Name: **hBooking_Coun**t
        - Key: Week of the year, it is the week of the year for the date booking is done
        - Value: Booking count, it is the count of booking for the week by the student

1.2. Implementation Design

1.2.1.1. Design Approach

Overall Design Architecture



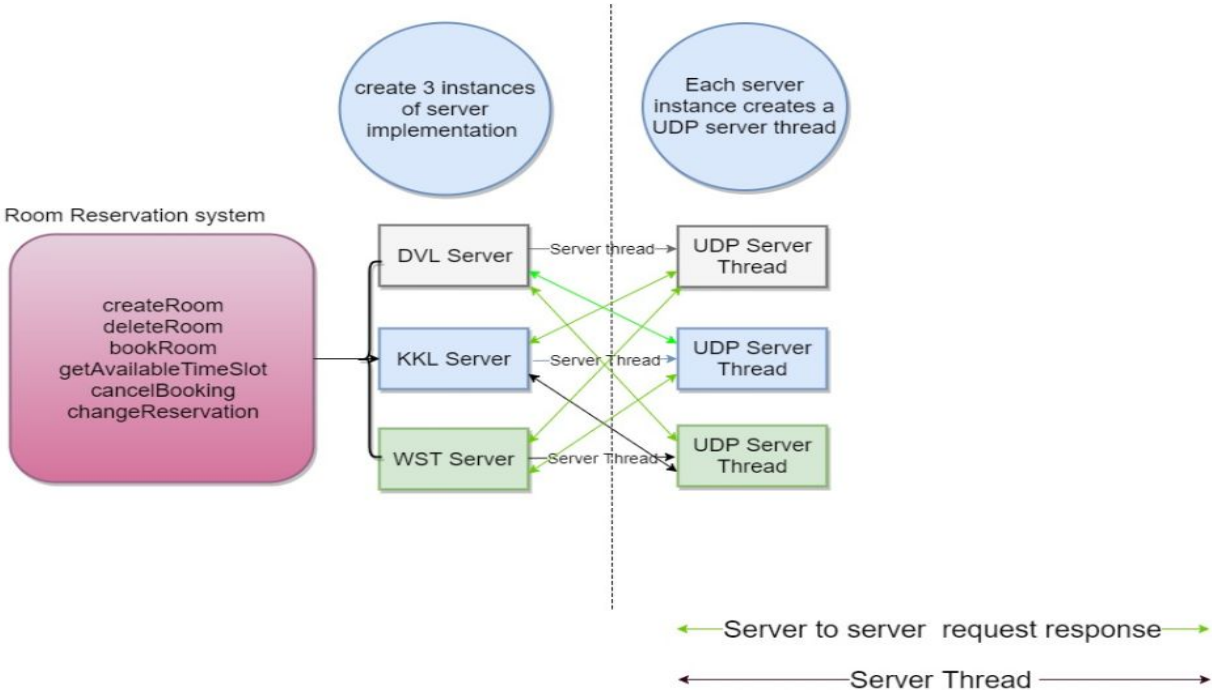
Overall workflow

- Front End will register CORBA remote reference with Naming Service
- Client will send request to Front End using CORBA remote invocation
- With the help of sequencer FE will generate unique request id and multicast it to all four replicas
- Sequencer is responsible to maintain total ordering
- Replicas will buffer each request and make sure to execute in same order received by checking sequence ID's
- Server implementations will process the request and send the result back to FE
- Replica will also buffer some last processed request ID/Sequence ID to avoid any duplicate request processing.
- In order to ensure UDP reliability a replica checks sequence ID every time it receives a request, if it finds any message is lost it sends a negative acknowledgment(**NACK**) in order to receive the missed message.
- On receiving response from all replicas. FE will do following
    - in case of any inconsistency in result, FE will repost error to all replica managers with request ID and replica ID.
    - If FE does not receive response from replica for a reasonable time then it reports crash to replica manager.
    - FE will respond back to client based with most appropriate answer, which is nothing but majority response given by replicas(by having the correct servers outvote the failed servers).
- Replica Manager will declare a replica as corrupted and initializes new replica instance
    - if it receives consecutive three error reports from FE or unresponsive for the same replica
    - And backup is done from previously logged files.

Replica failure recovery

- In case of failure replica will keep all new requests in the buffer for that replica and stop executing them.
- When RM restarts the replica, it will parse through the journal to create required hashmap
- And replica will complete all pending requests in buffer
- On catching up all pending requests, new replica will start operating smoothly

Server Architecture



Components

- **Front End**:

  The frontend is a corba object takes requests from the client. It identifies which functionality is to be invoked and for which campus.It forwards these requests to the sequencer using the sequencer send method.
  Furthermore, the frontend receives the response from the sequencer using its receive method and then depending on the response it invokes the replica manager or forwards the response to the client. If it finds any process crash or non malicious byzantin error it notifies the RM's

- **Replica Manage**r :

  The replica manager decides which replica to call for execution.In case of byzantine failure or process crash ,the replica manager verifies the failure by running that replica. Once verified the manager decides the next replica which should be called for successful result.Below are the methods defined in RM.
  - Creation of listener
  - Creation of instance of replica
  - Method (complain) defines to be invoked by FE to pass the message for Byzantine error or server failure

Replicas:

A replica is the system where the entire logic for various functionalities resides. It will execute the requests as received by the frontend .
In our implementation we have 4 replicas with its own logic but serving similar functionalities.

Design Techniques.

Concurrency

- The Servers creates new threads to communicate with each other.
- UDP server creates a new worker thread for each coming request and delegates the service request

Synchronization

- Request access to data set is synchronized.Hence only one thread can get hold on to it .

Logging

- Each server (DVL, KKL, WST) logs will be saved in below mentioned path
  Logs/Server/
- Each client logs will be saved in below mentioned path
  Logs/ Client

Reliability:

Journalling

It is the technique to recreate the data structure in the case of system failure, when entire data structure is lost and to synch with other replicas , it looks for the the all previous request to recreate the data structure.

As hashmap remains in memory any server crash will result into whole data lost . To recreate the data structure, we are recording every events coming from FE in a journal logger folder as mentioned below

- Data_Backup/Journal / DVL (or KKL or WST)

When FE send a flag to RM about the replica unresponsiveness, it invokes the method “Recreate_hashMap” for the replica. it process the all the request which was sent by FE in the same order to create the hashmap structure as required.

Serialization

Counter objects are being used to generate unique record ID. Any inconsistency in that operation may cause duplicate record ID and inconsistent data.
So to counter this problem, we have used , **Serialization and Deserialization**. **Serialization** is a process of converting an object into a sequence of bytes which can be persisted to a disk

or can be sent through streams. The reverse process of creating object from sequence of bytes is called **deserialization**.

Test Scenarios

Example test case to book a room:

```
for(int i =0;i<4;i++){
    int o = i;
    new Thread(new Runnable() {
        @Override
        public void run() {
            StudentInterface st = getRemotereference("dv1");
            System.out.println(st.bookRoom("dv1", "dv1s4567", "34", "23-10-2017", "12:30 - 13:30"));
        }
    }).run();
}
```

so the first successful and remaining are failed.

```
<terminated> TestSlots [Java Application] C:\Program Files\Java\j
dv1_23-10-2017_34_1
fail
fail
fail
```

Concurrency test case:

Type of operation	Time Taken	Number of Requests
createRoom	7 seconds	10,000
createRoom	40-45 seconds	100,000
getAvailableTimeSlot	17 - 19 seconds(Due to two extra UDP connections for every request)	10,000
changeReservation	11 - 12 seconds(as each request may need an extra UDP connection)	10,000

S.NO	Scenario	Expected Output	Actual output	PASS/FAIL
1	Give Room Details	Give Room details as the available record. Room will be booked if Room available.	Give Room details as the available record. Room will be booked if Room available.	PASS
2	Enter as Admin and select delete option. Provide the wrong Room no, you want to delete	System won't find a Room with this id. It will tell that no Room found.	System won't find a Room with this id. It will tell that no Room found.	Pass
3	Enter as Student and select option to transfer the room you want to transfer with wrong details.	System will display message that no booking exist for the given booking Id.	System will display message that no booking exist for the given booking Id.	Pass

Integration Testing:

The phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing.

Failure and recovery testing:

1. Launch Server Replica 1, 2 ,3 and 4
2. Send GetAvailblaRoomCount() request
3. Receive Record Count for all 3 Servers (DVL, KKL, WST) and show it in the client screen.
4. Shutdown Server Replica 1
5. Send GetAvailableRoomCount() request
6. Receive same Record Count for all 3 Servers (DVL, KKL, WST) and show it in th client screen. Server will restart and gets back to it’s previous state.
7. Shutdown Server Replica 1
8. Send GetAvailableRoomCount() request

9. Receive same Record Count for all 3 Servers (DVL, KKL, WST) show it in the client screen. Server will restart and gets back to it's previous state.
10. Shutdown Server Replica 1,2,3
11. Send Get*AvailableRoomCount()* request
12. Receive same Record Count for all 3 Servers (DVL, KKL, WST) and show it in the client screen. Servers will restart and gets back to it's previous state.

Work done

Pallavi : Designed and implemented the front end (FE) which receives a client request as a CORBA invocation , forwards the request to the sequencer, receives the results from the replicas and sends a single correct result back to the client as soon as possible.The FE also informs all the RMs of a possibly failed replica that produced incorrect result.

Basant : Design and implemented the replica manager (RM) which creates and initializes the actively replicated server subsystem. The RM also implements the failure detection and recovery for both types of failures.

Manohar : Design and implement a failure-free sequencer which receives a client request from the FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the four server replicas.

Vinita : Designed proper test cases for all possible error situations and implemented a client program to run these test cases

.

References

1. https://www.javatpoint.com/serialization-in-java
2. https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html