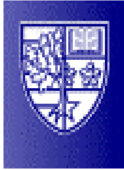
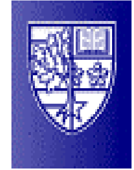
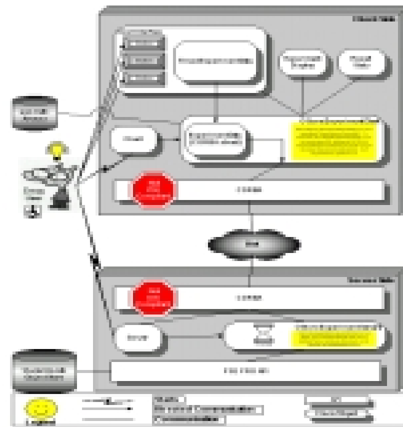


Heriot-Watt University
Edinburgh



Department of Computing & Electrical Engineering

Combined Coursework



Dr. Albert Burger

Group ?	Fotios Bassayiannis

"I interpret the word 'Query' in the more general sense and I thus use the get() method too!"

--Fotios

"...and they will perish, polishing spreadsheets and hunting windows, while we code our own Operating Systems."

--Digital Apocalypse

INTRODUCTION **4**
DELIVERABLES: 4
ABOUT THE SYSTEM **5**
OVERALL SYSTEM ARCHITECTURE **6**
OVERALL SYSTEM ARCHITECTURE (BUILT IN VISIO PRO 5) 7
OVERALL SYSTEM ARCHITECTURE IN UML (BUILT IN RATIONAL ROSE 98I) 8
FULL SYSTEM SCREENSHOT 9
THE CLIENT LAYER **10**
OBSERVER PATTERN UML CLASS DIAGRAM 11
CLIENT LAYER JAVA SOURCE CODE 12
Client.java *12*
OstoreExperimentClient.java *13*
UserInterface.java *24*
ExperimentDisplay.java *28*
ResultStats.java *31*
TimedExperimentData.java *33*
MIDDLEWARE (CORBA AND JDBC) **36**
Experiment.idl *36*
THE SERVER LAYER **38**
DESCRIPTION 38
SERVER SIDE SOURCE CODE 39
OstoreExperimentServer.java *39*
OstoreExperimentServant.java *40*
Student.java *50*
THE SYSTEM'S DATABASES **52**
THE ACCESS DATABASE (RELATIONAL) 52
THE OBJECTSTORE DATABASE (OBJECT ORIENTED) 53
OSVector Database Schema *53*
UML 53
ShowDB for .odb file with OSVector root with 3 added objects 54
OSHashMap Database Schema *57*
UML 57
ShowDB for .odb file with OSHashMap root with 3 added objects 58
OSTreeSet Database Schema *61*
UML 61
ShowDB for .odb file with OSTreeSet (and Index added) root with 3 added objects 62
RATIONALE OF EXPERIMENTS **65**
EXPERIMENT DESIGN 66
Collection Selection *66*
Experiment Design Prime Directives *70*
Mapping Experiment Requirements to System Requirements *71*
Integrity 71
Thouroughness & Precision 72
ObjectStore Database Design 72
EXPERIMENTS' ANALYSIS **74**
SECTION 1: COMPARING PICK POINT QUERIES 75
NoOpt *75*
HashingQuery *75*
HashingGet *76*
Indexing – None *76*
Indexing - ID *77*
End of Section 1 Conclusions *77*

SECTION 2: COMPARING SELECT (NON-PICK) POINT QUERIES 78
NoOpt *78*
HashingQuery *78*
Indexing – None *79*
Indexing – ID *79*
End of Section 2 Conclusions *80*
SECTION 3: COMPARING SELECT RANGE QUERIES 81
NoOpt *81*
HashingQuery *81*
Indexing – None *81*
Indexing – ID *81*
End of Section 3 Conclusions *82*
SECTION 4: PUTTING IT ALL TOGETHER 83
Pick Point Query Speed *83*
Select (non-Pick) Point Query Speed *83*
Select Range Query Speed *84*
OS Database files' Sizes & Population Times *84*
Comments on the Sizes & Population Times *84*
Some Extra General Conclusions & Comments *85*

APPENDIX 86

Introduction

This paper will present a distributed ObjectStore performance metrics and analysis tool that was developed in JDK 1.2.2 using the ODI (ObjectStore) and OMG (CORBA) APIs. A series of proposed ObjectStore Collection-Query performance experiments will also be presented and their stored (within an Access Relational database) timing results analysed.

Throughout the whole process of appropriately designing the system, performing the experiments, and deciphering the produced results, a feeling of insecurity, pertaining to the used experimental method and the specific software implementation of it, was prevalent. Such feelings are of course (to some degree), kind of inevitable when it comes to performing experiments on what is - for you - uncharted territory, but except giving you a hard time they also serve a benevolent purpose: They motivate hard work and fuel the desire to work hard and exhaustively explore the involved technologies' theory, potential, breadth and complexity, to the best of your abilities. To this end and the end of producing what I perceive as a beautiful multi-tiered software system, using modern OO technologies, this assignment has helped a lot.

Deliverables:

Overall Architecture Diagram (in terms of layers of class packages) in UML	**In this paper and on the accompanying CD**
Observer Pattern UML Class Diagram	**In this paper and on the accompanying CD**
Observer Pattern UML Sequence Diagram	**On CD only, due to the size of the diagram. (i.e. it would not fit on a page and be legible)**
JAVA Source Code	**All System source code is included in this paper and on CD**
Screenshots	**In this paper and on CD**
IDL specification	**In this paper and on CD**
Description of experiments	**In this paper and on CD**
Description and analysis of the experiments'results.	**In this paper and on CD**
Description of object-oriented databases	**In this paper and on CD**
UML describing OO database schemas	**In this paper and on CD**
Description of the Access database	**In this paper and on CD**
ER Diagram of the Access database	**In this paper and on CD**

Also,

- The **"System User's Manual"** can be found in the project's blue delivery folder in "hardcopy" form, and on CD.
- All UML related information can be found in the "Rose" folder within the Project's CD-ROM. Both the .mdl file and a Web export of it are included.
- The **"ReadMe.txt"** file on the root folder of the supplied Project CD describes the contents of the CD.
- **Please bear in mind that the only asked deliverable that is not included in this paper and can only be found in the supplied CD-ROM (within the Rose .mdl file), is the UML Sequence diagram of the Observer Pattern. This is because the diagram is long and would not fit nicely on paper.**

HAVE FUN!

About the System

This is a 3-tier system consisting of:

- A client application (JAVA using the OMG API)
- Middleware. (CORBA)
- A server application (JAVA using the OMG and ODI APIs)
- An ObjectStore database
- A Microsoft Access database

Communication between the client and server sides is done through the JDK 1.2.2 implementation of CORBA and the Experiment Configurations and Results are stored within an Access database that is manipulated by the Client application.

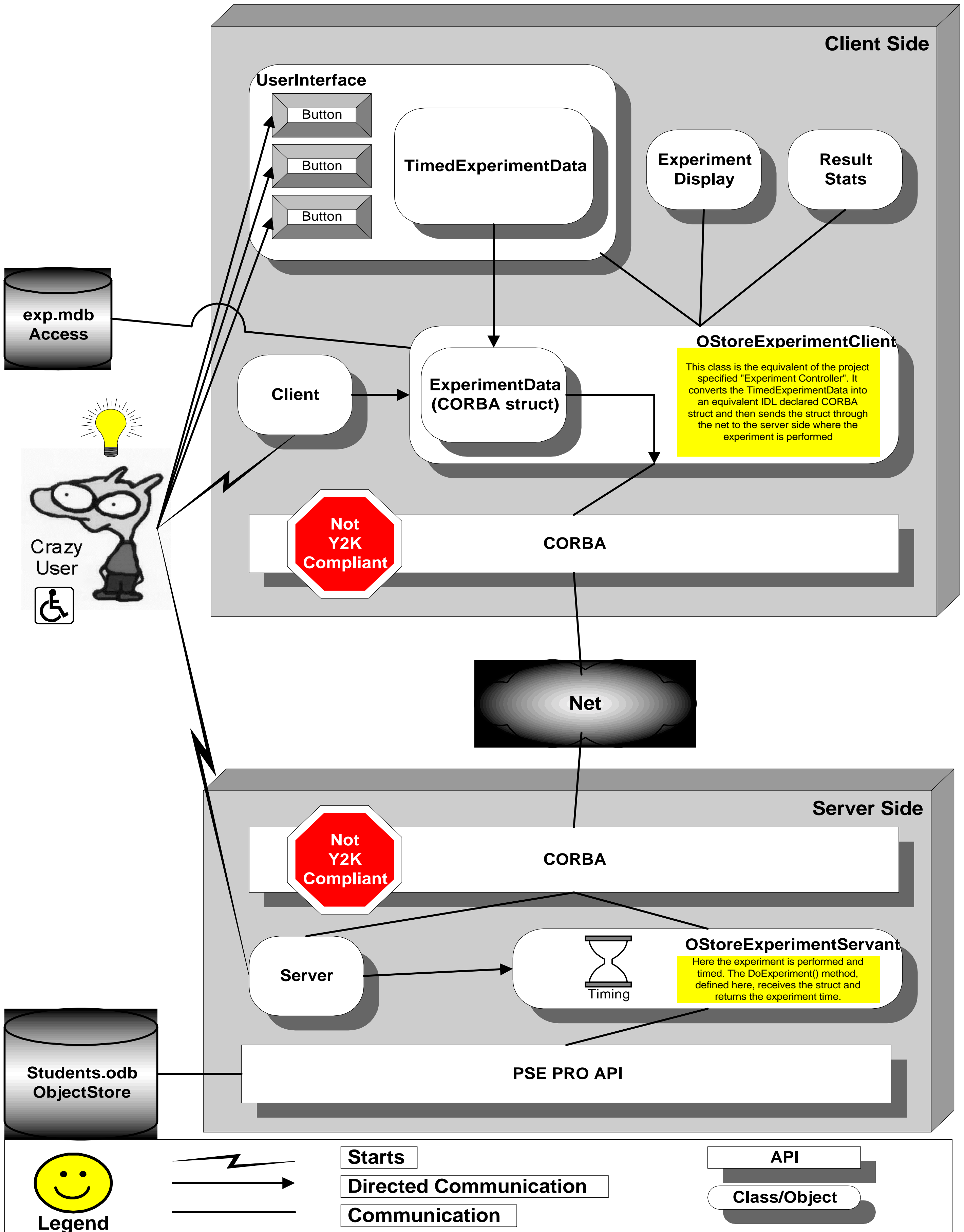
For information on how to build or run the System please consult the "System User's Manual" that comes as a separate document within this Project's blue folder. It would also be very helpful if you read the "Readme.txt" files that reside on each major directory of the project's CD-ROM (also found within the Project's blue folder).

Overall System Architecture

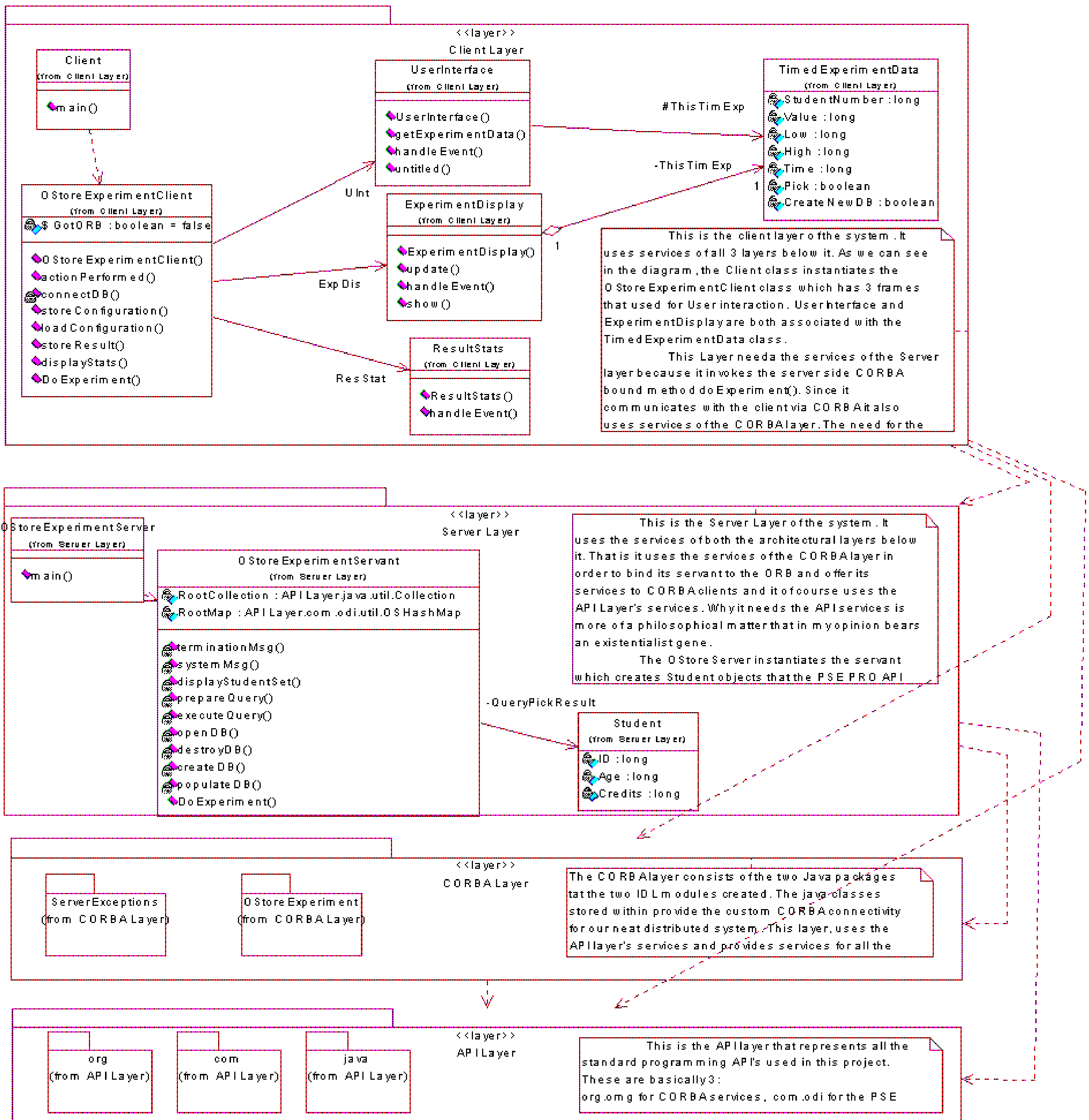
The following pages present:

1. A figure, depicting in an intuitive format, the Overall System Architecture.
2. The Overall System Architecture in UML notation.
3. Screenshots of the system in full deployment

Overall System Architecture (Built in VISIO PRO 5)



Overall System Architecture in UML (Built in Rational Rose 98i)



Full System Screenshot

User Interface

Experiment ID: 1
Student Field to Search: ID
Optimization Type: NoOpt
Student Field to Optimize: None
Number of Student Objects to create: 1000
Field Value to find: 0
Low Value for Range Query: -1
High Value for Range Query: -1
Use Pick Query: false
Create New DB: true

Load Configuration | Store Configuration
Execute | Store Result
Experiment Time | Result Statistics
Exit

Experiment Display

Experiment ID: 1
Experiment Status: Complete
Experiment Result: 1 secs & 210 ms

ostore notes 2.txt
idl specificat...
Osdd hlp

JAVA

```
Microsoft(R) Windows 98
   (C)Copyright Microsoft Corp 1981-1998.

G:\B\Project\System>client

G:\B\Project\System>java Client -ORBInitialPort 1050
Connection to DB established!
SQL Command being issued: SELECT * FROM ExperimentConfigurations
tID = 1
Experiment Configuration Data retrieved successfully!
ORB created and initialised!
Got root naming context!
Object reference resolved in naming!
Calling the DoExperiment method of the servant...
Beginning Experiment...
Experiment Time: 1210
```

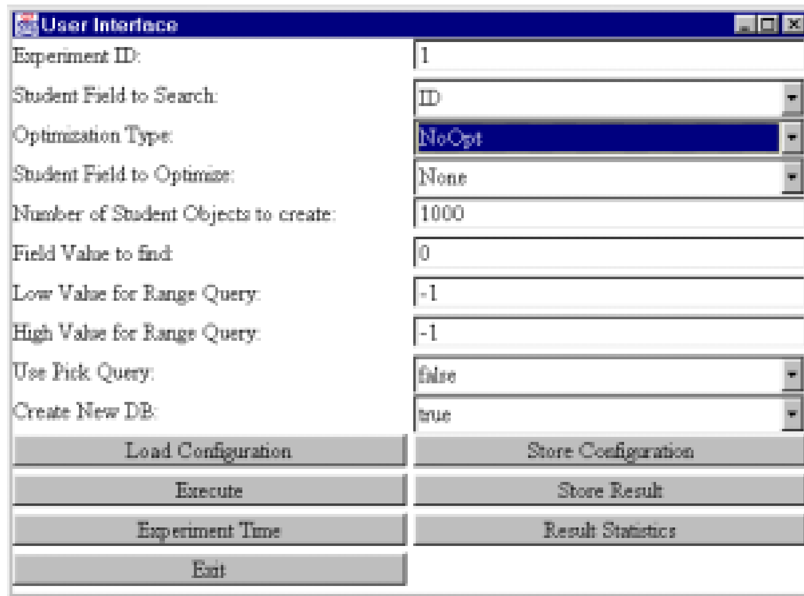
JAVA

```
G:\B\Project\System>server

G:\B\Project\System>java OStoreExperimentServer -ORBInitialPort 1050
Servant Created and registered with ORB!
Got root naming context
Object reference bound in naming
Waiting for client invocations...
Query prepared!
Value: 0 LH values: -1 -1
Database Destroyed!
CreateDB Transaction began, using NoOpt
OSVector Root Created!
CreateDB Transaction ended successfully, using NoOpt
Database Created!
Populating with 1000 Student objects...
(Performing Commit every 1000 objects)
Done in: 0seconds and 160ms
(Not counting time Commits take!)

executeQuery: Root acquired!
executeQuery: Timing began!
executeQuery: Timing Stopped!
EXPERIMENT TIME: 1210
########################################
```

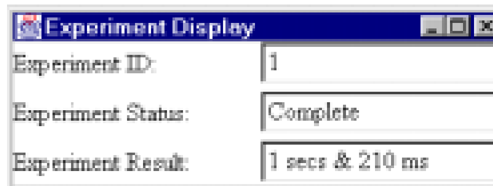
The Client Layer

The Client side consists of the:

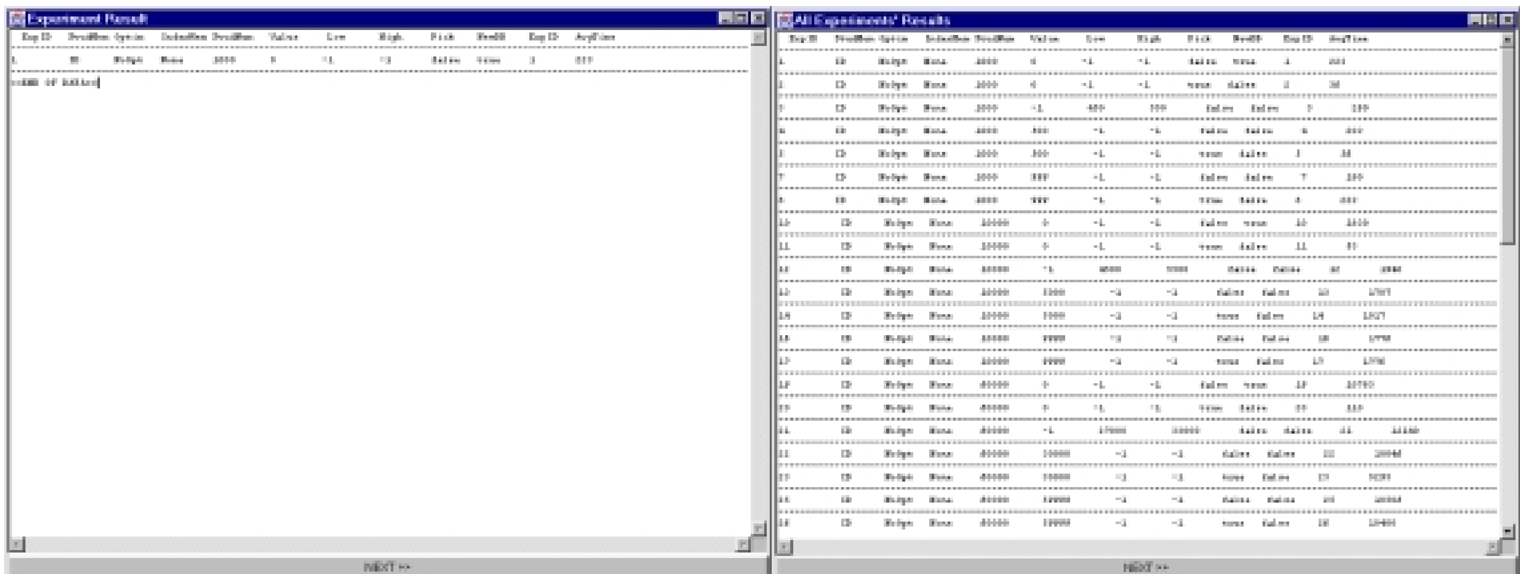
- **User Interface:** accepts user input for experiment specification and data management.



- **Experiment Display:** Displays current experiment number, status and result (once a result is available).



- **Result Statistics Window:** may either display the result of one experiment or of all experiments stored in the Access database.

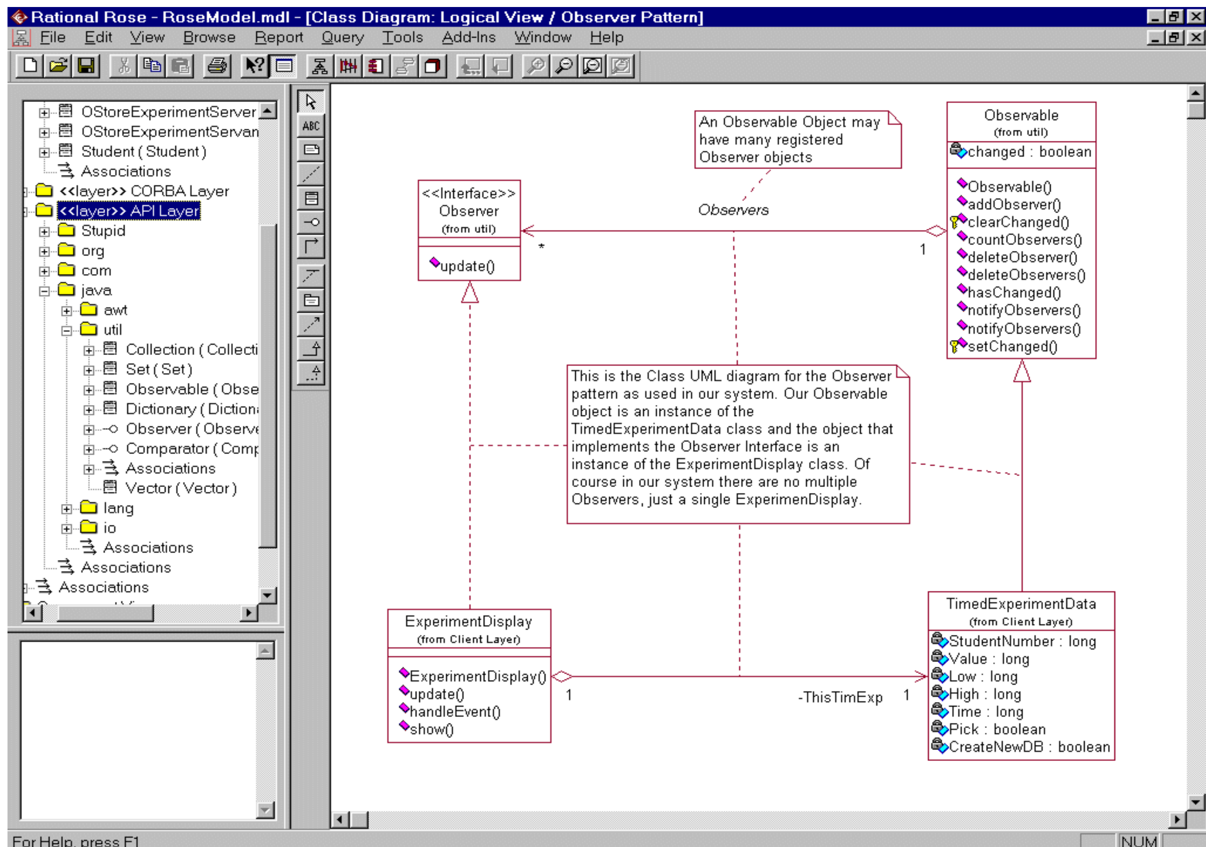


All Client Side processes are spawned from a single DOS session whose window looks like this (the user only has to type “client” and hit <enter>, to start the client side):

```
Object reference resolved in naming!
Calling the DoExperiment method of the servant...
Beginning Experiment...
Experiment Time: 1210
Connection to DB established!
SQL Command being issued: SELECT * FROM ExperimentConfigurations,(SELECT Experim
entID,AVG(ExperimentTime) FROM ExperimentResults GROUP BY ExperimentID)  WHERE (
(ExperimentConfigurations.ExperimentID = ExperimentResults.ExperimentID) AND (Ex
perimentResults.ExperimentID = 1));
Database Query Successfull!
Connection to DB established!
SQL Command being issued: SELECT * FROM ExperimentConfigurations,(SELECT Experim
entID,AVG(ExperimentTime) FROM ExperimentResults GROUP BY ExperimentID) WHERE Ex
perimentConfigurations.ExperimentID = ExperimentResults.ExperimentID;
Connection to DB established!
SQL Command being issued: SELECT * FROM ExperimentConfigurations,(SELECT Experim
entID,AVG(ExperimentTime) FROM ExperimentResults GROUP BY ExperimentID)  WHERE (
(ExperimentConfigurations.ExperimentID = ExperimentResults.ExperimentID) AND (Ex
perimentResults.ExperimentID = 1));
Database Query Successfull!
Connection to DB established!
SQL Command being issued: SELECT * FROM ExperimentConfigurations,(SELECT Experim
entID,AVG(ExperimentTime) FROM ExperimentResults GROUP BY ExperimentID) WHERE Ex
perimentConfigurations.ExperimentID = ExperimentResults.ExperimentID;
```

Observer Pattern UML Class Diagram

Observer Pattern Sequence Diagram can be found in the CD.



Client Layer JAVA Source Code

Client.java

```
/**
This little class is used as the launchpad for the
OStoreExperimentClient client-side object as it contains
the main() method. The reason I have dispatched the main()
method to this class by itself, is so that I may avoid
the complications that come from having a statically
declared-defined method (like the main() which has to
of course be static) within your class.
*/
public class Client
{
  public static void main(String args[])
  {
    OStoreExperimentClient Client;

    //Create the OStoreExperimentClient (which is the CORBA
    //client actually) and pass it the main() methods
    //command line arguments.
    Client = new OStoreExperimentClient(args);
  }
}
```

OstoreExperimentClient.java

```
import OStoreExperiment.*;
import ServerExceptions.*;

import org.omg.CosNaming.*;
import org.omg.CORBA.*;

import java.awt.event.*;

import java.sql.*;

/*************************************************************************/
/**
This is basic class of the client side of the system. It is what
the assignment handout calls the"Experiment Controller".
It contains the JDBC relevant methods and it also connects
to the CORBA ORB and invokes the DoExperiment() method on the
server side. It is capable of spawning 3 different kinds of frames:
UserInterface, ExperimentDisplay, ResultStats. The first two
are created right away when the client is started. They can
also be killed and they will be regenerated when needed. Killing
the UserInterface Frame object also kills the client JVM process.
OStoreExperimentClient implements the ActionListener interface
so that it can listen to button events generated within the User
Interface by the crazy user
*/

public class OStoreExperimentClient implements ActionListener
{
  //The client side frames
  UserInterface       UInt;
  ExperimentDisplay   ExpDis;
  ResultStats         ResStat;

  //This string will hold the command line args from the Client
  //object instance which has the main() method. As we can see in the
  //Client object's definition, the Client's main() method is
  //used as the "launchpad" for the OStorExperimentClient object.
  //Args specified there (e.g. -ORBInitialPort 1050) are then transferred
  //into this array.
  String args[];

  //The URL of the ODBC database
  //(e.g.Access, or any DB implementing an ODBC driver)
  private String       DBurl = "jdbc:odbc:Experiments";

  //A Connection object to the ODBC database
  private Connection DBCon = null;

  //JDBC statements that will hold our SQL queries
  private Statement UStatement = null, QStatement = null;

  //This ResultSet var will hold the
  //results of our SQL queries
  ResultSet Stats = null;

  //This boolean will be set to false
  //when the ResultStats window still
  //has data to display from the previous
  //executed SQL statement. It will be set
  //back to true when all data has been
  //displayed, so that the next call to the
  //ResultStats window, will invoke a new
  //SQL query, for the next set of data.
  boolean NewStats = true;
```

```
  //This var specifies whether an ORB connection
  //has been established. While testing the system
  //it has come to my attention that restablishing an
  //ORB connection each time a new client request came in
  //(even if the previous client side ORB object has been
  //shutdown before) was actually depleting my system's
  //resources (memory?) in about 8 subsequent CORBA calls.
  //Creating the ORB, connecting to it and establishing
  //initial references (and holding them) only once at
  //start solved this problem. So, this boolean var
  //is used to make sure that ORB creation and connection
  //is done only once. What is done repeatedly of course
  //is the resolution of the servant's ObjRef.
  private static boolean GotORB = false;

  //These vars keep the ORB and NamingContext objects alive.
  //In case we wanted more than one OStoreExperimentClient
  //to be running (not possible in this version of the program)
  //the same ORB would be used (since the vars are declared as
  //static) by that instance too, resulting in memory savings
  static ORB orb = null;
  static org.omg.CORBA.Object objRef = null;
  static NamingContext ncRef = null;

/****************************************************************************/
  //OStoreExperimentClient constructor
  //The string array passed to it is the command line
  //arguments from th main() method of the Client class
  //which is the class that instantiates
  //the OStoreExperimentClient
  public OStoreExperimentClient(String args[])
  {

    //Receive the user
    //supplied args
    //(from Client.java)
    this.args = args;


    //Create the UserInterface Frame object
    //The Uint creation, creates in its turn
    //(among other things) a TimedExperimentData
    //Object which basically contains Experiment
    //configuration data AND experiment time
    //(which is initially -1)
    UInt = new UserInterface();

    //Create the ExperimentDisplay Frame object
    //and pass it the reference of the TimedExperimentData
    //object, implicitly created by the UserInterface object
    ExpDis = new ExperimentDisplay(UInt.ThisTimExp);

    //Add to UInt's ThisTimExp instance's list of observers
    //the created instance of the experimentDisplay (ExpDis).
    //From now on the ExpDis object will be notified of changes
    //in the ThisTimExp object
    UInt.ThisTimExp.addObserver(ExpDis);

    //Specify which events (coming from the
    //UserInterface's buutons) this instance of the
    //OStoreExperimentClient will be monitoring.
    //this isnstance is registered as a listener for
    //the 7 UInt buttons
    UInt.Execute.addActionListener(this);
    UInt.Exit.addActionListener(this);
    UInt.StoreConfig.addActionListener(this);
    UInt.Load.addActionListener(this);
    UInt.StoreResult.addActionListener(this);
    UInt.ResultStats.addActionListener(this);
    UInt.ExpTime.addActionListener(this);
```

```
    //Add an action listener to the TextField
    //used for holding the experiment number.
    //This is for quickly loading an experiment
    //configuration by pressing the enter button
    UInt.ExperimentID_F.addActionListener(this);

  }
/****************************************************************************/


/****************************************************************************/
  //This method is called every time an action occurs in one
  //of the monitored UserInterface buttons. According to which
  //button generated the event an appropriate method is called.
  public void actionPerformed(ActionEvent ae)
  {

    //If the execute button is pressed...
    if (ae.getSource() == UInt.Execute)
    {
      //Get the user input data from the UserInterface instance.
      //By calling the getExperimentData() method all user input
      //is parsed from the UInt TextFields and Choice components
      //Into the UInt's instance of a TimedExperimentData object.
      //ThisTimExp object then represents the experiment to be
      //performed as specified by the user.
      UInt.getExperimentData();

      //Transfer all data from the TimedExperimentData java object
      //to the ExperimentData CORBA struct
      ExperimentData ThisExp =
        new ExperimentData( UInt.ThisTimExp.getStudentMember(),
                            UInt.ThisTimExp.getOptimizationType(),
                            UInt.ThisTimExp.getIndexableHashableMember(),

                            UInt.ThisTimExp.getStudentNumber(),
                            UInt.ThisTimExp.getValue(),
                            UInt.ThisTimExp.getLow(),
                            UInt.ThisTimExp.getHigh(),
                            UInt.ThisTimExp.getPick(),
                            UInt.ThisTimExp.getCreateNewDB() );

      //After the CORBA struct is ready,
      //Connect to the ORB in order to deliver it
      //and get the corresponding timing result
      try
      {

        //If te connection to the ORB is not
        //already established, then...
        if (!GotORB)
        {
          // create and initialize the ORB
          orb = ORB.init(args, null);
          System.out.println("ORB created and initialised!");

          // get the root naming context
          objRef = orb.resolve_initial_references("NameService");
          ncRef = NamingContextHelper.narrow(objRef);
          System.out.println("Got root naming context!");

          //Change the status of the GotORB flag
          GotORB = true;
        }

        //The following will be executed not only the first time,
        //but each time the "Execute" button is pressed...

        // resolve the Object Reference in Naming
        NameComponent nc = new NameComponent("ExperimentServant", "");
        NameComponent path[] = {nc};
        Experiment OSERef = ExperimentHelper.narrow(ncRef.resolve(path));
        System.out.println("Object reference resolved in naming!");
```

```
    // call the OStoreExperimentServant DoExperiment method
    System.out.println("Calling the DoExperiment " +
                       "method of the servant...");

    System.out.println("Beginning Experiment...");

    //Show the ExperimentDisplay Frame
    //in case the user has closed it
    ExpDis.show();

    //Get the ExperimentID number from the UInt and display
    //it in the appropriate field on the ExperimentDisplay
    //Frame instance
    ExpDis.ExperimentID_F.setText( UInt.ExperimentID_F.getText() );

    //Take care of the rest two TextFields of the ExpDis
    ExpDis.ExperimentStatus_F.setText("Running...");
    ExpDis.ExperimentResult_F.setText("n/a");

    //Call the OStoreExperimentServant DoExperiment method
    UInt.ThisTimExp.setTime(OSERef.DoExperiment(ThisExp));

    //Display the Experiment Time on the client side textual (DOS)
    //window. (Mainly for debugging purposes)
    System.out.println("Experiment Time: " + UInt.ThisTimExp.getTime());

  }

  //Catch thrown on the server side, CORBA ServerExceptions.AnyException
  //objects, and display the error message within them
  catch (AnyException a)
  {
    System.out.println("Servant Exception: " + a.Msg);
  }

  //Catch any other exceptions thrown by the OStoreExperimentClient
  //instance itself, and differentiate them from the ServerExceptions
  //by displaying the error together with the source of it (i.e. CLIENT)
  catch (Exception e)
  {
   System.out.println("CORBA CLIENT ERROR : " + e) ;

   //Print out a dump of the stack
   //for debugging purposes
   e.printStackTrace(System.out);
  }

}

//If the Exit button is pressed, on the UInt instance,
//then do the following...
else if (ae.getSource() == UInt.Exit)
{
  //Just terminate the client side JVM man!
  //I like to terminate with a 0 exit code
  //when all is cool, and the system is basically
  //chilling
  System.exit(0);
}

//If the StoreConfig button is pressed
//then do the following...
else if (ae.getSource() == UInt.StoreConfig )
{
  //Connect to the odbc DB via jdbc
  //and jdbc-odbc bridge
  connectDB();

  //Call the storeConfiguration()
  //method of this client.
  //This will store the presently
  //displayed on UInt, experiment
  //config data in the relational DB
  storeConfiguration();
}
```

```
    //If Load or enter (while the caret is in the
    //Experiment ID TextField) is pressed...
    else if ( ae.getSource() == UInt.Load ||
              ae.getSource() == UInt.ExperimentID_F )
    {
      connectDB();

      //Get the experiment ID number from the UInt and
      //pass it to the loadConfiguration() method which
      //will basically load the corresponding data from
      //the Access DB
      loadConfiguration( Long.parseLong(UInt.ExperimentID_F.getText()) );
    }

    //If StoreResult is pressed, store the timing result
    //presently stored within the TimedExperimentData object
    //in the ExperimentResults table of the Access DB
    else if (ae.getSource() == UInt.StoreResult)
    {
      connectDB();
      storeResult();
    }

    //If the ResultStats button is pressed,
    //display the stats frame with average times for
    //all stored experiment IDs
    else if ( ae.getSource() == UInt.ResultStats )

    {
      NewStats = true;

      connectDB();
      displayStats("all");
    }


    //If the ExpTime button is pressed
    //Display all different timings in
    //db for the specific experiment ID only
    else if (ae.getSource() == UInt.ExpTime)
    {
      NewStats = true;

      connectDB();
      displayStats("this");
    }

    else if ( ae.getSource() == ResStat.Next)
    {
      displayStats("all");
    }

  }//END of actionPerformed()
/*****************************************************************************/


/*****************************************************************************/
  //This method creates a Connection object
  //to the Access DataBase
  private void connectDB()
  {
        try
        {
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            }
            catch (ClassNotFoundException e)
            {
                System.out.println("DB driver loading failed!");
                System.exit(1);
            }

            try
            {
                //Get a connection object to the DB by supplying
                //the DB's url to the JDBC DriverManager class
                DBCon = DriverManager.getConnection(DBurl);
            }
```

```
               //Catch and display thrown SQL exceptions
               catch (SQLException e)
               {
               System.out.println("Connection to the DB under the specified URL: "
                                          + DBurl
                                          + " failed!");

                 //I have decided that it is better to NOT terminate
                 //the client side JVM, in case such an exception
                 //is thrown
                //System.exit(1);
               }

               //Signify success of the process
               System.out.println("Connection to DB established!");
        }
/******************************************************************************/



/******************************************************************************/
  //This method generates an INSERT SQL command in the form
  //of a string and feeds it to the Access DB in order
  //to store the currently displayed Experiment Configuration
  //in the database
  public void storeConfiguration()
  {

    try
    {
      String SQLString =
        "INSERT INTO ExperimentConfigurations VALUES ("+
        Long.parseLong(UInt.ExperimentID_F.getText())+",'"+
        UInt.StudentMember_C.getSelectedItem()+"','"+
        UInt.OptimizationType_C.getSelectedItem()+"','"+
        UInt.IndexableHashableMember_C.getSelectedItem()+"',"+
        Long.parseLong(UInt.StudentNumber_F.getText())+","+
        Long.parseLong(UInt.Value_F.getText())+","+
        Long.parseLong(UInt.Low_F.getText())+","+
        Long.parseLong(UInt.High_F.getText())+","+
        Boolean.valueOf(UInt.Pick_C.getSelectedItem()).booleanValue()+","+
        Boolean.valueOf(UInt.CreateNewDB_C.getSelectedItem()).booleanValue()
        +")"
        ;

      //Display the SQL command being issued
      //Mainly for debug purpose, but it also looks cool
      System.out.println("SQL Command being issued: " + SQLString);

      //Create te JDBC statement to hold the SQL command
      UStatement = DBCon.createStatement();

      //Execute this statement containing an UPDATE type of
      //SQL command (i.e. INSERT)
      UStatement.executeUpdate(SQLString);

      //Not really needed since the JDBC subsystem
      //is in autocommit mode by default,
      //But still just in case... it also feels good!
      DBCon.commit();

      //Let the poor user know the good news
      System.out.println("Database Updated Successfully!");

      //Release the resources associated
      //with this statement object
      UStatement.close();

    }

    //Catch'em SQL exceptions!
    catch (SQLException e)
    {
    //Display Error message
    System.out.println("SQL Insert Statement failed!" + e);
```

```
      //Just let the user decide if he wants to retry
      //the SQL command or terminate the client and
      //restart it. Thus the Client is not terminated
      //automayically in case of an SQL exception
      //System.exit(1);
    }


  }
/*****************************************************************************/



/*****************************************************************************/
  //This method loads an Experiment Configuration
  //from the ODBC database, given an ExperimentID
  public void loadConfiguration(long ExperimentID)
  {
    try
    {
      //Form an appropriate SQL command string
      String SQLString = "SELECT * " +
                         "FROM ExperimentConfigurations " +
                         "WHERE ExperimentID = " + ExperimentID;

      System.out.println("SQL Command being issued: " + SQLString);

      //Create a statement to hold the SQL query command
      QStatement = DBCon.createStatement();

      //Store results of the query in a ResultSet object
      ResultSet Config = QStatement.executeQuery(SQLString);

      //Advance the ResultSet cursor
      //to the first (and only, since
      //the ID is unique) row of data
      Config.next();

      //Get each column value and display it in the appropriate
      //UserInterface component
      UInt.StudentMember_C.select(Config.getString(2));
      UInt.OptimizationType_C.select(Config.getString(3));
      UInt.IndexableHashableMember_C.select(Config.getString(4));

      UInt.StudentNumber_F.setText( Long.toString(Config.getLong(5)) );
      UInt.Value_F.setText( Long.toString(Config.getLong(6)) );
      UInt.Low_F.setText( Long.toString(Config.getLong(7)) );
      UInt.High_F.setText( Long.toString(Config.getLong(8)) );

      UInt.Pick_C.select( ( new Boolean(Config.getBoolean(9)) ).toString() );
      UInt.CreateNewDB_C.select( ( new Boolean(Config.getBoolean(10)) ).toString() );

      //Tell the user that all is A OK!
      System.out.println("Experiment Configuration Data retrieved successfully!");

      //Close the JDBC statement
      QStatement.close();

    }

    //Catch the thrown SQL exceptions
    catch (SQLException e)
    {
      System.out.println("SQL Select Statement failed!" + e);

      //Let the user decide about that
      //System.exit(1);
    }

  }
/*****************************************************************************/
```

```
/***************************************************************************/
  //This method stores the experiment result currently
  //kept within the client side TimedExperimentData object
  //under the ExperimentID displayed on the ExperimentDisplay
  //instance.
  public void storeResult()
  {
    try
    {
      //Create the appropriate INSERT SQL command
      String SQLString = "INSERT INTO ExperimentResults VALUES ("        +
                         Long.parseLong(ExpDis.ExperimentID_F.getText()) +
                         ","                                             +
                         UInt.ThisTimExp.getTime()                       +
                         ")";

      System.out.println("SQL Command being issued: " + SQLString);

      //Create a statement to hold the SQL command
      UStatement = DBCon.createStatement();

      //Execute the statement
      UStatement.executeUpdate(SQLString);

      //Not really needed since the JDBC subsystem is in autocommit mode by default,
      //But still just in case... it also feels good!
      DBCon.commit();

      //Let the user know
      System.out.println("Database Updated Successfully!");

      //Release system resources associated with the statement object
      UStatement.close();

    }

    //Catch the thrown SQL exceptions
    catch (SQLException e)
    {

      //Display the Exception's error message
      System.out.println("SQL Result Insert Statement failed!" + e);

      //Let the user decide if he wants
      //to exit or try and reissue the command
      //System.exit(1);
    }

  }
/***************************************************************************/



/***************************************************************************/
  //This method receives a string and according to its value.
  //either displays cumulative results of all stored experiment
  //configurations and their average execution times, or
  //it displays all separate timings of a single experiment
  //configuration. To do this it opens a new frame with two
  //columns (TextAreas) and feeds it with the results of
  //appropriate SQL SELECT commands
  public void displayStats(String Yo)
  {

    //If the ResultStats Frame does
    //not exist, create it
    if (ResStat == null)
    {
      //Call the ResultStats constructor
      //and pass it the Yo String
      ResStat = new ResultStats(Yo);

      //Monitor the ResultStats Frame's
      //"Next" button for user input
      ResStat.Next.addActionListener(this);
    }
```

```
//If the ResultStats Frame already exists,
//from a previous invocation of the method,
//and new data has been requested,
//dispose of it, so a new one can be created.
//We could create a new one without disposing
//of the old one, but that would result in 2
//different instances of the same window being
//open at the same time, possibly displaying
//different data while bearing the same title,
//which is not good!
//If the window exists and new data has not
//been requested (i.e. the "Result Stats" button
//has not been pressed), then that means that the
//"Next" button, on the ResStats Frame has been
//pressed. In that case we do not dispose of the frame
//but we display the next batch of data on it,
//from the data previously put in the ResultSet
else if (ResStat != null && NewStats == true)
{
  ResStat.dispose();

  //Call the ResultStats cpnstructor
  //and pass it the Yo String
  ResStat = new ResultStats(Yo);

  //Monitor the ResultStats Frame's
  //"Next" button for user input
  ResStat.Next.addActionListener(this);
}

else
  //Flush the TextArea Contents
  ResStat.Screen_T.setText("");

//If new data has been requested
//(i.e. we re not still continuing the
//display of data in one screen
//each time), then create and
//execute the proper SQL queries...
if (NewStats == true) try
           {
  String SQLString = null;

  //If Yo is "all" then create a SELECT statement
  //that will return average times for all stored
  //experiment configurations
  if ( Yo.equals("all") )
    SQLString = "SELECT * "                                    +
                "FROM ExperimentConfigurations,"               +
                "(SELECT ExperimentID,AVG(ExperimentTime) "    +
                "FROM ExperimentResults GROUP BY ExperimentID) "  +
                "WHERE ExperimentConfigurations.ExperimentID = "  +
                "ExperimentResults.ExperimentID;"
                ;

  //If Yo is "this" then create a SELECT statement
  //that will return all time results stored
  //for a specific experiment configuration.
  //This specific Experiment configuration is
  //the one whose ID number is currently displayed
  //on the UserInterface
  else if ( Yo.equals("this") )
    SQLString = "SELECT * "                                      +
                "FROM ExperimentConfigurations,"                 +
                "(SELECT ExperimentID,AVG(ExperimentTime) "      +
                "FROM ExperimentResults GROUP BY ExperimentID)  "   +
                "WHERE ((ExperimentConfigurations.ExperimentID = "  +
                "ExperimentResults.ExperimentID) AND ("          +
                "ExperimentResults.ExperimentID = "              +
                 UInt.ExperimentID_F.getText() + "));"
                ;

  System.out.println("SQL Command being issued: " + SQLString);

                  //Create statement
                  QStatement = DBCon.createStatement();
```

```
                    //Execute statement and get results
                    //Stats is the ResultSet that will hold
                    //the results of our SQL queries
                    ResultSet Stats2 = QStatement.executeQuery(SQLString);
                    Stats = Stats2;

       }

   //Catch SQL Exceptions
   catch (SQLException e)
   {
     System.out.println("SQL Select Statement failed!" + e);

     //Let the user decide
     //System.exit(1);
   }


   //The TextArea Screen_T, as all JAVA awt TextAreas,
   //is of limited capacity. Thus we constrain it to
   //display up to 50 records each time. The count
   //variable is used for counting the displayed records.
   int count = 0;

   try
   {
     //Traverse the ResultSet and display up to
     //50 results, in an appropriate manner, on
     //the ResultStats Frame instance already created
     while (Stats.next())
     {

       ResStat.Screen_T.append( Long.toString(Stats.getLong(1)) +
                                "          ");
       ResStat.Screen_T.append( Stats.getString(2) +
                                "      ");
       ResStat.Screen_T.append( Stats.getString(3) +
                                "   ");
       ResStat.Screen_T.append( Stats.getString(4) +
                                "     ");
       ResStat.Screen_T.append( Long.toString(Stats.getLong(5)) +
                                "      ");
       ResStat.Screen_T.append( Long.toString(Stats.getLong(6)) +
                                "        ");
       ResStat.Screen_T.append( Long.toString(Stats.getLong(7)) +
                                "        ");
       ResStat.Screen_T.append( Long.toString(Stats.getLong(8)) +
                                "       ");
       ResStat.Screen_T.append( new Boolean(Stats.getBoolean(9)).toString() +
                                "   ");
       ResStat.Screen_T.append( new Boolean(Stats.getBoolean(10)).toString()+
                                "     ");
       ResStat.Screen_T.append( Long.toString(Stats.getLong(11)) +
                                "       ");

       //Get this value as a Long as we are not very much
       //interested in decimal digits
       ResStat.Screen_T.append( Long.toString(Stats.getLong(12)) );

       ResStat.Screen_T.append("\n");

       //Put a horizontal bar right under
       //and along the column values
       ResStat.Screen_T.append("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" +
                               "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" +
                               "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" +
                               "~~~~~~~~\n");

       //Increment by one for each
       //displayed record
       count++;
```

```
        //When 50 records have
        //been displayed...
        if (count == 50)
        {
          //Inform the user of what to do
          ResStat.Screen_T.append("Press the \"Next\" button " +
                                  "for the next set of stats...\n");

          //Set the NewStats boolean
          //variable to false
          NewStats = false;

          //exit this method
          return;
        }
      }

      //Let the poor user know that all is good!
      System.out.println("Database Query Successfull!");

      ResStat.Screen_T.append("<<END OF DATA>>");

     //Due to an ODBC driver whim
     //the statement should not
     //be closed in this case
     //QStatement.close();

   }

   //Catch SQL Exceptions
   catch (SQLException e)
   {
     System.out.println("SQL Select Statement failed!" + e);

      //Let the user decide
      //System.exit(1);
   }


  }//END of displayStats()
/****************************************************************************/



}//END of OStoreExperimentClient class
```

UserInterface.java

```
import java.awt.*;

/**
This class defines the UserInterface object.
It is a complex Frame object incorporating various
awt components like Labels, TextFields, Choices, and
Buttons. Thease are layed on the Frame according to
an appropriate Grid Layout.
The UserInterface receives UserInput that defines
an Experiment Configuration and lets the user perform
various actions on the client and server sides of the
system, including executing the experiment on the server
side.
*/
public class UserInterface extends Frame
{
   //Declarations of the Frame's various
   //Components

   private Label       ExperimentID_L,
                       StudentMember_L,
                       OptimizationType_L,
                       IndexableHashableMember_L,
                       StudentNumber_L,
                       Value_L,
                       Low_L,
                       High_L,
                       Pick_L,
                       CreateNewDB_L;

   //Textfields and Choices are declared protected because they
   //will be accesed by the OStoreExperimentClient so that the
   //UserInterface is update when, for example, the user loads
   //an experiment configuration.
   protected TextField ExperimentID_F,
                       StudentNumber_F,
                       Value_F,
                       Low_F,
                       High_F;


   protected Choice    StudentMember_C,
                       OptimizationType_C,
                       IndexableHashableMember_C,
                       Pick_C,
                       CreateNewDB_C;

   //These Buttons will be monitored for
   //user evnts by the OStoreExperimentClient.
   protected Button    Load,
                       StoreConfig,
                       Execute,
                       Exit,
                       StoreResult,
                       ResultStats,
                       ExpTime;

   //The Font we will be
   //using in this Frame
   private Font f;

   //This will be the oblect holding the experiment
   //configuration as it is defined by the user actions
   //on the UserInterface Frame object and its interaction
   //components
   protected TimedExperimentData ThisTimExp = null;
```

```
/****************************************************************************/
   //UserInterface class constructor
   public UserInterface()
   {
     //Create a Frame with passes title
     super( "User Interface" );

     //Instantiate a TimedExperimentData object
     ThisTimExp = new TimedExperimentData();

     //Set a layout of 14 rows, 2 columns
     //and 5 pixels inbetween space
     setLayout( new GridLayout(14,2,5,5) );

     //Instantiate a font we really love
     //and set it as this Frame's Font
     f = new Font( "TimesRoman", Font.PLAIN, 16 );
     setFont(f);

     //Instantiate Label components
     ExperimentID_L = new Label("Experiment ID: ");
     StudentMember_L = new Label("Student Field to Search: ");
     OptimizationType_L = new Label("Optimization Type: ");
     IndexableHashableMember_L = new Label("Student Field to Optimize: ");
     StudentNumber_L = new Label("Number of Student Objects to create: ");
     Value_L = new Label("Field Value to find: ");
     Low_L = new Label("Low Value for Range Query: ");
     High_L = new Label("High Value for Range Query: ");
     Pick_L = new Label("Use Pick Query: ");
     CreateNewDB_L = new Label("Create New DB: ");


     //Instantiate Choice components
     StudentMember_C = new Choice();
     StudentMember_C.add("ID");
     StudentMember_C.add("Age");
     StudentMember_C.add("Credits");

     OptimizationType_C = new Choice();
     OptimizationType_C.add("NoOpt");
     OptimizationType_C.add("HashingGet");
     OptimizationType_C.add("HashingQuery");
     OptimizationType_C.add("Indexing");

     IndexableHashableMember_C  = new Choice();
     IndexableHashableMember_C.add("None");
     IndexableHashableMember_C.add("getID()");
     IndexableHashableMember_C.add("getAge()");
     IndexableHashableMember_C.add("getCredits()");

     Pick_C = new Choice();
     Pick_C.setFont(f);
     Pick_C.add("false");
     Pick_C.add("true");

     CreateNewDB_C = new Choice();
     CreateNewDB_C.setFont(f);
     CreateNewDB_C.add("true");
     CreateNewDB_C.add("false");

     //Instantiate TextFields
     ExperimentID_F = new TextField("0", 10);

     StudentNumber_F = new TextField("1000", 10);

     Value_F = new TextField("500", 10);

     Low_F = new TextField("-1", 10);

     High_F = new TextField("-1", 10);


     //Instantiate Buttons
     Load        = new Button("Load Configuration");
     StoreConfig = new Button("Store Configuration");
     Execute     = new Button("Execute");
     Exit        = new Button("Exit");
```

```
     StoreResult = new Button("Store Result");
     ResultStats = new Button("Result Statistics");
     ExpTime     = new Button("Experiment Time");

     //Add all instantiated components onto the Frame's
     //layout. The sequence in which they are added determines
     //where on the layout theuy will be displayed.
     add(ExperimentID_L);              add(ExperimentID_F);
     add(StudentMember_L);             add(StudentMember_C);
     add(OptimizationType_L);          add(OptimizationType_C);
     add(IndexableHashableMember_L);   add(IndexableHashableMember_C);
     add(StudentNumber_L);             add(StudentNumber_F);
     add(Value_L);                     add(Value_F);
     add(Low_L);                       add(Low_F);
     add(High_L);                      add(High_F);
     add(Pick_L);                      add(Pick_C);
     add(CreateNewDB_L);               add(CreateNewDB_C);

     add(Load);
     add(StoreConfig);

     add(Execute);
     add(StoreResult);

     add(ExpTime);
     add(ResultStats);

     add(Exit);

     //Set an appropriate size
     //for the frame, so that
     //all components are
     //displayed nicely
     resize( 570, 420 );

     //Show the UserInterface Frame
     show();
   }
/******************************************************************************/



/******************************************************************************/
   //This method is called when we want to strip the user
   //supplied data from the various UI components and then
   //store it within the TimedExperimentData object, that
   //the UI created when it was first instantiated.
   public void getExperimentData()
   {

      ThisTimExp.setStudentMember
                 (StudentMember_C.getSelectedItem());
      ThisTimExp.setOptimizationType
                 (OptimizationType_C.getSelectedItem());
      ThisTimExp.setIndexableHashableMember
                 (IndexableHashableMember_C.getSelectedItem());

      ThisTimExp.setStudentNumber
                 ( Long.parseLong(StudentNumber_F.getText()) );
      ThisTimExp.setValue
                 ( Long.parseLong(Value_F.getText()) );
      ThisTimExp.setLow
                 ( Long.parseLong(Low_F.getText()) );
      ThisTimExp.setHigh
                 ( Long.parseLong(High_F.getText()) );

      ThisTimExp.setPick
        (Boolean.valueOf(Pick_C.getSelectedItem()).booleanValue() );
      ThisTimExp.setCreateNewDB
        (Boolean.valueOf(CreateNewDB_C.getSelectedItem()).booleanValue() );

   }
/******************************************************************************/
```

```
/***************************************************************************/
   //This simple method is called whenever
   //a window event occurs. It only handles
   //the WINDOW_DESTROY event, which occurs
   //when the user closes the window, by
   //pressing the 'X' button or otherwise.
   //When this occurs, the Frame object is
   //hiden and disposed and then the client
   //side JVM is terminated. This means that
   //closing the UI window ALSO (deliberately)
   //terminates the whole client-side system.
   //This action is equivalent to the user
   //pressing the "Exit" button.
   public boolean handleEvent( Event e )
   {
      if ( e.id == Event.WINDOW_DESTROY )
      {
         //Hide frame
         hide();

         //Free resources
         //associated with it
         dispose();

         //Terminate JVM
         System.exit( 0 );

         //Tell the caller
         //we took care of bussiness
         return true;
      }

      //Otherwise dispatch event to
      //the superclass' default event handlers
      return super.handleEvent( e );
   }
/***************************************************************************/


}//End of UserInterface class
```

ExperimentDisplay.java

```
import java.awt.*;

import java.util.Observer;
import java.util.Observable;

/**
This class is a Frame that will be used for displaying the ID number,
status and result of the executing experiment in real time. In order
to accomplish the "real-time" thingy it uses the Java Observable-Observer
facility. This class is registered (by the OStoreExperimentClient object)
as an Observer of the Observable (which is the TimedExperimentData object
created by the UserInterface object). In order to do that it has to
implement the Java Observer Interface. The resources of ExperimentDisplay
are never released as long as the client-side JVM is alive. Although
the user may close the Frame, it just hides and it reappears the next
time the user runs an experiment.
*/
public class ExperimentDisplay extends Frame implements Observer
{
  //Label components
  private   Label     ExperimentID_L,
                      ExperimentStatus_L,
                      ExperimentResult_L;

  //TextField components
  protected TextField ExperimentID_F,
                      ExperimentStatus_F,
                      ExperimentResult_F;

  private Font f;

  //This private TimedExperimentData object
  //will mainly be used in the Observer-Observable
  //scheme.
  private TimedExperimentData ThisTimExp = null;



/*****************************************************************************/
  //This is the constructor method of the class that creates
  //a new ExperimentDisplay object. This method is passed a
  //reference of a TimedExperimentData object (the Observable)
  //which is assigned to this class' private TimedExperimentData
  //reference. Thus we now have two references in different classes
  //(or objects when they are instantiated) for the same
  //TimedExperimentData object (which is originally created by the
  //UserInterface class once its constructor is called). No new
  //TimedExperimentData object is created here. We just get a
  //reference to the existing one so that we may observe it.
  public ExperimentDisplay(TimedExperimentData ThisTimExp)
  {
    //First of all call upon the superclass' (Window)
    //constructor method passing a String parameter.
    //This String will become the Frame's window
    //title
    super( "Experiment Display" );

    //Here we set the layout of this Container
    //object to one of 3 rows and 2 columns
    //on which the added components will
    //allign themselves, leaving a space of 5
    //pixels between, horizontally & vertically
    setLayout( new GridLayout(3,2,5,5) );

    //Create a font we like and set it as the Frame's font
    f = new Font( "TimesRoman", Font.PLAIN, 16 );
    setFont(f);

    //Make this class' reference get the
    //value of the passed reference
    this.ThisTimExp = ThisTimExp;

    //Instantiate Label components
    ExperimentID_L     = new Label("Experiment ID: ");
```

```
    ExperimentStatus_L = new Label("Experiment Status: ");
    ExperimentResult_L = new Label("Experiment Result: ");

    //Instantiate TextField components
    ExperimentID_F = new TextField("n/a", 10);
    ExperimentID_F.setEditable(false);

    ExperimentStatus_F = new TextField("n/a", 10);
    ExperimentStatus_F.setEditable(false);

    ExperimentResult_F = new TextField("n/a", 10);
    ExperimentResult_F.setEditable(false);

    //Add all instantiated components with the order we
    //want them to appear on the container's layout grid
    add(ExperimentID_L);     add(ExperimentID_F);
    add(ExperimentStatus_L); add(ExperimentStatus_F);
    add(ExperimentResult_L); add(ExperimentResult_F);

    //Set the window size
    resize( 320, 120 );

    //Finally show the
    //constructed window
    show();
   }
/*****************************************************************************/



/*****************************************************************************/
   //This method is called whenever the Oservable is
   //changed. The "obs" parameter receives a reference
   //of the changed Observable object. In the case
   //that this Observer was registered as an Observer
   //for more than one Observables (not the case here),
   //this reference could be used in comparisons with
   //the references of the Observables to determine
   //which one is the one that changed. This comparison
   //is not needed here (since there is only one observable)
   //but we do it anyway since it is good programming
   //practice.
   public void update(Observable obs, Object obj)
   {
     //If the changed Observable
     //is the TimedExperimentData object,
     //it means that experiment is complete
     //and its time is set within the object
     //So, do the following...
     if (obs == ThisTimExp)
     {
       //Show on the display that the
       //experiment is complete
       ExperimentStatus_F.setText("Complete");

       //Get the experiment's time
       //and display it
       long time = ThisTimExp.getTime();
       ExperimentResult_F.setText( Long.toString(time / 1000L) +
                                   " secs & "                  +
                                   Long.toString(time % 1000L) +
                                   " ms"                       );

       //Show the Frame
       //in case it was hidden
       show();
     }
   }
/*****************************************************************************/
```

```
/*************************************************************************/
   //This simple method is called whenever
   //a window event occurs. It only handles
   //the WINDOW_DESTROY event, which occurs
   //when the user closes the window, by
   //pressing the 'X' button or otherwise.
   //When this occurs, the Frame object is
   //not really destroyed...it just hides.
   public boolean handleEvent( Event e )
   {
      if ( e.id == Event.WINDOW_DESTROY )
      {
        //Hide Frame
        hide();

        //Tell the calling object
        //that the event was handled
        return true;
      }

      //Otherwise send the event to the
      //superclass' default event handlers
      return super.handleEvent( e );
   }
/*************************************************************************/



}//END of ExperimentDisplay class
```

ResultStats.java

```
import java.awt.*;

/**
This class is a frame that the client side
of the system uses to display stored, in the
Access DB, data in a format that is usefull
to the experimenter. Specifically, it displays
the average time for each stored experiment
configuration or (depending on the passed to its
constructor String) all the separate timings for
a single experiment configuration.
*/
public class ResultStats extends Frame
{
   //This TextArea will be used for
   //the display of the experiment data
   //that will be retrieved from the
   //Access DB via SQL queries
   protected TextArea Screen_T;

   //Through this button the user will
   //request the next screen of data
   protected Button Next;

   //This will hold the
   //font we like
   private Font f;


/******************************************************************************/
   //This is the Frame's constructor method
   public ResultStats(String Yo)
   {
     //First create a Frame with
     //an empty title
     super("");

     //Check Da pazzd Streeng Obzzakt Yo
     //and set the window title accordingly
     if ( Yo.equals("all") )
       this.setTitle( "All Experiments' Results" );
     else if ( Yo.equals("this") )
       this.setTitle( "Experiment Result");

     //Create a font we like
     f = new Font( "Courier", Font.PLAIN, 10 );

     //Create our button
     Next = new Button("NEXT >>");

     //Create the screen TextArea object
     Screen_T = new TextArea(44, 100);
     Screen_T.setFont(f);
     Screen_T.setEditable(false);

     //Display name for each TextArea column
     Screen_T.append("  ExpID  ");
     Screen_T.append(" StudMem ");
     Screen_T.append("Optim  ");
     Screen_T.append(" IndexMem");
     Screen_T.append(" StudNum ");
     Screen_T.append("  Value  ");
     Screen_T.append("   Low   ");
     Screen_T.append("   High  ");
     Screen_T.append("   Pick  ");
     Screen_T.append("  NewDB  ");
     Screen_T.append("  ExpID  ");

     //The right-wing column displays
     //the experiment's average time
     //over a series of separate timings
     Screen_T.append(" AvgTime ");

     //Next line please
     Screen_T.append("\n");
```

```
      //Put a horizontal bar right under and
      //along the column titles
      for(int i = 0; i < 128; i++)
        Screen_T.append("-");
      Screen_T.append("\n");

      //Add the Screen TextArea
      //coponent on the Frame
      add(Screen_T, BorderLayout.NORTH);

      //Add the Next button to the
      //Frame's layout
      add(Next, BorderLayout.SOUTH);

      //Set the right size
      resize( 800, 600 );

      //Show the Frame
      show();
    }
/*****************************************************************************/



/*****************************************************************************/
    //This simple method is called whenever
    //a window event occurs. It only handles
    //the WINDOW_DESTROY event, which occurs
    //when the user closes the window, by
    //pressing the 'X' button or otherwise.
    public boolean handleEvent( Event e )
    {
       if ( e.id == Event.WINDOW_DESTROY )
       {
          // hide frame
          hide();

          // free resources
          dispose();

          //Tell the caller that
          //the event has been
          //successfully handled
          return true;
       }

       //Otherwise send the event to the
       //superclass' default event handlers
       return super.handleEvent( e );
    }
/*****************************************************************************/


}//END of ResultStats class
```

TimedExperimentData.java

```
import java.util.Observable;

/**
This class represents an object that can hold (while maintaining
data encapsulation) all data that characterises an experiment
configuration, as well as a milliseconds time (in the form a long)
that represents a timing result for that experiment configuration.
The main purpose of this class will be to transienlty represent,
on the client side of the system, the running experiment and then
temporarily hold its timing result. This object is (when the
"execute" button is pressed) "transcribed" into an ExperimentData
CORBA struct and passed through the ORB to the server side where it
feeds the OStoreExperimentServant object with the data it needs to
perform the experiment. Unfortunately, IDL declared CORBA structs
(although they are represented by an appropriate Java class (after
IDLtoJava translation) they are not real objects as they do not
implement data encapsulation (their attributes are public) and thus
a programmer who wishes to maintain adequate data encapsulation of
the program's objects has to use the CORBA structs ONLY as a mere ORB
transfer vehicle and not as a normal object. I think that this will
change with the advent of the CORBA 3.0 specification (as we have
been told by Dr.Burger in class.
*/

public class TimedExperimentData extends Observable
{
  //Attributes

  private String StudentMember,           //The Attribute we are searching
                 OptimizationType,        //Well..Optimization Type used
                 IndexableHashableMember; //Which attribute to optimize

  private long   StudentNumber,           //How many Students to create
                 Value,                   //The value we are looking for
                 Low,                     //Lower bound of range queries
                 High,                    //Higher bound of range queries
                 Time;                    //The experiment result in millis

  private boolean Pick,                   //Whether to use a Pick query
                  CreateNewDB;            //Whether to create a new OS DB



/****************************************************************************/
  //Constructor
  public TimedExperimentData()
  {
    //Default Student Attribute values
    this.StudentMember            = "ID";
    this.OptimizationType         = "NoOpt";
    this.IndexableHashableMember  = "None";
    this.StudentNumber            = 1000L;
    this.Value                    = 500L;
    this.Low                      = -1L;
    this.High                     = -1L;
    this.Pick                     = false;
    this.CreateNewDB              = true;

    //-1L means no time data is stored
    this.Time                     = -1L;
  }
/****************************************************************************/



/****************************************************************************/
  //Accessors
  public String getStudentMember()
  {
    return this.StudentMember;
  }

  public String getOptimizationType()
  {
```

```
    return this.OptimizationType;
  }

  public String getIndexableHashableMember()
  {
    return this.IndexableHashableMember;
  }

  public long getStudentNumber()
  {
    return this.StudentNumber;
  }

  public long getValue()
  {
    return this.Value;
  }

  public long getLow()
  {
    return this.Low;
  }

  public long getHigh()
  {
    return this.High;
  }

  public boolean getPick()
  {
    return this.Pick;
  }

  public boolean getCreateNewDB()
  {
    return this.CreateNewDB;
  }

  public long getTime()
  {
    return this.Time;
  }


/*****************************************************************************/
  //Modifiers

  public void setStudentMember(String StudentMember)
  {
    this.StudentMember = StudentMember;
  }

  public void setOptimizationType(String OptimizationType)
  {
    this.OptimizationType = OptimizationType;
  }

  public void setIndexableHashableMember(String IndexableHashableMember)
  {
    this.IndexableHashableMember = IndexableHashableMember;
  }

  public void setStudentNumber(long StudentNumber)
  {
    this.StudentNumber = StudentNumber;
  }

  public void setValue(long Value)
  {
    this.Value = Value;
  }

  public void setLow(long Low)
  {
    this.Low = Low;
  }
```

```
  public void setHigh(long High)
  {
    this.High = High;
  }

  public void setPick(boolean Pick)
  {
    this.Pick = Pick;
  }

  public void setCreateNewDB(boolean CreateNewDB)
  {
    this.CreateNewDB = CreateNewDB;
  }

  public void setTime(long Time)
  {
    this.Time = Time;

    setChanged();
    notifyObservers();
  }
/****************************************************************************/


}//End of TimedExperimentData class
```

MiddleWare (CORBA and JDBC)

The middleware of the system is specified in CORBA IDL and implemented by the JAVA classes resulting from the *idltojava* compiler. The JDBC middleware is implemented in the Client Side of the System.

A problem was encountered was CORBA at 3:00 am on the morning of 1/1/2000. CORBA would refuse client-server communication (producing a fatal exception) as long as the computer's clock would show 2000 as its year. Resetting the system clock to 1999 solved the problem.

The bug was immediately reported to Sun but not to OMG as their whole site went down as soon as the new year arrived.

Experiment.idl

```
/*This is the IDL file for the Object Store Distributed Experiment System.*/
/*It consists of 2 modules, 1 exception, 1 struct, and 1 interface       */


/*Create a module to hold the ServerExceptions.      */
/*These will be user defined exceptions thrown       */
/*on the server side and caught on the client side.  */
/*Time not permitting much, only one Exception is    */
/*declared here: AnyException, which will cover all  */
/*kinds of server exceptions. It will be used mainly */
/*as a vehicle to transport error information about  */
/*the server to the client, via CORBA.               */
module ServerExceptions
{
  exception AnyException
  {
    /*This string will hold  */
    /*the error text message */
    string Msg;
  };
};


/*This is the main IDL module.                                            */
/*It defines a CORBA struct that will be used to transport the experiment */
/*data as specified by the user, from Client to Server.                   */
/*It also defines an interface which basically specifies the server side  */
/*method to be called. This is the DoExperiment method which will receive */
/*the above it defined CORBA struct object as an IN parameter and will    */
/*return the time the experiment took.                                    */
module OStoreExperiment
{
  struct ExperimentData
  {
    /*The Student object's member on which */
    /*we will base the search (query).     */
    /*It may be ID, Age, or Credits        */
    string StudentMember;

    /*Optimization to be used in the DataBase */
    string OptimizationType;

    /*Student Member over which the optimization  */
    /*is to be performed. This way we may be able */
    /*to query members either different than or   */
    /*or the same as the one we have optimaized   */
    /*thus extracting valuable experiment data    */
    string IndexableHashableMember;

    /*How many Student objects to create*/
    long long StudentNumber;

    /*The value to be found,             */
    /*if we are performing a point query */
    long long Value;
```

```
    /*Values representing the bounds of our range query. */
    /*These bounds are inclusive                         */
    long long Low;
    long long High;

    /*Whether a Pick type of query should be used or not */
    boolean Pick;

    /*Whether to create a new DB or not */
    boolean CreateNewDB;
  };

  /*The experiment interface that declares the DoExperiment() method */
  interface Experiment
  {
    /*The DoExperiment() method receives a struct and returns a long       */
    /*While it is capable of raising (throwing in JAVA lingo) AnyException */
    long long DoExperiment(in ExperimentData ThisExp)
      raises (ServerExceptions::AnyException);
  };
};

/*Dat izzall!*/
```

The Server Layer

Description

The OstoreExperimentServer starts and registers OStoreExperimentServant with CORBA. It then waits for client invocations. Read inline comments for details of exactly how all this happens.

The Server side DOS window output looks like this:

```
JAVA
7 x 12
G:\O\Project\System>server

G:\O\Project\System>java OStoreExperimentServer -ORBInitialPort 1050
Servant Created and registered with ORB!
Got root naming context
Object reference bound in naming
Waiting for client invocations...
Query prepared!
Value: 0 LH values: -1 -1
Database Destroyed!
CreateDB Transaction began, using NoOpt
OSVector Root Created!
CreateDB Transaction ended successfully, using NoOpt
Database Created!
Populating with 1000 Student objects...
(Performing Commit every 1000 objects)
Done in: 0seconds and 160ms
(Not counting time Commits take!)

executeQuery: Root acquired!
executeQuery: Timing began!
executeQuery: Timing Stopped!
EXPERIMENT TIME: 1210
#######################################
```

Server side source code follows;

Server Side Source Code

OstoreExperimentServer.java

```
import OStoreExperiment.*;
import ServerExceptions.*;

import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;

/**
This class is the CORBA server, and is the one that
is called first when the system initialises on the
server side. It creates the server side ORB, creates
an instance of the OStoreExperimentServantObject,
registers it with the ORB's naming service and
finally it starts waiting for CORBA client invocations
*/
public class OStoreExperimentServer
{

  public static void main(String args[])
  {
    try
    {
      //Create and initialize the ORB
      ORB orb = ORB.init(args, null);

      //Create servant and register it with the ORB
      OStoreExperimentServant OSESRef = new OStoreExperimentServant();
      orb.connect(OSESRef);
      System.out.println("Servant Created and registered with ORB!");

      //Get the root naming context
      org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
      NamingContext ncRef = NamingContextHelper.narrow(objRef);
      System.out.println("Got root naming context");

      //Bind the Object Reference in Naming
      NameComponent nc = new NameComponent("ExperimentServant", "");
      NameComponent path[] = {nc};
      ncRef.rebind(path, OSESRef);
      System.out.println("Object reference bound in naming");

      //Wait for invocations from clients
      System.out.println("Waiting for client invocations...");
      java.lang.Object sync = new java.lang.Object();
      synchronized (sync)
      {
        sync.wait();
      }

    }

    //Catch all Exceptions thrown and print
    //all relevant debugging info
    catch (Exception e)
    {
      System.err.println("\n==CORBA SERVER ERROR: " + e + "==\n");
      e.printStackTrace(System.out);
    }
  }

}
```

OstoreExperimentServant.java

```
//This file contains the OStoreExperimentServant class definition

import com.odi.*;
import com.odi.util.*;
import com.odi.util.query.*;

import java.util.*;
import java.io.*;

import ServerExceptions.*;
import OStoreExperiment.*;

/*************************************************************************/
/**
This class is a CORBA servant that can be used by CORBA clients
to manipulate ObjectStore databases on the server side.
It implements a series of methods for the creation, destruction,
opening, closing, etc. of the server side OStore DB.
*/
public class OStoreExperimentServant extends    _ExperimentImplBase
                                     implements ObjectStoreConstants
{

  //Create a CORBA struct to receive the
  //data for the experiment to be performed
  private static ExperimentData ThisExp;

  //Other globally needed variables
  private Database              StudentDB;
  private Query                 Q;
  private FreeVariables         FreeVars;
  private FreeVariableBindings FreeVarsB;
  private Collection            RootCollection;
  private Set                   QuerySelectResult = null;
  private OSHashMap             RootMap;
  private Student               QueryPickResult = null;


/*************************************************************************/
  /**
  This method is used to display an error message
  and then terminate the client side JVM
  */
  private
  void
  terminationMsg(String Msg)
  {
    System.err.println(Msg);
    System.exit(1);
  }
/*************************************************************************/


/*************************************************************************/
  /**
  This method just prints a message to the
  error console in an easier way.
  */

  private
  void
  systemMsg(String Msg)
  {
    System.err.println(Msg);
  }
/*************************************************************************/
```

```
/*************************************************************************/
  /**
  This method receives a set of student objects
  and prints all of them using the toString()
  method as implemented in the Student object definition
  */
  private
  void
  displayStudentSet(Set StudentSet)
  {
    Iterator i = StudentSet.iterator();

    while (i.hasNext())
      systemMsg( i.next().toString() );
  }
/*************************************************************************/



/*************************************************************************/
  /**
  This method receives a series of appropriate parameters reflecting
  the user choices on the client side and constructs the required
  OS query
  */
  private
  void
  prepareQuery(String StudentMember,
               long Value, long Low, long High, long StudentNumber)
  {

    //If a range query is requested but just one of the bounds
    //is specified by the user (the other being -1L), then
    //implicitly make the unspecified bound the highest or lowest
    //student number (depending on which of the two bounds was not
    //specified)
    if (Low != -1L && High == -1L) High = StudentNumber - 1L;
    if (Low == -1L && High != -1L) Low  = 0L;


    //Queries can be on any of the 3 Student object instance vars
    //Thus Queries have to be dynamically created and thus we need
    //FreeVariables and FreeVariableBindings
    FreeVars  = new FreeVariables();
    FreeVarsB = new FreeVariableBindings();

    //If the query concerns the ID field of the Student objects
    //then do the folowing
    if (StudentMember.equals("ID"))
    {
      //If the requested query is of "point" type then...
      if (Value != -1L)
      {
        FreeVars.put("Value", Long.class);
        FreeVarsB.put("Value", new Long(Value));

        Q = new Query(Student.class,
                      "getID() == Value.longValue()", FreeVars);
      }

      //Otherwise, since the query is of "range" type...
      else
      {
        FreeVars.put("Low"  , Long.class);
        FreeVars.put("High" , Long.class);
        FreeVarsB.put("Low"  , new Long(Low));
        FreeVarsB.put("High" , new Long(High));

        Q = new Query(Student.class,
                      "getID() >= Low.longValue() && getID() " +
                      "<= High.longValue()", FreeVars);
      }
    }
```

```
    //Here we take care of the case that the requested query
    //involves the Age Student member field.
    //Everything is the same as in the previous case
    else
    if (StudentMember.equals("Age"))
    {
      if (Value != -1L)
      {
        FreeVars.put("Value", Long.class);
        FreeVarsB.put("Value", new Long(Value));

        Q = new Query(Student.class,
                      "getAge() == Value.longValue()", FreeVars);
      }
      else
      {
        FreeVars.put("Low"  , Long.class);
        FreeVars.put("High", Long.class);
        FreeVarsB.put("Low"  , new Long(Low));
        FreeVarsB.put("High" , new Long(High));

        Q = new Query(Student.class,
                      "getAge() >= Low.longValue() && getAge() " +
                      "<= High.longValue()", FreeVars);
      }
    }

    //The Credits field is covered here.
    //Same as before...
    else
    if (StudentMember.equals("Credits"))
    {
      if (Value != -1L)
      {
        FreeVars.put("Value", Long.class);
        FreeVarsB.put("Value", new Long(Value));

        Q = new Query(Student.class,
                      "getCredits() == Value.longValue()", FreeVars);
      }
      else
      {
        FreeVars.put("Low"  , Long.class);
        FreeVars.put("High", Long.class);
        FreeVarsB.put("Low"  , new Long(Low));
        FreeVarsB.put("High" , new Long(High));

        Q = new Query(Student.class,
                      "getCredits() >= Low.longValue() && getCredits() " +
                      "<= High.longValue()", FreeVars);
      }
    }

    //Signify the end of the query preparation
    systemMsg("Query prepared!");

  }//END prepareQuery()
/******************************************************************************/
```

```
/*************************************************************************/
  /**
  This method executes the globally declared OS Query object.
  In the case that HashingGet has been specified as the
  optimization to be used, the method just uses a straight get()
  method on the object we are looking for.
  Results are fed into a globally declared Set object if a Query is used.
  If a Pick query or HashingGet is used a single Student object is
  asigned to the appropriate global variable (instance variable).
  It returns the time the query execution
  took in milliseconds (as a long value)
  */

  private
  long
  executeQuery(String OptimizationType, boolean Pick, long Value)
  {
    Transaction.begin(UPDATE);

    //If the optimization type is Hashing get the DB root
    //as a OSHashMap, otherwise as a Collection
    if ( OptimizationType.equals("HashingGet")  ||
         OptimizationType.equals("HashingQuery")   )
      RootMap = (OSHashMap) StudentDB.getRoot("Hashing");
    else
      RootCollection = (Collection) StudentDB.getRoot(OptimizationType);

    systemMsg("executeQuery: Root acquired!");

    //Start timing the Query of the get() operation
    systemMsg("executeQuery: Timing began!");
    long start = System.currentTimeMillis();

    //If HashingGet is used then get the object whose field has
    //the value of interest
    if (OptimizationType.equals("HashingGet"))
      QueryPickResult = (Student) RootMap.get(new Long(Value), true);

    //If HashingQuery is used then execute the prepared query on the
    //Values View of the root HashMap
    else if (OptimizationType.equals("HashingQuery"))
    {
      //If Pick is not requested, execute the query normally
      if (!Pick)
        QuerySelectResult = Q.select(RootMap.values(), FreeVarsB);

      //If Pick is requested then execute the prepared query
      //as a Pick query
      else
        QueryPickResult = (Student) Q.pick(RootMap.values(), FreeVarsB);
    }

    else

    //For any other kind of optimization scheme used,
    //including no optimization, do the following...
    if (!Pick)
      QuerySelectResult = Q.select(RootCollection, FreeVarsB);
    else
      QueryPickResult   = (Student) Q.pick(RootCollection, FreeVarsB);

    //Stop timing of the search routine
    long stop = System.currentTimeMillis();
    systemMsg("executeQuery: Timing Stopped!");


    //Display results of search (mainly for debugging purposes)
    //Eventually COMMENTED OUT as they are not needed by the
    //system as specified in the assignment
    /*
    if (QueryPickResult != null)
      systemMsg("QueryPickResult: \n" + QueryPickResult.toString());

    else
```

```
    //In case the found objects are more than 6, do not display them.
    //Otherwise, display the objects whose refs are held in
    //the QuerySelectResult object.
    if (QuerySelectResult != null && QuerySelectResult.size() <= 6)
    {

      systemMsg("QuerySelectResult");
      displayStudentSet(QuerySelectResult);
    }
    */

    Transaction.current().commit();

    //Return the experiment time
    return stop - start;

  }
/****************************************************************************/



/****************************************************************************/
  /**
  This method is used to open the DB in the passed mode
  */
  private
  void
  openDB(String Mode) throws AnyException
  {
    try
    {
      if (Mode.equals("UPDATE"))
        StudentDB = Database.open("Students.odb", ObjectStore.UPDATE);

      else

      if (Mode.equals("READONLY"));
        StudentDB = Database.open("Students.odb", ObjectStore.READONLY);
    }
    catch (DatabaseNotFoundException e)
    {
      //If database is not found, display the generated exception on the
      //server side and and throw an AnyException object
      //for the client to catch and display to the remote user
      systemMsg("Database was not found: " + e);
      throw( new AnyException("Database was not found: " + e) );
    }
  }
/****************************************************************************/




/****************************************************************************/
  /**
  This method opens the DB in update mode and destroys it, if
  the user requests that a new DB be created. If the DB is
  not there, an appropriate message is displayed and program
  continues normally
  */
  private
  void
  destroyDB()
  {

    try
    {
      //Destroy Students DB
      Database.open("Students.odb", ObjectStore.UPDATE).destroy();
      systemMsg("Database Destroyed!");
    }

    catch(DatabaseNotFoundException e)
    {
      systemMsg("Database to destroy was not found: " + e);
    }
```

```
  }
/**************************************************************************/



/**************************************************************************/
  /**
  This method creates the student DB, and the DB root according
  to the optimization schee requested. The IndexableHashableMember
  parameter specifies the Student member var over which the
  optimization should be performed
  */
  private
  void
  createDB(String OptimizationType, String IndexableHashableMember)
    throws AnyException
  {

    //Create DB
    StudentDB = Database.create("Students.odb", ALL_READ | ALL_WRITE);

    //Begin transaction instance
    Transaction.begin(UPDATE);

    //Show what is going on, on screen
    systemMsg("CreateDB Transaction began, using " + OptimizationType);

    //If NoOpt (no optimnization) is used,
    //create the DB root as an OSVector.
    //In all cases the root is given the
    //name of the OptimizationTyp used.
    if (OptimizationType.equals("NoOpt"))
    {
      OSVector Coll = new OSVector();
      StudentDB.createRoot(OptimizationType, Coll);
      systemMsg("OSVector Root Created!");
    }

    else

    //If hashing is requested, implement the DB root
    //as an OSHashMap (So that we can use both the get()
    //method and queries over its views)
    if ( OptimizationType.equals("HashingGet")  ||
         OptimizationType.equals("HashingQuery")   )
    {
      OSHashMap MyMap = new OSHashMap();
      StudentDB.createRoot("Hashing", MyMap);
      systemMsg("OSHashMap Root Created!");
    }

    else

    //If indexing is requested create the root
    //as an OSTreeSet
    if (OptimizationType.equals("Indexing"))
    {
      IndexedCollection Coll = new OSTreeSet(StudentDB);
      StudentDB.createRoot(OptimizationType, Coll);
      systemMsg("OSTreeSet Root Created!");

      //Here we add the index to the OSTreeSet over
      //the requested Student member although it is still
      //empty. We did not have to do that here as we could
      //add the index after adding the Students in the OSTreeSet.
      //Doing it here has the advantage that the index is
      //implemented on each Student object as it is added,
      //instead of processing them all together at the end
      //which, if the DB is big, can be very memory and CPU
      //hungry.
      if (!IndexableHashableMember.equals("None"))
      try
      {
        Coll.addIndex(Student.class, IndexableHashableMember);
        systemMsg("Index Created!");
      }
```

```
      //Catch a possible index exception.
      catch (IndexException e)
      {
        systemMsg("Couldn't access specified student field: " + e);
        throw( new AnyException("Couldn't access specified student field: "
                                + e) );
      }

    }

    Transaction.current().commit();

    //Show what is going on, on screen
    systemMsg("CreateDB Transaction ended successfully, using "
              + OptimizationType);

    systemMsg("Database Created!");

  }//END createDB()
/*****************************************************************************/



/*****************************************************************************/
  /**
  This method is used to populate the created and open Students DB.
  It adds to it the specified number of Student objects (StudentNumber)
  according to the passed OptimizationType over IndexableHashableMember.
  */

  private
  void
  populateDB( long   StudentNumber,
              String OptimizationType,
              String StudentMember,
              String IndexableHashableMember )
  {
    //Vars
    Student Fotios;

    Collection Coll = RootCollection;

    //These vars will be used for the
    //timing of the DB population routine
    long Start, Stop, Time = 0L;

    //object counter
    long j = 0L;

    systemMsg( "Populating with " + StudentNumber + " Student objects...\n"
              +"(Performing Commit every 1000 objects)" );

    //Create StudentNumber Student objects in multiple transactions
    //so that the memory strain is reduced
    for (;j < StudentNumber;)
    {
      Transaction.begin(UPDATE);

      //Get the appropriate root object
      if ( OptimizationType.equals("HashingGet") ||
           OptimizationType.equals("HashingQuery")  )
        RootMap = (OSHashMap) StudentDB.getRoot("Hashing");

      else
        Coll = (Collection) StudentDB.getRoot(OptimizationType);

      //start timing population here
      Start = System.currentTimeMillis();


      //Commit after every 1000 Students to
      //limit the size of each transaction.
      for (int i = 0; i < 1000 && j < StudentNumber; i++)
      {

              //Create new Student object
              Fotios = new Student();
```

```
        //Set the ID field to be
        //equal to the counter's value
        //ID spans from 0 to (StudentNumber - 1)
             Fotios.setID(j);

        //Age spans from (StudentNumber - 1) to 0
        Fotios.setAge( (StudentNumber - 1) - j );

             //Credits take the exact same value that the ID
             //field takes. This is because we later want to
             //compare queries over these two fields in
             //unoptimized root collections.
             Fotios.setCredits(j++);

        //In the case hashing is used a member over which the hashing
        //will be implemented, HAS to be selected. Here Student objects
        //are hashed into the HashMap root over the specified member
        if ( OptimizationType.equals("HashingGet")  ||
             OptimizationType.equals("HashingQuery")   )
        {
          if (IndexableHashableMember.equals("getID()"))
            RootMap.put(new Long(Fotios.getID()), Fotios);

          else if (IndexableHashableMember.equals("getAge()"))
            RootMap.put(new Long(Fotios.getAge()), Fotios);

          else if (IndexableHashableMember.equals("getCredits()"))
            RootMap.put(new Long(Fotios.getCredits()), Fotios);
        }

        //All other root and optimization types are handled
        //by the following statement. I guess this is what they call
        //polymorphism. That means the object is actually stored in
        //the way that the existing root object dictates. In the case
        //of Indexing, indexing is also implemented "automatically"
        //since (if it was requested) the index has already
        //been added to the root.
        else
               Coll.add(Fotios);
      }

      //Stop timing of this 1000-object transaction.
      Stop = System.currentTimeMillis();

      //Add this time to the total time
      Time += Stop - Start;

      //Commit this 1000-object transaction
      Transaction.current().commit();

    }

    systemMsg( "Done in: " + (Time / 1000L)
                           + "seconds and "
                           + (Time % 1000L)
                           + "ms\n"
                           + "(Not counting time Commits take!)\n" );


  }//END populateDB()
/*****************************************************************************/
```

```
/***************************************************************************/
  /**
  This is the most important method in the OStoreExperimentServant class.
  It is exported to the CORBA services and may be called by CORBA clients.
  It returns the time the experiment took, which is actually the long
  millisecond value that the executeQuery() method above, returns.
  This method calls the other methods of the OstoreExperimentServant object
  in the proper sequence so that the experiment is executed properly.
  It receives as a parameter a CORBA ExperimentData struct (as defined
  in the IDL file) which represents the choices made by the user in the
  UserInterface object on the client side of the system. It is capable,
  of throwing a CORBA AnyException, which is declared in the IDL's
  ServerExceptions Module.
  */
  public
  long
  DoExperiment(ExperimentData ThisExp) throws AnyException
  {

    //This var represents the experiment time
    //and it will be returned by the method at the end
    long Time = -1L;

    //Create a new PSE Session
    Session session = Session.create(null, null);

    try
    {

      session.join();

      //Prepare the query to be executed.
      //This operation does not need to be
      //inside a transaction of course.
      prepareQuery( ThisExp.StudentMember,
                    ThisExp.Value,
                    ThisExp.Low,
                    ThisExp.High,
                    ThisExp.StudentNumber );

      //Output values we are looking for
      systemMsg("Value: "      + ThisExp.Value +
                " LH values: " + ThisExp.Low + " " + ThisExp.High);

      //If a new DB is requested by the client....
      if ( ThisExp.CreateNewDB )
      {

        //Destroy the old DB (if it exists)
        destroyDB();

        //Create the new DB
        createDB( ThisExp.OptimizationType,
                  ThisExp.IndexableHashableMember );


        //Populate it good!
        populateDB( ThisExp.StudentNumber,
                    ThisExp.OptimizationType,
                    ThisExp.StudentMember,
                    ThisExp.IndexableHashableMember );


        //Execute the prepared query and get the time it took
        Time = executeQuery(ThisExp.OptimizationType,
                            ThisExp.Pick,
                            ThisExp.Value);
      }
```

```
      //If a new DB is not requested open the old one,
      //assuming it exists. If it does not exist,
      //an exception (viewed by the client too via CORBA)
      //is generated and the client JVM (Java Virtual Machine)
      //is terminated
      else
      {
        //Open Sesame..,
        openDB("READONLY");

        //Time Yo!
        Time = executeQuery(ThisExp.OptimizationType,
                            ThisExp.Pick,
                            ThisExp.Value);

      }

    }

    //Catch'em OStore Exxepzions
    catch (ObjectStoreException e)
    {
      systemMsg(e.toString());
      throw( new AnyException("ObjectStore Exception:" + e) );
    }

    //Finally, terminate this nasty session...
    finally
    {
      session.terminate();
    }

    //Output the experiment time on the server side
    //(For Debugging mainly)
    systemMsg("EXPERIMENT TIME: " + Time);

    //Delimit Experiment results on the
    //server-side monitor DOS screen
    systemMsg("#######################################");

    //Return the time to the calling client
    return Time;

  }//END of DoExperiment()
/****************************************************************************/


}//END of OStoreExperimentServant class
```

Student.java

```
/**
Da StudanD Clazz!
This Class defines the Student objects that will become
persistent and be stored within the ObjectStore database.
It is designed according to the data encapsulation paradigm,
that is, all member attributes are declared private and may
be accessed or modified only by calling appropriate accessor
or modifier member methods. It also has overloaded constructors
and it also overrides the Object toString() method, so that one
may appropriately and easily display Student objects on-screen.
*/

public class Student
{

  private long ID;
  private long Age;
  private long Credits;


  //Constructors

/***************************************************************************/
  public Student()
  {
    this.ID      = 0L;
    this.Age     = 0L;
    this.Credits = 0L;
  }
/***************************************************************************/


/***************************************************************************/
  public Student(long ID, long Age, long Credits)
  {
    this.ID      = ID;
    this.Age     = Age;
    this.Credits = Credits;
  }
/***************************************************************************/



/***************************************************************************/
  //Accessors
  public long getID()
  {
    return this.ID;
  }

  public long getAge()
  {
    return this.Age;
  }

  public long getCredits()
  {
    return this.Credits;
  }
/***************************************************************************/




/***************************************************************************/
  //Modifiers
  public void setID(long ID)
  {
    this.ID = ID;
  }

  public void setAge(long Age)
  {
    this.Age = Age;
  }
```

```
  public void setCredits(long Credits)
  {
    this.Credits = Credits;
  }
/*************************************************************************/



/*************************************************************************/
  //Display the Student object
  //in three titled rows
  public String toString()
  {
    return ("Student:\n"      +
                 "  ID         = " + ID  + "\n" +
                 "  Age        = " + Age + "\n" +
                 "  Credits    = " + Credits);
  }
/*************************************************************************/


}//END of Student class
```

The System's Databases

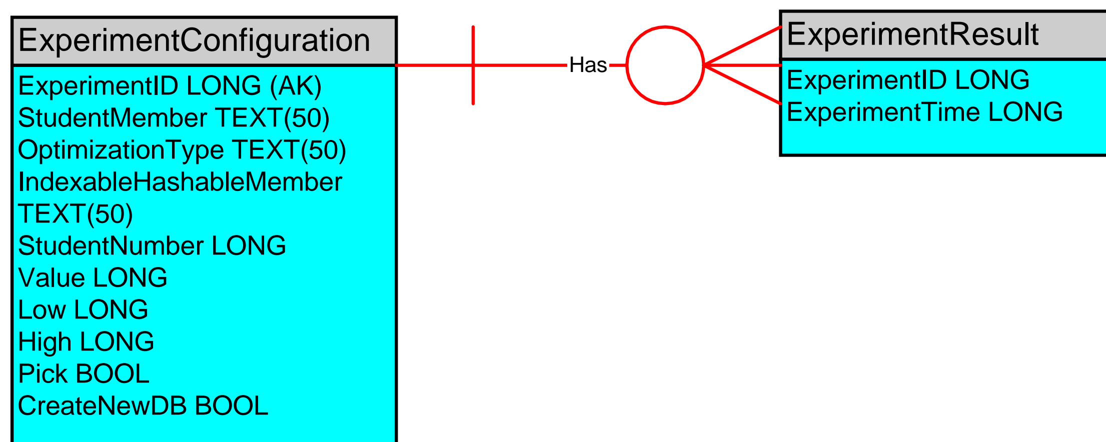
The Access Database (Relational)

The Access database is used for holding the System's Experiment Configurations and their related timing results. The Experiment Configurations are held in the *ExperimentConfigurations* Table (Relation) and the Experiment Results are held in the *ExperimentResults* Table (Relation).

Various queries that produce database data views that are helpful in making sense out of the numerous results and their corresponding experiment configurations, are also defined within the Microsoft Access database.

The ERD (diagram) of the Access database schema follows:

exp.mdb		Date :6/1/2000 (Digital Apocalypse)
Description: This relational database (MS ACCESS) holds the experiment configurations of the distributed ObjectStore testbed system and their corresponding multiple timing results.		
Target DB: Access	Rev: 0	Creator: LoneWolf
Filename: AccessERD.vsd		Company: PIT, inc.

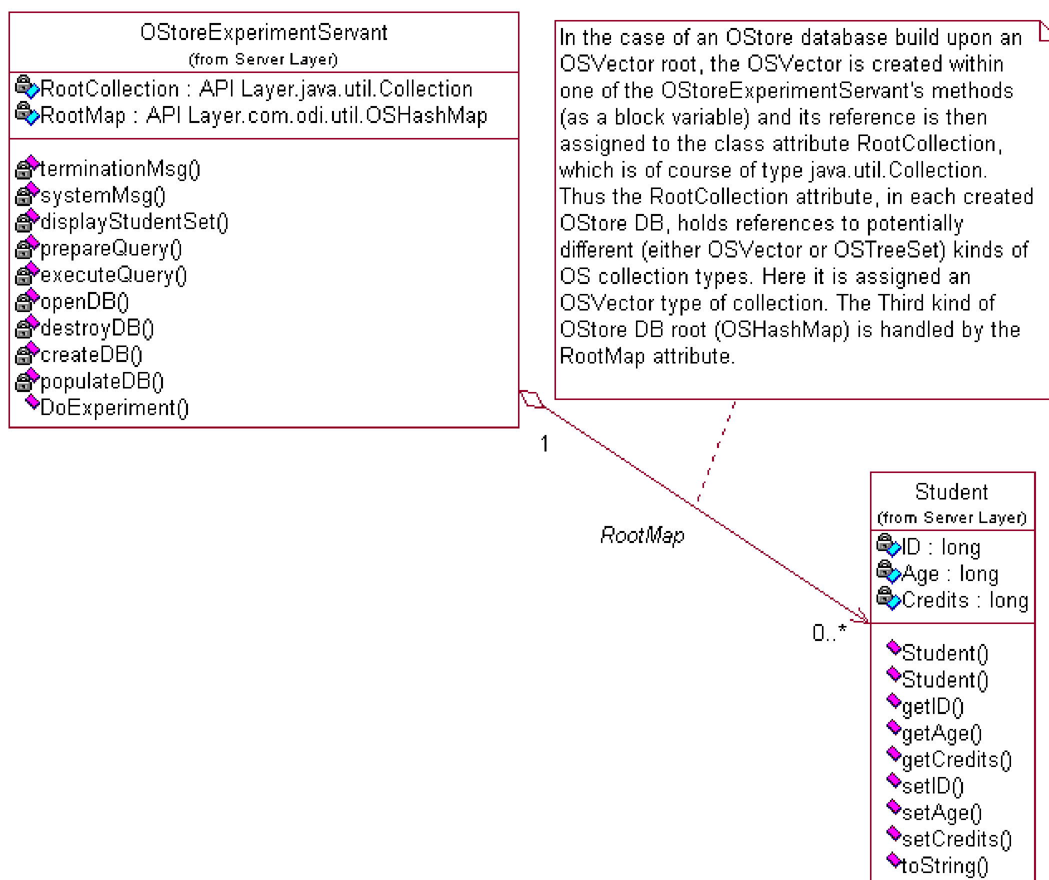


The ObjectStore Database (Object Oriented)

At any given time the server side maintains a single ObjectStore Database. This Database can have multiple schemas depending on the chosen root collection. The OStore database schemas used are described by the following UML diagrams. Each UML diagram is followed by an "under the hood" look of the corresponding .odb OStore database file.

OSVector Database Schema

UML



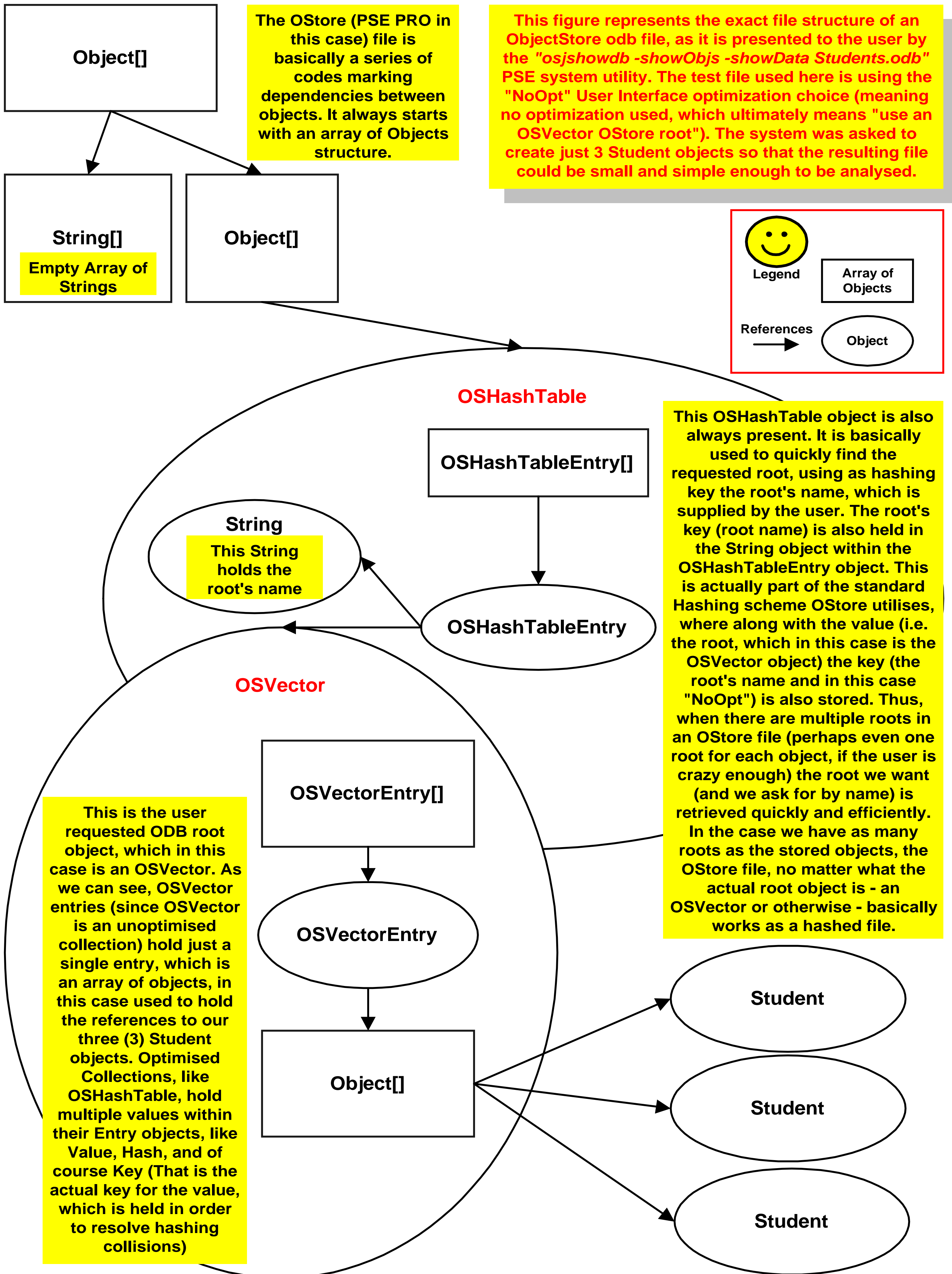
The raw textual data from the *osjshowdb -showObjs -showData Students.odb* command for each case, and an "Under the hood look" diagram analysing that data, follow;

ShowDB for .odb file with OSVector root with 3 added objects

```
> cd .
> osjshowdb -showObjs -showData Students.odb


G:\O\BigOne2>osjshowdb -showObjs -showData Students.odb
Name: Students.odb

There is one root:

 Name: NoOpt   Type: com.odi.util.OSVector


Destroyed Objects: 1

Segment: 0
Size: 3614 (4 Kbytes)

                                     Total
OID        Data Offset  Elements     Bytes  Type

<1|0|0|0>         1640         8        72  java.lang.Object[]
 Contains references to:
  0: <1|0|0|3>
  8: 6
  16: <1|0|0|1>
  24: null
  32: null
  40: null
  48: null
  56: null
  ...more nulls

<1|0|0|1>         2272         1        16  java.lang.String[]
<1|0|0|2>         1608                  32  com.odi.util.OSVector
 Contains references to:
  4: <1|0|0|10>
<1|0|0|3>          352         1        16  java.lang.Object[]
 Contains references to:
  0: <1|0|0|7>
<1|0|0|4>          264         5        16  java.lang.String
  Value: "NoOpt"
<1|0|0|5>          280                  32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|4>
  8: <1|0|0|2>
  16: null
<1|0|0|6>          144         2        24  com.odi.util.OSHashtableEntry[]
 Contains references to:
  0: <1|0|0|5>
  8: null
<1|0|0|7>          320                  32  com.odi.util.OSHashtable
 Contains references to:
  4: <1|0|0|6>
  20: null
<1|0|0|8>         1344        32       264  java.lang.Object[]
 Contains references to:
  0: <1|0|0|12>
  8: <1|0|0|13>
  16: <1|0|0|14>
  24: null
  32: null

<1|0|0|9>          712                  16  com.odi.util.OSVectorEntry
 Contains references to:
  0: <1|0|0|8>
<1|0|0|10>         728         1        16  com.odi.util.OSVectorEntry[]
 Contains references to:
  0: <1|0|0|9>
<1|0|0|12>         168                  32  Student
<1|0|0|13>         200                  32  Student
<1|0|0|14>         232                  32  Student
<1|0|0|15>        1728       537       544  java.lang.String
  Value:
"Ccom.odi.util.OSVector;{MODITheHashCode;fMelementEntryData;[]Tcom.odi.util.OSVectorEn
try;MelementCount;fMbucketIncrement;fMentryTableSize;f}Ccom.odi.util.OSDictionary;{MOD
```

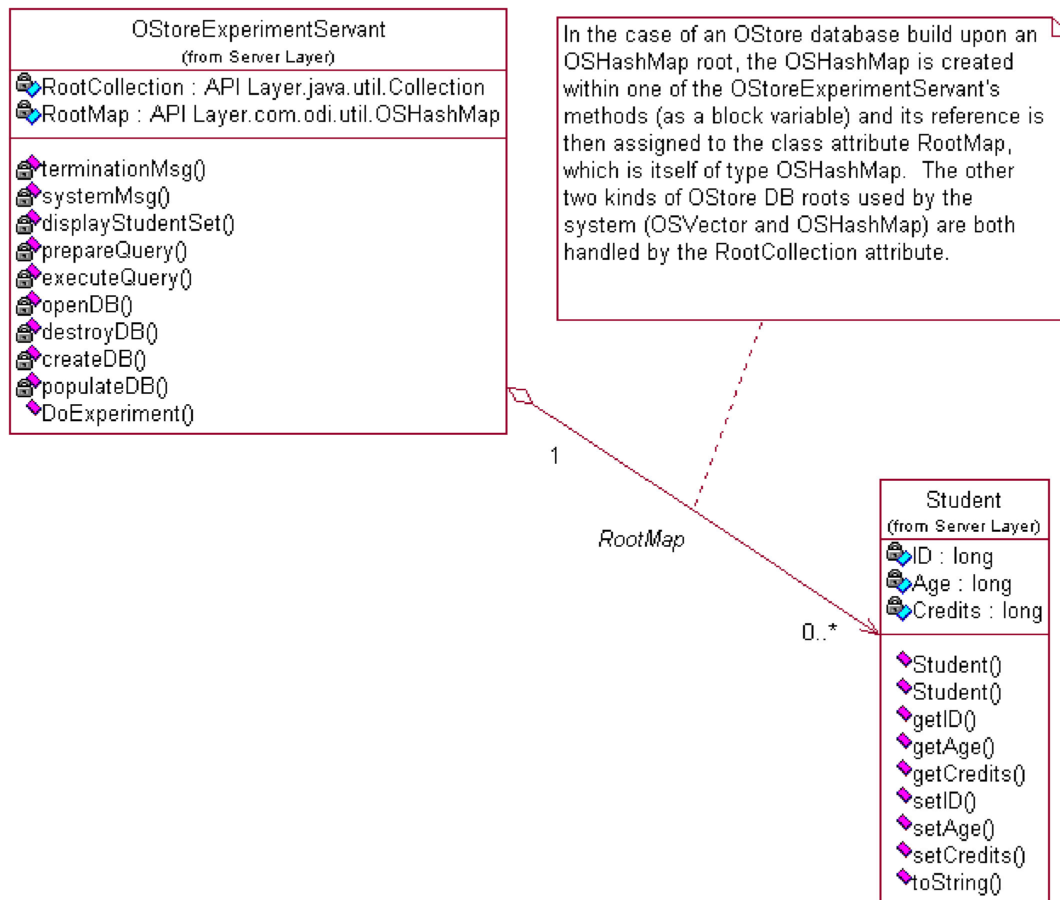
ITheHashCode;f}Ccom.odi.util.OSHashtable;{Bcom.odi.util.OSDictionary;Mtable;[]Tcom.odi
.util.OSHashtableEntry;MnTableEntries;fMreorgThreshold;fMaltRep;Tcom.odi.util.OSDictio
nary;}Ccom.odi.util.OSHashtableEntry;{Mkey;OMelement;OMnext;Tcom.odi.util.OSHashtableE
ntry;Mhash;f}Ccom.odi.util.OSVectorEntry;{MelementData;[]O}CStudent;{MODITheHashCode;f
MID;hMAge;hMCredits;h}"

Count	Tot Size (bytes)	Type
3	96	Student
1	32	com.odi.util.OSHashtable
1	32	com.odi.util.OSHashtableEntry
1	24	com.odi.util.OSHashtableEntry[]
1	32	com.odi.util.OSVector
1	16	com.odi.util.OSVectorEntry
1	16	com.odi.util.OSVectorEntry[]
3	352	java.lang.Object[]
2	560	java.lang.String
1	16	java.lang.String[]


Object[]
The OStore (PSE PRO in this case) file is basically a series of codes marking dependencies between objects. It always starts with an array of Objects structure.
This figure represents the exact file structure of an ObjectStore odb file, as it is presented to the user by the "osjshowdb -showObjs -showData Students.odb" PSE system utility. The test file used here is using the "NoOpt" User Interface optimization choice (meaning no optimization used, which ultimately means "use an OSVector OStore root"). The system was asked to create just 3 Student objects so that the resulting file could be small and simple enough to be analysed.
String[]
Empty Array of Strings
Object[]
Legend
Array of Objects
References
Object
OSHashTable
OSHashTableEntry[]
String
This String holds the root's name
OSHashTableEntry
This OSHashTable object is also always present. It is basically used to quickly find the requested root, using as hashing key the root's name, which is supplied by the user. The root's key (root name) is also held in the String object within the OSHashTableEntry object. This is actually part of the standard Hashing scheme OStore utilises, where along with the value (i.e. the root, which in this case is the OSVector object) the key (the root's name and in this case "NoOpt") is also stored. Thus, when there are multiple roots in an OStore file (perhaps even one root for each object, if the user is crazy enough) the root we want (and we ask for by name) is retrieved quickly and efficiently. In the case we have as many roots as the stored objects, the OStore file, no matter what the actual root object is - an OSVector or otherwise - basically works as a hashed file.
OSVector
OSVectorEntry[]
OSVectorEntry
Object[]
This is the user requested ODB root object, which in this case is an OSVector. As we can see, OSVector entries (since OSVector is an unoptimised collection) hold just a single entry, which is an array of objects, in this case used to hold the references to our three (3) Student objects. Optimised Collections, like OSHashTable, hold multiple values within their Entry objects, like Value, Hash, and of course Key (That is the actual key for the value, which is held in order to resolve hashing collisions)
Student
Student
Student

OSHashMap Database Schema

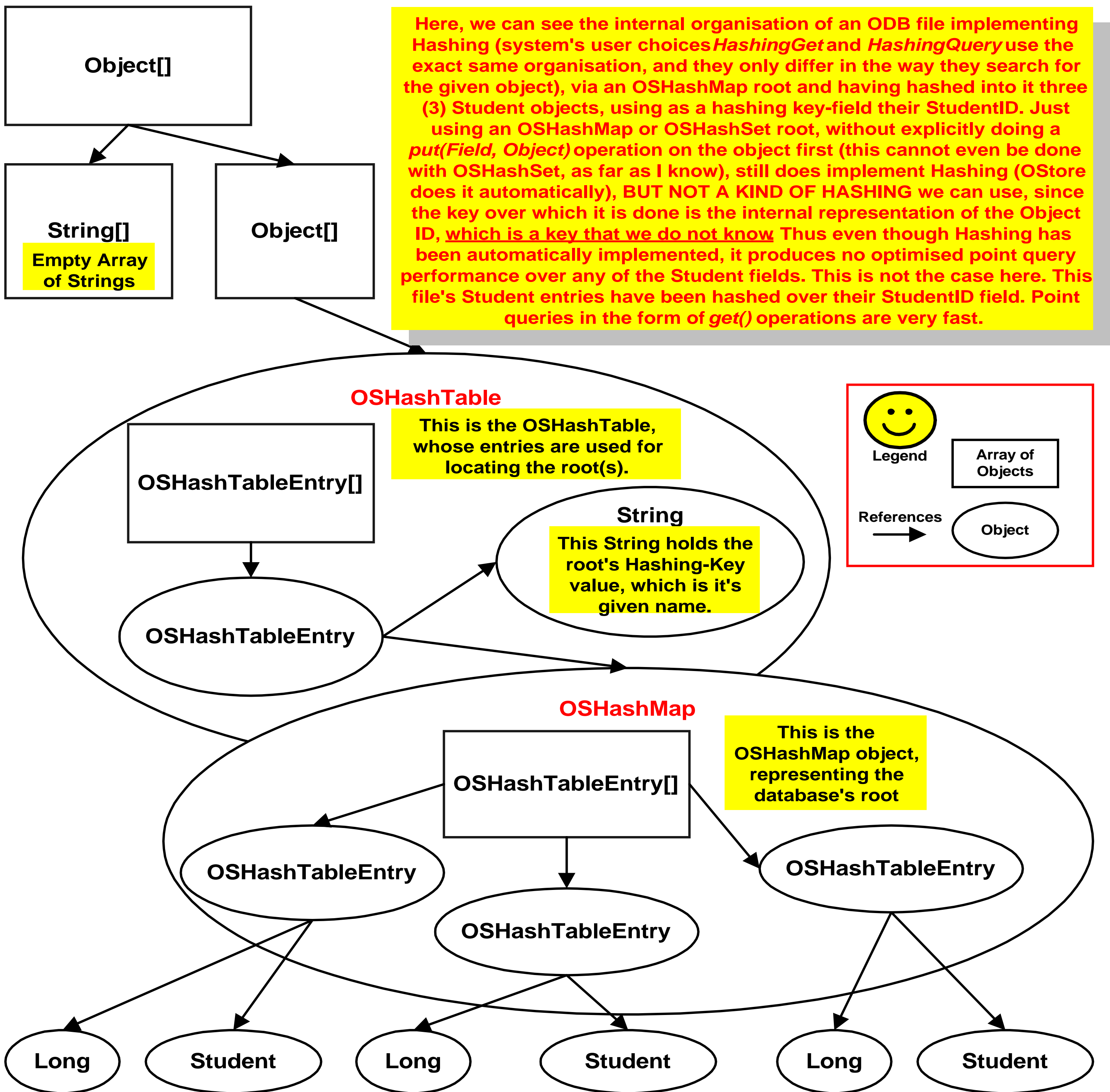
UML



<u>ShowDB for .odb file with OSHashMap root with 3 added objects</u>

```
> cd .
> osjshowdb -showObjs -showData Students.odb


G:\O\BigOne2>osjshowdb -showObjs -showData Students.odb
Name: Students.odb

There is one root:

 Name: Hashing   Type: com.odi.util.OSHashMap


Destroyed Objects: 1

Segment: 0
Size: 3175 (4 Kbytes)

                                     Total
OID        Data Offset  Elements     Bytes  Type

<1|0|0|0>         1240         8        72  java.lang.Object[]
 Contains references to:
  0: <1|0|0|3>
  8: 5
  16: <1|0|0|1>
  24: null
  32: null
  40: null
  48: null
  56: null
<1|0|0|1>         1776         1        16  java.lang.String[]
<1|0|0|2>         1200                  40  com.odi.util.OSHashMap
 Contains references to:
  4: <1|0|0|8>
  20: null
<1|0|0|3>          352         1        16  java.lang.Object[]
 Contains references to:
  0: <1|0|0|7>
<1|0|0|4>          264         7        16  java.lang.String
  Value: "Hashing"
<1|0|0|5>          280                  32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|4>
  8: <1|0|0|2>
  16: null
<1|0|0|6>          144         2        24  com.odi.util.OSHashtableEntry[]
 Contains references to:
  0: <1|0|0|5>
  8: null
<1|0|0|7>          320                  32  com.odi.util.OSHashtable
 Contains references to:
  4: <1|0|0|6>
  20: null
<1|0|0|8>         1152         5        48  com.odi.util.OSHashtableEntry[]
 Contains references to:
  0: <1|0|0|12>
  8: <1|0|0|15>
  16: <1|0|0|18>
  24: null
  32: null
<1|0|0|10>         976                  16  java.lang.Long
<1|0|0|11>         992                  32  Student
<1|0|0|12>        1024                  32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|10>
  8: <1|0|0|11>
  16: null
<1|0|0|13>        1056                  16  java.lang.Long
<1|0|0|14>        1072                  32  Student
<1|0|0|15>        1104                  32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|13>
  8: <1|0|0|14>
  16: null
<1|0|0|16>        1136                  16  java.lang.Long
```

```
<1|0|0|17>         168                   32  Student
<1|0|0|18>         200                   32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|16>
  8: <1|0|0|17>
  16: null
<1|0|0|19>        1344       422       432  java.lang.String
  Value:
"Ccom.odi.util.OSDictionary;{MODITheHashCode;f}Ccom.odi.util.OSHashtable;{Bcom.odi.uti
l.OSDictionary;Mtable;[]Tcom.odi.util.OSHashtableEntry;MnTableEntries;fMreorgThreshold
;fMaltRep;Tcom.odi.util.OSDictionary;}Ccom.odi.util.OSHashMap;{Bcom.odi.util.OSHashtab
le;MmodificationTick;f}Ccom.odi.util.OSHashtableEntry;{Mkey;OMelement;OMnext;Tcom.odi.
util.OSHashtableEntry;Mhash;f}CStudent;{MODITheHashCode;fMID;hMAge;hMCredits;h}"

     Count  Tot Size  Type
             (bytes)

         3        96  Student
         1        40  com.odi.util.OSHashMap
         1        32  com.odi.util.OSHashtable
         4       128  com.odi.util.OSHashtableEntry
         2        72  com.odi.util.OSHashtableEntry[]
         3        48  java.lang.Long
         2        88  java.lang.Object[]
         2       448  java.lang.String
         1        16  java.lang.String[]
```

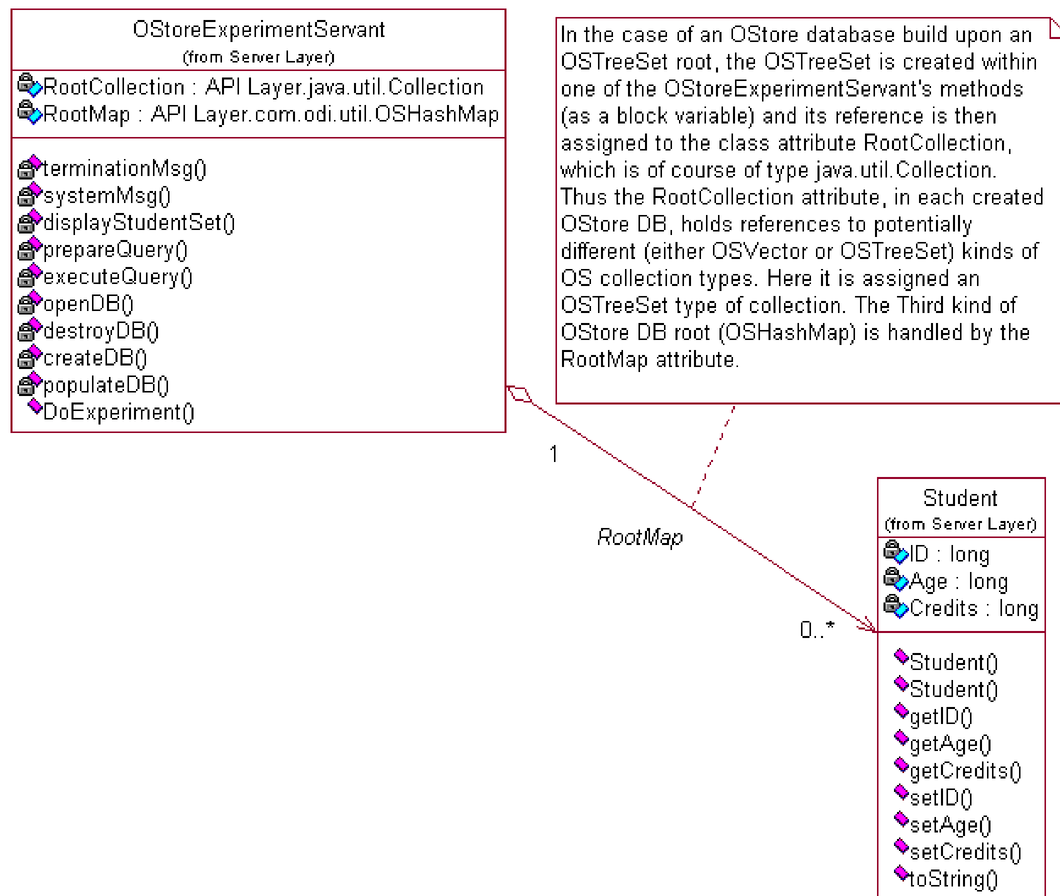


OSHashMap extends OSHashTable and implements the Java Map Interface. It thus has map views that can be queried and it also supports direct hashingput() and get() operations. As we can see, internally it uses an OSHashTableEntry[] Array of OSHashTableEntry objects (just like OSHashTable).

PSE PRO Hashing is roughly implemented as follows: The user supplies the field value (for the hashed field of the persistent object) he is looking for. This value is fed into a hash function that produces an array index number. This index number is the index for the OSHashMap's internal OSHashTableEntry[] array. Since the array objects are stored (virtually) sequentially, it is very fast to find the array element. All OStore does is that it multiplies the index number it got from the hash function by the OSHashTableEntry byte size or the defined bucket size and it thus gets the exact address of the element it is looking for. Once there (within RAM-speed access time, since after a couple of queries the OStore DB has been mapped onto the computer's Virtual memory), it just compares the given key to the stored key (Stored in the same OSHashTableEntry as the Student object). If they are the same the stored student object is fetched, otherwise there has been a collision there and thus the proper collision-handling agorithm is followed, which in the case of OStore, I believe it is sequential search of the next array elements. Note that, using a big enough number of buckets (even as big as the number of expected elements) can make *get()* really "fly", but iterations over the stored objects will be slow.

OSTreeSet Database Schema

UML

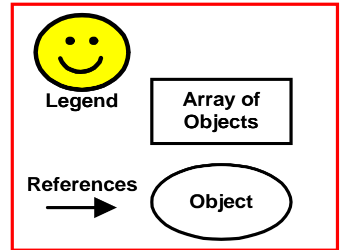


<u>ShowDB for .odb file with OSTreeSet (and Index added) root with 3 added objects</u>

```
> cd .
> osjshowdb -showObjs -showData Students.odb


G:\O\BigOne2>osjshowdb -showObjs -showData Students.odb
Name: Students.odb

There is one root:

 Name: Indexing   Type: com.odi.util.OSTreeSet


Destroyed Objects: 1

Segment: 0
Size: 21976 (22 Kbytes)

                                     Total
OID        Data Offset  Elements     Bytes  Type

<1|0|0|0>        18784         8        72  java.lang.Object[]
 Contains references to:
  0: <1|0|0|5>
  8: 13
  16: <1|0|0|1>
  24: null
  32: null
  40: null
  48: null
  56: null
<1|0|0|1>        20208         1        16  java.lang.String[]
<1|0|0|2>        14528                 104  com.odi.util.BTreeImpl
 Contains references to:
  8: <1|0|0|3>
  16: null
  40: null
  48: null
<1|0|0|3>        14656                4096  com.odi.util.BTreeNode4ByteKey
 Contains references to:
  0: null
  4: null
  2052:: 18
  2056:: 19
  2060:: 20
  2064: null
  2068: null
  ...more nulls due to default size
<1|0|0|4>         4784                  24  com.odi.util.OSTreeSet
 Contains references to:
  0: <1|0|0|2>
  8: <1|0|0|16>
<1|0|0|5>         4608         1        16  java.lang.Object[]
 Contains references to:
  0: <1|0|0|13>
<1|0|0|6>          264         8        16  java.lang.String
  Value: "Indexing"
<1|0|0|7>         4416                 104  com.odi.util.BTreeImpl
 Contains references to:
  8: <1|0|0|8>
  16: null
  40: null
  48: null
<1|0|0|8>        10432                4096  com.odi.util.BTreeNode8ByteKey
 Contains references to:
  0: null
  4: null
  2732:: 18
  2736:: 19
  2740:: 20
  2744: null
  2748: null
  ...more nulls due to default size of structure
<1|0|0|9>          144                  24  com.odi.util.BTreeIndex
 Contains references to:
  0: <1|0|0|7>
```

```
  8: <1|0|0|10>
<1|0|0|10>        280                   24  com.odi.util.Path
 Contains references to:
  0: Student
  8: getID()
<1|0|0|11>       4520                   32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|6>
  8: <1|0|0|4>
  16: null
<1|0|0|12>       4552         2         24  com.odi.util.OSHashtableEntry[]
 Contains references to:
  0: <1|0|0|11>
  8: null
<1|0|0|13>       4576                   32  com.odi.util.OSHashtable
 Contains references to:
  4: <1|0|0|12>
  20: null
<1|0|0|14>       4696                   32  com.odi.util.OSHashtableEntry
 Contains references to:
  0: <1|0|0|10>
  8: <1|0|0|9>
  16: null
<1|0|0|15>       4728         2         24  com.odi.util.OSHashtableEntry[]
 Contains references to:
  0: null
  8: <1|0|0|14>
<1|0|0|16>       4752                   32  com.odi.util.OSHashtable
 Contains references to:
  4: <1|0|0|15>
  20: null
<1|0|0|18>      18752                   32  Student
<1|0|0|19>      10384                   32  Student
<1|0|0|20>       4808                   32  Student
<1|0|0|21>      18856      1343       1352  java.lang.String
  Value:
"Ccom.odi.util.BTree;{}Ccom.odi.util.BTreeImpl;{Bcom.odi.util.BTree;Msize;fMmodificati
ons;fMtop;Tcom.odi.util.BTreeNode;MfreeNode;Tcom.odi.util.BTreeNode;MpageSize;dMmaxNod
eEntries;dMkeySize;bMfixedSizeKeys;kMduplicateKeys;kMextraByte;bMextraInt1;fMextraInt2
;fMextraObject1;OMextraObject2;OMinserts;fMoverflows;fMnodesAllocated;fMnodesAllocated
FromCache;fMheightIncreases;fMremoves;fMunderflows;fMnodesReturned;fMheightDecreases;f
MkeyOverflows;f}Ccom.odi.util.BTreeNode;{MnextLeaf;LMprevLeaf;LMnumEntries;dMflags;d}C
com.odi.util.BTreeNode4ByteKey;{Bcom.odi.util.BTreeNode;Mkeys;E[2040]bMvalues;E[510]L}
Ccom.odi.util.OSAbstractSet;{}Ccom.odi.util.OSTreeSet;{Bcom.odi.util.OSAbstractSet;Mtr
ee;Tcom.odi.util.BTree;Mindexes;Tcom.odi.util.OSDictionary;}Ccom.odi.util.BTreeNode8By
teKey;{Bcom.odi.util.BTreeNode;Mkeys;E[2720]bMvalues;E[340]L}Ccom.odi.util.BTreeIndex;
{Mtree;Tcom.odi.util.BTree;Mpath;Tcom.odi.util.Path;MisOrdered;kMhasDuplicates;k}Ccom.
odi.util.Path;{MclassString;[]oMpathString;[]o}Ccom.odi.util.OSDictionary;{MODITheHash
Code;f}Ccom.odi.util.OSHashtable;{Bcom.odi.util.OSDictionary;Mtable;[]Tcom.odi.util.OS
HashtableEntry;MnTableEntries;fMreorgThreshold;fMaltRep;Tcom.odi.util.OSDictionary;}Cc
om.odi.util.OSHashtableEntry;{Mkey;OMelement;OMnext;Tcom.odi.util.OSHashtableEntry;Mha
sh;f}CStudent;{MODITheHashCode;fMID;hMAge;hMCredits;h}"

     Count  Tot Size  Type
             (bytes)

         3        96  Student
         2       208  com.odi.util.BTreeImpl
         1        24  com.odi.util.BTreeIndex
         1      4096  com.odi.util.BTreeNode4ByteKey
         1      4096  com.odi.util.BTreeNode8ByteKey
         2        64  com.odi.util.OSHashtable
         2        64  com.odi.util.OSHashtableEntry
         2        48  com.odi.util.OSHashtableEntry[]
         1        24  com.odi.util.OSTreeSet
         1        24  com.odi.util.Path
         2        88  java.lang.Object[]
         2      1368  java.lang.String
         1        16  java.lang.String[]
```

Here, we can see the internal organisation of an ODB file with an OSTreeSet root and no added Index structure, with three (3) Student objects added



Object[]

String[]

Empty Array of Strings

Object[]

OSHashTable

This is the OSHashTable, whose entries are used for locating the root(s).

OSHashTableEntry[]

String

This String holds the root's Key value, which is it's given name.

OSHashTableEntry

OSTreeSet

This is the OSTreeSet object, representing the database's root. OSTreeSet implements indexing structures giving us the potential to index (i.e. optimise) a specific Student field.

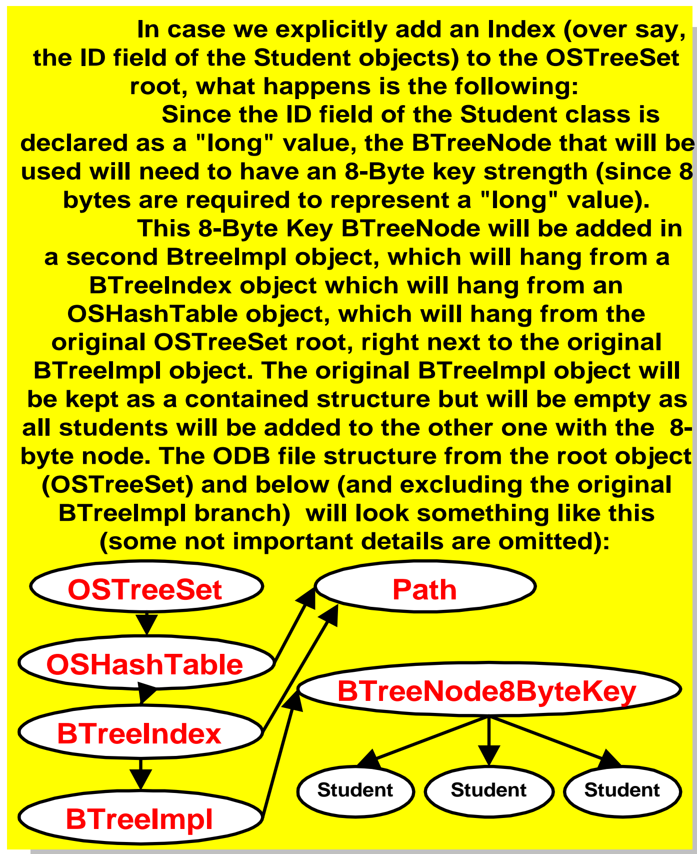
odi.util.BTreeImpl

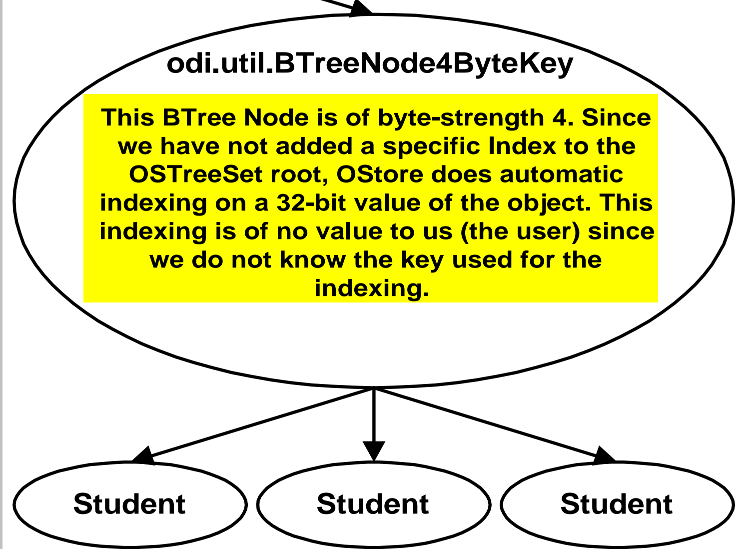
An OSTreeSet is basically a B-Tree structure and this is ODI's B-Tree implementation class. It references BTreeNodes of various Key strengths.

In case we explicitly add an Index (over say, the ID field of the Student objects) to the OSTreeSet root, what happens is the following:

Since the ID field of the Student class is declared as a "long" value, the BTreeNode that will be used will need to have an 8-Byte key strength (since 8 bytes are required to represent a "long" value).

This 8-Byte Key BTreeNode will be added in a second BtreeImpl object, which will hang from a BTreeIndex object which will hang from an OSHashTable object, which will hang from the original OSTreeSet root, right next to the original BTreeImpl object. The original BTreeImpl object will be kept as a contained structure but will be empty as all students will be added to the other one with the 8-byte node. The ODB file structure from the root object (OSTreeSet) and below (and excluding the original BTreeImpl branch) will look something like this (some not important details are omitted):





All information about how the OStore does it's business have been derived through the OStore Online documentation, the *osjshowdb* tool, reverse engineering ODI classes, studying the Java source code of used JAVA classes (e.g. HashTable, Map, etc., thanks Sun for being so Open!), and some experimentation.

Rationale of Experiments

I had some difficulty determining what was actually asked from us; Was it:

- to compare Point queries with Range queries when optimisations were used and not used? (with Range queries designed so they would return a single object, e.g. ID >=5000 && ID <= 5000), or was it:
- to compare Point queries with Point queries and Range queries with Range queries when optimisations were used or not?

The way this particular point is phrased in the assignment handout, leaves some doubt as to what is really meant.

Of course, once one considers that comparing Range with Point queries is a bit like comparing Apples with Oranges (as they are used differently – returning multiple elements - and in different occasions) and the fact that such a comparison would not produce any results suitable for interesting comments (i.e. all that could one say is there are no significant differences– or no differences at all), it is clear I think that this assignment is about the second possible interpretation given above.

This is what the experiments that follow try to establish, plus a little extra insight that should be welcome in any case.

So, here we go…

Select (Non-Pick) Collection queries, whether point or range, have at least one basic thing in common: they require a complete collection traversal (or iteration if you prefer) in order to complete. Pick Collection queries return the first result they find, and they thus sometimes have unpredictably varied performance (e.g. in Collections implemented as BTrees) and some other times predictably varied performance (e.g. in ordered collections - Vectors).

Based on this fundamental fact, we will conduct a series of experiments involving Pick and Select (non-Pick) Point and Range queries (with Range queries not implemented as Pick, in any case, because they are asked to return multiple results) over variations/combinations of the following factors:

- ♦ Three different kinds of root OS Collections (Four if you count OSTreeSet with Index added and without as two).
- ♦ Three different sizes (Small, Medium, and Large) for every used collection.
- ♦ Three different places (Start, Middle, and End) where Point queries are applied in each case.

Through this set-up, the rich set of resulting experiment configurations guarantees a body of experiment results that will be adequate for insightful drawn conclusions on the subject of our treatise.

Experiment Design

Collection Selection

The assignment specifically requests that our experiments be centred on comparing Range with Point queries when:

- No Hashing or Indexing Collection optimisation is used.
- Hashing is used.
- Indexing is used.

This means that we should find:

- An OS Collection type that permits queries and implements no optimisation at all.
- An OS Collection type that permits queries and can use Hashing.
- An OS Collection type that permits queries and can have Indexes added.

Right away one can identify inherent problems in these requirements. Specifically:

1. In case a collection type has the option to be optimised or not, (say, have an Index added or not), should it be used as both our selection for the non-optimised OS Collection type and for the optimised one? Or, should we have separate Collection types for each case?
2. Are there Collections that meet our optimisation requirements and at the same time permit queries to be performed on them?
3. Hashing optimisation is problematic in the query context, i.e. it is not useful for Range queries yet it is potentially very fast for Point queries; but how can we implement a true Hashed Point query when Collection queries involve a complete traversal of the collection? (Pick queries are different to that end and we will see what happens in their case)

The obvious route to be followed in our quest for the right kind of Collections would be to check the OStore documentation, find the offered kinds of collections, and then begin ruling out some of them based on whether they support queries and the right kind of optimisation options. Fortunately, the OS documentation is very comprehensive and offers a nice section on specifically this issue; namely "How to Choose a Collections Alternative". In this section the user will find the following table which summarises the various offered Collection alternatives and their characteristics.

Collection Class	Ordered/ Unordered	Duplicates/ No Duplicates	Quick Look-Up	Comparison Operations	Queries Allowed	Collection Size
OSHashBag	Unordered	Duplicate values allowed	Object look-up	Identity based	Can query	Medium
OSHashMap	Unordered	Duplicate values allowed No duplicate keys	Key look-up	Content based	No queries	Medium
OSHashSet	Unordered	No duplicates	Object look-up	Content based	Can query	Medium
OSHashtable	Unordered	Duplicate values allowed No duplicate keys	Key look-up	Identity based	No queries	Medium
OSTreeMap *xxx*	Ordered by key	Duplicate values allowed No duplicate keys	Key look-up	Content based	No queries	Large
OSTreeSet	Unordered	No duplicates	Object look-up	Content based	Can query and index	Large
OSVector	Ordered	Duplicate values allowed	None	Identity based	Can query	Small, medium
OSVectorList	Ordered	Duplicate values allowed	None	Content based	Can query	Small, medium

We are also informed that "the **OSHashtable** class is not compatible with the JDK 1.2 API" and that "all other collections in **com.odi.util** are compatible with the JDK 1.2 API."

The Collection Size field informs us of the recommended Collection size, e.g. The OSTreeSet Collection is designed to perform better than other Collections when size is big. According to the PSE documentation, definition of Collection size is roughly as follows:

- Small collections have fewer than 100 elements.
- Medium collections contain as many as 10,000 elements
- Large collections have more than 10,000 elements.

In our experiments we will follow this guideline closely, only changing the size of a small collection to 1,000 elements as fewer elements produce timing results that are so small that they are rendered unreliable.

Ruling out those collections that cannot be queried, we come up with the following table:

Collection Class	Ordered/ Unordered	Duplicates/ No Duplicates	Quick Look-Up	Comparison Operations	Queries Allowed	Collection Size
OSHashBag	Unordered	Duplicate values allowed	Object look-up	Identity based	Can query	Medium
OSHashSet	Unordered	No duplicates	Object look-up	Content based	Can query	Medium
OSTreeSet	Unordered	No duplicates	Object look-up	Content based	Can query and index	Large
OSVector	Ordered	Duplicate values allowed	None	Identity based	Can query	Small, medium
OSVectorList	Ordered	Duplicate values allowed	None	Content based	Can query	Small, medium

The problems with this table are:

- Collections that do not provide any inherent optimisation mechanism, or Quick Look-Up method as it is termed by the user-friendly PSE, (i.e. OSVector and OSVectorList) are ordered and allow duplicates while the ones that do implement optimisation techniques are unordered and do not allow duplicates (with the exception of OSHashBag). This means that if we decide to use the ordered Collections in our experiments we should be wary of where in the collection (i.e. start, middle, or end) the value we are looking for resides and we should also make sure that in all experiments (even those involving the duplicate allowing Vector type of collections) the stored, within the collection, objects are unique anyway, so that the experiments are performed over the same eperimental conditions in any case.
- Those collections internally implementing hashing (i.e. OSHashBag and OSHashSet) do not permit the user to do explicit hashing over a certain persistent object's attribute. This means that although hashing is used, it is not the hashing we would want for our experiments since it is done using as hash-key the OID value of the object, which we do not know, we are not supposed to know and we cannot use for quick single-object retrievals, so that we can compare the time the retrieval takes with say the time point queries over indexed collections take. **Comparing hashing with indexing, when no explicit hashing over an object attribute is performed (while on the other hand, for the indexed collection a specific index for one or more of its attributes has been added), would not be experimentally sane.**

So, what we need here, is an OS collection that:

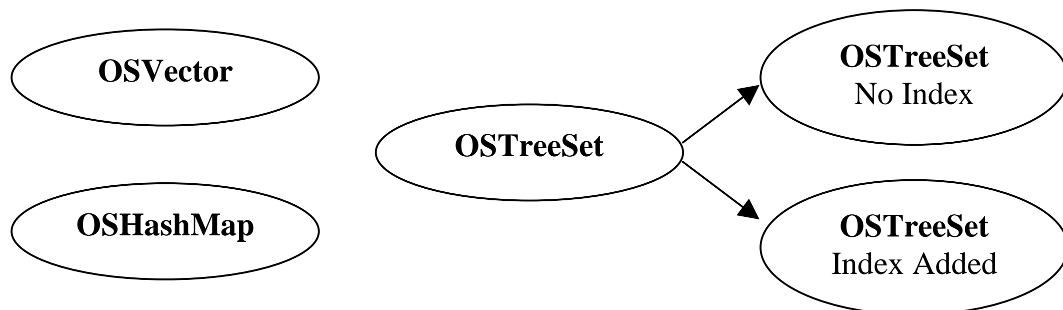
- Allows queries.
- Allows explicit hashing (or key look-ups, as PSE documentation terms it)

Fortunately, there is at least one OS Collection that does just that and it is **OSHashMap.** Although, the OS documentation says that it cannot be queried the user can kind of override that by querying the Map's views (i.e. the value pairs of [key,value]). This is way-cool and it solves our second problem.

The first problem will be solved by proper experiment design. Specifically:

- Values to be looked for will be specifically picked from the start, middle or end of collection, so we can see the difference ordering, of objects within a collection, makes.
- Created persistent objects will be unique.
- We will put in the "mix", an OS collection type that will be used both with and without its optimisation option "activated". Also, collections that can be optimised will be queried over optimised and non-optimised object attributes (when their optimisation option is activated).

Considering all of the above, 3 different collections types (one of them in 2 variations) have been chosen as the basis for our OS query performance experiments. In all of our experiments when our Collections are optimised they are optimised over the "ID" Student Attribute (although the system easily permits optimisation over a different Student object Attribute through a different User Interface option). The following figure presents the selected, for our experiments, OS Collections:



Experiment Design Prime Directives

Given the above and based on common sense and generally accepted as valid experimental method, we believe that the requirements (directives) guiding the design of our experiments should at least be the following:

- **Thoroughness**
  - Many experiments
  - Experiment configurations varying over all relevant collection aspects.
- **Precision**
  - Multiple timings.
  - Averaging of the timing values for each experiment configuration.
- **Integrity**
  - All experiments should be performed on the same system with the same standard processing load (same background processes running every time).
  - Thorough understanding (as much as it is "studently" possible) of underlying technologies so that external to the experiments factors can be recognised and extracted and/or compensated accordingly.
- **Experiments should accurately reflect their fundamental purpose (their cause of inception), as stated in the assignment handout.** Specifically, they should offer result data that will enable the experimenter to:
  - Compare the performance of the same kind of queries (select-point, pick-point, select-range), looking for the same object(s) on same size **but different kind of collections**, implementing (or not) various optimisation options.
  - Compare the performance of **various queries (say Pick with non-Pick Point queries)**, looking for the same object, over the exact same collection (same size, same root type, same optimisation options).
  - Compare the performance of the same query over the same collection, looking for a **different object**.
  - Compare the performance of the same query over the same collection looking for the same object but using a **different attribute** of that object (e.g. Age instead of StudentID)
- They should adequately map the used Collections' characteristics and intricacies to concrete timing results that can be commented on.
- The ObjectStore database should be designed in such a way as to help the experimentation directives described above.

Mapping Experiment Requirements to System Requirements

Integrity

Only one OS database is allowed each time on the server side. This is because we do not really know how OS manages its Virtual Memory when there are multiple .odb files involved. (i.e. issues like the amount of virtual memory granted to each file according to its size, or not, add complexity to our experiments that does not really help the prime directive of “Experiment Integrity”). Thus I have decided that these complications are better avoided for the sake of experiment integrity.

First time (after system boot) that OS searches a database file (3 files actually, .odb .odt .odf) while performing a query, it actually reads objects off the hard disk and maps them to Virtual Memory pages and of course also checks them with the issued query. All subsequent times that database is searched (until the next system boot) OS does not have to involve the hard disk in the operation (except if the database is very-very big) as all of it is actually held in RAM virtual memory. Not all virtual memory pages are held in RAM of course (most of it is usually held on the hard disk’s swap file), but in our system the portion of RAM virtual memory managed by ObjectStore proved to be enough to hold the whole of even the largest of the databases we used.

Also note that whether we terminate and restart the Java Virtual Machine, from where the OS PSE process originally spawned (or even terminate the DOS session from which the JVM was run and then start a new one), does not cause OStore to re-map the database to virtual memory. This is because OStore works on a lower system level, and thus each time we re-run it, it actually regains the handle of its reserved portion of the system’s Virtual Memory pool. Of course this is one of the big advantages of using OStore.

The significance that this advantage bears for our experiments is that the very first (after system reboot) timing result involving a pre-existing database or the very first timing result involving a newly created database should be ignored as they actually involve a hard disk operation which as we know involves all kinds of external factors (e.g. fragmentation, competing I/O operations, etc.) that do not make our timing result very reliable and thus do not help the prime directive of experiment integrity. Bear in mind here that if the OStore database, existing before the newly created one, is **exactly the same as the one we ask the system to create effectively destroying the old one,** then OStore will just use the old Virtual Memory and the first timing of a query on the newly created database will not be any slower than the next ones.

Collection queries of the *select()* kind involve a complete collection element traversal, while queries of the *pick()* kind return the first result they get as soon as they get it, thus not needing a complete collection traversal in order to finish. It is obvious that under normal database use, pick queries are only useful when:

- **The database elements are all unique** (they usually are; e.g. our Student objects are all unique although we could have made them to not be so).
- **When a Point query is used.** This is because in normal situations, Range queries do not involve queries that are so constructed that will return a single element (e.g. ID <= 5000 && ID >= 5000)

In our experiments we have a *pick()* query experiment configuration corresponding to each experiment configuration that involves a Point query. Although pick queries can also be used with Range queries, we did not deem that necessary for

the purpose of our experiments (since we want our Range queries to return multiple objects).

Thouroughness & Precision

- More than 130 different experiment configurations covering as many, of the discussed issues, as possible.
- 3 separate timings of each experiment, all stored in our Access database "ExperimentResults" table.
- Experiment results statistics output by the system present the user with the average time each experiment took over all of the 3 separate executions.
- All experiments performed on three different sizes of Ostore databases:
    - Small: 1,000 Student objects
    - Medium: 10,000 Student objects
    - Large: 60,000 Student objects
- All queries performed three times over each collection type and size for evenly distributed values (over the range of possible values for each case). Specifically a query is performed for a value at:
    - Start of collection
    - Middle of collection
    - End of collection

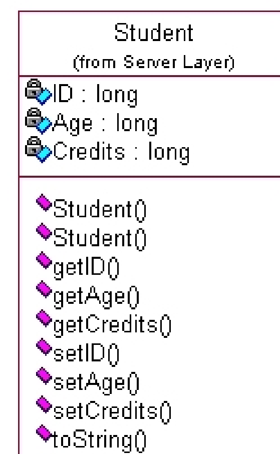
ObjectStore Database Design

The created OS databases record data about a number of alien (Extra Terrestrial) Students that:

- Can live for millennia.
- Have no names.
- Are very smart (can register for thousands of college credits and still pass all courses).
- Multiply like rabbits

Human Students were not preferred, as their data did not serve well our experimental purposes. The following paragraphs explain why.

The OS databases consist of Student objects that are created so that:



The Student Class

- All attribute values are unique throughout the database. This means that no Student has the same "ID", the same "Age" and the same "Credits" as any other Student within the database. This is done in order to ensure that any kind of Point query over any attribute will only have a single possible returned Student object. We want that, so that it is experimentally permissible for Select Point and Pick Point query completion times to be compared. (i.e. if the possible results were more than one, then one pick query could finish finding one object, and the other finding another object, etc.)

- All fields are not of random value but in fact they are of sequential values. Specifically, the "ID", and "Credits" values of the same Student object are equal. They start with the value of 0 and increase by a value of 1 for each added object. The "Age" value starts off equal to the value of the number of total objects within the database (a number which is user defined through the User Interface), minus 1 since counting starts from 0, and decreases by a value of 1 for each added object (Thus having the value of 0 for the last added object). All this is done so that the "ID" and "Credits" values actually map the object position within the collection (in case of an ordered collection) and in any case reflect the sequence in which objects were added to the OStore database. At the same time the Age attribute is reversed. Also, the fact that all attribute values are not random, means that the user can issue queries to find any existing object through any of its attributes' values because he knows by definition the range of existing values.
- It was decided that the Credits value be equal to the ID value so that we could test, through proper experiment configurations, whether optimisation of say the ID attribute (via an Index or Hashing) would implicitly mean optimisation of the Credits attribute as well ("implicitly" meaning that the involved query would first check the Student object with *ID = 5* if asked to find the Student with *Credits = 5* and the optimised attribute was ID).
- It was also decided that the OStore databases could not be very small (say 100 objects) as they would then produce timing results so small that they would be really vulnerable to external factors. In general, with databases that are not very small, external factors (like competing computer processes or other non-deterministic issues) are less likely to affect the timings, to a degree that would suggest possibly unreliable drawn conclusions.

Other experiment guidelines stemming from the general experiment requirements, were:

- Use Pick and Select (non-Pick) Point and Select Range queries on all kinds of collections used.
- Record time it takes for databases based on different types of root collections to be created, and comment on how expensive each collection creation operation and/or optimisation option is, in terms of creation time required.
- Record disk space needed by each Collection/Optimisation combination for each database size (Small, Medium and Large).
- Check Pick query results for predictability over certain value ranges (Start, Middle, and End of collection), for certain collection types.
- Compare performance of OSVector based databases with OSTreeSet (with no index added) based databases for various queries.

Experiments' Analysis

Analysis of the produced experiment results will take the following form:

A number of tables, containing relevant data, will be presented in each section. After each table a number of comments and conclusions, based on the data presented by it, will follow. At the end of each section some comments and conclusions, correlating the data of all tables within that section, will be offered.

The "NewDB" field should be ignored as it bears no experimental value. The Boolean value within it was used in various places in the *ExperimentConfigurations* Access database table in order to create the new OS database required for the next batch of experiments. The reader should also note that all times are recorded in milliseconds.

Also, from this point on the following semantic mapping is decreed:

NoOpt	Experiment where the root collection is an OSVector
HashingGet	Experiment where the root collection is an OSHashMap and the single objects are retrieved by *get()* operations (instead of queries)
HashingQuery	Experiment where the root collection is an OSHashMap and queries over the map's views are used for retrieval of object(s)
Indexing - None	Experiment where the root collection is an OSTreeSet an no Student attribute is optimised (i.e. no Index is added)
Indexing - ID	Experiment where the root collection is an OSTreeSet an the "ID" Student attribute is optimised (i.e. Index for the "ID" attribute is added)

Section 1: Comparing Pick Point Queries

NoOpt

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
2	ID	NoOpt	None	1000	0	-1	-1	Yes	No	36.66666
5	ID	NoOpt	None	1000	500	-1	-1	Yes	No	56.66666
8	ID	NoOpt	None	1000	999	-1	-1	Yes	No	223.3333
11	ID	NoOpt	None	10000	0	-1	-1	Yes	No	53.33333
14	ID	NoOpt	None	10000	5000	-1	-1	Yes	No	1017.5
17	ID	NoOpt	None	10000	9999	-1	-1	Yes	No	1776.666
20	ID	NoOpt	None	60000	0	-1	-1	Yes	No	110
23	ID	NoOpt	None	60000	30000	-1	-1	Yes	No	5293.333
26	ID	NoOpt	None	60000	59999	-1	-1	Yes	No	10400

- Pick query time, over OSVector collections (and ordered collections in general) is directly proportional to the element's position within the collection.
- In larger OSVector collections, Pick query time for the first collection element (Student ID = 0), increases slightly as collection size increases.

HashingQuery

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
29	ID	HashingQuery	getID()	1000	0	-1	-1	Yes	No	76.66666
32	ID	HashingQuery	getID()	1000	500	-1	-1	Yes	No	493.3333
35	ID	HashingQuery	getID()	1000	999	-1	-1	Yes	No	586.6666
38	ID	HashingQuery	getID()	10000	0	-1	-1	Yes	No	90
41	ID	HashingQuery	getID()	10000	5000	-1	-1	Yes	No	10270
44	ID	HashingQuery	getID()	10000	9999	-1	-1	Yes	No	21036.66
47	ID	HashingQuery	getID()	60000	0	-1	-1	Yes	No	493.3333
50	ID	HashingQuery	getID()	60000	30000	-1	-1	Yes	No	89603.33
53	ID	HashingQuery	getID()	60000	59999	-1	-1	Yes	No	51003.33
146	Age	HashingQuery	getID()	1000	0	-1	-1	Yes	No	620
149	Age	HashingQuery	getID()	1000	500	-1	-1	Yes	No	680
152	Age	HashingQuery	getID()	1000	999	-1	-1	Yes	No	53.33333
155	Age	HashingQuery	getID()	10000	0	-1	-1	Yes	No	23070
158	Age	HashingQuery	getID()	10000	5000	-1	-1	Yes	No	10490
161	Age	HashingQuery	getID()	10000	9999	-1	-1	Yes	No	90
164	Age	HashingQuery	getID()	60000	0	-1	-1	Yes	No	48866.66
167	Age	HashingQuery	getID()	60000	30000	-1	-1	Yes	No	94115
170	Age	HashingQuery	getID()	60000	59999	-1	-1	Yes	No	672.5

- OSHashMap collections show some signs of ordering, in the sense that Picks at the start of the collections have a very small result time relative to Picks at the end of the collection. This is not due to ordering but due to the hashing algorithm used (like the OS documentation says, "the OSHashMap collection is unordered"). As we can see Picks over the Age attribute produce roughly reversed results, as is expected since Age values are reversed (i.e. if we have 10 Students in our collection, then if a Student has ID = 0 , then Age = 9). This can further reinforce

the illusion of ordering, but the middle element retrieval times verify that the collection is in fact unordered, since the times recorded there do not follow the "ordered collection" pattern.

- For the Large collection (i.e. 60,000 objects) the middle element retrieval time is the slowest because of the massive hashing collisions in the middle of the collection (the usual hashing collision pattern), and thus the many overflow buckets that have to be searched.

HashingGet

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
136	ID	HashingGet	getID()	1000	0	-1	-1	No	Yes	16.66666
137	ID	HashingGet	getID()	1000	500	-1	-1	No	No	50
138	ID	HashingGet	getID()	1000	999	-1	-1	No	No	10
139	ID	HashingGet	getID()	10000	0	-1	-1	No	Yes	70
140	ID	HashingGet	getID()	10000	5000	-1	-1	No	No	56.66666
141	ID	HashingGet	getID()	10000	9999	-1	-1	No	No	36.66666
142	ID	HashingGet	getID()	60000	0	-1	-1	No	Yes	346.6666
143	ID	HashingGet	getID()	60000	30000	-1	-1	No	No	363.3333
144	ID	HashingGet	getID()	60000	59999	-1	-1	No	No	403.3333

- get() operations over the hashed attribute are very fast for small and medium collections, but they get relatively slow for Large collections due to massive collisions for the default number of buckets. Still, all retrieval times are less than half a second.

Indexing – None

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
56	ID	Indexing	None	1000	0	-1	-1	Yes	No	786.6666
59	ID	Indexing	None	1000	500	-1	-1	Yes	No	566.6666
62	ID	Indexing	None	1000	999	-1	-1	Yes	No	546.6666
65	ID	Indexing	None	10000	0	-1	-1	Yes	No	8310
68	ID	Indexing	None	10000	5000	-1	-1	Yes	No	2690
71	ID	Indexing	None	10000	9999	-1	-1	Yes	No	3146.666
74	ID	Indexing	None	60000	0	-1	-1	Yes	No	17886.66
77	ID	Indexing	None	60000	30000	-1	-1	Yes	No	58840
80	ID	Indexing	None	60000	59999	-1	-1	Yes	No	41323.33

- Retrieval times between start, middle, and of the collection (and in general for any value), follow an unpredictable pattern due to the Btree structure used.

Indexing - ID

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
83	ID	Indexing	getID()	1000	0	-1	-1	Yes	No	73.33333
86	ID	Indexing	getID()	1000	500	-1	-1	Yes	No	315
89	ID	Indexing	getID()	1000	999	-1	-1	Yes	No	67.5
92	ID	Indexing	getID()	10000	0	-1	-1	Yes	No	36.66666
95	ID	Indexing	getID()	10000	5000	-1	-1	Yes	No	130
98	ID	Indexing	getID()	10000	9999	-1	-1	Yes	No	56.66666
101	ID	Indexing	getID()	60000	0	-1	-1	Yes	No	36.66666
104	ID	Indexing	getID()	60000	30000	-1	-1	Yes	No	36.66666
107	ID	Indexing	getID()	60000	59999	-1	-1	Yes	No	57.5
110	Age	Indexing	getID()	1000	0	-1	-1	Yes	No	640
113	Age	Indexing	getID()	1000	500	-1	-1	Yes	No	910
116	Age	Indexing	getID()	1000	999	-1	-1	Yes	No	916.6666
119	Age	Indexing	getID()	10000	0	-1	-1	Yes	No	3956.666
122	Age	Indexing	getID()	10000	5000	-1	-1	Yes	No	8053.333
125	Age	Indexing	getID()	10000	9999	-1	-1	Yes	No	3956.666
128	Age	Indexing	getID()	60000	0	-1	-1	Yes	No	7926.666
131	Age	Indexing	getID()	60000	30000	-1	-1	Yes	No	12393.33
134	Age	Indexing	getID()	60000	59999	-1	-1	Yes	No	52216.66

- Pick times are virtually unaffected by the size of the collection.
- Result times are very fast in all cases.
- Queries over the unoptimised attribute “Age” produce times that are roughly the same as the ones over OSTreeSet with no Index added. (The times are not quite the same because of the Btree stucture’s unpredictability).
- Pick queries over the unoptimised “Age” attribute are, of course, slower than Pick queries over the optimised “ID” attribute (they better be too as if they are not the experimenter may have several sequential anxiety attacks! ☺).

<u>End of Section 1 Conclusions</u>

- OSHashMap Pick Queries are much slower than OSVector Pick queries, for all sizes and all collection positions. Of course the *get()* operations are a different story.
- Get operations over OSHashMap are faster than Pick queries over OSVector and OSTreeSet for small and medium collections, but in Large collections OSTreeSet (with Index added) has faster Pick query times than OSHashMap *get()* operations.
- Pick queries over OSTreeSet, with no Index added, are about as slow as Pick queries over OSHashMap.
- Pick queries over the indexed attribute in OSTreeSet collections are, of course, much faster than Pick queries over the same attribute but in OSTreeSet collections with no Index added.

Section 2: Comparing Select (non-Pick) Point Queries

NoOpt

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
1	ID	NoOpt	None	1000	0	-1	-1	No	Yes	223.3333
4	ID	NoOpt	None	1000	500	-1	-1	No	No	203.3333
7	ID	NoOpt	None	1000	999	-1	-1	No	No	180
10	ID	NoOpt	None	10000	0	-1	-1	No	Yes	1830
13	ID	NoOpt	None	10000	5000	-1	-1	No	No	1787.5
16	ID	NoOpt	None	10000	9999	-1	-1	No	No	1776.666
19	ID	NoOpt	None	60000	0	-1	-1	No	Yes	10783.33
22	ID	NoOpt	None	60000	30000	-1	-1	No	No	10046.66
25	ID	NoOpt	None	60000	59999	-1	-1	No	No	10506.66

No special comments in this section.

HashingQuery

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
28	ID	HashingQuery	getID()	1000	0	-1	-1	No	Yes	623.3333
31	ID	HashingQuery	getID()	1000	500	-1	-1	No	No	566.6666
34	ID	HashingQuery	getID()	1000	999	-1	-1	No	No	550
37	ID	HashingQuery	getID()	10000	0	-1	-1	No	Yes	21406.66
40	ID	HashingQuery	getID()	10000	5000	-1	-1	No	No	21420
43	ID	HashingQuery	getID()	10000	9999	-1	-1	No	No	21900
46	ID	HashingQuery	getID()	60000	0	-1	-1	No	Yes	112136.6
49	ID	HashingQuery	getID()	60000	30000	-1	-1	No	No	124955
52	ID	HashingQuery	getID()	60000	59999	-1	-1	No	No	130683.3
145	Age	HashingQuery	getID()	1000	0	-1	-1	No	Yes	626.6666
148	Age	HashingQuery	getID()	1000	500	-1	-1	No	No	710
151	Age	HashingQuery	getID()	1000	999	-1	-1	No	No	643.3333
154	Age	HashingQuery	getID()	10000	0	-1	-1	No	Yes	22733.33
157	Age	HashingQuery	getID()	10000	5000	-1	-1	No	No	21330
160	Age	HashingQuery	getID()	10000	9999	-1	-1	No	No	21206.66
163	Age	HashingQuery	getID()	60000	0	-1	-1	No	Yes	130560
166	Age	HashingQuery	getID()	60000	30000	-1	-1	No	No	131293.3
169	Age	HashingQuery	getID()	60000	59999	-1	-1	No	No	132260

- Select queries over the optimised attribute “ID” and the unoptimised (and reversed) “Age” produce roughly the same timing results in all cases.

Indexing – None

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
55	ID	Indexing	None	1000	0	-1	-1	No	Yes	880
58	ID	Indexing	None	1000	500	-1	-1	No	No	880
61	ID	Indexing	None	1000	999	-1	-1	No	No	896.6666
64	ID	Indexing	None	10000	0	-1	-1	No	Yes	10220
67	ID	Indexing	None	10000	5000	-1	-1	No	No	9190
70	ID	Indexing	None	10000	9999	-1	-1	No	No	10330
73	ID	Indexing	None	60000	0	-1	-1	No	Yes	48280
76	ID	Indexing	None	60000	30000	-1	-1	No	No	66700
79	ID	Indexing	None	60000	59999	-1	-1	No	No	58750

No Special Comments in this section.

Indexing – ID

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
82	ID	Indexing	getID()	1000	0	-1	-1	No	Yes	56.66666
85	ID	Indexing	getID()	1000	500	-1	-1	No	No	165
88	ID	Indexing	getID()	1000	999	-1	-1	No	No	122.5
91	ID	Indexing	getID()	10000	0	-1	-1	No	Yes	110
94	ID	Indexing	getID()	10000	5000	-1	-1	No	No	123.3333
97	ID	Indexing	getID()	10000	9999	-1	-1	No	No	93.33333
100	ID	Indexing	getID()	60000	0	-1	-1	No	Yes	33.33333
103	ID	Indexing	getID()	60000	30000	-1	-1	No	No	33.33333
106	ID	Indexing	getID()	60000	59999	-1	-1	No	No	40
109	Age	Indexing	getID()	1000	0	-1	-1	No	Yes	893.3333
112	Age	Indexing	getID()	1000	500	-1	-1	No	No	1063.333
115	Age	Indexing	getID()	1000	999	-1	-1	No	No	916.6666
118	Age	Indexing	getID()	10000	0	-1	-1	No	Yes	10065
121	Age	Indexing	getID()	10000	5000	-1	-1	No	No	9706.666
124	Age	Indexing	getID()	10000	9999	-1	-1	No	No	9376.666
127	Age	Indexing	getID()	60000	0	-1	-1	No	Yes	60453.33
130	Age	Indexing	getID()	60000	30000	-1	-1	No	No	57636.66
133	Age	Indexing	getID()	60000	59999	-1	-1	No	No	54810

- There are, of course, big differences in times when the collection (of any size) is queried over the optimised “ID” attribute and the unoptimised “Age” attribute.

End of Section 2 Conclusions

- Since all the above are Select (non-Pick) queries, it comes as no surprise that position (start, middle or end of collection) of the searched element makes no difference in the timing result. **This is because all the collection elements have to be traversed anyway.**
- Also, it is not surprising that in all cases, the bigger the collection, the longer the time the query takes, since it involves more traversed elements and more comparison operations.
- Slowest (except in small collections – there they are third fastest) are OSHashMap queries.
- Fastest queries, in all cases, are within the OSTreeSet collection with Index added and over the optimised attribute of course.
- Second fastest are the OSVector queries (in all cases).
- For small collections, OSHashMap queries are faster than OSTreeSet (with no Index added) queries, or queries in OSTreeSet collections with Index added, but over an unoptimised attribute.
- Except in small collections, third fastest queries are the ones in OSTreeSet collections with no Index added.
- Queries in OSTreeSet collections with no Index added and queries in OSTreeSet collections with Index added but over an unoptimised attribute, produce roughly the same time results (as expected).

Section 3: Comparing Select Range Queries

NoOpt

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
3	ID	NoOpt	None	1000	-1	450	550	No	No	260
12	ID	NoOpt	None	10000	-1	4500	5500	No	No	1940
21	ID	NoOpt	None	60000	-1	2700	3300	No	No	12140

No special comments in this section.

HashingQuery

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
30	ID	HashingQuery	getID()	1000	-1	450	550	No	No	586.6666
39	ID	HashingQuery	getID()	10000	-1	4500	5500	No	No	22576.66
48	ID	HashingQuery	getID()	60000	-1	2700	3300	No	No	132716.6
147	Age	HashingQuery	getID()	1000	-1	450	550	No	No	600
156	Age	HashingQuery	getID()	10000	-1	4500	5500	No	No	22500
165	Age	HashingQuery	getID()	60000	-1	2700	3300	No	No	116113.3

- Select Range queries over the optimised attribute “ID” and the unoptimised attribute “Age”, produce roughly the same times. This is because hashing is of no use in Select queries, whether Point or Range.

Indexing – None

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
57	ID	Indexing	None	1000	-1	450	550	No	No	986.6666
66	ID	Indexing	None	10000	-1	4500	5500	No	No	10120
75	ID	Indexing	None	60000	-1	2700	3300	No	No	61113.33

No special comments in this section.

Indexing – ID

Ex	StudMem	Optimization	IndexMem	StudNum	Value	Low	High	Pick	NewDB	AvgTime
84	ID	Indexing	getID()	1000	-1	450	550	No	No	166.6666
93	ID	Indexing	getID()	10000	-1	4500	5500	No	No	310
102	ID	Indexing	getID()	60000	-1	2700	3300	No	No	1646.666
111	Age	Indexing	getID()	1000	-1	450	550	No	No	936.6666
120	Age	Indexing	getID()	10000	-1	4500	5500	No	No	10286.66
129	Age	Indexing	getID()	60000	-1	2700	3300	No	No	61056.66

- As expected, there is huge difference in result time between Select Range queries over the “ID” attribute (optimised) and the “Age” attribute (unoptimised).

End of Section 3 Conclusions

- In all cases, as collection size increases, the time taken by Select queries over it also increases. This is because:
  i. There are more elements to be traversed.
  ii. There are more comparison operations to be performed.
  iii. There are more elements to be returned (within a Set object, that gets bigger as more object references are added to it)
- Queries in OSTreeSet (with Index added) over the unoptimised "Age" attribute are as fast (or as slow if you like) as queries in OSTreeSet (without Index added) over any attribute.
- Query Speed:
  - OSTreeSet with Index added comes 1st in all cases.
  - OSVector comes 2nd in all cases.
  - OSTreeSet with no Index added comes 3rd for medium and large sizes of collections, and comes 4th for small sizes of collections.
  - OSHashMap comes 4th for medium and large sizes of collections and 3rd for small sizes of collections.

Section 4: Putting it All Together

Pick Point Query Speed

Collection Size	Collection Type	1st Place	2nd Place	3rd Place	4th Place	5th Place
SMALL	OSVector			✔		
	OSHashMap(Query)					✔
	OSHashMap(Get)	✔				
	OSTreeSet(No Index)				✔	
	OSTreeSet(Index Added)		✔			
MEDIUM	OSVector			✔		
	OSHashMap(Query)					✔
	OSHashMap(Get)	✔				
	OSTreeSet(No Index)				✔	
	OSTreeSet(Index Added)		✔			
LARGE	OSVector			✔		
	OSHashMap(Query)					✔
	OSHashMap(Get)		✔			
	OSTreeSet(No Index)				✔	
	OSTreeSet(Index Added)	✔				

Select (non-Pick) Point Query Speed

Collection Size	Collection Type	1st Place	2nd Place	3rd Place	4th Place
SMALL	OSVector		✔		
	OSHashMap(Query)			✔	
	OSTreeSet(No Index)				✔
	OSTreeSet(Index Added)	✔			
MEDIUM	OSVector		✔		
	OSHashMap(Query)				✔
	OSTreeSet(No Index)			✔	
	OSTreeSet(Index Added)	✔			
LARGE	OSVector		✔		
	OSHashMap(Query)				✔
	OSTreeSet(No Index)			✔	
	OSTreeSet(Index Added)	✔			

Select Range Query Speed

Collection Size	Collection Type	1st Place	2nd Place	3rd Place	4th Place
SMALL	OSVector		✔		
	OSHashMap(Query)			✔	
	OSTreeSet(No Index)				✔
	OSTreeSet(Index Added)	✔			
MEDIUM	OSVector		✔		
	OSHashMap(Query)				✔
	OSTreeSet(No Index)			✔	
	OSTreeSet(Index Added)	✔			
LARGE	OSVector		✔		
	OSHashMap(Query)				✔
	OSTreeSet(No Index)			✔	
	OSTreeSet(Index Added)	✔			

OS Database files' Sizes & Population Times

	.odb File Size (KB)			.odt File Size (KB)			.odf File Size (KB)			File Creation Time (sec)		
Number of Objects	1,000	10,000	60,000	1,000	10,000	60,000	1,000	10,000	60,000	1,000	10,000	60,000
OSVector	**43**	**405**	**2,684**	**18**	**167**	**998**	**2**	**13**	**49**	**0.170**	**0.500**	**3.910**
OSHashMap	**81**	**867**	**5,761**	**48**	**470**	**2,814**	**3**	**21**	**93**	**0.110**	**52.160**	**681.11**
OSTreeSet – No Index	**51**	**583**	**12,125**	**17**	**158**	**941**	**2**	**13**	**237**	**0.380**	**2.480**	**21.830**
OSTreeSet – Index Added	**71**	**767**	**13.629**	**17**	**158**	**944**	**2**	**17**	**245**	**0.330**	**3.900**	**34.820**

Comments on the Sizes & Population Times

- Small difference in file size when an Index is added to the OSTreeSet root. This suggests that only the 8-Byte Tree-Node is used and the 4-Byte Tree-Node, although it remains as a structure within the odb file, is empty.
- Most expensive operation, when it comes to database creation time, is by far Hashing (OSHashMap) with explicit hashing over an object attribute's value (via a *put()* operation).
- Most expensive operation, when it comes to disk space required, is by far Indexing, whether an Index is added or not. This is because even if we do not explicitly ask for an index on say the "ID" attribute to be added, an Index is added anyway on the objects' OID value. The file gets a bit bigger when we add an explicit index over the "ID" attribute because then an 8-Byte BtreeNode is used (since "ID" is a long value) instead of the 4-Byte BtreeNode used in the case of the OID (since OID is a 32-bit value). This also means that Ostore cannot "handle" a database say handling data of all of planet Earth's people. ☺

Some Extra General Conclusions & Comments

- Hashing possesses the potential to be as fast, or even faster, (using the *get()* method) than Indexing, for large sizes of collections where it is slower, as the data above shows. This can be done by overriding the default number of buckets to a much bigger number (say as big as the intended number of objects stored in the Database). This would give very fast *get()* times but it would make iterations even slower and file size would probably “skyrocket”.
- In general, hashed collections are slow when it comes to iterations through them. In fact the more the buckets, the worse the time iterations (or traversals if you prefer) take.
- OSTreeSet based OS database files, created to be identical (same objects…same everything) can differ in size very slightly. We believe that this is due to the way that the BTree algorithm works.
- In OSTreeSet, with Index added over the ID attribute, querying over the Credits attribute (which always has the same value as the ID attribute), does not produce any optimised time results (as expected, but just checking ☺)
- Same goes for OSHashMap.
- For OSTreeSet collections, Pick Point and Select Point queries (over the optimised attribute) are as fast.
- Not so for OSHashMap though, because, as we have seen, in OSHashMap position of the searched element is significant when it comes to Pick time results. This is due to the way the Hashing algorithm works.

APPENDIX

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Note: Some Ex. ID nums are missing due to deprecation.

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
1	ID	NoOpt	None	1000	0	-1	-1	No	Yes	223.33333
2	ID	NoOpt	None	1000	0	-1	-1	Yes	No	36.666666
3	ID	NoOpt	None	1000	-1	450	550	No	No	260
4	ID	NoOpt	None	1000	500	-1	-1	No	No	203.33333
5	ID	NoOpt	None	1000	500	-1	-1	Yes	No	56.666666
7	ID	NoOpt	None	1000	999	-1	-1	No	No	180
8	ID	NoOpt	None	1000	999	-1	-1	Yes	No	223.33333
10	ID	NoOpt	None	10000	0	-1	-1	No	Yes	1830
11	ID	NoOpt	None	10000	0	-1	-1	Yes	No	53.333333
12	ID	NoOpt	None	10000	-1	4500	5500	No	No	1940
13	ID	NoOpt	None	10000	5000	-1	-1	No	No	1787.5
14	ID	NoOpt	None	10000	5000	-1	-1	Yes	No	1017.5
16	ID	NoOpt	None	10000	9999	-1	-1	No	No	1776.6666
17	ID	NoOpt	None	10000	9999	-1	-1	Yes	No	1776.6666
19	ID	NoOpt	None	60000	0	-1	-1	No	Yes	10783.333
20	ID	NoOpt	None	60000	0	-1	-1	Yes	No	110
21	ID	NoOpt	None	60000	-1	27000	33000	No	No	12140
22	ID	NoOpt	None	60000	30000	-1	-1	No	No	10046.666
23	ID	NoOpt	None	60000	30000	-1	-1	Yes	No	5293.3333
25	ID	NoOpt	None	60000	59999	-1	-1	No	No	10506.666
26	ID	NoOpt	None	60000	59999	-1	-1	Yes	No	10400

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
28	ID	HashingQuery	getID()	1000	0	-1	-1	No	Yes	623.33333
29	ID	HashingQuery	getID()	1000	0	-1	-1	Yes	No	76.666666
30	ID	HashingQuery	getID()	1000	-1	450	550	No	No	586.66666
31	ID	HashingQuery	getID()	1000	500	-1	-1	No	No	566.66666
32	ID	HashingQuery	getID()	1000	500	-1	-1	Yes	No	493.33333
34	ID	HashingQuery	getID()	1000	999	-1	-1	No	No	550
35	ID	HashingQuery	getID()	1000	999	-1	-1	Yes	No	586.66666
37	ID	HashingQuery	getID()	10000	0	-1	-1	No	Yes	21406.666
38	ID	HashingQuery	getID()	10000	0	-1	-1	Yes	No	90
39	ID	HashingQuery	getID()	10000	-1	4500	5500	No	No	22576.666
40	ID	HashingQuery	getID()	10000	5000	-1	-1	No	No	21420
41	ID	HashingQuery	getID()	10000	5000	-1	-1	Yes	No	10270
43	ID	HashingQuery	getID()	10000	9999	-1	-1	No	No	21900
44	ID	HashingQuery	getID()	10000	9999	-1	-1	Yes	No	21036.666
46	ID	HashingQuery	getID()	60000	0	-1	-1	No	Yes	112136.66
47	ID	HashingQuery	getID()	60000	0	-1	-1	Yes	No	493.33333
48	ID	HashingQuery	getID()	60000	-1	27000	33000	No	No	132716.66
49	ID	HashingQuery	getID()	60000	30000	-1	-1	No	No	124955
50	ID	HashingQuery	getID()	60000	30000	-1	-1	Yes	No	89603.333
52	ID	HashingQuery	getID()	60000	59999	-1	-1	No	No	130683.33
53	ID	HashingQuery	getID()	60000	59999	-1	-1	Yes	No	51003.333

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
55	ID	Indexing	None	1000	0	-1	-1	No	Yes	880
56	ID	Indexing	None	1000	0	-1	-1	Yes	No	786.66666
57	ID	Indexing	None	1000	-1	450	550	No	No	986.66666
58	ID	Indexing	None	1000	500	-1	-1	No	No	880
59	ID	Indexing	None	1000	500	-1	-1	Yes	No	566.66666
61	ID	Indexing	None	1000	999	-1	-1	No	No	896.66666
62	ID	Indexing	None	1000	999	-1	-1	Yes	No	546.66666
64	ID	Indexing	None	10000	0	-1	-1	No	Yes	10220
65	ID	Indexing	None	10000	0	-1	-1	Yes	No	8310
66	ID	Indexing	None	10000	-1	4500	5500	No	No	10120
67	ID	Indexing	None	10000	5000	-1	-1	No	No	9190
68	ID	Indexing	None	10000	5000	-1	-1	Yes	No	2690
70	ID	Indexing	None	10000	9999	-1	-1	No	No	10330
71	ID	Indexing	None	10000	9999	-1	-1	Yes	No	3146.6666
73	ID	Indexing	None	60000	0	-1	-1	No	Yes	48280
74	ID	Indexing	None	60000	0	-1	-1	Yes	No	17886.666
75	ID	Indexing	None	60000	-1	27000	33000	No	No	61113.333
76	ID	Indexing	None	60000	30000	-1	-1	No	No	66700
77	ID	Indexing	None	60000	30000	-1	-1	Yes	No	58840
79	ID	Indexing	None	60000	59999	-1	-1	No	No	58750
80	ID	Indexing	None	60000	59999	-1	-1	Yes	No	41323.333

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
82	ID	Indexing	getID()	1000	0	-1	-1	No	Yes	56.666666
83	ID	Indexing	getID()	1000	0	-1	-1	Yes	No	73.333333
84	ID	Indexing	getID()	1000	-1	450	550	No	No	166.66666
85	ID	Indexing	getID()	1000	500	-1	-1	No	No	165
86	ID	Indexing	getID()	1000	500	-1	-1	Yes	No	315
88	ID	Indexing	getID()	1000	999	-1	-1	No	No	122.5
89	ID	Indexing	getID()	1000	999	-1	-1	Yes	No	67.5
91	ID	Indexing	getID()	10000	0	-1	-1	No	Yes	110
92	ID	Indexing	getID()	10000	0	-1	-1	Yes	No	36.666666
93	ID	Indexing	getID()	10000	-1	4500	5500	No	No	310
94	ID	Indexing	getID()	10000	5000	-1	-1	No	No	123.33333
95	ID	Indexing	getID()	10000	5000	-1	-1	Yes	No	130
97	ID	Indexing	getID()	10000	9999	-1	-1	No	No	93.333333
98	ID	Indexing	getID()	10000	9999	-1	-1	Yes	No	56.666666
100	ID	Indexing	getID()	60000	0	-1	-1	No	Yes	33.333333
101	ID	Indexing	getID()	60000	0	-1	-1	Yes	No	36.666666
102	ID	Indexing	getID()	60000	-1	27000	33000	No	No	1646.6666
103	ID	Indexing	getID()	60000	30000	-1	-1	No	No	33.333333
104	ID	Indexing	getID()	60000	30000	-1	-1	Yes	No	36.666666
106	ID	Indexing	getID()	60000	59999	-1	-1	No	No	40
107	ID	Indexing	getID()	60000	59999	-1	-1	Yes	No	57.5

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
109	Age	Indexing	getID()	1000	0	-1	-1	No	Yes	893.33333
110	Age	Indexing	getID()	1000	0	-1	-1	Yes	No	640
111	Age	Indexing	getID()	1000	-1	450	550	No	No	936.66666
112	Age	Indexing	getID()	1000	500	-1	-1	No	No	1063.3333
113	Age	Indexing	getID()	1000	500	-1	-1	Yes	No	910
115	Age	Indexing	getID()	1000	999	-1	-1	No	No	916.66666
116	Age	Indexing	getID()	1000	999	-1	-1	Yes	No	916.66666
118	Age	Indexing	getID()	10000	0	-1	-1	No	Yes	10065
119	Age	Indexing	getID()	10000	0	-1	-1	Yes	No	3956.6666
120	Age	Indexing	getID()	10000	-1	4500	5500	No	No	10286.666
121	Age	Indexing	getID()	10000	5000	-1	-1	No	No	9706.6666
122	Age	Indexing	getID()	10000	5000	-1	-1	Yes	No	8053.3333
124	Age	Indexing	getID()	10000	9999	-1	-1	No	No	9376.6666
125	Age	Indexing	getID()	10000	9999	-1	-1	Yes	No	3956.6666
127	Age	Indexing	getID()	60000	0	-1	-1	No	Yes	60453.333
128	Age	Indexing	getID()	60000	0	-1	-1	Yes	No	7926.6666
129	Age	Indexing	getID()	60000	-1	27000	33000	No	No	61056.666
130	Age	Indexing	getID()	60000	30000	-1	-1	No	No	57636.666
131	Age	Indexing	getID()	60000	30000	-1	-1	Yes	No	12393.333
133	Age	Indexing	getID()	60000	59999	-1	-1	No	No	54810
134	Age	Indexing	getID()	60000	59999	-1	-1	Yes	No	52216.666

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
136	ID	HashingGet	getID()	1000	0	-1	-1	No	Yes	16.666666
137	ID	HashingGet	getID()	1000	500	-1	-1	No	No	50
138	ID	HashingGet	getID()	1000	999	-1	-1	No	No	10
139	ID	HashingGet	getID()	10000	0	-1	-1	No	Yes	70
140	ID	HashingGet	getID()	10000	5000	-1	-1	No	No	56.666666
141	ID	HashingGet	getID()	10000	9999	-1	-1	No	No	36.666666
142	ID	HashingGet	getID()	60000	0	-1	-1	No	Yes	346.66666
143	ID	HashingGet	getID()	60000	30000	-1	-1	No	No	363.33333
144	ID	HashingGet	getID()	60000	59999	-1	-1	No	No	403.33333
145	Age	HashingQuery	getID()	1000	0	-1	-1	No	Yes	626.66666
146	Age	HashingQuery	getID()	1000	0	-1	-1	Yes	No	620
147	Age	HashingQuery	getID()	1000	-1	450	550	No	No	600
148	Age	HashingQuery	getID()	1000	500	-1	-1	No	No	710
149	Age	HashingQuery	getID()	1000	500	-1	-1	Yes	No	680
151	Age	HashingQuery	getID()	1000	999	-1	-1	No	No	643.33333
152	Age	HashingQuery	getID()	1000	999	-1	-1	Yes	No	53.333333
154	Age	HashingQuery	getID()	10000	0	-1	-1	No	Yes	22733.333
155	Age	HashingQuery	getID()	10000	0	-1	-1	Yes	No	23070
156	Age	HashingQuery	getID()	10000	-1	4500	5500	No	No	22500
157	Age	HashingQuery	getID()	10000	5000	-1	-1	No	No	21330
158	Age	HashingQuery	getID()	10000	5000	-1	-1	Yes	No	10490

Appendix: Experiment Configurations and their Results

Ex	ExperimentID
StudMember	StudentMember
Optimization	OptimizationType
IndexMem	IndexableMember
StudentNum	StudentNumber
Value	Value

Low	Low
High	High
Pick	Pick
CreateNewDB	CreateNewDB
AvgTime	Experiment Average Time

Ex	StudMember	Optimization	IndexMem	StudentNum	Value	Low	High	Pick	CreateNewDB	AvgTime
160	Age	HashingQuery	getID()	10000	9999	-1	-1	No	No	21206.666
161	Age	HashingQuery	getID()	10000	9999	-1	-1	Yes	No	90
163	Age	HashingQuery	getID()	60000	0	-1	-1	No	Yes	130560
164	Age	HashingQuery	getID()	60000	0	-1	-1	Yes	No	48866.666
165	Age	HashingQuery	getID()	60000	-1	27000	33000	No	No	116113.33
166	Age	HashingQuery	getID()	60000	30000	-1	-1	No	No	131293.33
167	Age	HashingQuery	getID()	60000	30000	-1	-1	Yes	No	94115
169	Age	HashingQuery	getID()	60000	59999	-1	-1	No	No	132260
170	Age	HashingQuery	getID()	60000	59999	-1	-1	Yes	No	672.5