CMPSC 380
Principles of Database Systems
Fall 2014

Laboratory Assignment Three
Combining Imperative and Declarative Programming: Implement and Empirical Evaluation

Group: Ginoza, Mulvay, Ballinger, Landgrebe, McCurdy

Pledge:

Summary:

This project is to allow us to implement a benchmarking framework that compares the performance of JoSQL to a hand-coded-baseline program that uses iteration constructs.. JoSQL is a Java library that allows for SQL calls to perform operations on objects. We have decided to compare the performance of these two data entry tools by using different data structures (e.g. **ArrayList**, **LinkedList**, **ArrayDeque**, and **Vector**).

We also decided to compare the performance of these data structures on different hardware. The hardware we will be comparing is a Dell OptiPlex 380 – Core 2 Duo E7500 2.93 GHz with 4 GB of RAM versus a Mid 2011 MacBook Pro Core 2 Duo Core i7 (I7-3520M) 2.9 Ghz with 8GB of RAM.

1. **A five paragraph commentary on the work that each team member completed.**

   Every member of the group will individually be submitting this required deliverable.

2. **A full-featured description of the key features provided by the JoSQL library.**

   - **Performance:**
     The more work you expect JoSQL to do the longer it will take to do it. If your WHERE clause does not significantly limit the number of objects that the rest of the query should work on, then obviously the rest of the query has more work to do, which equates to more processing time.
   - **Query:**
     This class in JoSQL allows for a developer of the Java programming language to apply arbitrary SQL statements to a collection of Java objects.

     Simple Example:
     ```
     Query q = new Query ();
     q.parse (sql);
     List results = q.execute (java.util.List);
     ```
   - **Where:**
     Allow for searching through data sets where objects—Strings in this lab—match what is begin searched for in the **Where** clause.

Simple Example:
WHERE name LIKE '%SEARCH HERE'

- **Bind Variables:**
  Bind variables allow you to place "markers" inside the SQL statement that will then be replaced at execution time with the values supplied.

  Simple Example:
  q.setVariable (":name", myObj)

- **Custom Functions:**
  Custom functions are basically those specified by the developer to extend the JoSQL functionality in some way. Custom functions can override the built-in functions.

  Simple Example:
  public String to_string (Object o)

  Referred to in SQL with:
  SELECT to_string (12345)
  FROM java.lang.Object

  Note: Custom functions are searched for in the order in which their "handler Object" are added to the Query object.

  ```
  Query q = new Query ();
  q.addFunctionHandler (new MyFunctionHandler1 ());
  q.addFunctionHandler (new MyFunctionHandler2 ());
  ```

- **Accessors:**
  An accessor is basically a way of accessing information held with an object (this is performed by using Java Reflection). The accessor provides a textual way of representing the public methods names or fields that are accessed, separated by ".".

  Example, class java.io.File:
  To access the getName method, use: name or getName.

- **Save Values:**
  A save value is just an "identifier", usually a java.lang.String(although any object can be used), that acts as a key for a value. The value can be any object.. Save values are stored in the Query object and are available to the developer after the query has completed. Save values are assigned either by a function or in the EXECUTE ON clause.

- **Expressions:**
  Expressions (modeled using the Expression object) are the basic "unit" within JoSQL. As such expressions can be interchanged and used just about everywhere (this means that even binary expressions such as x = y can be used in the SELECT part of the statement). Each expression in JoSQL must return a value, even if the value is null. In this way it means that comparisons between expressions can occur and the result of evaluating an expression is suitable for returning in query execution results.

Various types of JoSQL Expressions

Binary expression - this type of expression represents the result of comparing a LHS with a RHS. For example $1 = 1$. Just about all expressions that can be used in a Where clause are binary expressions.

Aliased expression - this type of expression represents an expression that has an alias associated with it.

Value Expression - this type of expression represents a value, such as a constant, a bind variable, the result of executing a function or even a sub-query.

3. **The output from each team member's use of the FileFinder.java program.**

   These can be found after the seven posed questions.

4. **The properly formatted and documented version of the source code for your benchmarks.**

   These can be found after the seven posed questions and the output of our runs of FileFinder.java.

5. **The CSV data files resulting from all of the runs of your benchmarking framework.**

   The data files containing the results can be found after our source code.

6. **A comprehensive report describing our benchmarks, evaluation metrics, and results.**

   The purpose of this laboratory assignment was to compare the performance of JoSQL—SQL for Java Objects—to a hand-coded implementation of a search algorithm. We wrote four hand-coded versions of this algorithm which implement different data structures: the **ArrayList, ArrayDeque, LinkedList,** and **Vector**. We also decided to compare the hardware constraints of a Dell OptiPlex 380 – Core 2 Duo E7500 2.93 GHz with 4 GB of RAM and a Mid 2011 MacBook Pro Core 2 Duo Core i7 (I7-3520M) 2.9 Ghz with 8GB of RAM.

   We found that JoSQL was not the fastest search algorithm. When compared with the ArrayDeque, the JoSQL was up to 32 times slower than our hand-coded implemetation.

   We used differently-sized inputs to determine whether size was a factor; size did not exhibit a significant difference when comparing the speed in nanoseconds of the two algorithms.. In some cases the larger data set was actually searched more quickly. When searching for "medium", the most common word in the CSV file, with the data structure ArrayList, we observed that the hand-coded version completed the search in 760000 ns compared to the 835000 ns while searching for "winter", the least common word. Here, we can conclude that size is not significant, but in future studies we will have to consider that we did not vary size as much as we could have.

   Concerning hardware constraints, the Dell ran every algorithm in less than than the Mac, even when controlling for confounding variables such as other open programs. We think the minute increase in speed per core that the Dell has may have allowed it to perform these tasks in fewer nanoseconds, but we acknowledge that this is not enough to account for the hugh variablility we observed.

One noticable trend was that the performance of the ArrayList, LinkedList, Vector, and ArrayDeque data structures was much quicker than the JoSQL. A comparison of averages showed that the JoSQL was much slower than the hand-coded version.

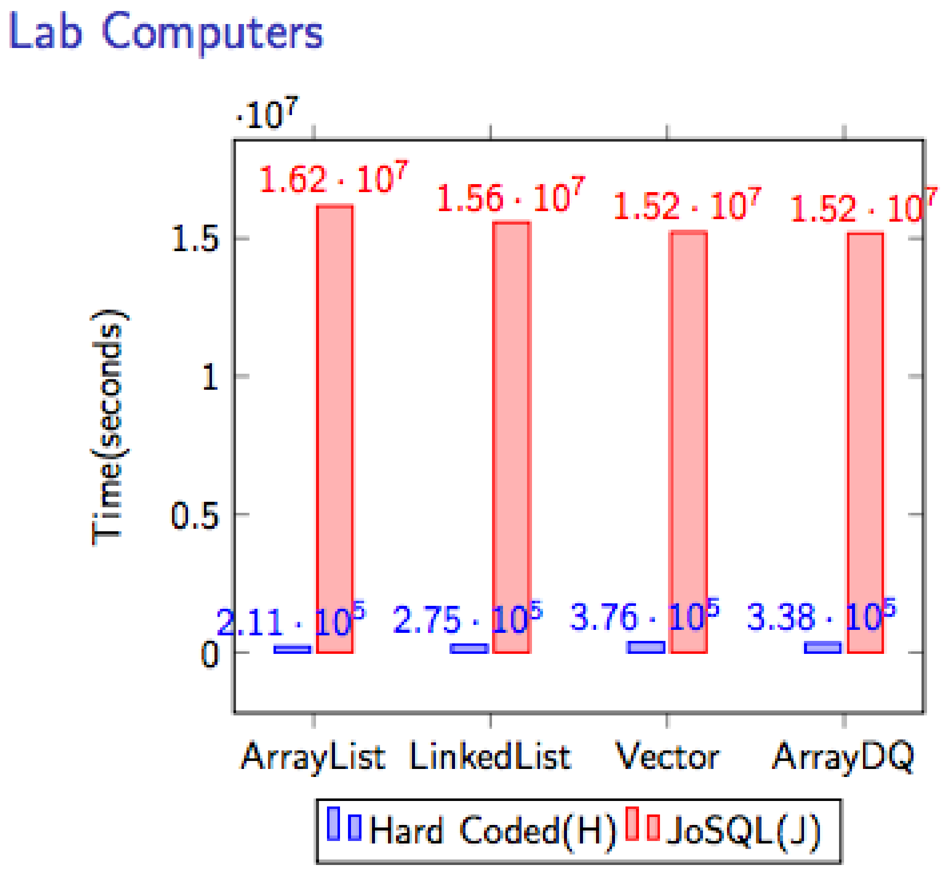


Figure 1: Times when run on Optiplex 380

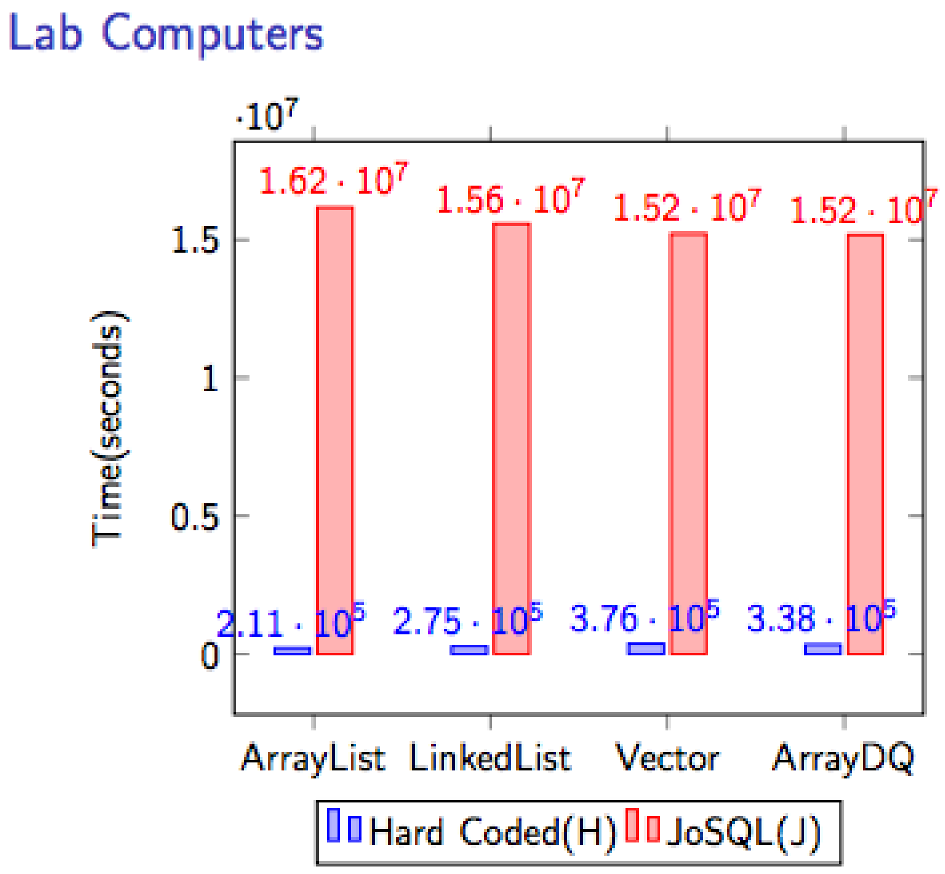
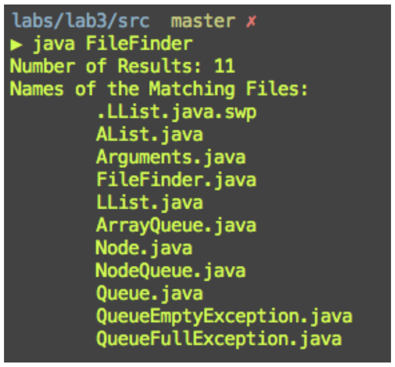


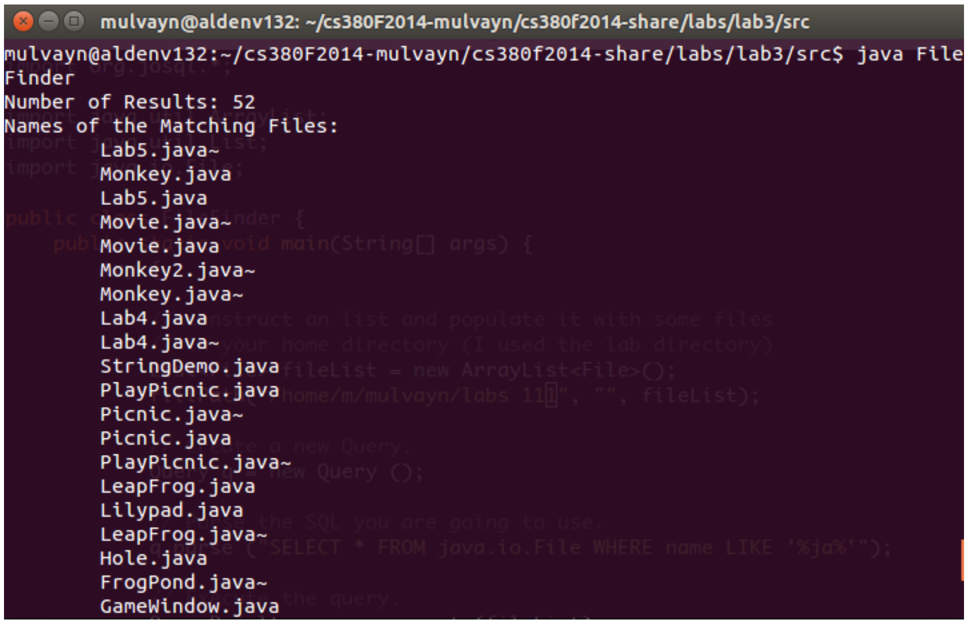
Figure 2: Times when run on MacBook Pro

7. **The presentation slides to support your team's five minute presentation.**
   This will be presented in the laboratory session on September 24, 2014.



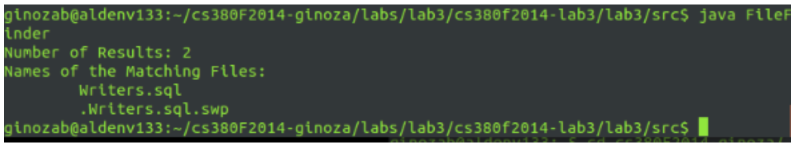
```
labs/lab3/src  master ✗
▶ java FileFinder
Number of Results: 11
Names of the Matching Files:
        .LList.java.swp
        AList.java
        Arguments.java
        FileFinder.java
        LList.java
        ArrayQueue.java
        Node.java
        NodeQueue.java
        Queue.java
        QueueEmptyException.java
        QueueFullException.java
```

Figure 3: Output of Colton's FileFinder.java



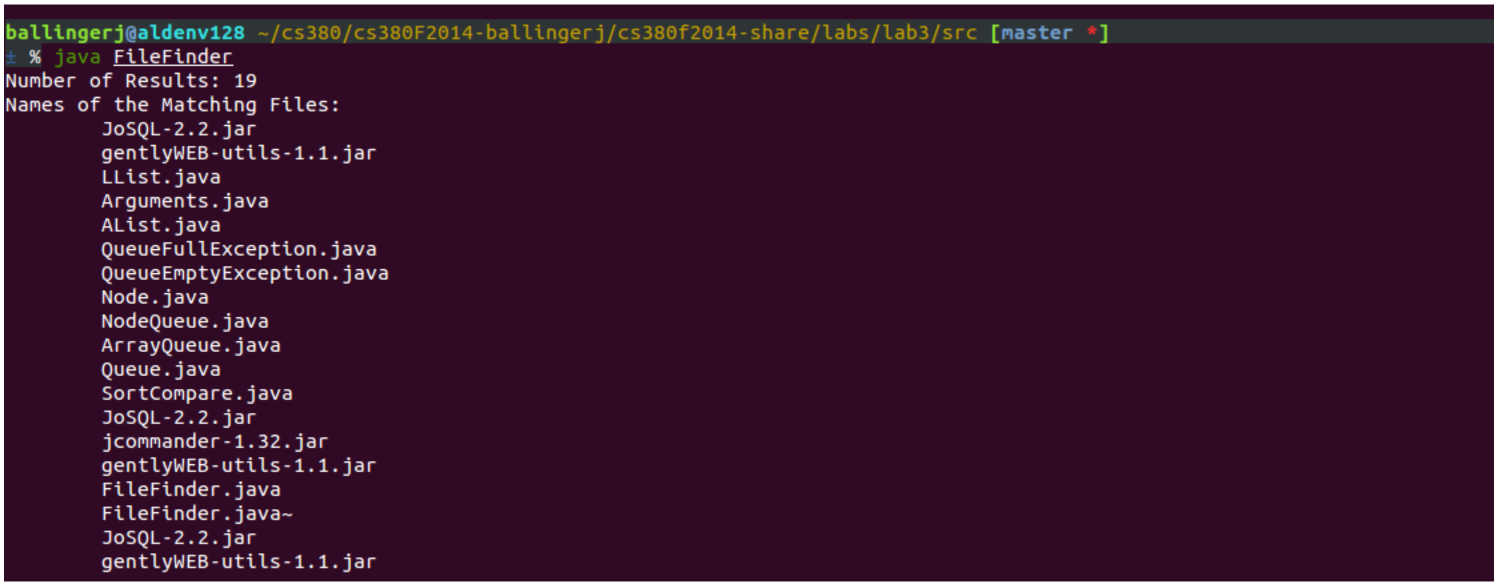
```
mulvayn@aldenv132: ~/cs380F2014-mulvayn/cs380f2014-share/labs/lab3/src
mulvayn@aldenv132:~/cs380F2014-mulvayn/cs380f2014-share/labs/lab3/src$ java File
Finder
Number of Results: 52
Names of the Matching Files:
        Lab5.java~
        Monkey.java
        Lab5.java
        Movie.java~
        Movie.java
        Monkey2.java~
        Monkey.java~
        Lab4.java
        Lab4.java~
        StringDemo.java
        PlayPicnic.java
        Picnic.java~
        Picnic.java
        PlayPicnic.java~
        LeapFrog.java
        Lilypad.java
        LeapFrog.java~
        Hole.java
        FrogPond.java~
        GameWindow.java
```

Figure 4: Output of Nick's FileFinder.java



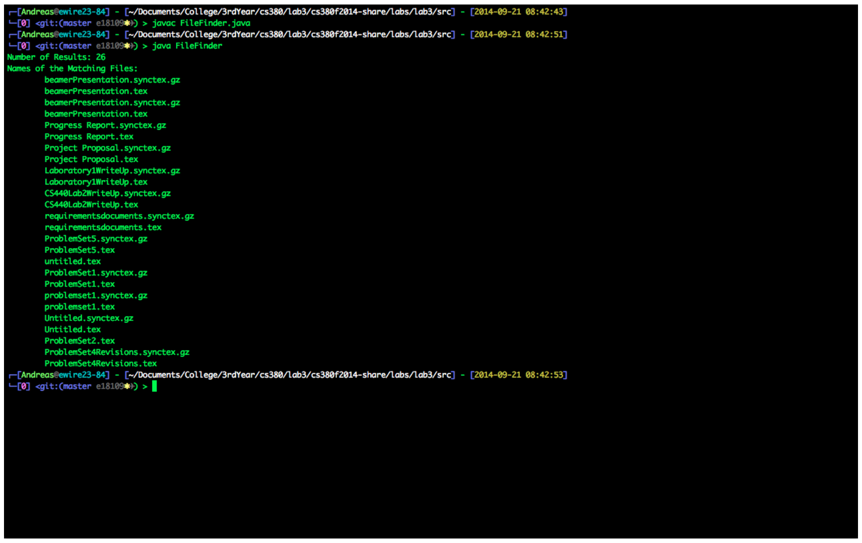
```
ginozab@aldenv133:~/cs380F2014-ginoza/labs/lab3/cs380f2014-lab3/lab3/src$ java FileF
inder
Number of Results: 2
Names of the Matching Files:
        Writers.sql
        .Writers.sql.swp
ginozab@aldenv133:~/cs380F2014-ginoza/labs/lab3/cs380f2014-lab3/lab3/src$
```

Figure 5: Output of Brandon's FileFinder.java



```
ballingerj@aldenv128 ~/cs380/cs380F2014-ballingerj/cs380f2014-share/labs/lab3/src [master *]
± % java FileFinder
Number of Results: 19
Names of the Matching Files:
        JoSQL-2.2.jar
        gentlyWEB-utils-1.1.jar
        LList.java
        Arguments.java
        AList.java
        QueueFullException.java
        QueueEmptyException.java
        Node.java
        NodeQueue.java
        ArrayQueue.java
        Queue.java
        SortCompare.java
        JoSQL-2.2.jar
        jcommander-1.32.jar
        gentlyWEB-utils-1.1.jar
        FileFinder.java
        FileFinder.java~
        JoSQL-2.2.jar
        gentlyWEB-utils-1.1.jar
```

Figure 6: Output of Jake's FileFinder.java



```
┌─[Andreas@ewire23-84] - [~/Documents/College/3rdYear/cs380/lab3/cs380f2014-share/labs/lab3/src] - [2014-09-21 08:42:43]
└─[0] <git:(master e18109✱)> > javac FileFinder.java
┌─[Andreas@ewire23-84] - [~/Documents/College/3rdYear/cs380/lab3/cs380f2014-share/labs/lab3/src] - [2014-09-21 08:42:51]
└─[0] <git:(master e18109✱)> > java FileFinder
Number of Results: 26
Names of the Matching Files:
        beamerPresentation.synctex.gz
        beamerPresentation.tex
        beamerPresentation.synctex.gz
        beamerPresentation.tex
        Progress Report.synctex.gz
        Progress Report.tex
        Project Proposal.synctex.gz
        Project Proposal.tex
        Laboratory1WriteUp.synctex.gz
        Laboratory1WriteUp.tex
        CS440Lab2WriteUp.synctex.gz
        CS440Lab2WriteUp.tex
        requirementsdocuments.synctex.gz
        requirementsdocuments.tex
        ProblemSet5.synctex.gz
        ProblemSet5.tex
        untitled.tex
        ProblemSet1.synctex.gz
        ProblemSet1.tex
        problemset1.synctex.gz
        problemset1.tex
        Untitled.synctex.gz
        Untitled.tex
        ProblemSet2.tex
        ProblemSet4Revisions.synctex.gz
        ProblemSet4Revisions.tex
┌─[Andreas@ewire23-84] - [~/Documents/College/3rdYear/cs380/lab3/cs380f2014-share/labs/lab3/src] - [2014-09-21 08:42:53]
└─[0] <git:(master e18109✱)> >
```

Figure 7: Output of Andreas's FileFinder.java

The entirety of the source code for our benchmarking framework is on the following pages.

```
import org.josql.*;
import com.beust.jcommander.Parameter;
import com.beust.jcommander.JCommander;
import java.util.ArrayList;
import java.util.List;
import java.io.File;
import java.util.*;

public class AList {

public static ArrayList<String> list = new ArrayList<String>(); // holds lines of algae

public static void main(String[] args) {

Arguments params = new Arguments();
        JCommander cmd = new JCommander(params, args);
        try {
            cmd.parse(args);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            cmd.usage();
        }

        // The debugging flag which will determine whether or not
        // we produce output
        boolean debugOutput = params.getOutput();

// The pull flag will determine which data attritute to pull
String pullOutput = params.getPull().toString();

try{
Scanner scan = new Scanner(new File("../data/algae.csv")); // scan algae

/* Scan until there are no more lines, add them to an arrayList---list---
 * and scan every line in list and search for lines that contain 'winter'
 * and add them to winterList */
while (scan.hasNextLine())
{
String next = scan.nextLine();
list.add(next);
}
System.out.println("By hand:\n");
findByHand(pullOutput);
```

```
System.out.println("\nBy JoSQL:\n");
findByJo(pullOutput);

} catch (java.io.FileNotFoundException FNF) {
System.out.println("Couldn't Find File");
}
}

public static void findByJo(String pullOutput)
{

try {

// Create a new Query.
Query q = new Query ();
//String winter = "winter";
String query = "SELECT * FROM java.lang.String WHERE toString LIKE '%" + pullOutput + "%''

long startTime = System.nanoTime();

// Parse the SQL you are going to use.
q.parse(query);

// Execute the query.
QueryResults qr = q.execute(list);

// Cycle over the query results
List<String> res =  qr.getResults();

long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;

for (String result : res)
{
System.out.println(result);
}
System.out.println("Time of findByJo() with data-structure ArrayList: " + estimatedTime +
}

catch(org.josql.QueryParseException e) {
System.out.println("Error One");
}

catch(org.josql.QueryExecutionException e) {
System.out.println("Error Two");
}
}

/* Hand-coded version that finds occurences of winter in algae.csv
 * Does it by scanning each line of algae adding them to an arrayList
 * then in that arrayList searches each line for 'Winter,' then adds
 * them to another arrayList */
public static void findByHand(String pullOutput) {

ArrayList<String> pullList = new ArrayList<String>(); // holds lines that contain the pul

long startTime = System.nanoTime();

for (String str : list)
{
if (str.contains(pullOutput))
pullList.add(str);
}
long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;
System.out.println("Time of findByHand() with data-structure ArrayList: " + estimatedTime

for (String contains : pullList)
 {
  System.out.println(contains);
 }
}
}
```

```
import org.josql.*;
import com.beust.jcommander.Parameter;
import com.beust.jcommander.JCommander;
import java.util.ArrayList;
import java.util.List;
import java.io.File;
import java.util.*;

public class LList {

private static LinkedList<String> list = new LinkedList<String>(); // holds lines of algae

public static void main(String[] args) {

Arguments params = new Arguments();
        JCommander cmd = new JCommander(params, args);
        try {
            cmd.parse(args);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            cmd.usage();
        }

        // The debugging flag which will determine whether or not
        // we produce output
        boolean debugOutput = params.getOutput();

// The pull flag will determine which data attritute to pull
String pullOutput = params.getPull().toString();

try{
Scanner scan = new Scanner(new File("../data/algae.csv")); // scan algae

/* Scan until there are no more lines, add them to an arrayList---list---
 * and scan every line in list and search for lines that contain 'winter'
 * and add them to winterList */
while (scan.hasNextLine())
{
String next = scan.nextLine();
list.add(next);
}
System.out.println("By Hand:\n");
findByHand(pullOutput);
System.out.println("\nUsing JoSQL:\n");
findByJo(pullOutput);

} catch (java.io.FileNotFoundException FNF) {
System.out.println("Couldn’t Find File");
}
}

public static void findByJo(String pullOutput)
{
try {

// Create a new Query.
Query q = new Query ();
String query = "SELECT * FROM java.lang.String WHERE toString LIKE ’%" + pullOutput + "%’'

long startTime = System.nanoTime();

// Parse the SQL you are going to use.
q.parse(query);

// Execute the query.
QueryResults qr = q.execute(list);

// Cycle over the query results
List<String> res =  qr.getResults();

long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;

for (String result : res)
{
System.out.println(result);
}
System.out.println("Time of findWinterByJo() with data-structure LinkedList: " + estimatec
}

catch(org.josql.QueryParseException e) {
System.out.println("Error One");
}

catch(org.josql.QueryExecutionException e) {
System.out.println("Error Two");
}
}
```

```
/* Hand-coded version that finds occurences of the pull value in algae.csv
 * Does it by scanning each line of algae adding them to an LinkedList
 * then in that LinkedList searches each line for the pull value, then adds
 * them to another LinkedList */

public static void findByHand(String pullOutput) {


LinkedList<String> pullList = new LinkedList<String>(); // holds lines that contain the pu

long startTime = System.nanoTime();

for (String str : list)
{
if (str.contains(pullOutput))
pullList.add(str);
}
long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;
System.out.println("Time of findByHand() with data-structure LinkedList: " + estimatedTime

for (String contains : pullList)
{
System.out.println(contains);
}

}
}
```

```
import org.josql.*;
import com.beust.jcommander.Parameter;
import com.beust.jcommander.JCommander;
import java.util.ArrayList;
import java.util.List;
import java.io.File;
import java.util.*;

public class Vect {

private static Vector<String> list = new Vector<String>(); // holds lines of algae

public static void main(String[] args) {

Arguments params = new Arguments();
        JCommander cmd = new JCommander(params, args);
        try {
            cmd.parse(args);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            cmd.usage();
        }

        // The debugging flag which will determine whether or not
        // we produce output
        boolean debugOutput = params.getOutput();

// The pull flag will determine which data attritute to pull
String pullOutput = params.getPull().toString();

try{
Scanner scan = new Scanner(new File("../data/algae.csv")); // scan algae

/* Scan until there are no more lines, add them to an arrayList---list---
 * and scan every line in list and search for lines that contain 'winter'
 * and add them to winterList */
while (scan.hasNextLine())
{
String next = scan.nextLine();
list.add(next);
}
System.out.println("By Hand:\n");
findByHand(pullOutput);
System.out.println("\nUsing JoSQL:\n");
findByJo(pullOutput);

} catch (java.io.FileNotFoundException FNF) {
System.out.println("Couldn’t Find File");
}
}

public static void findByJo(String pullOutput)
{

try {

// Create a new Query.
Query q = new Query ();
String query = "SELECT * FROM java.lang.String WHERE toString LIKE ’%" + pullOutput + "%’'
long startTime = System.nanoTime();

// Parse the SQL you are going to use.
q.parse (query);

// Execute the query.
QueryResults qr = q.execute(list);

// Cycle over the query results
List<String> res =  qr.getResults();

long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;

for (String result : res)
{
System.out.println(result);
}
System.out.println("Time of findWinterByJo() with data-structure Vector: " + estimatedTime
}

catch(org.josql.QueryParseException e) {
System.out.println("Error One");
}

catch(org.josql.QueryExecutionException e) {
System.out.println("Error Two");
}
}
```

```
/* Hand-coded version that finds occurences of the pull value in algae.csv
 * Does it by scanning each line of algae adding them to an LinkedList
 * then in that LinkedList searches each line for the pull value, then adds
 * them to another LinkedList */

public static void findByHand(String pullOutput) {


Vector<String> pullList = new Vector<String>(); // holds lines that contain the pull value

long startTime = System.nanoTime();

for (String str : list)
{
if (str.contains(pullOutput))
pullList.add(str);
}
long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;
System.out.println("Time of findByHand() with data-structure Vector: " + estimatedTime + "

for (String contains : pullList)
{
System.out.println(contains);
}

}
}
```

```
import org.josql.*;
import com.beust.jcommander.Parameter;
import com.beust.jcommander.JCommander;
import java.util.ArrayDeque;
import java.util.List;
import java.io.File;
import java.util.*;

public class ADeque {

public static ArrayDeque<String> list = new ArrayDeque<String>(); // holds lines of algae

public static void main(String[] args) {

Arguments params = new Arguments();
        JCommander cmd = new JCommander(params, args);
        try {
            cmd.parse(args);
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
            cmd.usage();
        }

        // The debugging flag which will determine whether or not
        // we produce output
        boolean debugOutput = params.getOutput();

// The pull flag will determine which data attritute to pull
String pullOutput = params.getPull().toString();

try{
Scanner scan = new Scanner(new File("../data/algae.csv")); // scan algae

/* Scan until there are no more lines, add them to an arrayList---list---
 * and scan every line in list and search for lines that contain 'winter'
 * and add them to winterList */
while (scan.hasNextLine())
{
String next = scan.nextLine();
list.add(next);
}
System.out.println("By hand:\n");
findByHand(pullOutput);
```

```
System.out.println("\nBy JoSQL:\n");
findByJo(pullOutput);

} catch (java.io.FileNotFoundException FNF) {
System.out.println("Couldn't Find File");
}
}

public static void findByJo(String pullOutput)
{

try {

// Create a new Query.
Query q = new Query ();
//String winter = "winter";
String query = "SELECT * FROM java.lang.String WHERE toString LIKE '%" + pullOutput + "%''

long startTime = System.nanoTime();

// Parse the SQL you are going to use.
q.parse(query);

// Execute the query.
QueryResults qr = q.execute(list);

// Cycle over the query results
List<String> res =  qr.getResults();

long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;

for (String result : res)
{
System.out.println(result);
}
System.out.println("Time of findByJo() with data-structure ArrayDeque: " + estimatedTime +
}

catch(org.josql.QueryParseException e) {
System.out.println("Error One");
}

catch(org.josql.QueryExecutionException e) {
System.out.println("Error Two");
}
}

/* Hand-coded version that finds occurences of winter in algae.csv
 * Does it by scanning each line of algae adding them to an arrayList
 * then in that arrayList searches each line for 'Winter,' then adds
 * them to another arrayList */
public static void findByHand(String pullOutput) {

ArrayDeque<String> pullList = new ArrayDeque<String>(); // holds lines that contain the pu

long startTime = System.nanoTime();

for (String str : list)
{
if (str.contains(pullOutput))
pullList.add(str);
}
long endTime = System.nanoTime();
long estimatedTime = endTime - startTime;
System.out.println("Time of findByHand() with data-structure ArrayDeque: " + estimatedTime

for (String contains : pullList)
 {
  System.out.println(contains);
 }
}
}
```

```
import com.beust.jcommander.Parameter;
import com.beust.jcommander.JCommander;

public class Arguments {
    @Parameter(names = "-debug", description = "Debug mode", arity = 1)
    private boolean debug;

    @Parameter(names = "-pull", required = true, description = "determines what to output'
    private String pull;

    public boolean getOutput() {
        return debug;
    }

    public String getPull() {
        return pull;
    }
}
```