Criterion C: Development

Techniques Used (126 Words)

- Software
  - Netbeans IDE for Java
  - Derby SQL database plugin for Java
  - JFrame
- Data structures
  - ArrayList
    - Used to store data that was retrieved from the SQL database
    - Multiple arrayLists were used to hold objects
      - Variable set was used to create an easier end-user experience with more friendly column names
      - River was used to hold all information about each element in the database
  - SQL databases
    - An SQL database was used, so the program can be slightly modified and have online connectivity features
    - SQL databases were used because of their advanced querying features, offloading some of the sorting and grouping work
  - ResultSet
    - Result set is the data structure that SQL databases return
    - Works similarly to a stack

Evidence of Algorithmic Thinking (429 Words)

Algorithmic thinking was required multiple times to overcome obstacles faced during the development process. To make the program efficient, I had to mind the amount of loops I used, and use pseudocode to plan out the program I was going to make. Because this was working with an SQL database, I demonstrated an understanding of the algorithms used in complex data structures. Another way I demonstrated algorithmic thinking was through the statistics methods, which were created for the "Data Viewer" program.

An example of the algorithmic thinking necessary when handling complex data structures can be found in the "Data Importer" program:

```
try {
    if(rs.next()){
      rs.first();
       while(rs.next()) {
        elementsInList++;
       }
```

```
        }
        rs.first();
       }
```

This demonstrates an understanding of the data structure because it required a very specific method of iteration.

Another example can be found in the “Data Viewer” program, with the “mean()” method, which calculates mean and standard deviation. To complete this method, I had to demonstrate an understanding of the method by which one can reach the mean, and translate that method into code. This required algorithmic thinking:

```
public void mean (ArrayList<Integer> dataToAnalyze) {
    double mean;
    int total = 0;
    for(int i = 0; i < dataToAnalyze.size(); i++){
       total += dataToAnalyze.get(i);
    }
    mean = ((double) total)/((double) dataToAnalyze.size());

    double totalStDev = 0;
    for(int j = 0; j < dataToAnalyze.size(); j++){
       double a = Math.abs((dataToAnalyze.get(j)));
       totalStDev += a/mean;
    }
    totalStDev = totalStDev/(double) dataToAnalyze.size();

    String meanSt = "";
    String stDevSt = "";
    DecimalFormat decform = new DecimalFormat("#.00");
    meanSt = decform.format(mean);
    stDevSt = decform.format(totalStDev);
```

```
        JOptionPane.showMessageDialog(null, "mean: " + meanSt + "\nst. dev: " + stDevSt);
    }
```

Another example can be found in the varStats method, which is very efficient because algorithmic thinking was used. This program uses no loops, because it was designed to be efficient. This is another demonstration of an understanding of complex methods. I used the characteristics of the ArrayList object, which also shows an understanding of the functionality of Java objects:

```
    public void varStats(ArrayList<Integer> dataToAnalyze) {
        int min = dataToAnalyze.get(0);
        int quartile1 = dataToAnalyze.get((dataToAnalyze.size()/4));
        int med = dataToAnalyze.get((dataToAnalyze.size()/2));
        int quartile3 = dataToAnalyze.get((3*(dataToAnalyze.size()/4)));
        int max = dataToAnalyze.get(dataToAnalyze.size()-1);
        JOptionPane.showMessageDialog(null, "min: \t" +  min + "\nquartile1: \t" + quartile1 + "\nmed: \t" + med + "\nquartile3: \t" + quartile3 + "\nmax: \t" + max);
    }
```

I also demonstrated algorithmic thinking in the method that adds columns to the table. This required an understanding of the data structures and methods associated with the TableColumn, Array, and ArrayList objects:

```
    String[] colNames = new String[queriedColumns.size()];
    for(int i = 0; i < queriedColumns.size(); i++){
        TableColumn temp = new TableColumn();
        colNames[i] = queriedColumns.get(i).getJavaName();
        model.addColumn(temp);
    }
    model.setColumnIdentifiers(colNames);
```