1423963

Java – Databases Coursework

Within this coursework I have made a number of different classes, which I have discussed below:

1. **Record.java**

The main decision made within this stage of development was how to store the fields. I decided to use an ArrayList<String>. This, to me, was seen as a sensible decision because it would allow me to access the individual records while maintaining order.

Most of the methods were public as they will need to be accessed to retrieve the individual elements of the record.

Within each class I decided to do testing instead of having a testing class. This is because you can test it directly and the user will know where the problem is occurring. I decided to use claims and tested each method within the record class. I made the Database.java class so that I was able to test the outcome of the record class.

2. **Table.java**

Throughout the whole process, the Table class was hugely refactored at each stage. Here I wanted to create basic functions for creating a database and included insert Row, update Row, insert Column name, insert Column, delete Column, select Row, remove Row, remove multiple rows etc. The main refactoring occurred because I had over complicated the record class and needed to add lots of methods to the table class.

The testing within this class was very similar to the testing in the record class. It highlighted a few issues for me such as the column size being plus one when adding in the autoID later on and also highlighted the issue of what to use to separate aspects of the record. E.g. should a comma be used or a space to represent the different fields. I decided to choose a comma to separate the fields and a new line to separate the records.

I needed to decide how to implement the column headings. I added method 'createColumnName' where I was able to add the first input into the table to be the column headings. Here I was also able to update the column width with the length of the data being inputted into the table.

3. **ChangeFile.java**

In this class, I implemented a simple read to a file and write to a file method. I decided to use BufferedReader and BufferedWriter because this does simple execution of reading and writing and does not do any special parsing. To ensure that all the records were read I had to make sure that the input checked that there was a next line and then secondly read that line otherwise, the records would be read incorrectly.

4. Print.java

The main aim here was to get a suitable layout for a table and input the records into the table. Here I needed to add methods in the table class to get records and column names into the table. Here is where I decided to divide records by a comma and have different records separated by new lines.

I realised I needed to keep a track of how many columns there were and also the width of the individual fields being inputted into the table to ensure the layout of the table was lined up when it was printed. I therefore had to further refactor table.java to ensure that the column width was updated throughout and each time a column was added the column counter increased. This would ensure the correct number of "-" and "|" were printed and alignment was correct. I used 'substring' to ensure that I got the entirety of the record.

5. Keys.java

A lot of refactoring was involved here because I needed to add in another column which automatically gave me a key even if there was one already there. I decided to do automatic ID values incrementing from zero assigned to each record within the table. I had to create a new column name to ensure that the 0 index was 'autoID' and therefore I updated the method 'createColumnName'. One of the main issues I struggled with here was when the user was writing to the file more than once, the 'autoID' would automatically assign another key if duplicates were added. To fix this I created an 'isDuplicate' method. Here it would compare the ID number and check to see if it already exists. If it was a duplicate then the entire record would not be entered into the table.

6. Databases.java

To wrap up the entire database folder I included a list of the files, create a new file and delete a file, create a new directory and move a file from one directory to another. The code for each function was very similar. When listing the files within the directory using the 'listContents' function I was able to make it so that when the user prints the contents it shows whether it is a file or directory. Here I had to make sure that the path to the files was correct and therefore inserted "/" between the file name that the user would enter and the folder name.

In the 'moveFileToDirectory' function I decided to rename the original file to the same file but in a different directory.

7. Test.java

Used to run all the tests in the different classes. To test each class individually you can run javac *.java followed by java "class name".

1423963

Here I have included some examples of the use of the functions and how they interact with the classes:

Using different files and directories

```
createNewDirectory("/Users/username/Documents/Computer Science/Java/Assignment 3/Testing");
createNewFile("hi.txt", "/Users/username/Documents/Computer Science/Java/Assignment 3");
moveFileToDirectory("hi.txt", "Testing", "/Users/username/Documents/Computer Science/Java/Assignment 3");
```

Inputting data into records

```
tab.createColumnName("hi, whos, there");
tab.insertRow("hello, how, are");
tab.insertRow("what, is, up");
tab.insertRow("howdie, 25, ok?");
```

Reading a file and inputting it into the table

```
finishing = cf.readFile("helloworld.txt");
tab.getTablefromFile(finishing);
```

Displaying the table

```
print.createWidth(tab.maxWidth());
print.createColNum(tab.maxCol());
print.printOut(tab.tableOut());
```

Creating a new file

```
createNewFile("test1.txt");
```

Listing all the files and deleting a specific file

```
listContents("/Users/username/Documents/Computer Science/Java/Assignment 3");
deleteFile("test1.txt");”A
```