Program 1 – Hurricane Database

The purpose of this assignment is to practice using the various concepts learned in Chapter 9 of our textbook. Specifically, this assignment will require some familiarity with the following concepts.

- Keyboard input (CPS141)
- Reading data from a file (CPS141)
- Writing a basic class (CPS141)
- Overriding toString (CPS141)
- More String methods (9.3)
- StringBuilder class (9.4)
- Tokenizing Strings (9.5)
- Numeric Wrapper Classes (9.6)
- Working with CSV data (9.7)

This assignment involves creating a simple command line application to find the strongest and fastest hurricane for a given name. The program will prompt the user to enter the name of a hurricane (e.g., "Camille") and press "return". The program will then print out a simple report listing details about the strongest (highest wind speed) and fastest hurricanes for that name in the database. After displaying the report, the program will prompt the user to continue with another search or quit. The database for this program will be provided by the "htracks_na_cps142_dcc.csv" file included with this assignment. Place the file in the root folder of your project.

All program specifications for our course will follow the same pattern used below.

Section 1 describes how the user should interact with the program, or the "External Operation" of the program. Read this section first to understand how the program should work from the user's perspective. It is a guide for what the user should do and see when using the program.

Section 2 describes how the program should be written to work properly, or the "Internal Operation" of the program. Write your code by following the instructions in this section in order.

Section 3 describes the deliverables, or Java files, that should be submitted with the assignment

Section 4 gives some sample output of what the program should look like when it is run.

Section 5 outlines the General Submission Policy for all programming assignments.

Section 6 outlines the General Requirements for all programming assignments.

Section 7 outlines how this programming assignment will be evaluated for a grade.

1 User Interface Specifications

The program should be written to meet the following User Interface specifications. This section corresponds to Item 3 of the "Program Rubric" at the end of this specification. Please refer to the sample output in Section 4. The following command-line interface should be implemented.

Program flow:

1. Display a user-friendly welcome message.
2. Prompt the user to enter a search term.
3. Read the search term from the keyboard.
4. If the search term is found, generate and display a well-formatted report containing the following information:
    a. Display the search term entered by the user.
    b. Display the total number of records found for this search term.
    c. Display data for the hurricane with highest wind speed
        i. Name of Hurricane
        ii. Year
        iii. Number
        iv. ID
        v. Timestamp string
        vi. Category
        vii. Status
        viii. Latitude with one decimal place
        ix. Longitude with one decimal place
        x. Wind speed (knots) with one decimal place
        xi. Pressure (millibars) with one decimal place
        xii. Speed (of the storm in knots) with one decimal place
        xiii. Direction (of the storm in degrees) with one decimal place
    d. Display data for the hurricane with fastest storm speed
        i. Name of Hurricane
        ii. Year
        iii. Number
        iv. ID
        v. Timestamp string
        vi. Category
        vii. Status
        viii. Latitude with one decimal place
        ix. Longitude with one decimal place
        x. Wind speed (knots) with one decimal place
        xi. Pressure (millibars) with one decimal place
        xii. Speed (of the storm in knots) with one decimal place
        xiii. Direction (of the storm in degrees) with one decimal place
5. If the search term is not found, display a "not found" message.

6. Prompt the user to generate another search report.
    a. If user answers affirmatively return to Step 2.
    b. If user answers negatively display a friendly "thank you" message and end the program.

2 Java Implementation

The program should be implemented in Java using the Eclipse IDE as specified below. This section corresponds to Item 4 of the Program Rubric. This program will require two Java (.java) files that must be uploaded to Brightspace. It is recommended that you work on each file separately and in the order that they are described below. It is also recommended that you create a "Playground.java" file to use for testing out the various components prior to putting them all together in the main program.

2.1 StormRecord.java

The StormRecord Java class represents a single storm tracking event record in the database. It will consist of a set of fields that will hold information that we will read from the "htracks_na_cps142_dcc.csv" file. It will also define a single Constructor method taking a String as the argument, "getter" methods only for all fields, and a "toString" method that will be used in generating the report output for the StormRecord Java object.

2.1.1 Fields

The StormRecord class should include the fields listed in the following table. Please note both the type and name of the field. All fields should be declared private.

Do not write any setters for these fields. See the list of required getters in the next section.

Field Type	Field Name	UML
String	id	-id : String
int	year	-year : int
int	number	-number : int
String	name	-name : String
String	timestamp	-timestamp : String
double	latitude	-latitude : double
double	longitude	-longitude: double
String	status	-status : String
double	wind	-wind : double
double	pressure	-pressure : double
int	category	-category : int
double	speed	-speed : double
double	direction	-direction : double

2.1.2 Methods

The methods for the StormRecord class consist of a single constructor, getter (i.e., accessor) methods for all fields, and an override of the `toString` method.

Constructor `+StormRecord(string : String)`

This constructor takes a String object which must be tokenized using a comma "," delimiter. Once the string is tokenized, the array of tokens String objects are used to initialize the instance fields of this class. Refer to Section 9.5 on how to tokenize a String. Be sure to properly trim each token before converting to a numeric value. Refer to Section 9.6 on how to convert String tokens into double values. Tokens are mapped to the instance fields as follows.

Field Type	Field Name	Token Position
String	id	18
int	year	1
int	number	2
String	name	5
String	timestamp	6
double	latitude	19
double	longitude	20
String	status	21
double	wind	22
double	pressure	23
int	category	24
double	speed	25
double	direction	26

Special Notes for mapping tokens:

1. All tokens should be trimmed before using them to set the corresponding field named in the table above. Use the String `trim()` instance method for this.
2. For each `token`, if the trimmed length is 0 then it will not be possible to convert the token String to an appropriate data value. In this case leave the default value. Only set the value if the token as length greater than 0.
3. For all `int` field types use `Integer.parseInt` to convert the trimmed token to an int.
4. For all `double` field types use `Double.parseDouble` to convert the trimmed token to a `double`.

Getters

Write the getter methods for all fields. **DO NOT WRITE ANY SETTER METHODS.**

Method `+toString() : String`

This method will return a well-formatted String object to be used in the final report. Be sure to use the proper @Override annotation to indicate to the compiler that you are overriding this method.

NOTE: You must use the StringBuilder class to create the String object to be returned. See Section 9.4 of the textbook. **DO NOT USE "System.out.println" anywhere in this method**.

When printed to the console this String should produce output similar to the following:

```
Name:          FREDERIC
Year:          1979
Number:        65
ID:            AL111979
Timestamp:     9/12/79 12:00
Category:      4
Status:        HU
Latitude(N):   27.4
Longitude(E): -87.0
Wind(kts):     115.0
Pressure(mb): 943.0
Speed(kts):    11.0
Direction(d): 325.0
```

Note that each line contains a single data item including a label (on the left) and the value (on the right). Notice also that the data columns are nicely left justified. Floating point (double) values are limited to one decimal place. The String data in the right-hand column is displayed as is from the data with no special formatting.

2.2 Main.java

The Main.java file should contain a class called "Main" that has a `public static void main` method declared. Note that capitalization is important in Java. Class names should always have the first character capitalized. Variable names and method names should always start with a lower-case letter. The main program should use the Scanner class and `System.out` methods to implement the command line user interface described in Section 1.

You may write the code for the user interface and report output as seems best to you as long as it fully implements the UI specification in Section 1 and uses the StormRecord `toString` instance method in creating the report. In other words, ***no other StormRecord instance methods should be used in creating the report output except the toString method***. Remember that System.out.print methods use `toString` internally, so it is not always necessary to call it explicitly.

Use the sample output in Section 4 as a guide. It doesn't have to match exactly but must contain all the relevant parts and be well formatted as shown.

Required methods for Main class are listed below.

2.2.1 Methods

The Main class should define and use the following required public static method to help implement the program.

Method `+loadData() : ArrayList<StormRecord>`

A public static method that is responsible for loading the data from the "htracks_na_cps142_dcc.csv" file provided with the project. It should create and return an `ArrayList` consisting of a StormRecord observation record for each row of data in the file.

Things to note about the `loadData` method:

1. Skip the first two lines of the file since they contain the header rows. You can easily do this by using a loop and a Scanner instance to check that the Scanner has a next line and by calling the `nextLine()` method to skip the line.
2. This method will need to add a "`throws FileNotFoundException`" clause to the method header and to the main method header.

Method `main`

Implementation of the console UI should be completed in the main method of your program. You may also write and use private helper methods to help build the UI if you wish. There are many ways this could be accomplished but one possible algorithm is described below. Note that the algorithm is written in **pseudocode**.

ALGORITHM: partialConsoleUserInterface

```
boolean shouldContinue = true

while shouldContinue:

   StormRecord maxWind
   StormRecord maxSpeed
   String searchTerm = get input from the user
   int recordCount = 0

   for StormRecord record in data:

      if searchTerm.length()>0 and record.getName() contains searchTerm:

         recordCount++

         if maxWind is null or record.getWind()>maxWind.getWind():
            maxWind = record
         endIf

         if maxSpeed is null or record.getSpeed()>maxSpeed.getSpeed():
            maxSpeed = record
         endIf

      endIf

   endFor

   display the results for maxWind and maxSpeed

   shouldContinue = ask use if they want to try again

endWhile
```

2.3 Comments

Your programs must always include a project documentation comment at the top of your main program file. For example.

```
/**
 * <Your Name Goes Here>
 * CPS142 – Fall 2024
 * <write the date here>
 * Instructor: Adam Divelbiss
 * Assignment: Program01
 * Purpose: <write the purpose of your program here>
 */
```

Where text between "<" ">" angle brackets is for you to fill in. Remove the angle brackets in your comments. They are just placeholders for demonstration.

All classes and methods, except for methods that override well documented superclass methods, must have a documentation comment. Also be sure to use a single line comment to document/describe each major section in the code

3 Deliverables

The deliverable files will be uploaded to the assignment in Brightspace. Deliverables consist of the following .java files. DO NOT SEND THE .class FILES. I CANNOT USE THEM.

- Main.java
- StormRecord.java

The due date/time is:
Saturday, September 7, 2024, at 11:59PM

4 Sample Output

The following output demonstrates one possibility that would get full credit for Section 3 of the Program Rubric. Text in green is user input.

```
Welcome to the DCC Hurricane Database!

Enter all or part of the name of a North Atlantic Hurricane to search.
The program will find records for hurricanes with the highest wind speed
and highest storm speed from the database for the given name.

Total records: 54111

Hurricane name: Ivan


************************************************************
Hurricane report for: IVAN

Total number of records found: 339

Highest Wind:
Name:          IVAN
Year:          2004
Number:        66
ID:            AL092004
Timestamp:     9/11/04 18:00
Category:      5
Status:        HU
Latitude(N):   18.0
Longitude(E): -79.0
Wind(kts):     145.0
Pressure(mb): 920.0
Speed(kts):    6.0
Direction(d): 295.0


Fastest Storm:
Name:          IVAN
Year:          1980
Number:        81
ID:            AL141980
Timestamp:     10/11/80 15:00
Category:      1
Status:        HU
Latitude(N):   48.2
Longitude(E): -29.5
Wind(kts):     65.0
Pressure(mb): 990.0
Speed(kts):    39.0
Direction(d): 45.0
************************************************************

Would you like to try another search? n

Thanks for using the DCC Hurricane Database!
```

5 General Submission Policy

You may submit your code multiple times up until the due date/time. Any submission after the due date/time will be subject to the late assignment policy described in the course Syllabus. In all cases the last code you submit will be used for the grade.

6 General Requirements

All code you write for this course should meet the following general requirements:

1. All code should be your own work and not copied from another student, the internet, or any solution guide for the textbook.

2. Code from the following sources may be used or modified to complete the project:
    a. Any code provided by me in the class notes.
    b. Any code provided by the textbook itself (excluding any solution guides).
    c. Any code written by you for this or another course.

3. All Java classes should have a documentation comment including, author, date, course (i.e., CPS142-Fall 2024), and a purpose or brief description statement.

4. All methods must have properly formatted and descriptive documentation comments. The only exception is methods that override well documented interface or superclass methods. Private methods must also be commented.

5. Any method with more than just a few lines of code should have additional single-line comments describing important parts of the algorithm.

6. The use of break, continue, or exit statements inside loops is not acceptable in the implementation of any class. If found points will be subtracted from Item 5 of the Programming Rubric.

NOTE: The above requirements should be followed only where applicable.

7 Program Rubric

Your grade for this assignment will be computed using the following rubric.

Item	Description	Percent of Grade
1	Compilation: Compiles with no syntax errors.	10
2	Execution: Runs with no crashes or runtime errors.	10
3	External Operation: Written to meet all requirements for input and output as specified or implied by the problem.	30
4	Internal Operation: Written to meet all requirements for internal operation as specified as specified or implied by the problem statement.	30
5	Coding Style: Written and formatted in accordance with best practices presented in class (indentation, class names, variable names, method names, etc.).	5
6	Project Comment: Contains a **documentation comment** with the assignment name, author, date, class (i.e., CPS142-Fall 2024), and purpose/description at the top of each file.	5
7	Documentation: All methods, classes, and interfaces have **documentation comments**. Methods that override well-documented interface method and the main method are exempt.	5
8	Comments: Includes sufficient other regular comments to describe important parts of the code.	5
	Total	**100**