

Introduction

The product is a Java program coded on Netbeans which allows users to log injury and treatment data of athletes. The Netbeans GUI using Java Swing components such as text fields and combo boxes creates an interface where users access and input data. Algorithms will search and sort rosters of players and calculate statistics at the click of a button.

Word Count: 60

Summary of Techniques Used

- Use of Flag Values
 - E.g. -999, not set yet
- Parsing
- Inheritance Through a Hierarchy
 - Abstraction
- Instantiation of Classes
- Parameter Passing
- For Loop
- Simple and Compound Selection (If/Else)
- Use of ||, &&, !, .equals() on Conditionals
- Methods Returning a value
- Methods Taking in Parameters
- Arrays
- Array of Objects
- User Defined Objects Made From an OOP "Template" Class
- Encapsulation of Private Methods with Accessors
- Bubble Sort
- Sequential Search
- Error Handling
- JOptionPane Popups
- GUI Tabs
- GUI Display Table
 - Printing data on jTable using for loop with conditionals (If)
- Use of GUI features: TextField.setText(), ComboBox.getSelectedItem(), etc.

Structure of the Program

What: Inheritance was implemented to create a hierarchy which extends the Player class. The subclass InjuryAndTreatment, includes attributes such as injured body part, injury type, injury date, treatment type and etc. The player class has an array of injuries assigned to each player (element of the players array). There is also the main class, MainGUI, which contains methods that allow users to interact with the program, input data and view outputs. It also uses sorting and searching methods from the SortAndSearchPlayers class.

Why: Inheritance was used as the injuries and treatments belongs to players. Object oriented programming with subprograms and objects was beneficial as inheritance and encapsulation meant attributes could be passed on and modified only when appropriate accessor and modifier methods are called. Abstraction allowed for ease of debugging, maintenance and testing where I was able to identify and solve problems one at a time.

Word Count: 143

Data Structures Used

Some examples of general data structures include arrays, array lists, linked lists, stacks, queues etc. I personally implemented the use of arrays to organize the players and their injuries when they were added. This use of arrays allowed for efficient organization of objects into a structure necessary in this program. Players have injuries therefore having an injuries array belonging to each player (an element in the players array) helped the above. This also allowed for easy searching and sorting necessary in the program. Searching for a player was necessary to view their data or add injuries to them, and using arrays allowed for the use of a sequential search which starts at the beginning of the array and goes through each element looking for the key. Sorting was also a necessity in the data tables and the use of arrays allowed for the implementation of a bubble sort which with successive passes through the array compares neighboring array element pares and sorts it.

Word Count: 163

Main Unique Algorithms

1. Entering Players and Associated Error Handling

```
819 private void addPlayerButtonMouseReleased(java.awt.event.MouseEvent evt) {
820     // TODO add your handling code here:
821     String name = "not_set_yet";
822     int gradeLevel = -999;
823     boolean nonStudent = nonStudentCheckBox.isSelected();
824     boolean nonAthlete = nonAthleteCheckBox.isSelected();
825     int studentID = Integer.parseInt(studentIDTF.getText());
826     String gender = "";
827     if(genderMaleRadioButton.isSelected()){
828         gender = "Male";
829     }
830     if(genderFemaleRadioButton.isSelected()){
831         gender = "Female";
832     }
833     String sport = sportComboBox.getSelectedItem() + "";
834     String playingLevel = playingLevelComboBox.getSelectedItem() + "";
835     String season = seasonComboBox.getSelectedItem() + "";
836
837     if(fullNameTF.getText().isEmpty()){
838         JOptionPane.showMessageDialog(null, "Please input a name.", "Error", JOptionPane.ERROR_MESSAGE);
839     }else if(Integer.parseInt(gradeLevelTF.getText()) < 1 || Integer.parseInt(gradeLevelTF.getText()) > 12){
840         JOptionPane.showMessageDialog(null, "Please input a grade level from 1 to 12.", "Error", JOptionPane.ERROR_MESSAGE);
841     }else{
842         name = fullNameTF.getText();
843         gradeLevel = Integer.parseInt(gradeLevelTF.getText());
844         players[counter] = new Player(name, nonStudent, nonAthlete, studentID, gender, gradeLevel, sport, playingLevel, season, injurie
845         counter++;
846     }
847
848     fullNameTF.setText("");
849     nonStudentCheckBox.setSelected(false);
850     nonAthleteCheckBox.setSelected(false);
851     studentIDTF.setText("");
852     genderMaleRadioButton.setSelected(false);
853     genderFemaleRadioButton.setSelected(false);
854     gradeLevelTF.setText("");
855     sportComboBox.setSelectedIndex(0);
856     playingLevelComboBox.setSelectedIndex(0);
857     seasonComboBox.setSelectedIndex(0);

```

Input from the GUI is taken in via GUI features for attributes of the Player class and assigned to an element of the Players array. If the name isn't entered or the grade level's input is out of

range, there will be an error message that pops up via a JOptionPane and the player won't be added.

2. Entering Injuries and Associated Error Handling

```
782 private void addButtonMouseReleased(java.awt.event.MouseEvent evt) {
783     // TODO add your handling code here:
784     SortAndSearchPlayers sortClass = new SortAndSearchPlayers();
785     int key = sortClass.sequentialSearchForPlayerName(players, playerTF.getText());
786
787     String injuryBodyPart = "not set yet";
788     String injuryType = "not set yet";
789     String injuryEvaluation = evaluationTF.getText();
790     String injuryDay = dateDayComboBox.getSelectedItem() + "";
791     String injuryMonth = dateMonthComboBox.getSelectedItem() + "";
792     String injuryYear = dateYearComboBox.getSelectedItem() + "";
793     String treatmentType = treatmentTypeComboBox.getSelectedItem() + "";
794     String treatmentNotes = notesTF.getText();
795
796     if(playerTF.getText().isEmpty()){
797         JOptionPane.showMessageDialog(null, "Please input a player's name.", "Error", JOptionPane.ERROR_MESSAGE);
798     }else if(bodyPartTF.getText().isEmpty()){
799         JOptionPane.showMessageDialog(null, "Please input the player's injured body part.", "Error", JOptionPane.ERROR_MESSAGE);
800     }else if(injuryTypeTF.getText().isEmpty()){
801         JOptionPane.showMessageDialog(null, "Please input the player's injury type.", "Error", JOptionPane.ERROR_MESSAGE);
802     }else{
803         bodyPartTF.getText();
804         injuryTypeTF.getText();
805         InjuryAndTreatment injuryAndTreatment = new InjuryAndTreatment(injuryBodyPart, injuryType, injuryEvaluation, injuryDay, injuryM
806         players[key].setNextInjury(injuryAndTreatment);
807     }
808
809     bodyPartTF.setText("");
810     injuryTypeTF.setText("");
811     evaluationTF.setText("");
812     dateDayComboBox.setSelectedIndex(0);
813     dateMonthComboBox.setSelectedIndex(0);
814     dateYearComboBox.setSelectedIndex(0);
815     treatmentTypeComboBox.setSelectedIndex(0);
816     notesTF.setText("");
817 }
```

Input from the GUI is taken in via GUI features for attributes of the InjuryAndTreatment class and assigned to an element of the injuries array. If the player's name, injury type or injured body part isn't entered, there will be an error message that pops up via a JOptionPane and the injury won't be added. Error messages here and above were programmed using if/else statements to prevent adding injuries or players when there is no input for essential information or input is incorrect.

3. Output on JTable and Sorting

```
39 private void refreshDisplayTable(){
40     int rowToOverWrite = 0;
41     for(int row = 0; row < players.length; row++){
42         if(players[row].getStudentID() != -999){
43             playerTable.setValueAt(players[row].getName(), rowToOverWrite, 0);
44             playerTable.setValueAt(players[row].getStudentID(), rowToOverWrite, 1);
45             playerTable.setValueAt(players[row].getGender(), rowToOverWrite, 2);
46             playerTable.setValueAt(players[row].getGradeLevel(), rowToOverWrite, 3);
47             playerTable.setValueAt(players[row].getSport(), rowToOverWrite, 4);
48             playerTable.setValueAt(players[row].getPlayingLevel(), rowToOverWrite, 5);
49             playerTable.setValueAt(players[row].getSeason(), rowToOverWrite, 6);
50             rowToOverWrite++;
51         }
52     }
53 }
```

```
865 private void sortByNameButtonMouseReleased(java.awt.event.MouseEvent evt) {
866     // TODO add your handling code here:
867     SortAndSearchPlayers sortClass = new SortAndSearchPlayers();
868     sortClass.sortByPlayerNames(players);
869     for(int i = 0; i < players.length; i++){
870         System.out.println(players[i].getName());
871     }
872     refreshDisplayTable();
873 }
```

```

22 public void sortByPlayerNames(Player[] players) {
23     int n = players.length;
24     boolean sorted = false;
25     while (!sorted) {
26         n--; //It is the n which will result in one less comparison happening each outer pass;
27         //whereas, with the first bubble sort we could use the 'pass' variable used for the for loop.
28         sorted = true;
29         for (int i=0; i < n; i++) {
30             if (players[i].getName().compareTo(players[i+1].getName()) > 0){
31                 Player temp = players[i];
32                 players[i] = players[i+1];
33                 players[i+1] = temp;
34                 sorted = false; //as in the second bubble sort, if swapping happens we'll want to continue, and so
35                                 //with sorted re-set to false again, the while loop continues
36             }
37         }
38     }
39 }

```

Displaying players on the data table goes through a for loop to display all added players and their information for all columns of the table. An example of sorting done is by name. The `sortByNameButtonMouseReleased` method calls the `sortByPlayerNames` method in the `SortAndSearchPlayers` class which uses a bubble sort then calls the `refreshDisplayTable` method to show the sorted data. This same structure is used to sort by sport, playing level and season.

4. View Player Data JOptionPane Popup, Sequential Search For Name and Associated Error Handling

```

905 private void viewPlayerDataButtonMouseReleased(java.awt.event.MouseEvent evt) {
906     // TODO add your handling code here:
907     SortAndSearchPlayers sortClass = new SortAndSearchPlayers();
908     int key = sortClass.sequentialSearchForPlayerName(players, nameTF.getText());
909
910     String playersInjuredBodyPart = "";
911     String playersInjuryType = "";
912     String playersTreatmentType = "";
913
914     if(nameTF.getText().isEmpty()){
915         JOptionPane.showMessageDialog(null, "Please input the name of the player you would like to see.", "Error", JOptionPane.ERROR_ME
916     }else if(key == -1){
917         JOptionPane.showMessageDialog(null, "The player you are looking for doesn't exist.", "Error", JOptionPane.ERROR_MESSAGE);
918     }else{
919         InjuryAndTreatment [] injuries = players[key].getInjuries();
920         for(int i = 0; i < injuries.length; i++){
921             playersInjuredBodyPart = injuries[i].getInjuryBodyPart();
922             playersInjuryType = injuries[i].getInjuryType();
923             playersTreatmentType = injuries[i].getTreatmentType();
924         }
925         JOptionPane.showMessageDialog(null, "Name: " + (players[key].getName()) + "\nID#: " + (players[key].getStudentID()) + "\nGender
926         + "\nGrade: " + (players[key].getGradeLevel()) + "\nSport: " + (players[key].getSport()) + "\nPlaying Level: " + (players[key].
927         + "\nSeason: " + (players[key].getSeason()) + "\nInjury Part: " + playersInjuredBodyPart + "\nInjury Type: " + playersInjuryTyp
928         + "\nTreatment Type: " + playersTreatmentType, "Player Data", JOptionPane.PLAIN_MESSAGE);
929     }
930 }

```

```

13 public int sequentialSearchForPlayerName(Player arr[], String player){
14     for(int i = 0; i < arr.length; i++){
15         if(arr[i].getName().equals(player)){
16             return i;
17         }
18     }
19     return -1;
20 }

```

Firstly, the player's name who's data is to be shown is taken in using GUI features and that is searched in the `sequentialSearchForPlayerName` method from the `SortAndSearchPlayers` class. A sequential search was implemented because it was the most efficient way to search. Then the Key for which element from the players array the player is located is returned and essential information is gotten using get methods. There is also a for loop looping through the injuries array to get information for the player's injury. If the name isn't entered or is incorrect, there will be an error message that pops up via a `JOptionPane`.

Word Count: 347

User Interface/GUI Work

What: GUI was used in the creation of this program for users to be able to interact with the program in a convenient, user friendly setting.

- JTextFields: For users to enter subjective information.
- JButtons: For users to click and prompt the functioning of methods.
- JTables: Used to display data in a compact, organized manner.
- JComboBoxes: For users to select from a list of available options.
- JCheckBoxes: For users to assign true/false for boolean in the code.
- JRadioButtons and JButtonGroups: For users to be prompted to select one of the possible inputs.
- JTabbedPane and JPanels: Organizes user interface and separates related functions into individual tabs.

Why: JTextFields were used for example in Label 1, to input the player's full name. This is good for the client as they can input subjective information as every player's name is different. Next, JRadioButtons and JButtonGroups were used as seen in Label 2, to input the player's gender. This GUI element prompts the user to select one of the two options as an athlete's gender must be either male or female. JComboBoxes were used in example 3, to input what sport the player plays. There is a set amount of sports an athlete can play at the school, so the JComboBox was an efficient tool which users use to input this information. JTabbedPane and JPanels were used as seen in Label 5 to organize the user interface and separate related functions into pages so the user can navigate the program easily. Lastly, JButtons were used in examples 4, 6 and 7 to be able to execute certain actions like adding a player, prompting a popup to see player data and sorting the table.

The screenshot shows a Java Swing GUI with two panels. The left panel is for adding a player, and the right panel is for displaying and managing a table of players.

Left Panel (Add Player):

- Import Roster: URL Import
- OR
- Full Name: **1**
- Non-Student: Yes
- Non-Athlete: Yes
- Student ID#:
- Gender: Male **2** Female
- Grade Level:
- Sport: **3**
- Playing Level:
- Season:
- Add Player **4**

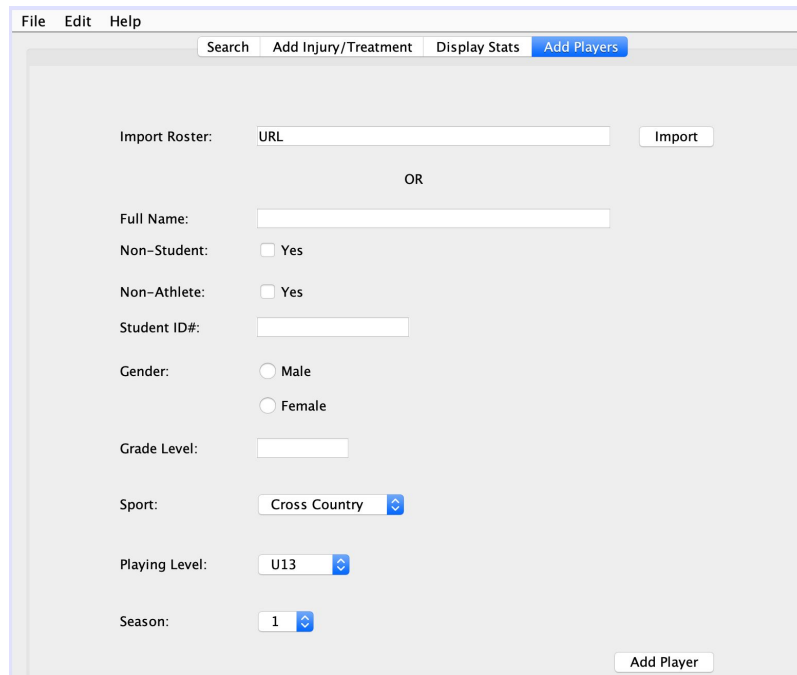
Right Panel (Table View):

- Name: View Player Data **6**
- 7** Sort By Name Sort By Sport Sort By Playing Level Sort By Season
- *Sorting is in alphabetical order from A to Z
- Table with columns: Name, ID#, Gender, Grade, Sport, Level, Season
- Refresh Table

Word Count: 198

Software Tools Used

Netbeans, an integrated developmental program for java used by professionals around the world was used in the development and coding of this product. It was used as it allows for creation of a user interface using Java Swing components. It also supplies prewritten methods, constructors, and shortcuts used for object oriented programming, which made the development much more efficient.



The screenshot shows a Java Swing window titled "MainGUI.java" with a menu bar containing "File", "Edit", and "Help". Below the menu bar are four buttons: "Search", "Add Injury/Treatment", "Display Stats", and "Add Players". The main content area contains a form for adding a player. It starts with "Import Roster:" followed by a text field labeled "URL" and an "Import" button. Below this is an "OR" separator. The form then includes several input fields: "Full Name:", "Non-Student:" (with a "Yes" checkbox), "Non-Athlete:" (with a "Yes" checkbox), "Student ID#:", "Gender:" (with radio buttons for "Male" and "Female"), "Grade Level:", "Sport:" (with a dropdown menu showing "Cross Country"), "Playing Level:" (with a dropdown menu showing "U13"), and "Season:" (with a dropdown menu showing "1"). At the bottom right of the form is an "Add Player" button.

```
779     System.exit(0);
780 }
781
782 private void addButtonMouseReleased(java.awt.event.MouseEvent evt) {
783     // TODO add your handling code here:
784     SortAndSearchPlayers sortClass = new SortAndSearchPlayers();
785     int key = sortClass.sequentialSearchForPlayerName(players, playerTF.getText());
786
787     String injuryBodyPart = "not set yet";
788     String injuryType = "not set yet";
789     String injuryEvaluation = evaluationTF.getText();
790     String injuryDay = dateDayComboBox.getSelectedIndex() + "";
791     String injuryMonth = dateMonthComboBox.getSelectedIndex() + "";
792     String injuryYear = dateYearComboBox.getSelectedIndex() + "";
793     String treatmentType = treatmentTypeComboBox.getSelectedIndex() + "";
794     String treatmentNotes = notesTF.getText();
795
796     if(playerTF.getText().isEmpty()){
797         JOptionPane.showMessageDialog(null, "Please input a player's name.", "Error", JOptionPane.ERROR_MESSAGE);
798     }else if(bodyPartTF.getText().isEmpty()){
799         JOptionPane.showMessageDialog(null, "Please input the player's injured body part.", "Error", JOptionPane.ERROR_MESSAGE);
800     }else if(injuryTypeTF.getText().isEmpty()){
801         JOptionPane.showMessageDialog(null, "Please input the player's injury type.", "Error", JOptionPane.ERROR_MESSAGE);
802     }else{
803         bodyPartTF.getText();
804         injuryTypeTF.getText();
805         InjuryAndTreatment injuryAndTreatment = new InjuryAndTreatment(injuryBodyPart, injuryType, injuryEvaluation, injuryDay,
806             players[key]).setNextInjury(injuryAndTreatment);
807     }
808
809     bodyPartTF.setText("");
810     injuryTypeTF.setText("");
811     evaluationTF.setText("");
812     dateDayComboBox.setSelectedIndex(0);
813     dateMonthComboBox.setSelectedIndex(0);
814     dateYearComboBox.setSelectedIndex(0);
815     treatmentTypeComboBox.setSelectedIndex(0);
816     notesTF.setText("");
817 }
```