PassTrack is a program created to track multiple statistics of the NYIT Ultimate Frisbee Team. It includes the following features:

**Features**

* Saving and Loading Statistics
  + Completions (Throws)
  + Catches
  + Goals
  + Drops
  + Bad Throws
  + Blocks
  + Assists
* Adding Players
* Adding Teams
* Listing Players
* Tracking Games

**Concepts**

1. Polymorphism
   1. Throughout the PassTrack, polymorphism was used to simplify code, and make methods more efficient.
2. Abstraction
   1. Abstraction was used to clean up code, and make classes simpler.
3. Encapsulation
   1. Encapsulation was implemented to hide secure and simplify numerous parts of the program.
4. Exception Handling
   1. Exception handling was used in reading or writing to a file.
   2. Was very useful to ensure PassTrack could safely interact with file storage to save / load statistics.

**Code Examples**

**Capitalization**: This code capitalizes the first letter of player names, then returning the result.

|  |
| --- |
| //Capitalize and Return Value  public static String capitalizeFirstLetter(player index) {  String result = "Player Not Found";  if(!"".equals(index.getName()))  result = index.getName().substring(0,1).toUpperCase() + index.getName().substring(1).toLowerCase();  return result;    } |

**Rank Points**: To obtain an understanding of a player’s skill level, this calculation was used. Taking in the majority of statistics, a number was assigned and used to sort the “Stats Selection” Sheet

|  |
| --- |
| **//Find rank points  public double getRankPoint() {  double rankPoint=0;  rankPoint = (this.goals\*2 + this.assists + this.blocks\*2 //Points with bonuses  + this.completions + this.catches) // Normal one point   / (this.badThrows\*.5 + this.drops\*.5 + 1) ; // Negative stats  return rankPoint;  }** |

**Exception Handling:** The following was used to write text to a file and save data to the program

|  |
| --- |
| public static void writeFile(String file,String text) throws IOException {  FileOutputStream outputStream = new FileOutputStream(file);  try {  byte[] strToBytes = text.getBytes();  outputStream.write(strToBytes);  outputStream.close();  }catch(IOException e) {  e.printStackTrace();  }  } |

The following section is what is saved to a text file: The first number is the unique ID of the player, the word being the action, and the last number being the quantity. While I could have mapped the word to a number, I chose not to for readability purposes.

|  |
| --- |
| 1 completions 1 1 catches 1 6 drops 1 10 completions 1 11 completions 1 11 catches 1 15 completions 1 15 catches 1 10 completions 1 11 completions 1 11 catches 1 1 catches 1 6 drops 1 10 completions 1 |

The following code is used for encapsulation for the “Player” class.

|  |
| --- |
| //Goals  public void addGoal() {  this.goals++;  }  public void removeGoal() {  this.goals--;  }  public int getGoals() {  return this.goals;  }  public void setGoals(int goals) {  this.goals = goals;  }    //Completions  public void addCompletion() {  this.completions++;  }  public void removeCompletion() {  this.completions--;  }  public int getCompletions() {  return this.completions;  }  public void setCompletions(int completions) {  this.completions = completions;  } |

**Abstraction**: The last example, shows an example of abstraction. All methods larger than a line are placed into a separate method for organizational and efficiency purposes. The switch structure was used to eliminate “bad input” from the user.

|  |
| --- |
| //Main loop and menu  while(running) {  System.out.println("\nFrisbee Tracker 1.2");  System.out.println("1. List Players");  System.out.println("2. List Stats");  System.out.println("3. Add Players");  System.out.println("4. Add Team");  System.out.println("5. Start Match");  System.out.println("6. Save");  System.out.println("7. Exit");  input = r.nextLine();  switch(input) {    //List All Players  case "1":  method.listPlayers();  break;    //Sort Player Array / Print Stats  case "2": player.playerArray.sort(Comparator.comparing(player::getRankPoint).reversed());  method.listStats();  break;    //Add Players  case "3":  method.addPlayers();  break; |

**Notes**

While I’m pleased with how PassTrack turned out, there were a few key concepts I would have liked to change.

* Saving and Loading Statistics
  + One feature that could have been added is to append the file instead of completely re-writing it from scratch.
* Keeping track of Positions
  + Instead of an extra attribute of the player object, I would have rather used a buffer to temporarily store the passes from the game.