

Criterion C: Development

Data Structure

The data structure I choose for storing the player data was the **ArrayList**. The ArrayList class is part of Java's Collection classes. Since the list to be maintained was fairly small in size (less than 300 players) efficiency was not a major concern therefore I opted for the convenience and ease of use that the ArrayList class provides. Its capability to easily add and remove records combined with storing data of an unknown length made it the perfect choice.

In writing the program I utilized many programming techniques. Below is a list of some of those techniques.

List of Techniques

- Class Decomposition
- Method Decomposition – both void and non-void methods
- Parameter Passing
- Data Validation
- Linear Search
- Sorting
- File I/O
- Printer Output

Class Decomposition

I decomposed the problem into seven classes: Player, Group, MaxDatabase, WeightLiftingProgram, WeightTraining, PrintWeightProgram, and PrintGroups. Organizing the program into classes made it easier to design, write, and debug the program. Below is a section of code from the Player class.

```
public class Player implements Serializable
{
    // instance variables
    private String name;
    private int benchMax;
    private int squatMax;
    private int inclineMax;
    private int powerMax;
    private int classification;

    // constructors
    public Player()
    {
        name = "";
        benchMax = 0;
        squatMax = 0;
        inclineMax = 0;
        powerMax = 0;
        classification = 9;
    }
}
```

The Player class encapsulates all the information for one player. By grouping the data this way, I was able to create an **ArrayList** of Players.

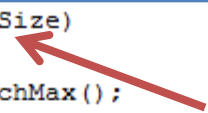
I chose an ArrayList as my data structure because it allows for an unknown number of Players. It also provides convenient methods for adding and removing Players from the list.

Method Decomposition

Method decomposition allowed me to organize the code into classes according to the task the code performed. Below is an example of a non-void, also none as a function, which takes one parameter of type integer. This method returns an ArrayList of Group objects organized into groups according to the players' bench press max.

```
public ArrayList<Group> createGroups(int groupSize)
{
    ArrayList <Player> list = sortPlayersByBenchMax();
    ArrayList <Group> groups = new ArrayList<>();

    Group group = new Group(groupSize);
    for(int i = 0; i < list.size(); i++)
    {
        group.addPlayer(list.get(i));
        if(i != 0 && i % groupSize == 0)    // add group every groupSize play
        {
            groups.add(group);
            group = new Group(groupSize);
            group.addPlayer(list.get(i));
        }
    }
    // if you still have players left create a group smaller than groupsize
    if(groups.size() * groupSize < list.size())
    {
        int num = list.size() - groups.size() * groupSize;
        group = new Group(groupSize);
        for(int i = 0, j = num-1; i < num; i++, j--)
        {
            group.addPlayer(list.get(list.size() - j - 1));
        }
        groups.add(group);
    }
    return groups;
}
```



parameter passing

Data Validation

The program uses a menu system to navigate all of its features. Users enter integer values to select the different options. The method below is called each time the user is supposed to enter an integer. If an integer value is not entered the method prompts the user to enter the value again. A user-friendly program should do its best to prevent a user from crashing the program.

```

public int validateIntegerInput(String prompt)
{
    int ans = 0;
    boolean flag;

    do
    {
        flag = true;
        System.out.print(prompt); // display input prompt
        if(keyboard.hasNextInt()) // if input is an integer
        {
            ans = keyboard.nextInt();
        }
        else // not an integer
        {
            System.out.println("Invalid Entry. Try again.");
            flag = false;
        }
        keyboard.nextLine(); // clear buffer
    }
    while(flag == false);

    return ans;
}

```

The Scanner class provides methods for data validation. The **hasNextInt** method returns false if the data input is not an integer. This allowed me to trap any bad input so I could prompt the user to enter the data again.

When the user wants to print a workout program card he must enter the current week of the program (The workout program is 10 weeks). The method below requires that the user type an integer between 1 and 10 if he does not the method prompts him to enter the value again until it is in the proper range.

```

public int validateWeekNum(int week)
{
    while(week < 1 || week > 10)
    {
        week = validateIntegerInput("Enter Program Week (1-10) -->");
    }

    return week;
}

```

Sorting

Sorting the players by bench press max was necessary in order to group the players by their max. Below is the method that sorts the players by bench press max using the selection sort algorithm.

```
public ArrayList <Player> sortPlayersByBenchMax()
{
    // create new list and copy player's data into it
    ArrayList <Player> list = copyList(players);

    // selection sort algorithm

    int i, j;
    int max;
    Player temp;

    for (i = 0; i < list.size()-1; i++) // advance through list one player at a time
    {
        max = i;
        for (j = i+1; j < list.size(); j++) // find largest player max in list
        {
            if (list.get(j).getBenchMax() > list.get(max).getBenchMax())
                max = j;
        }
        // swap largest max with current max
        temp = list.get(i);
        list.set(i, list.get(max));
        list.set(max, temp);
    }
    return list; // return sorted list
}
```

The selection algorithm was chosen over faster algorithms because the data set is fairly small and it is easy to implement.

Linear Search

When a user wants to print a player's workout program card he enters the player's name and the program performs a linear search through the ArrayList to find the player. Below is the method that implements this algorithm.

```

public Player searchByName(String name)
{
    // linear search algorithm
    for(Player player : players)
    {
        if(player.getName().equals(name))
        {
            return player;
        }
    }

    return null; // player not in list
}

```

The Linear Search algorithm was chosen over the Binary Search algorithm because the data set was fairly small and the data is not sorted. In order to use the Binary Search the data would first need to be sorted which would negate the efficiency benefit of the Binary Search.

File I/O

Serialization was chosen as the method used to read and write the database to a file on a secondary storage device. Since the ArrayList class supports serializing this was the best option. Below is the method used to save the data to a file.

```

public void saveFile()
{
    FileOutputStream fileID;
    ObjectOutputStream outFile;

    try
    {
        // Create the output stream
        fileID = new FileOutputStream(FILENAME);
        outFile = new ObjectOutputStream(fileID);

        // Write the ArrayList to the file
        outFile.writeObject(players);

        // Close the file
        outFile.close();
    }
    catch (IOException e)
    {
        System.out.println("Error writing to data file: " + e.getMessage());
    }
}

```

Printer Output

Printer output was the most important feature implemented in the program. This feature was implemented in the class PrintWeightProgram. The methods used in this class were adopted from the following site: <https://docs.oracle.com/javase/tutorial/2d/printing/>.

You can see sample printouts in ([Appendix C](#)).

Words: 640