```java
package org.bwagner;

import java.io.Serializable; // needed to save to data file as an object

/*
 This class stores each player’s name, classification(9, 10, 11, 12),
 and weight max for each of the four exercises: bench, squat, incline,
 and power clean.
 */
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;
    }

    public Player(String n, int b, int s, int i, int p, int c) {
        name = n;
        benchMax = b;
        squatMax = s;
        inclineMax = i;
        powerMax = p;
        classification = c;
    }

    // accessor methods
    public String getName() {
        return name;
    }

    public int getBenchMax() {
        return benchMax;
    }

    public int getSquatMax() {
        return squatMax;
    }

    public int getInclineMax() {
        return inclineMax;
```

```
54      }
55
56      public int getPowerMax() {
57          return powerMax;
58      }
59
60      public int getClassification() {
61          return classification;
62      }
63
64      // mutator method
65      public void setName(String n) {
66          name = n;
67      }
68
69      public void setBenchMax(int b) {
70          benchMax = b;
71      }
72
73      public void setSquatMax(int s) {
74          squatMax = s;
75      }
76
77      public void setInclineMax(int i) {
78          inclineMax = i;
79      }
80
81      public void setPowerMax(int p) {
82          powerMax = p;
83      }
84
85      public void setClassification(int c) {
86          classification = c;
87      }
88
89      // toString
90      @Override
91      public String toString() {
92          return String.format("%-17s%4s%n%-17s%4s%n%-17s%4d%n%-17s%4d%n%-17s%4d%n%-17s%4d",
93                  "Name:", name,
94                  "Classification:", classification,
95                  "Bench Max:", benchMax,
96                  "Squat Max:", squatMax,
97                  "Incline Max:", inclineMax,
98                  "Power Clean Max:", powerMax);
99      }
100 }
```

```
 1 package org.bwagner;
 2
 3 import java.util.*;
 4 import java.io.*;
 5
 6 /*
 7   This class maintains a list of Player's. It also provides methods for
 8   manipulating this list.
 9
10 */
11
12 public class MaxDatabase implements Serializable
13 {
14     // constants
15     public static String FILENAME = "weightTraining.dat";  // data file name
16     public static String BACKUP = "weightTraining.bak";  // backup file name
17
18     // instance variables
19     private ArrayList<Player> players;  // database of Players
20
21     // constructor
22     public MaxDatabase()
23     {
24         players = new ArrayList<>();
25         readFile();
26     }
27
28     /*
29         This method reads the data file if it exists and loads the data into the
30         database. If the data file does not exist it creates it.
31     */
32     public void readFile()
33     {
34         FileInputStream fileID;
35         ObjectInputStream inFile;
36
37         try
38         {
39             // Create a stream for reading in objects
40             fileID = new FileInputStream(FILENAME);
41             inFile  = new ObjectInputStream(fileID);
42
43             // Read all the objects and put them in the ArrayList
44             players = (ArrayList <Player>) inFile.readObject();
45
46             makeBackupFile();  // make backup
47
48             // Close the stream
49             inFile.close();
50         }
51         catch(FileNotFoundException e) // Data file does not exist; create it
52         {
```

```
53              FileOutputStream newFileID;
54              ObjectOutputStream outFile;
55
56              try
57              {
58                  // Create the output stream
59                  newFileID = new FileOutputStream(FILENAME);
60                  // create new data file
61                  outFile   = new ObjectOutputStream(newFileID);
62
63                  // Close the file
64                  outFile.close();
65
66                  return;   // exit method now;
67              }
68              catch (IOException ex) // can't create data file
69              {
70                  System.out.println("Error creating data file: " + ex.getMessage());
71              }
72          }
73          catch(IOException exception) // a general IO error possibly corrupt file
74          {
75              System.out.println("Error reading data file: " + exception.getMessage());
76          }
77          catch(ClassNotFoundException e)  // needed because of cast above
78          {
79              System.out.println("Error trying to open file: " + e.getMessage());
80          }
81      }
82
83      /*
84         This method save the database to the data file.
85      */
86      public void saveFile()
87      {
88          FileOutputStream fileID;
89          ObjectOutputStream outFile;
90
91          try
92          {
93              // Create the output stream
94              fileID = new FileOutputStream(FILENAME);
95              outFile   = new ObjectOutputStream(fileID);
96
97              // Write the ArrayList to the file
98              outFile.writeObject(players);
99
100             // Close the file
101             outFile.close();
102         }
103         catch (IOException e)
104         {
```

```
104         {
105             System.out.println("Error writing to data file: " + e.getMessage());
106         }
107     }
108
109     /* This method saves the current data in the data file to a backup file before
110        the data from the data file is loaded into the database.
111     */
112     public void makeBackupFile()
113     {
114         FileOutputStream fileID;
115         ObjectOutputStream outFile;
116
117         try
118         {
119             // Create the output stream
120             fileID = new FileOutputStream(BACKUP);
121             outFile   = new ObjectOutputStream(fileID);
122
123             // Write the ArrayList to the file
124             outFile.writeObject(players);
125
126             // Close the file
127             outFile.close();
128         }
129         catch (IOException e)
130         {
131             System.out.println("Error writing to backup file: " + e.getMessage());
132         }
133
134     }
135
136     /*
137         @return the number of players in database
138     */
139     public int getSize()
140     {
141         return players.size();
142     }
143
144     /*
145         Adds a Player to database
146         @param player the player to be added
147     */
148     public void addPlayer(Player player)
149     {
150         players.add(player);
151     }
152
153     /* Deletes a Player from database
154        @param player the player to be removed
155     */
```

```
156     public void deletePlayer(Player player)
157     {
158         players.remove(player);
159     }
160 
161     /*
162        Deletes every player in the database and clears data file.
163     */
164     public void clearDatabase()
165     {
166         players = new ArrayList<>();  // clear ArrayList
167 
168         // clear data file
169         FileOutputStream newFileID;
170         ObjectOutputStream outFile;
171         try
172         {
173             // Create the output stream
174             newFileID = new FileOutputStream(FILENAME);
175             // create new data file
176             outFile   = new ObjectOutputStream(newFileID);
177 
178             // Close the file
179             outFile.close();
180         }
181         catch (IOException ex) // can't create data file
182         {
183             System.out.println("Error deleting data file: " + ex.getMessage());
184         }
185 
186     }
187 
188     /*
189         An accessor method for the list of players
190         @ return a reference to the ArrayList players
191     */
192     public ArrayList<Player> getPlayers()
193     {
194         return players;
195     }
196 
197     /*
198        Performs a linear search for a player in the database
199        @param name the player's name
200        @return the Player found or null if player not found
201     */
202     public Player searchByName(String name)
203     {
204         // linear search algorithm
205         for(Player player : players)
206         {
207             if(player.getName().equals(name))
```

```
207             if(player.getName().equals(name))
208             {
209                 return player;
210             }
211         }
212 
213         return null;  // player not in list
214     }
215 
216     /*
217        @return an Arraylist that is a copy of the database that is
218                sorted using the selection sort algorithm in
219                alphabetical order by player name
220     */
221     public ArrayList <Player> sortPlayersByName()
222     {
223         // create new list and copy player's data into it
224         ArrayList <Player> list = copyList(players);
225 
226         // selection sort algorithm
227 
228         int i, j;
229         int min;
230         Player temp;
231 
232         for (i = 0; i < list.size()-1; i++)
233         {
234            min = i;
235            for (j = i+1; j < list.size(); j++)
236            {
237               if (list.get(j).getName().compareTo(list.get(min).getName()) < 0)
238                  min = j;
239            }
240            // swap
241            temp = list.get(i);
242            list.set(i, list.get(min));
243            list.set(min, temp);
244          }
245 
246         return list;
247     }
248 
249     /*
250        @return an Arraylist that is a copy of the database that is
251                sorted using the selection sort algorithm in
252                numerical order by player bench max
253     */
254     public ArrayList <Player> sortPlayersByBenchMax()
255     {
256         // create new list and copy player's data into it
257         ArrayList <Player> list = copyList(players);
258 
```

```
259         // selection sort algorithm
260 
261         int i, j;
262         int max;
263         Player temp;
264 
265         for (i = 0; i < list.size()-1; i++) // advance through list one player at a time
266         {
267            max = i;
268            for (j = i+1; j < list.size(); j++)  // find largest player max in list
269            {
270               if (list.get(j).getBenchMax() > list.get(max).getBenchMax())
271                  max = j;
272            }
273            // swap largest max with current max
274            temp = list.get(i);
275            list.set(i, list.get(max));
276            list.set(max, temp);
277          }
278         return list;  // return sorted list
279     }
280 
281     /*
282         return an ArrayList that is a copy of the database
283     */
284     public ArrayList<Player> copyList(ArrayList<Player> list)
285     {
286         ArrayList <Player> temp = new ArrayList<>();
287         for(Player player: list)
288         {
289             temp.add(player);
290         }
291         return temp;
292     }
293 
294     /*
295        @return an ArrayList of Groups organize in groups according
296           by player bench max
297     */
298     public ArrayList<Group> createGroups(int groupSize)
299     {
300         ArrayList <Player> list = sortPlayersByBenchMax();
301         ArrayList <Group> groups = new ArrayList<>();
302 
303         Group group = new Group(groupSize);
304         for(int i = 0; i < list.size(); i++)
305         {
306             group.addPlayer(list.get(i));
307             if(i != 0 && i % groupSize == 0)   // add group every groupSize players
308             {
309                 groups.add(group);
310                 group = new Group(groupSize);
```

```
310                 group = new Group(groupSize);
311                 group.addPlayer(list.get(i));
312             }
313         }
314         // if you still have players left create a group smaller than groupsize
315         if(groups.size() * groupSize < list.size())
316         {
317             int num = list.size() - groups.size() * groupSize;
318             group = new Group(groupSize);
319             for(int i = 0, j = num-1; i < num; i++, j--)
320             {
321                 group.addPlayer(list.get(list.size() - j - 1));
322             }
323             groups.add(group);
324         }
325         return groups;
326     }
327 }
```

```java
package org.bwagner;

import java.util.*;
import java.io.*;

/*
   This class is the program's user interface. It is responsible for interacting
   with the user through a menu system. It contains the program's main method.
*/

public class WeightTraining
{
    //instance variables
    private MaxDatabase max;   // needed to communicate with the database
    private Scanner keyboard;
    private boolean modified = false; // tracks whether database needs to be saved

    // constructor
    public WeightTraining()
    {
        max = new MaxDatabase();
        keyboard = new Scanner(System.in);

        mainMenu();
    }

    /*
       This is the main menu for the program. All interaction with the user
       originates from this menu.
    */
    public void mainMenu()
    {
        int ans = 0;

        do
        {
            System.out.println();
            System.out.println("======================");
            System.out.println("       Main Menu   ");
            System.out.println("======================");
            System.out.println("  1. Add Player");
            System.out.println("  2. Update Player Maxes");
            System.out.println("  3. View List of Player Names");
            System.out.println("  4. View a Player's Maxes");
            System.out.println("  5. Delete Players");
            System.out.println("  6. Print");
            System.out.println("  7. Closeout School Year");
            System.out.println("  8. Save");
            System.out.println("  9. Exit");

            ans = validateIntegerInput("Selection -->");
            System.out.println();
            if(ans == 1)
                addPlayer();
            if(ans == 2)
                updatePlayers();
            if(ans == 3)
                viewAllPlayers();
            if(ans == 4)
                searchForPlayer();
            if(ans == 5)
                delete();
            if(ans == 6)
                print();
            if(ans == 7)
                closeOutYear();
            if(ans == 8)
            {
```

```
                saveDataFile();
                modified = false;
            }
        }
        while(ans != 9);

        if(ans == 9)
        {
            if(modified == true)
            {
                System.out.println();
                System.out.println("Caution: you have made changes to the database.");
                System.out.print("Would like to Save[y/n]?");
                String response = keyboard.next();
                if(response.equalsIgnoreCase("y"))
                {
                    saveDataFile();
                }
            }
        }

        System.out.println();
        System.out.println("Good Bye!");
        System.out.println();
        System.exit(0);       // close terminal window
    }

    /*
       This method allows the user to enter an integer value. It then verifies
       that the input value is an integer. If it is not an integer the method
       prompts the user to re-enter the value again.
       @return the input value
       @param prompt the input prompt
    */
    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;
    }

    /*
       This method validates that the parameter week is between
       1 <= week <= 10. If it is not it requires the user to enter
       a valid number.
       @param the week value(1-10)
    */
    public int validateWeekNum(int week)
    {
        while(week < 1 || week > 10)
```

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

        return week;
    }

    /*
       This method prompts the user to enter a player's info and then adds
       the player to the database.
    */
    public void addPlayer()
    {
        String ans = "";

        do
        {
            System.out.println("====================");
            System.out.println("     Add Player");
            System.out.println("====================");
            System.out.print("Enter Player Name (lastname, firstname)-->");
            String name = keyboard.nextLine();
            int classification = validateIntegerInput("Enter Player Classification (9,10,11,12)-->");
            while(classification < 9 || classification > 12)
            {
                classification = validateIntegerInput("Enter Player Classification (9,10,11,12)-->");
            }
            int bench = validateIntegerInput("Enter Bench Max -->");
            int squat = validateIntegerInput("Enter Squat Max -->");
            int incline = validateIntegerInput("Enter Incline Max -->");
            int power = validateIntegerInput("Enter Power Clean Max -->");

            max.addPlayer(new Player(name, bench, squat, incline, power, classification));  // add player to database
            modified = true;
            System.out.println();
            System.out.print("Add another player[Y/N]?");

            ans = keyboard.nextLine();
        }
        while(ans.equalsIgnoreCase("y"));
    }

    /*
       This method allows a user to modify all players or single player
       max values.
    */

    public void updatePlayers()
    {
        System.out.println("======================");
        System.out.println("   Update Players Max");
        System.out.println("======================");
        System.out.println("1. Update a Player's Max");
        System.out.println("2. Update All Players' Max");
        int ans = validateIntegerInput("Selection -->");

        if(ans == 1)
        {
            String response = "";
            do
            {
                System.out.println();
                System.out.print("Enter Player Name (lastname, firstname)-->");
                String name = keyboard.nextLine();
                Player player = max.searchByName(name);
                if(player == null)
                {
                    System.out.println("Sorry " + name + " is not in database.");
```

```
                    System.out.println("Sorry, " + name + " is not in database.");
                }
                else
                {
                     System.out.println(player);
                     System.out.println();

                     int bench = validateIntegerInput("Enter new Bench Max -->");
                     int squat = validateIntegerInput("Enter new Squat Max -->");
                     int incline = validateIntegerInput("Enter new Incline Max -->");
                     int power = validateIntegerInput("Enter new Power Clean Max -->");

                     player.setBenchMax(bench);
                     player.setSquatMax(squat);
                     player.setInclineMax(incline);
                     player.setPowerMax(power);

                     modified = true;

                     System.out.println();
                     System.out.print("Update Another Player[Y/N]-->");
                     response = keyboard.nextLine();
                }
            }
            while(response.equalsIgnoreCase("y"));
        }
        if(ans == 2)
        {
            updateAllMaxes();
            modified = true;
        }
    }

    /* This method is a helper method for updatePlayers. It allows
       the user to update max values for all players.
    */
    private void updateAllMaxes()
    {
        for(Player player: max.getPlayers())
        {
            System.out.println();
            System.out.println("Current Player's Maxes");
            System.out.println("--------------------");
            System.out.println(player);
            System.out.println();
            int bench = validateIntegerInput("Enter new Bench Max -->");
            int squat = validateIntegerInput("Enter new Squat Max -->");
            int incline = validateIntegerInput("Enter new Incline Max -->");
            int power = validateIntegerInput("Enter new Power Clean Max -->");

            player.setBenchMax(bench);
            player.setSquatMax(squat);
            player.setInclineMax(incline);
            player.setPowerMax(power);
        }
    }

    /* This method allows the user to remove a player from the database or
       clear the database of all players.
    */
    public void delete()
    {
        System.out.println("====================");
        System.out.println("   Delete Player");
        System.out.println("====================");
        System.out.println("  1. Delete a Player");
        System.out.println("  2. Clear Database");
        int ans = validateIntegerInput("Selection -->");
```

```
272
273         if(ans == 1)
274         {
275            System.out.println();
276            System.out.print("Enter Player Name (lastname, firstname)-->");
277            String name = keyboard.nextLine();
278            Player player = max.searchByName(name);
279            if(player == null)
280            {
281                System.out.println("Sorry," + name + " is not in database.");
282            }
283            else
284            {
285                System.out.println("Found the following player:");
286                System.out.println(player);
287                System.out.print("Are you sure you want to delete this player[Y/N]?");
288                String response = keyboard.nextLine();
289                if(response.equalsIgnoreCase("y"))
290                {
291                   max.deletePlayer(player);
292                   modified = true;
293                   System.out.println("Player Deleted!");
294                }
295                else
296                {
297                    System.out.println("Player Not Deleted!");
298                }
299            }
300         }
301         if(ans == 2)
302         {
303             System.out.print("This process will delete all players. Continue[Y/N]?");
304             String response = keyboard.nextLine();
305             if(response.equalsIgnoreCase("y"))
306             {
307                max.clearDatabase();
308                modified = true;
309                System.out.println("Entire Database Deleted");
310             }
311             else
312             {
313                 System.out.println("Database Not Deleted");
314             }
315         }
316     }
317
318     /*
319        This method displays a list in alphabetical of all players in the
320        database. It displays each player's name and classification.
321     */
322     public void viewAllPlayers()
323     {
324         System.out.println("====================");
325         System.out.println("   View All Players");
326         System.out.println("====================");
327
328         ArrayList<Player> list = max.sortPlayersByName();
329
330         for(int i = 0; i < list.size(); i++)
331         {
332             System.out.println((i+1) + "." + list.get(i).getName() + " " + list.get(i).getClassification());
333         }
334          System.out.println();
335     }
336
337     /* This method searches the database by player name. If the player is found
338        it displays the Player's exercise maxes.
339     */
```

```
340     public void searchForPlayer()
341     {
342         System.out.println("====================");
343         System.out.println("  Search For Player");
344         System.out.println("====================");
345         System.out.print("Enter Player Name (lastname, firstname)-->");
346         String name = keyboard.nextLine();
347 
348         Player player = max.searchByName(name);
349         if(player == null)
350         {
351             System.out.println("Sorry, \"" + name + "\" is not in database.");
352         }
353         else
354         {
355             System.out.println(player);
356         }
357     }
358 
359     /* This method allows the user to print two documents.
360        1. A player or players workout program.
361        2. A list of players organized in groups of four by
362           their bench max.
363     */
364     public void print()
365     {
366         System.out.println("====================");
367         System.out.println("       Print");
368         System.out.println("====================");
369         System.out.println("1. Print Weight Lifting Program");
370         System.out.println("2. Print PlayerGroups");
371         int ans = validateIntegerInput("Selection -->");
372 
373         if(ans == 1)
374         {
375             printWeightLiftingProgram();
376         }
377         if(ans == 2)
378         {
379             int size = validateIntegerInput("Enter size of groups -->");
380             PrintGroups print = new PrintGroups(max.createGroups(size));
381          /* ArrayList<Group> groups = max.createGroups(size);
382             for(Group group : groups)
383             {
384                 for(Player player : group.getGroup())
385                 {
386                     if(player != null)
387                        System.out.println(player.getName()+ " "+player.getBenchMax());
388                 }
389                 System.out.println();
390             }
391          */
392         }
393     }
394 
395     /*
396        This method is a helper method for print. It prints weight lifting
397        workout programs.
398     */
399     private void printWeightLiftingProgram()
400     {
401         System.out.println("============================");
402         System.out.println(" Print Weight Lifting Program");
403         System.out.println("============================");
404         System.out.println("1. Print a Player");
405         System.out.println("2. Print All Players");
406         int ans = validateIntegerInput("Selection -->");
407 
```

```
        if(ans == 1)
        {
            String response = "";
            do
            {
                System.out.print("Enter Player Name (lastname, firstname)-->");
                String name = keyboard.nextLine();
                int week = validateIntegerInput("Enter Program Week (1-10) -->");
                week = validateWeekNum(week);
                Player player = max.searchByName(name);
                if(player != null)
                {
                    PrintWeightProgram print = new PrintWeightProgram(player, week);
                }
                else
                {
                    System.out.println("Player not Found");
                }
                System.out.println();
                System.out.print("Print another player[Y/N]?");
                response = keyboard.nextLine();
            }
            while(response.equalsIgnoreCase("y"));
        }
        if(ans == 2)
        {
            int week = validateIntegerInput("Enter Program Week (1-10) -->");
            validateWeekNum(week);
            PrintWeightProgram print = new PrintWeightProgram(max.getPlayers(), week);
        }
    }

    /*
        This method updates the database by deleting all seniors and promoting all
        underclassmen to the next grade level.
    */
    public void closeOutYear()
    {
        System.out.println("======================");
        System.out.println(" Closeout School Year");
        System.out.println("======================");
        System.out.println("Caution: This feature will remove all seniors from the database");
        System.out.println("and promote underclassmen to the next grade level.");
        System.out.println();
        System.out.print("Are you sure you would like to continue[y/n]?");
        String ans = keyboard.next();
        if(ans.equalsIgnoreCase("y"))
        {
            ArrayList <Player> list = max.getPlayers();
            int i = 0;
            while(i < list.size())
            {
                Player player = list.get(i);
                if(player.getClassification() == 12)
                {
                    list.remove(i);
                }
                else
                {
                    i++;
                    player.setClassification(player.getClassification() + 1);
                }
            }
            modified = true;
            System.out.println();
            System.out.println("Closeout Complete!");
            System.out.println();
        }
```

```
        }
    }

    /*
       This method saves the database to the data file.
    */
    public void saveDataFile()
    {
        System.out.println("====================");
        System.out.println("   Save Data File");
        System.out.println("====================");

        max.saveFile();
    }

    /*
        This is the program's main menu.
    */
    public static void main(String[] args)
    {
        WeightTraining app = new WeightTraining();
    }
}
```

```java
package org.bwagner;

import java.awt.*;
import javax.swing.*;
import java.awt.print.*;
import java.util.*;

/*
    This class provides methods that communicate with a printer. It can print a
    weekly workout program for every player.
*/

public class PrintWeightProgram extends JFrame implements Printable
{
    final static int RECORD_SIZE = 9;  // data plus 2 blank lines after record

    private ArrayList <Player> players;   // stores list of Players
    private String[] textLines;  // stores each line to be printed per record
    private int week;            //week to be printed
    private Player player;
    private int[] pageBreaks;   // array of page break line positions
    private int[] weekReps;     // stores number of reps for week

    int[][] repNum = {{10,8,8}, //how many times they are doing that weight for that day
                      {8,6,4},
                      {4,4,2},
                      {2,2,1},
                     {10,10,10},
                     {10,8,8},
                      {8,6,4},
                      {4,4,2},
                      {2,2,1},
                      {1,1,1}};

    /*
        @param list ArrayList of players
        @param w the workout program week
    */
    public PrintWeightProgram(ArrayList<Player> list, int w) // prints all players
    {
        players =  list;
week = w;
        setVisible(false);       //don't show gui window

weekReps = new int[3];

        for(int c = 0; c < 3; c++)
        {
            weekReps[c] = repNum[week-1][c];
        }

        PrinterJob job = PrinterJob.getPrinterJob();

        PageFormat pf = job.defaultPage();
        Paper paper = new Paper();
        double margin = 72/2;        // half inch margin
        paper.setImageableArea(margin, margin, paper.getWidth() - margin * 2, paper.getHeight()
            - margin * 2);
        pf.setPaper(paper);

        job.setPrintable(this, pf);

         boolean ok = job.printDialog();
         if (ok)
         {
             try
             {
                 job.print();
             }
             catch (PrinterException ex)
             {
                // The job did not successfully complete
                System.out.println("Printing Error");
             }
         }
    }

    /*
       @param p a Player to be printed
       @param w workout program week
    */
    public PrintWeightProgram(Player p, int w) //prints one student
    {
    player = p;
    week = w;
        setVisible(false);       //don't show gui window

        PrinterJob job = PrinterJob.getPrinterJob();

        PageFormat pf = job.defaultPage();
        Paper paper = new Paper();
        double margin = 72/2;        // half inch margin
        paper.setImageableArea(margin, margin, paper.getWidth() - margin * 2, paper.getHeight()
            - margin * 2);
        pf.setPaper(paper);

        job.setPrintable(this, pf);
        boolean ok = job.printDialog();

        weekReps = new int[3];

        for(int c = 0; c < 3; c++)
        {
            weekReps[c] = repNum[week-1][c];
        }


         if (ok)
         {
             try
             {
                 job.print();
             }
             catch (PrinterException ex)
             {
                 // The job did not successfully complete
             }
         }
    }

    /*
        Prepares the data for printing by filling the textLines array with all
        the data from the array of players into string format.
    */
    private void initTextLines()
    {
        if (textLines == null)
        {
            if(players == null)
            {
                int numLines= RECORD_SIZE;
                textLines = new String[numLines];

                textLines[0] = String.format("%-16s%s%n","Name:", player.getName());
                textLines[1] = String.format("%-16s%s%n","Program Week:", week);
                textLines[2] = String.format("%-16s%s%d%s%d%s%d%s%d%n","Current Maxes:", "Bench - ", player.getBenchMax(),"  Squat - ", player.getSquatMax(),"  Incline - ", player.getInclineMax(),"  Power Clean - ", player.getPowerMax());
                textLines[3] = "-------------------------------------------------------------------------------------------";
                textLines[4] = String.format("%-8s%d%s%d%s%d%n","Reps: ",weekReps[0],", ",weekReps[1],", ",weekReps[2]);
                textLines[5] = String.format("%-8s%d%s%-16s%d%n","Squat: ", WeightLiftingProgram.calculateSquat(player.getSquatMax(), week),"          ","Power Clean:", WeightLiftingProgram.calculatePowerClean(player.getPowerMax(), week));
                textLines[6] = String.format("%-8s%d%s%-16s%d%n","Bench: ", WeightLiftingProgram.calculateBench(player.getBenchMax(), week),"          ","Incline: " , WeightLiftingProgram.calculateIncline(player.getInclineMax(), week));
                textLines[7] = "\n";
```

```
                textLines[8] = "\n";
            }
            else
            {
                int numLines=players.size() * RECORD_SIZE;
                textLines = new String[numLines];
                for (int i=0;i<numLines;i+=RECORD_SIZE)
                {
                    Player aPlayer = (players.get(i/RECORD_SIZE));
                    textLines[i]   = String.format("%-16s%s%n","Name:", aPlayer.getName());
                    textLines[i+1] = String.format("%-16s%s%n","Program Week:", week);
                    textLines[i+2] = String.format("%-16s%s%d%s%d%s%d%s%d%n","Current Maxes:", "Bench - ", aPlayer.getBenchMax()," Squat - ", aPlayer.getSquatMax()," Incline - ", aPlayer.getInclineMax()," Power Clean - ", aPlayer.getPowerMax());
                    textLines[i+3] = "----------------------------------------------------------------------------------------------------";
                    textLines[i+4] = String.format("%-8s%d%s%d%s%d%n","Reps: ",weekReps[0],", ",weekReps[1],", ",weekReps[2]);
                    textLines[i+5] = String.format("%-8s%d%s%-16s%d%n","Squat: ", WeightLiftingProgram.calculateSquat(aPlayer.getSquatMax(), week),"          ","Power Clean:", WeightLiftingProgram.calculatePowerClean(aPlayer.getPowerMax(), week));
                    textLines[i+6] = String.format("%-8s%d%s%-16s%d%n","Bench: ", WeightLiftingProgram.calculateBench(aPlayer.getBenchMax(), week),"          ","Incline: " , WeightLiftingProgram.calculateIncline(aPlayer.getInclineMax(), week));
                    textLines[i+7] = "\n";
                    textLines[i+8] = "\n";
                }
            }
        }
    }

    @Override
    public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
    {
        Font font = new Font("Consolas", Font.PLAIN, 12);
        g.setFont(font);

        FontMetrics metrics = g.getFontMetrics(font);
        int lineHeight = metrics.getHeight();

        if (pageBreaks == null)
        {
            initTextLines();
            int linesPerPage = (int)(pf.getImageableHeight()/lineHeight);
            int recordsPerPage = linesPerPage/RECORD_SIZE;
            int numRecords = textLines.length/RECORD_SIZE;
            int numBreaks = (numRecords * RECORD_SIZE) / linesPerPage;
            pageBreaks = new int[numBreaks];
            for (int b=0; b<numBreaks; b++)
            {
                pageBreaks[b] = (b+1)* recordsPerPage * RECORD_SIZE;
            }
        }

        if (pageIndex > pageBreaks.length)
        {
            return NO_SUCH_PAGE;
        }

        /* User (0,0) is typically outside the imageable area, so we must
         * translate by the X and Y values in the PageFormat to avoid clipping
         * Since we are drawing text we
         */
        Graphics2D g2d = (Graphics2D)g;
        g2d.translate(pf.getImageableX(), pf.getImageableY());

        /* Draw each line that is on this page.
         * Increment 'y' position by lineHeight for each line.
         */
        int y = 0;
        int start = (pageIndex == 0) ? 0 : pageBreaks[pageIndex-1];
        int end   = (pageIndex == pageBreaks.length)
                          ? textLines.length : pageBreaks[pageIndex];
        for (int line=start; line<end; line++)
        {
            y += lineHeight;
            g.drawString(textLines[line], 0, y);
        }

        /* tell the caller that this page is part of the printed document */
        return PAGE_EXISTS;
    }

}
```

```
 1 package org.bwagner;
 2
 3 import java.util.*;
 4
 5 /*
 6    This class provides static methods for calculating
weekly workout weights for
 7    each of the four main exercises: bench, squat,
incline, and power clean.
 8    A workout weight is calculated by multipling the max
for that exercise
 9    by the weight percentage determined by the workout
week(1 - 10). Each week
10    the weight percentage increases by 5%. Weights are
rounded to multiples of 5.
11 */
12
13 public class WeightLiftingProgram
14 {
15     public static double[] formulas = {0.60, 0.65, 0.70,
0.75, 0.80,
16                                        0.80, 0.85, 0.90,
0.95, 1.0};
17
18
19     public static int calculateBench(int b, int w)
20     {
21         double percent = formulas[w - 1];
22         double repWeight = b * percent;
23
24         return 5*((int)Math.round(repWeight/5));
//rounds to nearest multiple of 5
25     }
26
27     public static int calculateSquat(int s, int w)
28     {
29         double percent = formulas[w - 1];
```

```
30          double repWeight = s * percent;
31
32          return 5*((int)Math.round(repWeight/5));
33      }
34
35      public static int calculateIncline(int i, int w)
36      {
37          double percent = formulas[w - 1];
38          double repWeight = i * percent;
39
40          return 5*((int)Math.round(repWeight/5));
41      }
42
43      public static int calculatePowerClean(int pc, int w)
44      {
45          double percent = formulas[w - 1];
46          double repWeight = pc * percent;
47
48          return 5*((int)Math.round(repWeight/5));
49      }
50
51  }
```

```java
package org.bwagner;

import java.util.*;

/*
    This class stores up to four Players to form a workout group.
*/

public class Group
{
    private Player[] group;
    private int maxGroupSize;
    private int numPlayers;

    //constructor
    public Group(int groupSize)
    {
        maxGroupSize = groupSize;
        group = new Player[maxGroupSize];
        numPlayers = 0;
    }

    /*
        Adds a player to the group
    */
    public void addPlayer(Player player)
    {
        if(numPlayers < maxGroupSize)
        {
            group[numPlayers] = player;
            numPlayers++;
        }
    }

    /*
        @return the number of players in this group
    */
    public int getGroupSize()
```

```
39      {
40          return numPlayers;
41      }
42
43      public Player[] getGroup()
44      {
45          return group;
46      }
47
48  }
```

```
1 package org.bwagner;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.print.*;
6 import static java.awt.print.Printable.NO_SUCH_PAGE;
7 import static java.awt.print.Printable.PAGE_EXISTS;
8 import java.util.*;
9
10 /*
11    This class is responsible for interacting with the printer and printing a
12    list of Groups.
13 */
14
15 public class PrintGroups extends JFrame implements Printable
16 {
17     final static int RECORD_SIZE = 8;  // data plus 2 blank lines after record
18                                        // and 2 lines for header
19
20     private ArrayList <Group> groups;  // stores workout groups to be printed
21     private String[] textLines;        // stores each line of data to be printed per record
22     private int[] pageBreaks;          // array of page break line positions
23
24     /*
25        @param list an ArrayLis of Groups
26     */
27     public PrintGroups(ArrayList<Group> list) // prints all groups
28     {
29         setVisible(false);       //don't show gui window
30
31         groups = list;
32
33         PrinterJob job = PrinterJob.getPrinterJob();
34
35         PageFormat pf = job.defaultPage();
36         Paper paper = new Paper();
37         double margin = 72/2;        // half inch margin
38         paper.setImageableArea(margin, margin, paper.getWidth() - margin * 2, paper.getHeight()
39             - margin * 2);
40         pf.setPaper(paper);
41
42         job.setPrintable(this, pf);
43         boolean ok = job.printDialog();
44         if (ok)
45         {
46             try
47             {
48                 job.print();
49             }
50             catch (PrinterException ex)
51             {
52                // The job did not successfully complete
53             }
54          }
55     }
56     /*
57         Prepares the data for printing by filling the textLines array with all
58         the data from the array of groups into string format.
59     */
60     private void initTextLines()
61     {
62         if (textLines == null)
63         {
64
65             int numLines=groups.size() * RECORD_SIZE;
66             textLines = new String[numLines];
67             int groupCount = 1;
68             for (int i=0;i < numLines; i+=RECORD_SIZE)
69             {
70                 Group group = (groups.get(i/RECORD_SIZE));
71                 //Group group = groups.get(groupCount-1);
72                 Player[] players = group.getGroup();
73
74                 textLines[i]   = String.format("%s%d%n","Group", groupCount);
75                 textLines[i+1] = String.format("%s%n","----------");
76                 int j = 2;
77                 for(; j <= group.getGroupSize()+1; j++)
78                 {
79                     if(players[j-2] != null)
```

```
                        textLines[i+j] = String.format("%-22s%s%d%n", players[j-2].getName(), "Bench Max = ", players[j-2].getBenchMax());
                }
                textLines[i+j] = "\n";
                textLines[i+j+1] = "\n";

                if(i % RECORD_SIZE == 0 || group.getGroupSize() < 4)
                {
                    groupCount++;
                }
            }

        }
     }

     @Override
    public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
    {
         Font font = new Font("Consolas", Font.PLAIN, 14);
         g.setFont(font);
         FontMetrics metrics = g.getFontMetrics(font);
         int lineHeight = metrics.getHeight();

         if (pageBreaks == null)
         {
             initTextLines();
             int linesPerPage = (int)(pf.getImageableHeight()/lineHeight);
             int recordsPerPage = linesPerPage/RECORD_SIZE;
             int numRecords = textLines.length/RECORD_SIZE;
             int numBreaks = (numRecords * RECORD_SIZE) / linesPerPage;
             pageBreaks = new int[numBreaks];
             for (int b=0; b<numBreaks; b++)
             {
                 pageBreaks[b] = (b+1)* recordsPerPage * RECORD_SIZE;
             }
         }

         if (pageIndex > pageBreaks.length)
         {
             return NO_SUCH_PAGE;
         }

         /* User (0,0) is typically outside the imageable area, so we must
          * translate by the X and Y values in the PageFormat to avoid clipping
          * Since we are drawing text we
          */

         Graphics2D g2d = (Graphics2D)g;
         g2d.translate(pf.getImageableX(), pf.getImageableY());

         /* Draw each line that is on this page.
          * Increment 'y' position by lineHeight for each line.
          */

         int y = 0;
         int start = (pageIndex == 0) ? 0 : pageBreaks[pageIndex-1];
         int end   = (pageIndex == pageBreaks.length)
                         ? textLines.length : pageBreaks[pageIndex];
         for (int line=start; line<end; line++)
         {
             y += lineHeight;
             if(textLines[line] != null)
                g.drawString(textLines[line], 0, y);
         }



         /* tell the caller that this page is part of the printed document */
         return PAGE_EXISTS;
     }

}
```