

```

1 package org.bwagner;
2
3 import java.io.Serializable; // needed to save to data file as an object
4
5 /*
6  This class stores each player's name, classification(9, 10, 11, 12),
7  and weight max for each of the four exercises: bench, squat, incline,
8  and power clean.
9  */
10 public class Player implements Serializable {
11
12     // instance variables
13     private String name;
14     private int benchMax;
15     private int squatMax;
16     private int inclineMax;
17     private int powerMax;
18     private int classification;
19
20     // constructors
21     public Player() {
22         name = "";
23         benchMax = 0;
24         squatMax = 0;
25         inclineMax = 0;
26         powerMax = 0;
27         classification = 9;
28     }
29
30     public Player(String n, int b, int s, int i, int p, int c) {
31         name = n;
32         benchMax = b;
33         squatMax = s;
34         inclineMax = i;
35         powerMax = p;
36         classification = c;
37     }
38
39     // accessor methods
40     public String getName() {
41         return name;
42     }
43
44     public int getBenchMax() {
45         return benchMax;
46     }
47
48     public int getSquatMax() {
49         return squatMax;
50     }
51
52     public int getInclineMax() {
53         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        */
113     public void makeBackupFile()
114     {
115         FileOutputStream fileID;
116         ObjectOutputStream outFile;
117
118         try
119         {
120             // Create the output stream
121             fileID = new FileOutputStream(BACKUP);
122             outFile = new ObjectOutputStream(fileID);
123
124             // Write the ArrayList to the file
125             outFile.writeObject(players);
126
127             // Close the file
128             outFile.close();
129         }
130         catch (IOException e)
131         {
132             System.out.println("Error writing to backup file: " + e.getMessage());
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 }

```

```

1 package org.bwagner;
2
3 import java.util.*;
4 import java.io.*;
5
6 /*
7  This class is the program's user interface. It is responsible for interacting
8  with the user through a menu system. It contains the program's main method.
9  */
10
11 public class WeightTraining
12 {
13     //instance variables
14     private MaxDatabase max;    // needed to communicate with the database
15     private Scanner keyboard;
16     private boolean modified = false; // tracks whether database needs to be saved
17
18     // constructor
19     public WeightTraining()
20     {
21         max = new MaxDatabase();
22         keyboard = new Scanner(System.in);
23
24         mainMenu();
25     }
26
27     /*
28      This is the main menu for the program. All interaction with the user
29      originates from this menu.
30      */
31     public void mainMenu()
32     {
33         int ans = 0;
34
35         do
36         {
37             System.out.println();
38             System.out.println("=====");
39             System.out.println("      Main Menu      ");
40             System.out.println("=====");
41             System.out.println(" 1. Add Player");
42             System.out.println(" 2. Update Player Maxes");
43             System.out.println(" 3. View List of Player Names");
44             System.out.println(" 4. View a Player's Maxes");
45             System.out.println(" 5. Delete Players");
46             System.out.println(" 6. Print");
47             System.out.println(" 7. Closeout School Year");
48             System.out.println(" 8. Save");
49             System.out.println(" 9. Exit");
50
51             ans = validateIntegerInput("Selection -->");
52             System.out.println();
53             if(ans == 1)
54                 addPlayer();
55             if(ans == 2)
56                 updatePlayers();
57             if(ans == 3)
58                 viewAllPlayers();
59             if(ans == 4)
60                 searchForPlayer();
61             if(ans == 5)
62                 delete();
63             if(ans == 6)
64                 print();
65             if(ans == 7)
66                 closeOutYear();
67             if(ans == 8)
68                 {

```

```

69         saveDataFile();
70         modified = false;
71     }
72 }
73 while(ans != 9);
74
75 if(ans == 9)
76 {
77     if(modified == true)
78     {
79         System.out.println();
80         System.out.println("Caution: you have made changes to the database.");
81         System.out.print("Would like to Save[y/n]?");
82         String response = keyboard.next();
83         if(response.equalsIgnoreCase("y"))
84         {
85             saveDataFile();
86         }
87     }
88 }
89
90 System.out.println();
91 System.out.println("Good Bye!");
92 System.out.println();
93 System.exit(0);    // close terminal window
94 }
95
96 /*
97 This method allows the user to enter an integer value. It then verifies
98 that the input value is an integer. If it is not an integer the method
99 prompts the user to re-enter the value again.
100 @return the input value
101 @param prompt the input prompt
102 */
103 public int validateIntegerInput(String prompt)
104 {
105     int ans = 0;
106     boolean flag;
107
108     do
109     {
110         flag = true;
111
112         System.out.print(prompt); // display input prompt
113         if(keyboard.hasNextInt()) // if input is an integer
114         {
115             ans = keyboard.nextInt();
116         }
117         else // not an integer
118         {
119             System.out.println("Invalid Entry. Try again.");
120             flag = false;
121         }
122         keyboard.nextLine();    // clear buffer
123     } while(flag == false);
124
125     return ans;
126 }
127
128 /*
129 This method validates that the parameter week is between
130 1 <= week <= 10. If it is not it requires the user to enter
131 a valid number.
132 @param the week value(1-10)
133 */
134 public int validateWeekNum(int week)
135 {
136     while(week < 1 || week > 10)

```

```

137     {
138         week = validateIntegerInput("Enter Program Week (1-10) -->");
139     }
140
141     return week;
142 }
143
144 /*
145  This method prompts the user to enter a player's info and then adds
146  the player to the database.
147  */
148 public void addPlayer()
149 {
150     String ans = "";
151
152     do
153     {
154         System.out.println("=====");
155         System.out.println("    Add Player");
156         System.out.println("=====");
157         System.out.print("Enter Player Name (lastname, firstname)-->");
158         String name = keyboard.nextLine();
159         int classification = validateIntegerInput("Enter Player Classification (9,10,11,12)-->");
160         while(classification < 9 || classification > 12)
161         {
162             classification = validateIntegerInput("Enter Player Classification (9,10,11,12)-->");
163         }
164         int bench = validateIntegerInput("Enter Bench Max -->");
165         int squat = validateIntegerInput("Enter Squat Max -->");
166
167         int incline = validateIntegerInput("Enter Incline Max -->");
168         int power = validateIntegerInput("Enter Power Clean Max -->");
169
170         max.addPlayer(new Player(name, bench, squat, incline, power, classification)); // add player to database
171         modified = true;
172         System.out.println();
173         System.out.print("Add another player[Y/N]?");
174
175         ans = keyboard.nextLine();
176     } while(ans.equalsIgnoreCase("y"));
177 }
178
179 /*
180  This method allows a user to modify all players or single player
181  max values.
182  */
183
184 public void updatePlayers()
185 {
186     System.out.println("=====");
187     System.out.println("    Update Players Max");
188     System.out.println("=====");
189     System.out.println("1. Update a Player's Max");
190     System.out.println("2. Update All Players' Max");
191     int ans = validateIntegerInput("Selection -->");
192
193     if(ans == 1)
194     {
195         String response = "";
196         do
197         {
198             System.out.println();
199             System.out.print("Enter Player Name (lastname, firstname)-->");
200             String name = keyboard.nextLine();
201             Player player = max.searchByName(name);
202             if(player == null)
203             {
204                 System.out.println("Sorry " + name + " is not in database.");

```

```

204         System.out.println("Sorry, " + name + " is not in database. ");
205     }
206     else
207     {
208         System.out.println(player);
209         System.out.println();
210
211         int bench = validateIntegerInput("Enter new Bench Max -->");
212         int squat = validateIntegerInput("Enter new Squat Max -->");
213         int incline = validateIntegerInput("Enter new Incline Max -->");
214         int power = validateIntegerInput("Enter new Power Clean Max -->");
215
216         player.setBenchMax(bench);
217         player.setSquatMax(squat);
218         player.setInclineMax(incline);
219         player.setPowerMax(power);
220
221         modified = true;
222
223         System.out.println();
224         System.out.print("Update Another Player[Y/N]-->");
225         response = keyboard.nextLine();
226     }
227 }
228 while(response.equalsIgnoreCase("y"));
229 }
230 if(ans == 2)
231 {
232     updateAllMaxes();
233     modified = true;
234 }
235 }
236
237 /* This method is a helper method for updatePlayers. It allows
238    the user to update max values for all players.
239 */
240 private void updateAllMaxes()
241 {
242     for(Player player: max.getPlayers())
243     {
244         System.out.println();
245         System.out.println("Current Player's Maxes");
246         System.out.println("-----");
247         System.out.println(player);
248         System.out.println();
249         int bench = validateIntegerInput("Enter new Bench Max -->");
250         int squat = validateIntegerInput("Enter new Squat Max -->");
251         int incline = validateIntegerInput("Enter new Incline Max -->");
252         int power = validateIntegerInput("Enter new Power Clean Max -->");
253
254         player.setBenchMax(bench);
255         player.setSquatMax(squat);
256         player.setInclineMax(incline);
257         player.setPowerMax(power);
258     }
259 }
260
261 /* This method allows the user to remove a player from the database or
262    clear the database of all players.
263 */
264 public void delete()
265 {
266     System.out.println("=====");
267     System.out.println("    Delete Player");
268     System.out.println("=====");
269     System.out.println("  1. Delete a Player");
270     System.out.println("  2. Clear Database");
271     int ans = validateIntegerInput("Selection -->");
272 }

```

```

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

```

```

408     if(ans == 1)
409     {
410         String response = "";
411         do
412         {
413             System.out.print("Enter Player Name (lastname, firstname)-->");
414             String name = keyboard.nextLine();
415             int week = validateIntegerInput("Enter Program Week (1-10) -->");
416             week = validateWeekNum(week);
417             Player player = max.searchByName(name);
418             if(player != null)
419             {
420                 PrintWeightProgram print = new PrintWeightProgram(player, week);
421             }
422             else
423             {
424                 System.out.println("Player not Found");
425             }
426             System.out.println();
427             System.out.print("Print another player[Y/N]?");
428             response = keyboard.nextLine();
429         }
430         while(response.equalsIgnoreCase("y"));
431     }
432     if(ans == 2)
433     {
434         int week = validateIntegerInput("Enter Program Week (1-10) -->");
435         validateWeekNum(week);
436         PrintWeightProgram print = new PrintWeightProgram(max.getPlayers(), week);
437     }
438 }
439
440 /*
441     This method updates the database by deleting all seniors and promoting all
442     underclassmen to the next grade level.
443 */
444 public void closeOutYear()
445 {
446     System.out.println("=====");
447     System.out.println(" Closeout School Year");
448     System.out.println("=====");
449     System.out.println("Caution: This feature will remove all seniors from the database");
450     System.out.println("and promote underclassmen to the next grade level.");
451     System.out.println();
452     System.out.print("Are you sure you would like to continue[y/n]?");
453     String ans = keyboard.next();
454     if(ans.equalsIgnoreCase("y"))
455     {
456         ArrayList <Player> list = max.getPlayers();
457         int i = 0;
458         while(i < list.size())
459         {
460             Player player = list.get(i);
461             if(player.getClassification() == 12)
462             {
463                 list.remove(i);
464             }
465             else
466             {
467                 i++;
468                 player.setClassification(player.getClassification() + 1);
469             }
470         }
471         modified = true;
472         System.out.println();
473         System.out.println("Closeout Complete!");
474         System.out.println();
475     }

```



```
475     }
476 }
477
478 /*
479     This method saves the database to the data file.
480 */
481 public void saveDataFile()
482 {
483     System.out.println("=====");
484     System.out.println("    Save Data File");
485     System.out.println("=====");
486
487     max.saveFile();
488 }
489
490 /*
491     This is the program's main menu.
492 */
493 public static void main(String[] args)
494 {
495     WeightTraining app = new WeightTraining();
496 }
497 }
```

```

1 package org.bwagner;
2
3 import java.awt.*;
4 import javax.swing.*;
5 import java.awt.print.*;
6 import java.util.*;
7
8 /*
9  This class provides methods that communicate with a printer. It can print a
10 weekly workout program for every player.
11 */
12
13 public class PrintWeightProgram extends JFrame implements Printable
14 {
15     final static int RECORD_SIZE = 9; // data plus 2 blank lines after record
16
17     private ArrayList<Player> players; // stores list of Players
18     private String[] textLines; // stores each line to be printed per record
19     private int week; // week to be printed
20     private Player player;
21     private int[] pageBreaks; // array of page break line positions
22     private int[] weekReps; // stores number of reps for week
23
24     int[][] repNum = {{10,8,8}, //how many times they are doing that weight for that day
25                       {8,6,4},
26                       {4,4,2},
27                       {2,2,1},
28                       {10,10,10},
29                       {10,8,8},
30                       {8,6,4},
31                       {4,4,2},
32                       {2,2,1},
33                       {1,1,1}};
34
35     /*
36     @param list ArrayList of players
37     @param w the workout program week
38     */
39     public PrintWeightProgram(ArrayList<Player> list, int w) // prints all players
40     {
41         players = list;
42         week = w;
43         setVisible(false); //don't show gui window
44
45         weekReps = new int[3];
46
47         for(int c = 0; c < 3; c++)
48         {
49             weekReps[c] = repNum[week-1][c];
50         }
51
52         PrinterJob job = PrinterJob.getPrinterJob();
53
54         PageFormat pf = job.defaultPage();
55         Paper paper = new Paper();
56         double margin = 72/2; // half inch margin
57         paper.setImageableArea(margin, margin, paper.getWidth() - margin * 2, paper.getHeight()
58                               - margin * 2);
59         pf.setPaper(paper);
60
61         job.setPrintable(this, pf);
62
63         boolean ok = job.printDialog();
64         if (ok)
65         {
66             try
67             {
68                 job.print();
69             }
70             catch (PrinterException ex)
71             {
72                 // The job did not successfully complete
73                 System.out.println("Printing Error");
74             }
75         }
76     }
77
78     /*
79     @param p a Player to be printed
80     @param w workout program week
81     */
82     public PrintWeightProgram(Player p, int w) //prints one student
83     {
84         player = p;
85         week = w;
86         setVisible(false); //don't show gui window
87
88         PrinterJob job = PrinterJob.getPrinterJob();
89
90         PageFormat pf = job.defaultPage();
91         Paper paper = new Paper();
92         double margin = 72/2; // half inch margin
93         paper.setImageableArea(margin, margin, paper.getWidth() - margin * 2, paper.getHeight()
94                               - margin * 2);
95         pf.setPaper(paper);
96
97         job.setPrintable(this, pf);
98         boolean ok = job.printDialog();
99
100         weekReps = new int[3];
101
102         for(int c = 0; c < 3; c++)
103         {
104             weekReps[c] = repNum[week-1][c];
105         }
106
107         if (ok)
108         {
109             try
110             {
111                 {
112                     job.print();
113                 }
114                 catch (PrinterException ex)
115                 {
116                     // The job did not successfully complete
117                 }
118             }
119         }
120     }
121
122     /*
123     Prepares the data for printing by filling the textLines array with all
124     the data from the array of players into string format.
125     */
126     private void initTextLines()
127     {
128         if (textLines == null)
129         {
130             if (players == null)
131             {
132                 int numLines = RECORD_SIZE;
133                 textLines = new String[numLines];
134
135                 textLines[0] = String.format("%-16s\n", "Name:", player.getName());
136                 textLines[1] = String.format("%-16s\n", "Program Week:", week);
137                 textLines[2] = String.format("%-16s\n", "Current Maxes:", "Bench - ", player.getBenchMax(), " Squat - ", player.getSquatMax(), " Incline - ", player.getInclineMax(), " Power Clean - ", player.getPowerClean());
138                 textLines[3] = "-----";
139                 textLines[4] = String.format("%-8s\n", "Reps: ", weekReps[0], " ", weekReps[1], " ", weekReps[2]);
140                 textLines[5] = String.format("%-8s\n", "Squat: ", WeightLiftingProgram.calculateSquat(player.getSquatMax(), week), " ", "Power Clean:", WeightLiftingProgram.calculatePowerClean(player.getPowerClean(), week));
141                 textLines[6] = String.format("%-8s\n", "Bench: ", WeightLiftingProgram.calculateBench(player.getBenchMax(), week), " ", "Incline: ", WeightLiftingProgram.calculateIncline(player.getInclineMax(), week));
142                 textLines[7] = "\n";
143             }
144         }
145     }
146
147     public int getRecordSize()
148     {
149         return RECORD_SIZE;
150     }
151
152     public String[] getTextLines()
153     {
154         return textLines;
155     }
156
157     public void setRecordSize(int recordSize)
158     {
159         RECORD_SIZE = recordSize;
160     }
161
162     public void setTextLines(String[] textLines)
163     {
164         this.textLines = textLines;
165     }
166
167     public void setPlayers(ArrayList<Player> players)
168     {
169         this.players = players;
170     }
171
172     public void setWeek(int week)
173     {
174         this.week = week;
175     }
176
177     public void setPlayer(Player player)
178     {
179         this.player = player;
180     }
181
182     public void setPageBreaks(int[] pageBreaks)
183     {
184         this.pageBreaks = pageBreaks;
185     }
186
187     public void setWeekReps(int[] weekReps)
188     {
189         this.weekReps = weekReps;
190     }
191
192     public void setVisible(boolean visible)
193     {
194         this.setVisible(visible);
195     }
196
197     public void setPrinterJob(PrinterJob job)
198     {
199         this.job = job;
200     }
201
202     public void setPageFormat(PageFormat pf)
203     {
204         this.pageFormat = pf;
205     }
206
207     public void setPaper(Paper paper)
208     {
209         this.paper = paper;
210     }
211
212     public void setMargin(double margin)
213     {
214         this.margin = margin;
215     }
216
217     public void setPrinterException(PrinterException ex)
218     {
219         this.printerException = ex;
220     }
221
222     public void setSystemOutPrintln(String str)
223     {
224         System.out.println(str);
225     }
226
227     public void setJobPrintable(Printable printable)
228     {
229         this.job.setPrintable(printable, this.pageFormat);
230     }
231
232     public void setJobPrintDialog(boolean ok)
233     {
234         this.job.printDialog();
235     }
236
237     public void setJobPrint(PrinterJob job)
238     {
239         job.print();
240     }
241
242     public void setJobPrintException(PrinterException ex)
243     {
244         this.job.printException(ex);
245     }
246
247     public void setJobPrintSystemOutPrintln(String str)
248     {
249         System.out.println(str);
250     }
251
252     public void setJobPrintable(Printable printable)
253     {
254         this.job.setPrintable(printable, this.pageFormat);
255     }
256
257     public void setJobPrintDialog(boolean ok)
258     {
259         this.job.printDialog();
260     }
261
262     public void setJobPrint(PrinterJob job)
263     {
264         job.print();
265     }
266
267     public void setJobPrintException(PrinterException ex)
268     {
269         this.job.printException(ex);
270     }
271
272     public void setJobPrintSystemOutPrintln(String str)
273     {
274         System.out.println(str);
275     }
276
277     public void setJobPrintable(Printable printable)
278     {
279         this.job.setPrintable(printable, this.pageFormat);
280     }
281
282     public void setJobPrintDialog(boolean ok)
283     {
284         this.job.printDialog();
285     }
286
287     public void setJobPrint(PrinterJob job)
288     {
289         job.print();
290     }
291
292     public void setJobPrintException(PrinterException ex)
293     {
294         this.job.printException(ex);
295     }
296
297     public void setJobPrintSystemOutPrintln(String str)
298     {
299         System.out.println(str);
300     }
301
302     public void setJobPrintable(Printable printable)
303     {
304         this.job.setPrintable(printable, this.pageFormat);
305     }
306
307     public void setJobPrintDialog(boolean ok)
308     {
309         this.job.printDialog();
310     }
311
312     public void setJobPrint(PrinterJob job)
313     {
314         job.print();
315     }
316
317     public void setJobPrintException(PrinterException ex)
318     {
319         this.job.printException(ex);
320     }
321
322     public void setJobPrintSystemOutPrintln(String str)
323     {
324         System.out.println(str);
325     }
326
327     public void setJobPrintable(Printable printable)
328     {
329         this.job.setPrintable(printable, this.pageFormat);
330     }
331
332     public void setJobPrintDialog(boolean ok)
333     {
334         this.job.printDialog();
335     }
336
337     public void setJobPrint(PrinterJob job)
338     {
339         job.print();
340     }
341
342     public void setJobPrintException(PrinterException ex)
343     {
344         this.job.printException(ex);
345     }
346
347     public void setJobPrintSystemOutPrintln(String str)
348     {
349         System.out.println(str);
350     }
351
352     public void setJobPrintable(Printable printable)
353     {
354         this.job.setPrintable(printable, this.pageFormat);
355     }
356
357     public void setJobPrintDialog(boolean ok)
358     {
359         this.job.printDialog();
360     }
361
362     public void setJobPrint(PrinterJob job)
363     {
364         job.print();
365     }
366
367     public void setJobPrintException(PrinterException ex)
368     {
369         this.job.printException(ex);
370     }
371
372     public void setJobPrintSystemOutPrintln(String str)
373     {
374         System.out.println(str);
375     }
376
377     public void setJobPrintable(Printable printable)
378     {
379         this.job.setPrintable(printable, this.pageFormat);
380     }
381
382     public void setJobPrintDialog(boolean ok)
383     {
384         this.job.printDialog();
385     }
386
387     public void setJobPrint(PrinterJob job)
388     {
389         job.print();
390     }
391
392     public void setJobPrintException(PrinterException ex)
393     {
394         this.job.printException(ex);
395     }
396
397     public void setJobPrintSystemOutPrintln(String str)
398     {
399         System.out.println(str);
400     }
401
402     public void setJobPrintable(Printable printable)
403     {
404         this.job.setPrintable(printable, this.pageFormat);
405     }
406
407     public void setJobPrintDialog(boolean ok)
408     {
409         this.job.printDialog();
410     }
411
412     public void setJobPrint(PrinterJob job)
413     {
414         job.print();
415     }
416
417     public void setJobPrintException(PrinterException ex)
418     {
419         this.job.printException(ex);
420     }
421
422     public void setJobPrintSystemOutPrintln(String str)
423     {
424         System.out.println(str);
425     }
426
427     public void setJobPrintable(Printable printable)
428     {
429         this.job.setPrintable(printable, this.pageFormat);
430     }
431
432     public void setJobPrintDialog(boolean ok)
433     {
434         this.job.printDialog();
435     }
436
437     public void setJobPrint(PrinterJob job)
438     {
439         job.print();
440     }
441
442     public void setJobPrintException(PrinterException ex)
443     {
444         this.job.printException(ex);
445     }
446
447     public void setJobPrintSystemOutPrintln(String str)
448     {
449         System.out.println(str);
450     }
451
452     public void setJobPrintable(Printable printable)
453     {
454         this.job.setPrintable(printable, this.pageFormat);
455     }
456
457     public void setJobPrintDialog(boolean ok)
458     {
459         this.job.printDialog();
460     }
461
462     public void setJobPrint(PrinterJob job)
463     {
464         job.print();
465     }
466
467     public void setJobPrintException(PrinterException ex)
468     {
469         this.job.printException(ex);
470     }
471
472     public void setJobPrintSystemOutPrintln(String str)
473     {
474         System.out.println(str);
475     }
476
477     public void setJobPrintable(Printable printable)
478     {
479         this.job.setPrintable(printable, this.pageFormat);
480     }
481
482     public void setJobPrintDialog(boolean ok)
483     {
484         this.job.printDialog();
485     }
486
487     public void setJobPrint(PrinterJob job)
488     {
489         job.print();
490     }
491
492     public void setJobPrintException(PrinterException ex)
493     {
494         this.job.printException(ex);
495     }
496
497     public void setJobPrintSystemOutPrintln(String str)
498     {
499         System.out.println(str);
500     }
501
502     public void setJobPrintable(Printable printable)
503     {
504         this.job.setPrintable(printable, this.pageFormat);
505     }
506
507     public void setJobPrintDialog(boolean ok)
508     {
509         this.job.printDialog();
510     }
511
512     public void setJobPrint(PrinterJob job)
513     {
514         job.print();
515     }
516
517     public void setJobPrintException(PrinterException ex)
518     {
519         this.job.printException(ex);
520     }
521
522     public void setJobPrintSystemOutPrintln(String str)
523     {
524         System.out.println(str);
525     }
526
527     public void setJobPrintable(Printable printable)
528     {
529         this.job.setPrintable(printable, this.pageFormat);
530     }
531
532     public void setJobPrintDialog(boolean ok)
533     {
534         this.job.printDialog();
535     }
536
537     public void setJobPrint(PrinterJob job)
538     {
539         job.print();
540     }
541
542     public void setJobPrintException(PrinterException ex)
543     {
544         this.job.printException(ex);
545     }
546
547     public void setJobPrintSystemOutPrintln(String str)
548     {
549         System.out.println(str);
550     }
551
552     public void setJobPrintable(Printable printable)
553     {
554         this.job.setPrintable(printable, this.pageFormat);
555     }
556
557     public void setJobPrintDialog(boolean ok)
558     {
559         this.job.printDialog();
560     }
561
562     public void setJobPrint(PrinterJob job)
563     {
564         job.print();
565     }
566
567     public void setJobPrintException(PrinterException ex)
568     {
569         this.job.printException(ex);
570     }
571
572     public void setJobPrintSystemOutPrintln(String str)
573     {
574         System.out.println(str);
575     }
576
577     public void setJobPrintable(Printable printable)
578     {
579         this.job.setPrintable(printable, this.pageFormat);
580     }
581
582     public void setJobPrintDialog(boolean ok)
583     {
584         this.job.printDialog();
585     }
586
587     public void setJobPrint(PrinterJob job)
588     {
589         job.print();
590     }
591
592     public void setJobPrintException(PrinterException ex)
593     {
594         this.job.printException(ex);
595     }
596
597     public void setJobPrintSystemOutPrintln(String str)
598     {
599         System.out.println(str);
600     }
601
602     public void setJobPrintable(Printable printable)
603     {
604         this.job.setPrintable(printable, this.pageFormat);
605     }
606
607     public void setJobPrintDialog(boolean ok)
608     {
609         this.job.printDialog();
610     }
611
612     public void setJobPrint(PrinterJob job)
613     {
614         job.print();
615     }
616
617     public void setJobPrintException(PrinterException ex)
618     {
619         this.job.printException(ex);
620     }
621
622     public void setJobPrintSystemOutPrintln(String str)
623     {
624         System.out.println(str);
625     }
626
627     public void setJobPrintable(Printable printable)
628     {
629         this.job.setPrintable(printable, this.pageFormat);
630     }
631
632     public void setJobPrintDialog(boolean ok)
633     {
634         this.job.printDialog();
635     }
636
637     public void setJobPrint(PrinterJob job)
638     {
639         job.print();
640     }
641
642     public void setJobPrintException(PrinterException ex)
643     {
644         this.job.printException(ex);
645     }
646
647     public void setJobPrintSystemOutPrintln(String str)
648     {
649         System.out.println(str);
650     }
651
652     public void setJobPrintable(Printable printable)
653     {
654         this.job.setPrintable(printable, this.pageFormat);
655     }
656
657     public void setJobPrintDialog(boolean ok)
658     {
659         this.job.printDialog();
660     }
661
662     public void setJobPrint(PrinterJob job)
663     {
664         job.print();
665     }
666
667     public void setJobPrintException(PrinterException ex)
668     {
669         this.job.printException(ex);
670     }
671
672     public void setJobPrintSystemOutPrintln(String str)
673     {
674         System.out.println(str);
675     }
676
677     public void setJobPrintable(Printable printable)
678     {
679         this.job.setPrintable(printable, this.pageFormat);
680     }
681
682     public void setJobPrintDialog(boolean ok)
683     {
684         this.job.printDialog();
685     }
686
687     public void setJobPrint(PrinterJob job)
688     {
689         job.print();
690     }
691
692     public void setJobPrintException(PrinterException ex)
693     {
694         this.job.printException(ex);
695     }
696
697     public void setJobPrintSystemOutPrintln(String str)
698     {
699         System.out.println(str);
700     }
701
702     public void setJobPrintable(Printable printable)
703     {
704         this.job.setPrintable(printable, this.pageFormat);
705     }
706
707     public void setJobPrintDialog(boolean ok)
708     {
709         this.job.printDialog();
710     }
711
712     public void setJobPrint(PrinterJob job)
713     {
714         job.print();
715     }
716
717     public void setJobPrintException(PrinterException ex)
718     {
719         this.job.printException(ex);
720     }
721
722     public void setJobPrintSystemOutPrintln(String str)
723     {
724         System.out.println(str);
725     }
726
727     public void setJobPrintable(Printable printable)
728     {
729         this.job.setPrintable(printable, this.pageFormat);
730     }
731
732     public void setJobPrintDialog(boolean ok)
733     {
734         this.job.printDialog();
735     }
736
737     public void setJobPrint(PrinterJob job)
738     {
739         job.print();
740     }
741
742     public void setJobPrintException(PrinterException ex)
743     {
744         this.job.printException(ex);
745     }
746
747     public void setJobPrintSystemOutPrintln(String str)
748     {
749         System.out.println(str);
750     }
751
752     public void setJobPrintable(Printable printable)
753     {
754         this.job.setPrintable(printable, this.pageFormat);
755     }
756
757     public void setJobPrintDialog(boolean ok)
758     {
759         this.job.printDialog();
760     }
761
762     public void setJobPrint(PrinterJob job)
763     {
764         job.print();
765     }
766
767     public void setJobPrintException(PrinterException ex)
768     {
769         this.job.printException(ex);
770     }
771
772     public void setJobPrintSystemOutPrintln(String str)
773     {
774         System.out.println(str);
775     }
776
777     public void setJobPrintable(Printable printable)
778     {
779         this.job.setPrintable(printable, this.pageFormat);
780     }
781
782     public void setJobPrintDialog(boolean ok)
783     {
784         this.job.printDialog();
785     }
786
787     public void setJobPrint(PrinterJob job)
788     {
789         job.print();
790     }
791
792     public void setJobPrintException(PrinterException ex)
793     {
794         this.job.printException(ex);
795     }
796
797     public void setJobPrintSystemOutPrintln(String str)
798     {
799         System.out.println(str);
800     }
801
802     public void setJobPrintable(Printable printable)
803     {
804         this.job.setPrintable(printable, this.pageFormat);
805     }
806
807     public void setJobPrintDialog(boolean ok)
808     {
809         this.job.printDialog();
810     }
811
812     public void setJobPrint(PrinterJob job)
813     {
814         job.print();
815     }
816
817     public void setJobPrintException(PrinterException ex)
818     {
819         this.job.printException(ex);
820     }
821
822     public void setJobPrintSystemOutPrintln(String str)
823     {
824         System.out.println(str);
825     }
826
827     public void setJobPrintable(Printable printable)
828     {
829         this.job.setPrintable(printable, this.pageFormat);
830     }
831
832     public void setJobPrintDialog(boolean ok)
833     {
834         this.job.printDialog();
835     }
836
837     public void setJobPrint(PrinterJob job)
838     {
839         job.print();
840     }
841
842     public void setJobPrintException(PrinterException ex)
843     {
844         this.job.printException(ex);
845     }
846
847     public void setJobPrintSystemOutPrintln(String str)
848     {
849         System.out.println(str);
850     }
851
852     public void setJobPrintable(Printable printable)
853     {
854         this.job.setPrintable(printable, this.pageFormat);
855     }
856
857     public void setJobPrintDialog(boolean ok)
858     {
859         this.job.printDialog();
860     }
861
862     public void setJobPrint(PrinterJob job)
863     {
864         job.print();
865     }
866
867     public void setJobPrintException(PrinterException ex)
868     {
869         this.job.printException(ex);
870     }
871
872     public void setJobPrintSystemOutPrintln(String str)
873     {
874         System.out.println(str);
875     }
876
877     public void setJobPrintable(Printable printable)
878     {
879         this.job.setPrintable(printable, this.pageFormat);
880     }
881
882     public void setJobPrintDialog(boolean ok)
883     {
884         this.job.printDialog();
885     }
886
887     public void setJobPrint(PrinterJob job)
888     {
889         job.print();
890     }
891
892     public void setJobPrintException(PrinterException ex)
893     {
894         this.job.printException(ex);
895     }
896
897     public void setJobPrintSystemOutPrintln(String str)
898     {
899         System.out.println(str);
900     }
901
902     public void setJobPrintable(Printable printable)
903     {
904         this.job.setPrintable(printable, this.pageFormat);
905     }
906
907     public void setJobPrintDialog(boolean ok)
908     {
909         this.job.printDialog();
910     }
911
912     public void setJobPrint(PrinterJob job)
913     {
914         job.print();
915     }
916
917     public void setJobPrintException(PrinterException ex)
918     {
919         this.job.printException(ex);
920     }
921
922     public void setJobPrintSystemOutPrintln(String str)
923     {
924         System.out.println(str);
925     }
926
927     public void setJobPrintable(Printable printable)
928     {
929         this.job.setPrintable(printable, this.pageFormat);
930     }
931
932     public void setJobPrintDialog(boolean ok)
933     {
934         this.job.printDialog();
935     }
936
937     public void setJobPrint(PrinterJob job)
938     {
939         job.print();
940     }
941
942     public void setJobPrintException(PrinterException ex)
943     {
944         this.job.printException(ex);
945     }
946
947     public void setJobPrintSystemOutPrintln(String str)
948     {
949         System.out.println(str);
950     }
951
952     public void setJobPrintable(Printable printable)
953     {
954         this.job.setPrintable(printable, this.pageFormat);
955     }
956
957     public void setJobPrintDialog(boolean ok)
958     {
959         this.job.printDialog();
960     }
961
962     public void setJobPrint(PrinterJob job)
963     {
964         job.print();
965     }
966
967     public void setJobPrintException(PrinterException ex)
968     {
969         this.job.printException(ex);
970     }
971
972     public void setJobPrintSystemOutPrintln(String str)
973     {
974         System.out.println(str);
975     }
976
977     public void setJobPrintable(Printable printable)
978     {
979         this.job.setPrintable(printable, this.pageFormat);
980     }
981
982     public void setJobPrintDialog(boolean ok)
983     {
984         this.job.printDialog();
985     }
986
987     public void setJobPrint(PrinterJob job)
988     {
989         job.print();
990     }
991
992     public void setJobPrintException(PrinterException ex)
993     {
994         this.job.printException(ex);
995     }
996
997     public void setJobPrintSystemOutPrintln(String str)
998     {
999         System.out.println(str);
1000     }
1001
1002     public void setJobPrintable(Printable printable)
1003     {
1004         this.job.setPrintable(printable, this.pageFormat);
1005     }
1006
1007     public void setJobPrintDialog(boolean ok)
1008     {
1009         this.job.printDialog();
1010     }
1011
1012     public void setJobPrint(PrinterJob job)
1013     {
1014         job.print();
1015     }
1016
1017     public void setJobPrintException(PrinterException ex)
1018     {
1019         this.job.printException(ex);
1020     }
1021
1022     public void setJobPrintSystemOutPrintln(String str)
1023     {
1024         System.out.println(str);
1025     }
1026
1027     public void setJobPrintable(Printable printable)
1028     {
1029         this.job.setPrintable(printable, this.pageFormat);
1030     }
1031
1032     public void setJobPrintDialog(boolean ok)
1033     {
1034         this.job.printDialog();
1035     }
1036
1037     public void setJobPrint(PrinterJob job)
1038     {
1039         job.print();
1040     }
1041
1042     public void setJobPrintException(PrinterException ex)
1043     {
1044         this.job.printException(ex);
1045     }
1046
1047     public void setJobPrintSystemOutPrintln(String str)
1048     {
1049         System.out.println(str);
1050     }
1051
1052     public void setJobPrintable(Printable printable)
1053     {
1054         this.job.setPrintable(printable, this.pageFormat);
1055     }
1056
1057     public void setJobPrintDialog(boolean ok)
1058     {
1059         this.job.printDialog();
1060     }
1061
1062     public void setJobPrint(PrinterJob job)
1063     {
1064         job.print();
1065     }
1066
1067     public void setJobPrintException(PrinterException ex)
1068     {
1069         this.job.printException(ex);
1070     }
1071
1072     public void setJobPrintSystemOutPrintln(String str)
1073     {
1074         System.out.println(str);
1075     }
1076
1077     public void setJobPrintable(Printable printable)
1078     {
1079         this.job.setPrintable(printable, this.pageFormat);
1080     }
1081
1082     public void setJobPrintDialog(boolean ok)
1083     {
1084         this.job.printDialog();
1085     }
1086
1087     public void setJobPrint(PrinterJob job)
1088     {
1089         job.print();
1090     }
1091
1092     public void setJobPrintException(PrinterException ex)
1093     {
1094         this.job.printException(ex);
1095     }
1096
1097     public void setJobPrintSystemOutPrintln(String str)
1098     {
1099         System.out.println(str);
1100     }
1101
1102     public void setJobPrintable(Printable printable)
1103     {
1104         this.job.setPrintable(printable, this.pageFormat);
1105     }
1106
1107     public void setJobPrintDialog(boolean ok)
1108     {
1109         this.job.printDialog();
1110     }
1111
1112     public void setJobPrint(PrinterJob job)
1113     {
1114         job.print();
1115     }
1116
1117     public void setJobPrintException(PrinterException ex)
1118     {
1119         this.job.printException(ex);
1120     }
1121
1122     public void setJobPrintSystemOutPrintln(String str)
1123     {
1124         System.out.println(str);
1125     }
1126
1127     public void setJobPrintable(Printable printable)
1128     {
1129         this.job.setPrintable(printable, this.pageFormat);
1130     }
1131
1132     public void setJobPrintDialog(boolean ok)
1133     {
1134         this.job.printDialog();
1135     }
1136
1137     public void setJobPrint(PrinterJob job)
1138     {
1139         job.print();
1140     }
1141
1142     public void setJobPrintException(PrinterException ex)
1143     {
1144         this.job.printException(ex);
1145     }
1146
1147     public void setJobPrintSystemOutPrintln(String str)
1148     {
1149         System.out.println(str);
1150     }
1151
1152     public void setJobPrintable(Printable printable)
1153     {
1154         this.job.setPrintable(printable, this.pageFormat);
1155     }
1156
1157     public void setJobPrintDialog(boolean ok)
1158     {
1159         this.job.printDialog();
1160     }
1161
1162     public void setJobPrint(PrinterJob job)
1163     {
1164         job.print();
1165     }
1166
1167     public void setJobPrintException(PrinterException ex)
1168     {
1169         this.job.printException(ex);
1170     }
1171
1172     public void setJobPrintSystemOutPrintln(String str)
1173     {
1174         System.out.println(str);
1175     }
1176
1177     public void setJobPrintable(Printable printable)
1178     {
1179         this.job.setPrintable(printable, this.pageFormat);
1180     }
1181
1182     public void setJobPrintDialog(boolean ok)
1183     {
1184         this.job.printDialog();
1185     }
1186
1187     public void setJobPrint(PrinterJob job)
1188     {
1189         job.print();
1190     }
1191
1192     public void setJobPrintException(PrinterException ex)
1193     {
1194         this.job.printException(ex);
1195     }
1196
1197     public void setJobPrintSystemOutPrintln(String str)
1198     {
1199         System.out.println(str);
1200     }
1201
1202     public void setJobPrintable(Printable printable)
1203     {
1204         this.job.setPrintable(printable, this.pageFormat);
1205     }
1206
1207     public void setJobPrintDialog(boolean ok)
1208     {
1209         this.job.printDialog();
1210     }
1211
1212     public void setJobPrint(PrinterJob job)
1213     {
1214         job.print();
1215     }
1216
1217     public void setJobPrintException(PrinterException ex)
1218     {
1219         this.job.printException(ex);
1220     }
1221
1222     public void setJobPrintSystemOutPrintln(String str)
1223     {
1224         System.out.println(str);
1225     }
1226
1227     public void setJobPrintable(Printable printable)
1228     {
1229         this.job.setPrintable(printable, this.pageFormat);
1230     }
1231
1232     public void setJobPrintDialog(boolean ok)
1233     {
1234         this.job.printDialog();
1235     }
1236
1237     public void setJobPrint(PrinterJob job)
1238     {
1239         job.print();
1240     }
1241
1242     public void setJobPrintException(PrinterException ex)
1243     {
1244         this.job.printException(ex);
1245     }
1246
1247     public void setJobPrintSystemOutPrintln(String str)
1248     {
1249         System.out.println(str);
1250     }
1251
1252     public void setJobPrintable(Printable printable)
1253     {
1254         this.job.setPrintable(printable, this.pageFormat);
1255     }
1256
1257     public void setJobPrintDialog(boolean ok)
1258     {
1259         this.job.printDialog();
1260     }
1261
1262     public void setJobPrint(PrinterJob job)
1263     {
1264         job.print();
1265     }
1266
1267     public void setJobPrintException(PrinterException ex)
1268     {
1269         this.job.printException(ex);
1270     }
1271
1272     public void setJobPrintSystemOutPrintln(String str)
1273     {
1274         System.out.println(str);
1275     }
1276
1277     public void setJobPrintable(Printable printable)
1278     {
1279         this.job.setPrintable(printable, this.pageFormat);
1280     }
1281
1282     public void setJobPrintDialog(boolean ok)
1283     {
1284         this.job.printDialog();
1285     }
1286
1287     public void setJobPrint(PrinterJob job)
1288     {
1289         job.print();
1290     }
1291
1292     public void setJobPrintException(PrinterException ex)
1293     {
1294         this.job.printException(ex);
1295     }
1296
1297     public void setJobPrintSystemOutPrintln(String str)
1298     {
1299         System.out.println(str);
1300     }
1301
1302     public void setJobPrintable(Printable printable)
1303     {
1304         this.job.setPrintable(printable, this.pageFormat);
1305     }
1306
1307     public void setJobPrintDialog(boolean ok)
1308     {
1309         this.job.printDialog();
1310     }
1311
1312     public void setJobPrint(PrinterJob job)
1313     {
1314         job.print();
1315     }
1316
1317     public void setJobPrintException(PrinterException ex)
1318     {
1319         this.job.printException(ex);
1320     }
1321
1322     public void setJobPrintSystemOutPrintln(String str)
1323     {
1324         System.out.println(str);
1325     }
1326
1327     public void setJobPrintable(Printable printable)
1328     {
1329         this.job.setPrintable(printable, this.pageFormat);
1330     }
1331
1332     public void setJobPrintDialog(boolean ok)
1333     {
1334         this.job.printDialog();
1335     }
1336
1337     public void setJobPrint(PrinterJob job)
1338     {
1339         job.print();
1340     }
1341
1342     public void setJobPrintException(PrinterException ex)
1343     {
1344         this.job.printException(ex);
1345     }
1346
1347     public void setJobPrintSystemOutPrintln(String str)
1348     {
1349         System.out.println(str);
1350     }
1351
1352     public void setJobPrintable(Printable printable)
1353     {
1354         this.job.setPrintable(printable, this.pageFormat);
1355     }
1356
1357     public void setJobPrintDialog(boolean ok)
1358     {
1359         this.job.printDialog();
1360     }
1361
1362     public void setJobPrint(PrinterJob job)
1363     {
1364         job.print();
1365     }
1366
1367     public void setJobPrintException(PrinterException ex)
1368     {
1369         this.job.printException(ex);
1370     }
13
```

```

142         textLines[8] = "\n";
143     }
144     else
145     {
146         int numLines=players.size() * RECORD_SIZE;
147         textLines = new String[numLines];
148         for (int i=0;i<numLines;i+=RECORD_SIZE)
149         {
150             Player aPlayer = (players.get(i/RECORD_SIZE));
151             textLines[i] = String.format("%-16s%3n", "Name:", aPlayer.getName());
152             textLines[i+1] = String.format("%-16s%3n", "Program Week:", week);
153             textLines[i+2] = String.format("%-16s%3d%3s%3d%3s%3d%3n", "Current Maxes:", "Bench - ", aPlayer.getBenchMax(), " Squat - ", aPlayer.getSquatMax(), " Incline - ", aPlayer.getInclineMax(), " Power Clean - ", aPlayer.getPowerMax());
154             textLines[i+3] = "-----";
155             textLines[i+4] = String.format("%-8s%3d%3s%3d%3n", "Reps: ", weekReps[0], ", ", weekReps[1], ", ", weekReps[2]);
156             textLines[i+5] = String.format("%-8s%3d%3s%3d%3n", "Squat: ", WeightLiftingProgram.calculateSquat(aPlayer.getSquatMax(), week), " ", "Power Clean:", WeightLiftingProgram.calculatePowerClean(aPlayer.getPowerMax(), week));
157             textLines[i+6] = String.format("%-8s%3d%3s%3d%3n", "Bench: ", WeightLiftingProgram.calculateBench(aPlayer.getBenchMax(), week), " ", "Incline: ", WeightLiftingProgram.calculateIncline(aPlayer.getInclineMax(), week));
158             textLines[i+7] = "\n";
159             textLines[i+8] = "\n";
160         }
161     }
162 }
163
164
165 @Override
166 public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
167 {
168     Font font = new Font("Consolas", Font.PLAIN, 12);
169     g.setFont(font);
170
171     FontMetrics metrics = g.getFontMetrics(font);
172     int lineHeight = metrics.getHeight();
173
174     if (pageBreaks == null)
175     {
176         initTextLines();
177         int linesPerPage = (int)(pf.getImageableHeight()/lineHeight);
178         int recordsPerPage = linesPerPage/RECORD_SIZE;
179         int numRecords = textLines.length/RECORD_SIZE;
180         int numBreaks = (numRecords * RECORD_SIZE) / linesPerPage;
181         pageBreaks = new int[numBreaks];
182         for (int b=0; b<numBreaks; b++)
183         {
184             pageBreaks[b] = (b+1)* recordsPerPage * RECORD_SIZE;
185         }
186     }
187
188     if (pageIndex > pageBreaks.length)
189     {
190         return NO_SUCH_PAGE;
191     }
192
193     /* User (0,0) is typically outside the imageable area, so we must
194     * translate by the X and Y values in the PageFormat to avoid clipping
195     * Since we are drawing text we
196     */
197     Graphics2D g2d = (Graphics2D)g;
198     g2d.translate(pf.getImageableX(), pf.getImageableY());
199
200     /* Draw each line that is on this page.
201     * Increment 'y' position by lineHeight for each line.
202     */
203     int y = 0;
204     int start = (pageIndex == 0) ? 0 : pageBreaks[pageIndex-1];
205     int end = (pageIndex == pageBreaks.length)
206         ? textLines.length : pageBreaks[pageIndex];
207     for (int line=start; line<end; line++)
208     {
209         y += lineHeight;
210         g.drawString(textLines[line], 0, y);
211     }
212
213     /* tell the caller that this page is part of the printed document */
214     return PAGE_EXISTS;
215 }
216
217 }

```

```
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 multiplying 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 }
```

```
1 package org.bwagner;
2
3 import java.util.*;
4
5 /*
6     This class stores up to four Players to form a workout group.
7 */
8
9 public class Group
10 {
11     private Player[] group;
12     private int maxGroupSize;
13     private int numPlayers;
14
15     //constructor
16     public Group(int groupSize)
17     {
18         maxGroupSize = groupSize;
19         group = new Player[maxGroupSize];
20         numPlayers = 0;
21     }
22
23     /*
24         Adds a player to the group
25     */
26     public void addPlayer(Player player)
27     {
28         if(numPlayers < maxGroupSize)
29         {
30             group[numPlayers] = player;
31             numPlayers++;
32         }
33     }
34
35     /*
36         @return the number of players in this group
37     */
38     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 ArrayList 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     /*
58     Prepares the data for printing by filling the textLines array with all
59     the data from the array of groups into string format.
60     */
61     private void initTextLines()
62     {
63         if (textLines == null)
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)

```



```

80         textLines[i+j] = String.format("%-22s%s%d\n", players[j-2].getName(), "Bench Max = ", players[j-2].getBenchMax());
81     }
82     textLines[i+j] = "\n";
83     textLines[i+j+1] = "\n";
84
85     if(i % RECORD_SIZE == 0 || group.getGroupSize() < 4)
86     {
87         groupCount++;
88     }
89 }
90
91 }
92 }
93
94 @Override
95 public int print(Graphics g, PageFormat pf, int pageIndex) throws PrinterException
96 {
97     Font font = new Font("Consolas", Font.PLAIN, 14);
98     g.setFont(font);
99     FontMetrics metrics = g.getFontMetrics(font);
100    int lineHeight = metrics.getHeight();
101
102    if (pageBreaks == null)
103    {
104        initTextLines();
105        int linesPerPage = (int)(pf.getImageableHeight()/lineHeight);
106        int recordsPerPage = linesPerPage/RECORD_SIZE;
107        int numRecords = textLines.length/RECORD_SIZE;
108        int numBreaks = (numRecords * RECORD_SIZE) / linesPerPage;
109        pageBreaks = new int[numBreaks];
110        for (int b=0; b<numBreaks; b++)
111        {
112            pageBreaks[b] = (b+1)* recordsPerPage * RECORD_SIZE;
113        }
114    }
115
116    if (pageIndex > pageBreaks.length)
117    {
118        return NO_SUCH_PAGE;
119    }
120
121    /* User (0,0) is typically outside the imageable area, so we must
122     * translate by the X and Y values in the PageFormat to avoid clipping
123     * Since we are drawing text we
124     */
125
126    Graphics2D g2d = (Graphics2D)g;
127    g2d.translate(pf.getImageableX(), pf.getImageableY());
128
129    /* Draw each line that is on this page.
130     * Increment 'y' position by lineHeight for each line.
131     */
132
133    int y = 0;
134    int start = (pageIndex == 0) ? 0 : pageBreaks[pageIndex-1];
135    int end = (pageIndex == pageBreaks.length)
136             ? textLines.length : pageBreaks[pageIndex];
137    for (int line=start; line<end; line++)
138    {
139        y += lineHeight;
140        if(textLines[line] != null)
141            g.drawString(textLines[line], 0, y);
142    }
143
144
145
146    /* tell the caller that this page is part of the printed document */
147    return PAGE_EXISTS;
148 }
149
150 }

```