```java
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

/**
 * Name: Zane Emerick
 * Class: CS 1450 Section 001
 * Assignment #4
 * Due: Feb 21, 2020
 *
 * Description: Design a pinball machine with certain targets to be specified
 *  in a file. The machine needs to be able to store these targets as Objects
 *  in a 2D array and be able to be played based upon information written in
 *  a second file. Next, the program needs to be able to generate a report
 *  describing which targets were hit and which were not, and how many points
 *  hit would be worth. Finally, all of this data needs to be printed out to
 *  the user.
 */

public class EmerickZaneAssignment4 {
    public static void main(String[] args) throws IOException {
        File pinballTargetsFile = new File("PinballMachineTargets.txt");

        Scanner targetReader = new Scanner(pinballTargetsFile);

        int numRows = targetReader.nextInt();
        int numCols = targetReader.nextInt();

        PinBallMachine playField = new PinBallMachine(numRows, numCols);

        int currRow, currColumn, id, points;
        Target currTarget;
        String type;

        //loop to create the targets and load them into machine
        while (targetReader.hasNext()) {
            currRow = targetReader.nextInt();
            currColumn = targetReader.nextInt();
            type = targetReader.next();
            id = targetReader.nextInt();
            points = targetReader.nextInt();

            currTarget = new Target(type, id, points);

            playField.addTargetToPlayingField(currRow, currColumn,
currTarget);
        }

        //display, play, print results
        playField.displayPlayingField();

        play(playField);
        System.out.println("\n\n");
        printReport(playField);

        targetReader.close();
    }

```

```
60      /**
61       * method to play the actual game using the simulated targets and numbers
62       * @param pinBallMachine the PinBallMachine to be played on
63       * @throws IOException
64       */
65      public static void play(PinBallMachine pinBallMachine) throws IOException
    {
66          File playFile = new File("Play.txt");
67          Scanner playReader = new Scanner(playFile);
68
69          int row, column;
70          Target location;
71          int currentScore = 0;
72
73          System.out.println("---------------------------------");
74          System.out.println("      Simulated Pinball Game     ");
75          System.out.println("---------------------------------");
76          System.out.printf("%-12s %-6s %-8s %-5s%n", "Target", "ID", "Points",
    "Score");
77
78          while (playReader.hasNext()) {
79              row = playReader.nextInt();
80              column = playReader.nextInt();
81
82              if (pinBallMachine.getTarget(row, column) != null) {
83                  location = pinBallMachine.getTarget(row, column);
84
85                  location.incrementHits();
86                  currentScore += location.getPoints();
87
88                  System.out.printf("%-12s %-6d %-8d %-5d%n",
    location.getType(), location.getId(), location.getPoints(),
89                          currentScore);
90              }
91          }
92          playReader.close();
93      }
94
95      /**
96       * prints a report showing how the game went and which targets were hit
97       * @param pinBallMachine the pinBallMachine the game was played on
98       */
99      public static void printReport(PinBallMachine pinBallMachine) {
100         ArrayList<TargetReport> reportsList = new ArrayList<>();
101         TargetReport report;
102         Target location;
103
104         for(int i = 0; i < pinBallMachine.getRows(); i++) {
105             for(int j = 0; j < pinBallMachine.getCols(); j++) {
106                 location = pinBallMachine.getTarget(i, j);
107                 if( location != null) {
108                     report = new TargetReport(i, j, location.getId(),
    location.getPoints(), location.getHits(), location.getType());
109                     reportsList.add(report);
110                 }
111             }
112         }
113
114         /**
115          * Collections.sort() prints a random string of binary?
```

```
116          * Not really sure why and couldn't figure out how to fix it.
117          * Code runs correctly otherwise.
118         */
119         Collections.sort(reportsList, Collections.reverseOrder());
120 
121 
122         System.out.println("\n-------------------------------------------
--");
123         System.out.println("       Pinball Machine Targets Hit Report
");
124         System.out.println("-------------------------------------------
");
125         System.out.printf("%-5s %-10s %-10s %-5s %-10s %-5s%n", "Row",
"Column", "Type", "ID", "Points", "Hits");
126         System.out.println("-------------------------------------------
");
127 
128         for(int i = 0; i < reportsList.size(); i++) {
129             System.out.print(reportsList.get(i).print());
130         }
131 
132     }
133 
134 }
135 
136 
137 class PinBallMachine {
138     private int numberRows, numberCols;
139     private Target[][] playingField;
140 
141     public PinBallMachine(int numberRows, int numberCols) {
142         this.numberRows = numberRows;
143         this.numberCols = numberCols;
144 
145         playingField = new Target[numberRows][numberCols];
146     }
147 
148     public int getRows() {
149         return numberRows;
150     }
151 
152     public int getCols() {
153         return numberCols;
154     }
155 
156     public void addTargetToPlayingField(int row, int column, Target target) {
157         playingField[row][column] = target;
158     }
159 
160     public Target getTarget(int row, int column) {
161         Target location = playingField[row][column];
162 
163         if(location == null) {
164             location = null;
165         }
166         return location;
167     }
168 
169     /**
170      * Displays the playing field as a grid
```

```
     */
    public void displayPlayingField() {
        System.out.print("       ");
        for(int i = 0; i < numberCols; i++) {
            System.out.printf("%-12s", "column: " + i);
        }
        System.out.println();

        for(int i = 0; i < (numberCols * 12) + 4; i++) {
            System.out.print("-");
        }
        System.out.println();

        Target currentLocation;
        for(int i = 0; i < numberRows; i++) {
            System.out.print("row: " + i);

            for(int j = 0; j < numberCols; j++) {

                currentLocation = playingField[i][j];

                if(currentLocation == null) {
                    System.out.printf(" %-11s", "---------");
                } else {
                    System.out.printf(" %-11s", currentLocation.getType());
                }
            }
            System.out.println("\n");
        }
    }
}

class Target {
    private String type;
    private int id, points, hits;

    public Target(String type, int id, int points) {
        this.type = type;
        this.id = id;
        this.points = points;
    }

    public int getId() {
        return id;
    }

    public int getPoints() {
        return points;
    }

    public int getHits() {
        return hits;
    }

    public String getType() {
        return type;
    }

    public void incrementHits() {
        hits++;
```

```
    }
}

class TargetReport implements Comparable<TargetReport> {
    private int rowNumber, columnNumber, id, points, hits;
    private String type;

    public TargetReport(int rowNumber, int columnNumber, int id, int points,
int hits, String type) {
        this.type = type;
        this.rowNumber = rowNumber;
        this.columnNumber = columnNumber;
        this.id = id;
        this.points = points;
        this.hits = hits;
    }

    public String print() {
        return String.format("%-5d %-10d %-10s %-5d %-10d %-5d%n", rowNumber,
columnNumber, type, id, points, hits);
    }

    /**
     * Overrides the compareTo method to use for sorting
     * @param otherReport the report to be compared to
     * @return -1 if other has more hits, and +1 if this has more hits
     */
    @Override
    public int compareTo(TargetReport otherReport) {
        int token = 0;
        if(this.hits > otherReport.hits) {
            token = 1;
        } else if (this.hits < otherReport.hits) {
            token = -1;
        }
        System.out.print(token);
        return token;
    }
}
```