**CS 1450 Data Structures and Algorithms – Spring 2020**

**Assignment #4**

Due Date: **Feb 19, 2020** at 1:40pm (MW class), **Feb 20, 2020** at 1:40pm (TR classes)

Purpose: Learn to use objects that contain objects and work with collections

Effort: Individual

Points: **100**

Deliverables: Upload a **.zip** file with **ONLY** your source code (the one .java file) to Canvas.

**Hand in a hardcopy version of the code and your design notes at beginning of class.**

**Assignment Description**

This assignment provides more practice creating and manipulating classes as well as the opportunity to use Java’s Comparable interface. For this assignment, create a program that simulates a Pinball Machine. Here are some terms related to pinball and their meaning for this assignment. See the document **PinballTargets.docx** for a visual.

*Playing Field*

* The flat surface of the game on which targets are arranged. The ball rolls along this surface hitting the targets giving the player points.

*Target*

* The targets we will use are bumpers, kickers, rollovers, and flippers. When the ball comes in contact with a bumper, kicker, or rollover points are added to the overall score.

*Hit*

* When the ball comes in contact with a target.

The pinball machine will contain **a playing field** of **targets**. The size of the playing field, is specified in the *PinballMachineTargets.txt* file. To setup the pinball machine’s **playing field**, read the number of rows and number of columns from the file and create a Pinball Machine object that contains a playing field (2D array) based on these values. Next, for each target in the file, read its details, create the target, and place the target into playing field at its specified row and column. Note, not all locations in the pinball machine’s playing field will contain a target. For example, in the test file shown below, in row 0 columns 0, 1, 3 and 4 are empty, in row 1 columns 0, 2, and 4 are empty, rows 2-6 also contain empty columns.

**After** all targets are placed into the pinball machine’s playing field, display the pinball machine, play a game, and print a report. In the report, targets must be displayed in order from the one with the most hits to the lowest hits. See output below.

**Specifications**

1. Create a Java class called **LastNameFirstNameAssignment4**
2. Follow "CS1450 Programming Assignments Policy"
3. Write a test program (i.e. main) to do the following:
   * Step 1: Create a **pinball machine** object
     + Contains a playing field (2D array) with the number of rows & columns read from file.
   * Step 2: Load pinball machine with all targets in file
     + For each target in the file:
       - Read details for the target from the file.
       - Create a target object using the details.
       - Add target to pinball machine’s playing field in row & column specified in file
         * Use the **addTargetToPlayingField** method in the **PinballMachine** class
   * Step 3: Display the loaded pinball machine.
     + Display a nicely formatted version of the pinball machine’s playing field
     + Use ***displayPlayingField()*** method in **PinballMachine** class (see 1st output example)
   * Step 4: Play a **game**. Write a method to play a game:

**public** **static** **void** play(PinballMachine pinballMachine) **throws** IOException

This method must:

* Be placed in your Assignment4 class
* *Play.txt* file simulates one game. This file provides different locations of the ball on playing field during the game. See #4 below for file details.
* For each location in the file, determine if a target was hit.
  + This means: if there is a target in the location, then a hit occurred.
  + When a target is hit, increment target’s hits, update the score with points from this target, then print the target’s **type**, **id,** **points**, and the current **score** at that point in game (see 2nd output example)
  + Step 5: Print a **report** for the pinball machine. Write a method to print the report:

**public** **static** **void** printReport (PinballMachine pinballMachine)

This method must:

* Be placed in your Assignment4 class
* Create one **TargetReport** for each target and place all reports into an ArrayList
* Sort all **TargetReports** in ArrayList using the ***Collections.sort()*** method
* Print all **TargetReports** in ArrayList (which are now sorted by number of hits)
  + - * + Use the ***print()*** method on **TargetReport** (see 3rd output example)

1. Test file information (PinballMachineTargets.txt and Play.txt)
2. Setup the pinball machine using the file **PinballMachineTargets.txt** This file is an example so **DO NOT** assume that your code should only work for a playing field with 7 rows and 5 columns and/or these targets in this specific order. The grader file will be different.

7 # of rows in playing field (array row index)

5 # of columns in playing field (array column index)

0 2 Bumper 1 100

1 1 Bumper 2 100

1 3 Bumper 3 100 Details for each target

2 0 Kicker 1 10

2 2 Rollover 1 10

2 4 Kicker 2 10

3 1 Rollover 2 50 Note

3 3 Rollover 3 50 - Row 0: columns 0, 1, 3, 4, are empty

4 2 Rollover 4 10 - Row 1: columns 0, 2, 4 are empty

5 0 Rollover 5 50 - Row 2: columns 1 & 3 are empty

5 1 Flipper 1 0 - Row 3: columns 0, 2, 4 are empty

5 3 Flipper 2 0 - Row 4: columns 0, 1, 3, 4 are empty

5 4 Rollover 6 50 - Row 5: columns 2 is empty

- Row 6: all columns are empty

* 1. 1st line is the number of rows in the pinball machine’s playing field
  2. 2nd line is the number of columns in the pinball machine’s playing field
  3. Remaining lines contain details for each **target**. The format is as follows:

Row# Column# Type Id Points

**0 1 Bumper 1 100**

1. Simulate playing a game (a ball rolling around) using the **Play.txt** file. This file is an example so **DO NOT** assume that your code should only work for the rows and columns specified in the file or the rows and columns in this specific order. The grader file will be different.

Row# Column#

**0 4**

**Classes**

Remember, build as much of the classes as possible before you start writing the code in main.

**PinballMachine Class**

* Description
  + Represents the pinball machine.
  + Pinball machine contains one playing field that is modeled by a **2-dimensional array**.
    - In the test file, the playing field has **7 rows** (0-6) and **5 columns** (0-4).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Column 0** | **Column 1** | **Column 2** | **Column 3** | **Column 4** |
| **Row 0** | -------- | -------- | Bumper | -------- | -------- |
| **Row 1** | -------- | Bumper | -------- | Bumper | -------- |
| **Row 2** | Kicker | -------- | Rollover | -------- | Kicker |
| **Row 3** | -------- | Rollover | -------- | Rollover | -------- |
| **Row 4** | -------- | -------- | Rollover | -------- | -------- |
| **Row 5** | Rollover | Flipper | -------- | Flipper | Rollover |
| **Row 6** |  |  |  |  |  |

* Private Data Fields
  + **numberRows** – number of rows in the playing field (i.e. row index )
  + **numberColumns** – number of columns in the playing field (i.e. column index)
  + **playingField** – array of **Target** objects (HAS-A relationship)
    - use a 2D array NOT ArrayList
    - see FAQ for help with 2D arrays
* Public Methods
  + Constructor: ***public PinballMachine (int numberRows, int numberColumns)***
    - Initializes numberRows and numberColumns to incoming values
    - Initializes size of playing field (2D array) to ***numberRows*** and ***numberColumns***

playingField = **new** Target[numberRows][numberColumns];

* + Getters: For data fields **numberRows** and **numberColumns**
  + Setters: None
  + ***addTargetToPlayingField (int row, int column, Target target)***
    - Places incoming target object in 2D *array* at a specified location (row/column)
  + ***getTarget (int row, int column)***
    - Returns the target stored in the playing field (2D array) at specific row/column
    - If no target in **row/column** location (i.e. empty array location), returns null
  + ***displayPlayingField()***
    - Displays nicely formatted version of pinball machine’s playing field (2D array)
    - Print row #, column #, and target type (see 1st example output below)

**Target Class**

* Description
  + Represents one target
* *Private Data Fields*
  + ***type*** *– string value for the target’s type (bumper, kicker, rollover, flipper)*
  + ***id*** *– integer value for the target’s unique id*
  + ***points*** *– integer value for the # of points this target is worth when it is hit by the ball*
  + ***hits*** *– integer value for the number of times this target is hit during play*
* Public Methods
  + Constructor ***public Target (String type, int id, int points)***
    - initializes all private data fields with incoming values
  + Getter: One for each private data field
  + Setters: None
  + ***incrementHits ()***
    - Updates the value stored in **hits** by one

**TargetReport Class**

* Description
  + Represents the report for one target.
  + Class must implement **Comparable<TargetReport>**
* Private Data Fields
  + **rowNumber** – row # target is stored in (i.e. array row index)
  + **columnNumber** – column # target is stored in (i.e. array column index)
  + **type** – string value for the target’s type (bumper, flipper, etc.)
  + **id** – integer value for the target’s unique id
  + **points** – integer value for the # of points this target is worth when hit by the ball
  + **hits** – integer value for the number of times this target is hit during play
* Public Methods
  + Constructor ***public TargetRecord (int rowNumber, int columnNumber, String type,***

***int id, int points, int hits)***

* + - initializes all private data fields with incoming values
  + Getters and Setter: None
  + ***public String print()*** 
    - returns string with row#, column#, type, id, points, and hits
  + ***public int compareTo(TargetReport otherReport)*** – overrides method in Comparable
    - returns integer value (-1, 0, 1) based on result of comparing two target reports
    - Compares two target reports based on the number of hits

**Mut Do and Tips**

**Must Do**

* Use a regular 2D array to model the pinball machine’s playing field (see FAQ on 2D arrays)
* Use an ArrayList to store **target reports**
* Use Java’s pre-defined Comparable interface. **Do not** create your own Comparable interface!

**Tip: Reading Targets from File**

* The number of targets is not part of the file as on previous assignments. On this assignment, use a while loop that reads until the end of the file is reached using the scanner’s **hasNext** method.

**Tip: Rows and Columns**

* Not all locations in the pinball machine’s playing field contain a target.
* Empty locations in the array contain the value **null**

**Tip: TargetReport Sorting and Comparable**

* In ***printReport*** method (step 5 above), use the sort method on the **Collections** class.
* Here’s how this works:
  + Because **TargetReport** implements **Comparable**, the class must contain a ***compareTo*** method. The ***compareTo*** compares two reports based on the number of hits.
  + The connection to ***Collections.sort()*** method: **Collections.sort** uses an object’s **compareTo** method to compare and sort objects! By sending the ArrayList of TargetReports to Collections.sort(), the ArrayList will be sorted based on target hits**.**
  + See Listing 13.9 p. 515 in Liang for ***compareTo*** example

**Tip: Build the Classes First and Code Incrementally**

* Create as much of each class before you write the code in main.
* Next get the code working in bits. Make sure you can read all the data from the file correctly, then create the target objects, etc.

**Tip: General**

* One way to return nicely formatted strings in the TargetReport print methods is to use the string format method. For example:

**return** String.*format*("%d\t%d\t%-10s\t%-1d\t%-4d\t%-4d",

rowNumber,

columnNumber,

type,

number,

points,

hits);

* See chapter 4 in the Liang book for details on format specifiers

**Output**

Your output will look like the following when running test file **PinballMachineTargets.txt** and **Play.txt**

Set up targets in pinball machine...

Column 0 Column 1 Column 2 Column 3 Column 4

Row 0 ------- ------- Bumper ------- -------

Row 1 ------- Bumper ------- Bumper -------

Row 2 Kicker ------- Rollover ------- Kicker

Row 3 ------- Rollover ------- Rollover -------

Row 4 ------- ------- Rollover ------- -------

Row 5 Rollover Flipper ------- Flipper Rollover

Row 6 ------- ------- ------- ------- -------

Dashes represent an

empty location in

playing field

----------------------------------------------

Simulate Pinball Game

----------------------------------------------

Target Hit ID Points Score

----------------------------------------------

Bumper 1 100 100

Bumper 3 100 200

Bumper 2 100 300

Kicker 2 10 310

Rollover 3 50 360

Rollover 4 10 370

Flipper 1 0 370

Rollover 4 10 380

Bumper 3 100 480

Rollover 1 10 490

Rollover 2 50 540

Rollover 5 50 590

Produced report

Using print method

on TargetReport

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

PINBALL MACHINE TARGET HIT REPORT

(From Most Hits to Least Hits)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Row Column Type Number Points Number Hits

------------------------------------------------------------------------

1 3 Bumper 3 100 2

4 2 Rollover 4 10 2

0 2 Bumper 1 100 1

1 1 Bumper 2 100 1

2 2 Rollover 1 10 1

2 4 Kicker 2 10 1

3 1 Rollover 2 50 1

3 3 Rollover 3 50 1

5 0 Rollover 5 50 1

5 1 Flipper 1 0 1

2 0 Kicker 1 10 0

5 3 Flipper 2 0 0

5 4 Rollover 6 50 0