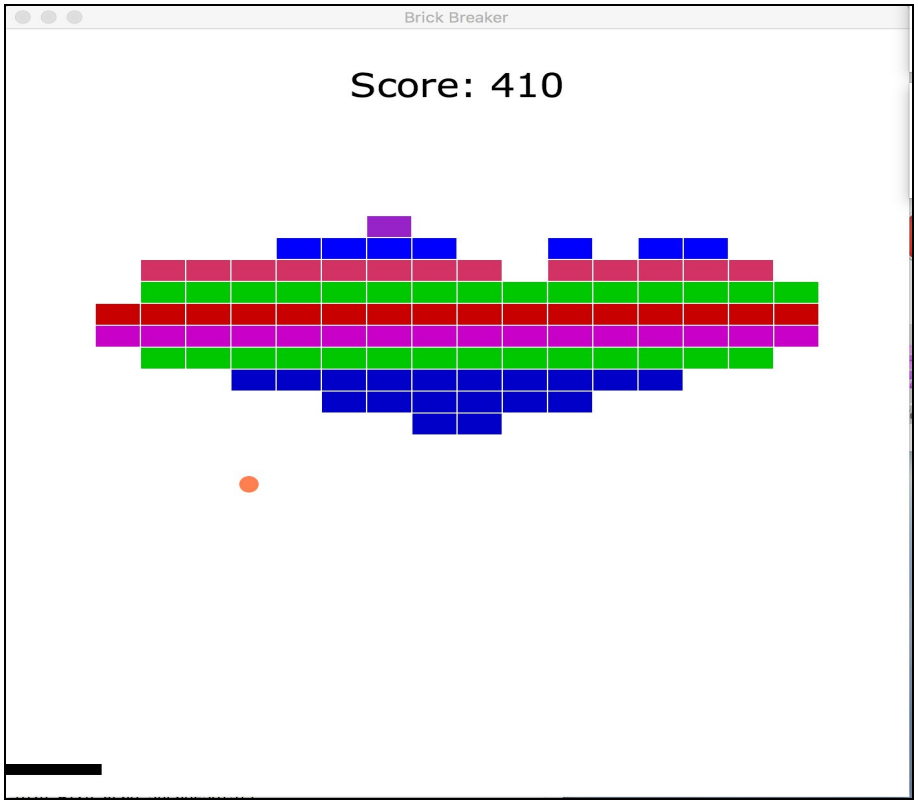


Project 1 - Brick Breaker

Overview

You will complete the code for a game named Brick Breaker for this project. Brick Breaker is similar to the classic console game BreakOut. The game board consists of rows of colored bricks, a paddle located at the bottom of the screen, and a bouncing ball. The object of the game is to use the paddle to keep the ball in play so that you can eliminate all the bricks on the screen. When a ball makes contact with a brick, the brick is eliminated (it disappears). Each brick will have an associated point value that will be added to the player's score when the brick is contacted by the ball. Once all the bricks on the screen are gone, you will level-up to a new set of bricks with more challenging arrangement.

Your task is to implement the classes in the Classes section of the assignment and implement the code to read the game configuration file to populate an array of Level objects (all described below). The Level objects are used to create the bricks on each game level. You will also write the file read and write methods for the player profiles. The player profile tracks how many times a player has played the game, their highest score and information about saved games. The save and load game portion of the game will be extra credit.

Goals

1. Be able to create classes containing encapsulated properties/data fields.
2. Be able to create and effectively use static methods and understand when their use is appropriate.
3. Be able to create modular code using methods.
4. Be able to create classes containing aggregated Array and ArrayList reference types.
5. Be able to use objects via the class contract without viewing the associated source code (Abstraction).
6. Be able to declare and manipulate Arrays and ArrayList using loops..

Concepts

- If-then-else conditionals
- Loops
- Methods
- File I/O
- Exception handling (try/catch for file reading and writing)
- Arrays and/or ArrayList
- Classes and Objects
- Static methods
- Abstraction
- Encapsulation
- UML Class Diagrams

Classes

Implement the classes described by the following UML class diagrams. The classes must be implemented exactly as described or the given code will not work. For example, if you change the case of a method name, then one of my methods that calls this renamed method will not compile.

BrickRow

Models one row of bricks on a level. All bricks on a row have the same color and same point value. Creative level designs are achieved by masking out bricks on a row (set mask value to false) to create different patterns

BrickRow	
-pointValue : int -color : Color -brickmask : int []	Point value for the bricks on this row Color of bricks on this brick row. 0 or 1 for each brick. 1=visible, 0=not visible
+BrickRow() +BrickRow(ptVal : int, clr : Color, bMask : String) +getBrickMaskValue(int:index) : boolean +setBrickMaskValue(index : int, value : boolean) : void +getBrickMaskLength() : int +toString() : String //Include getter and setters for primitive data types	Initialize all properties to their Java default values. bMask should contain char '0 for false, char '1' for true. Return the brick mask value at location index in brickMask. Set the mask value at location index to value. Return the length of the brick mask. See note below.

BrickRow Notes:

toString() : The format of the output of the toString method for the the Level class should be:

Brick row number

R G B color values (0..255) separated by one space

Bricks mask values (0=false, 1=true) i.e. mirror the input file format.

Level

Models a single game level. This is equivalent to a new set of bricks (i.e. an array of BrickRow objects).

Level	
-levelNum : int -brickRows : BrickRow[]	Numeric label for the level
+Level() +Level(levelNum : int, numBrickRows : int) +getBrickRow(index : int) : BrickRow +setBrickRow(index : int, bkRow : BrickRow) :void +toString() //getter and setter methods for primitive data types.	Set properties to Java defaults. Make sure you instantiate brickRows. Return the brickRow object at location index. Set value at location index to bkRow. See note below

Level Notes:

toString() : The format of the output of the toString method for the the Level class should be:

Level Number

Number of BrickRows in this level

$BrickRow_0$

$BrickRow_1$

$BrickRow_{n-1}$

Note: Each BrickRow is added to the string using the format described in the BrickRow section.

PlayerProfile

Encapsulates the data for a single player.

PlayerProfile	
-name : String -numGamesPlayed : int -highScore : int -savedGames : ArrayList<String>	Player gaming name Current number of games played Current high score List of saved games file names

+PlayerProfile()
+PlayerProfile(name : String)
+toString() : String
//See Handling ArrayLists section below
//Getter and Setter methods for primitives
	Constructors
	
	See explanation below.

PlayerProfile Notes:

toString() : The format of the output of the toString method for the the Level class should be:

Player Name
Number of games played
Player's highest score
Number of saved game files
Saved game file 0
Saved game file 1
...
Saved game file n-1

GameProfiles

Encapsulates the player data for all registered players.

GameProfiles	
-selectedProfile : PlayerProfile -highGameProfile : PlayerProfile -profiles : ArrayList<PlayerProfiles>	Reference to the profile that was selected at game start up. Reference to the player profile with highest score. ArrayList of PlayerProfiles
+GameProfiles() +getSelectedProfile() : PlayerProfile +getHighGameProfile : PlayerProfile +setHighGameProfile(PlayerProfile prof) :void //See Handling ArrayLists Section below +toString() : String	 Return a reference to the selected PlayerProfile. Return a reference to profile with highest score. Set the highGameProfile equal to the argument. Creates a string with each player profiles starting on a new line.

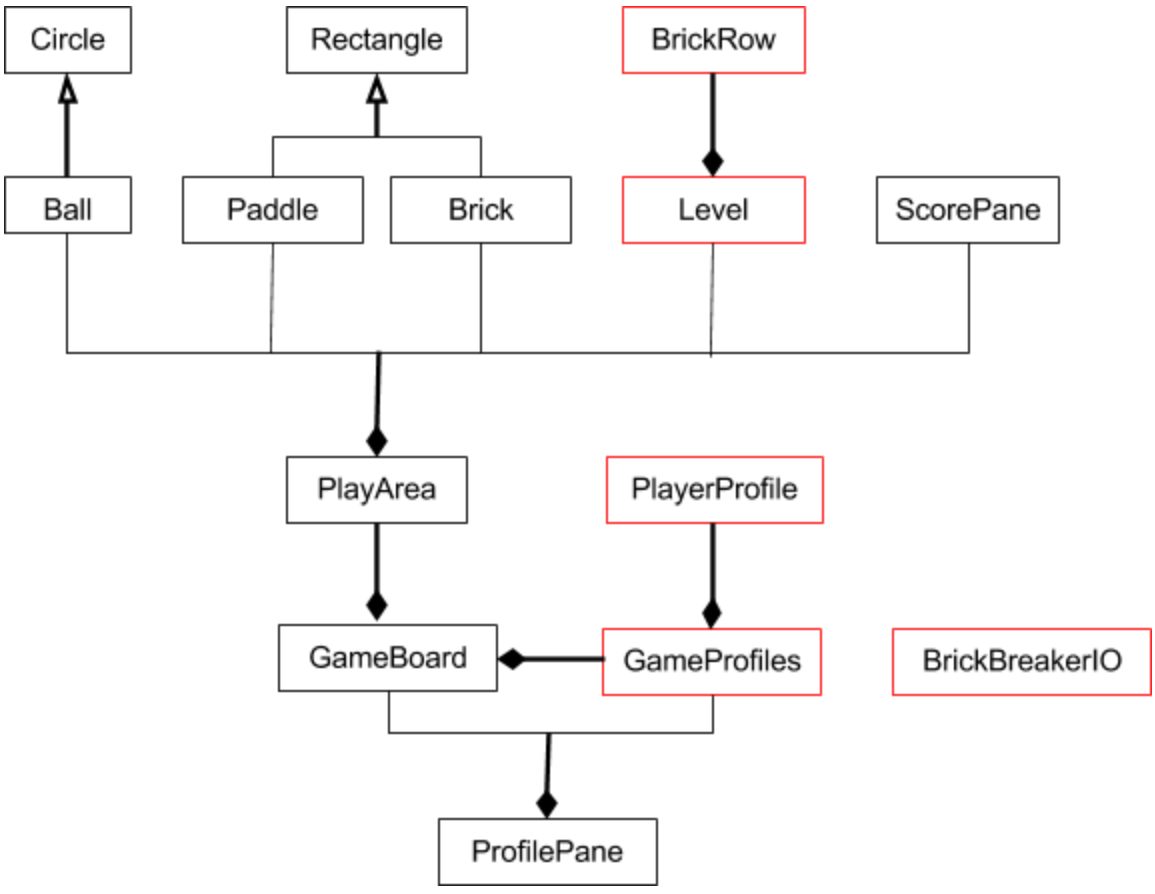
BrickBreakerIO

Used to aggregate all the file manipulation methods used in this application. Each of the implemented methods are static.

BrickBreakerIO	
+readConfigFile(cFileName : String) : void +readProfiles(gmProf : GameProfiles, pFileName : String) : void +writeProfiles(gmProf : GameProfiles, pFileName : String) : void +saveGame() +loadGame()	 implement later for extra credit implement later for extra credit

UML Domain Model

The following UML Domain Model is provided to help you understand how the above classes fit within the scope of the entire Brick Breaker application. Student implemented classes appear in the red boxes.



Milestones / Implementation Roadmap

Level 1

Implement the BrickRow and Level classes as described in the UML Class Diagrams in the Classes section. Follow these steps:

1. Create a new project named BrickBreakerStudent.
2. Netbeans will automatically create a brickbreakerstudent package for you; however, if you are using another IDE, create a package named brickbreakerstudent in your project directory.
3. Create your BrickRow and Level classes in the brickbreakerstudent package.
4. If your IDE does not create a class with a main method when it creates your project, create a class named BrickBreakerStudent and add the main() method. The main method will be used to instantiate and test the BrickRow and Level classes. Test your classes by first instantiating BrickRow and Level objects, then use the setter methods to change the default property values. Call the class toString() methods to validate the property values.

Level 2

Implement and test the BrickBreakerIO readConfigFile static method.

1. Create the BrickBreakerIO class in your brickbreakerstudent package.
2. Add the readConfigFile static method to the class. Refer to the File Formats section below for specifications of the configuration file format.
3. In Netbeans you can specify the directory where JVM will search for files without absolute pathnames (working directory) by selecting Run->Set Project Configuration->Customize. Click the Browse button next to Working Directory and navigate to the directory containing your Brick Breaker configuration file.
4. Modify your driver main method so that it will store the array of Level objects returned from the method. Next, the main method should call the toString() method for each Level object in the array. Verify that the output of the driver method corresponds with the configuration file input.

Level 3

Implement the PlayerProfile and GameProfiles classes.

1. Create the PlayerProfile and GameProfiles classes in the brickbreakerstudent package. Both of these classes have ArrayList properties. Do not create the same

get/set methods for these properties that you used on the other primitive properties. See the Handling ArrayLists section below for specifications of the methods that should be created.

2. Once you have created the classes, instantiate a few PlayerProfile objects, add new values using the class mutator methods and then add them to a GameProfiles object. You can call the class toString() methods to validate the data that you added to them.

Handling ArrayLists

Each ArrayList should have five associated methods to perform: getNum, add, get and set. So if you have an ArrayList named widgets that stored items of type Widget, then the associated UML would be:

```
+getNumWidgets() : int                          /Return the number of items in the ArrayList widgets.
+getWidget(index:int) : Widget                  //get the Widget at location index in ArrayList widgets
+setWidget(index:int, item:Widget):void         //store item at location index in the ArrayList widgets.
+addWidget(item:Widget):void                    //Append the Widget to the ArrayList.
+deleteWidget(index:int) : Widget               //Remove and return the Widget at location index
```

Level 4

Implement the static readProfiles and writeProfiles methods.

1. Create the readProfiles and writeProfiles static methods in the BrickBreakerIO class. See the File Formats section below for the specification of profile file format.
2. Update your main so that it calls the readProfiles method to read in the sample profiles file.
3. Call the GameProfiles toString method again to confirm that the contents of the profile file were input correctly.
4. Modify the driver method so that it reads in a profiles file, creates a new profile and adds the new profile to the GameProfiles object. You can also modify some of the properties of the objects read in from the file.
5. Output the augmented GameProfiles object using the writeProfiles method and verify the information is correct and matches the format of the profiles file format.

Level 5

At this point, you are ready to integrate your code with the given Brick Breaker code.

1. If you are using an IDE other than Netbeans, make sure that the classes you implemented are compiled into a single jar file. The class with the main method is not needed inside of your jar file.

2. Create a new project (name it BrickBreaker) and copy the given Java source file (BrickBreaker.java) into the project source directory. Link in the BrickBreakerStudent and the given BrickBreakerEngine jar files by right clicking the Libraries folder in your project directory. Set the default direct path to point to the directory containing your configuration and profile input files.
3. The game should now execute.

File Formats

Configuration File Format

Number of levels
Level #
Number of brick rows for this level
Brick row # (use 0 for the first one)
Color R G B values (0..255)
Brick Mask (0=brick visible, 1=brick not visible)
Brick row # (use 0 for the first one)
Color R G B values (0..255)
Brick Mask (0=brick visible, 1=brick not visible)
…
Level #
Number of brick rows for this level
Brick row # (use 0 for the first one)
Color R G B values (0..255)
Brick Mask (0=brick visible, 1=brick not visible)
Brick row # (use 0 for the first one)
Color R G B values (0..255)
Brick Mask (0=brick visible, 1=brick not visible)
…

Configuration File Example

1	The file will contain information about 1 level
0	The level number.
12	The level will contain 12 rows of bricks
0	The first brick row number
0 0 200	The RGB values for the brick row color
10	Each brick on the row will be worth 10 points
00000000011000000000	The brick mask. This row will only have 2 visible bricks.
1	The start of the second brick row (brick row 1)
0 20 200	
10	
00000001111110000000	

…
The other brick rows will follow the same format so they are omitted here.

Profile File Format

Player name 0
Number of games played
High game
Number of saved files
Saved file 0
Saved file 1
...
Player name 1
Number of games played
High game
Number of saved files
Saved file 0
Saved file 1
...
Player name n-1
Number of games played
High game
Number of saved files
Saved file 0
Saved file 1
...

Save File Format

Add later for more advanced asskgnment

Submission

Complete this section later using your university specific instructions.