2048, Part 1

This is a two-part project. You are going to write the popular game 2048, a numbers game that involves sliding tiles. This project will also be a good example of MVC, or the Model-View-Controller design pattern.

The game 2048 takes place on a two-dimensional 4x4 grid. When the game starts, there are two 2s randomly placed on the grid.

```
   2   2   -   -
   -   -   -   -
   -   -   -   -
   -   -   -   -
Move: a
```

The user is prompted to enter a move direction using w (up), a (left), s (down), or d (right). All numbered pieces slide in that direction, taking up all available spaces. Tiles of the same number will combine to form their sum.

```
Current score: 4
   4   -   -   -
   -   -   -   -
   -   -   -   4
   -   -   -   -
Move: a
```

Whenever a move is made, a new piece appears up on the board. 10% of the time it's a 4, and 90% of the time it's a 2.

```
Current score: 4
   4   -   -   -
   -   -   -   -
   4   -   -   2
   -   -   -   -
Move: w

Current score: 12
   8   -   -   2
   -   -   -   -
   -   -   2   -
   -   -   -   -
```

When pieces combine, their sum is added to the running score. The game is over when there are no more spaces left on the board, and nothing more can be combined.

To start out, you will create three classes:

`j2048Controller.java`

Keeps track of the board and creates instances of the other two classes. The Controller moves data between the Model and the View. For example, when the score is printed to the screen, the Controller calls the View's `printScore(int score)` method, passing in the Model's `getScore()` method as an argument.

`j2048Model.java`

This class initializes the pieces and also spawns a new piece every time the board is moved. The Model also moves the pieces on the board and determines if the game has been won.

`j2048View.java`

This class interacts with the user by printing the board to the screen along with the current score.

Just use a single-dimensional array of four elements for now, and just handle moving left and moving right. Accept a single key using Scanner and make sure to only accept a, d, and q to exit the game. Print the board to the screen after every move. Don't worry about keeping track of the score for now.

In your Controller, create the main method. Declare a variable called `DIM` and initialize it to 4. Create a single-dimensional int array of `DIM` elements for the board, and use `DIM` everywhere in your program instead of 4. If you are consistent, when you are done with the project you should be able to change the value of `DIM` in your Controller and play 2048 with a board of 6 or even 8 rows and columns, simply by changing one number. The Controller keeps track of the board and passes it as an argument to the Model and View as necessary.

In the Model, declare two instance variables: one for the score, and one for the dimension, called `DIM`. Create two constructors: one that takes a parameter and sets `DIM` to that value, and a default constructor that takes no parameters, and calls the other constructor passing in 4 as an argument.

Next, create the following methods:

```
// Fills board with two 2s in random locations.
public void init(int[] board)

// Iterates over each element of board and print its value to the
// screen. If the value is 0, print a "-" instead.
public void draw(int[] board)
```

```
// Spawns a new value in an empty location in the board.
// 90% of the time, it should be a 2.
// 10% of the time, it should be a 4.
public void spawn(int[] board)
```

In the View, declare an instance variable called `DIM`. Create two constructors: one that takes a parameter and sets `DIM` to that value, and a default constructor that takes no parameters, and calls the other constructor passing in 4 as an argument.

Then create a `draw()` method that takes the board as a parameter, iterates through the array, and draws the board:

```
// Draws the board.
public void draw(int[] board)
```

You can use a String formatter to format your numbers so they always take up the same number of characters in the grid, to keep everything aligned. The % is a placeholder and the 4 indicates we always want it to take up 4 places (since the numbers can get to be 4 places if they are over 1000. The 'c' means 'char' and 'd' means integer.

```
System.out.print("%4c", '-');
System.out.print("%4d",board[x]);
```

After every move, call `spawn()` to create a new number somewhere in the array.

Remember that when you pass in an array to a method, it is passed by reference. That means that changes you make to the array inside the method change the original array. You should not be returning arrays from your methods.

One error you will probably run into at some point during this project is the `IndexOutOfBounds` error. This occurs when you try to access an element that is outside the bounds of the array. For example, accessing `board[4]` will result in this error because the index of the last element in an array is always one less than its length; in this case, 3. If you run into this error check the boundaries of your loop headers. If you are accessing `i + 1` in your loop, then you want to make sure the highest possible value for `i` is `board[DIM] - 2`.

Here are some examples of what should happen with each move:

Before	Move	After
2 2 - -	left	4 - - -
- - 2 2	right	- - - 4
2 – 2 -	left	4 - - -

2 – - 2	right	- - - 4
4 2 - 2	left	4 4 - -
2 4 - 2	left	2 4 2 -
4 2 2 -	left	4 4 - -
4 2 2 -	right	- - 4 4
2 2 2 -	left	4 2 - -
2 2 2 -	right	- - 2 4
4 4 8 8	left	8 16 - -
4 4 8 8	right	- - 8 16