In the game of Memory, a deck of cards consists of several card pairs that have the same image. Cards are laid on the table face down in a grid and 2 players alternate in turning over two cards with the goal of finding a matching pair. The player with the most pairs wins the game. More information at https://en.wikipedia.org/wiki/Concentration_%28game%29

You are creating the game that will eventually allow 2 players to play a text-based version of Memory. In this assignment, you are to write a section of the program that will prepare the playing grid for a new game by randomly placing cards on the grid and printing the position of all the cards (face up).

The game uses the first 8 letters of the alphabet instead of images for the card faces, with same letters making up the matching pair. So 'A' and 'A' are a matching pair, as well as 'B' and 'B', etc. The game grid is 4 cards by 4 cards with 16 cards total, consisting of 8 matching pairs.

There are several approaches of varying efficiency to solve this problem. Although not very efficient, one way to accomplish this task is to create a character array `deck` that holds the letters that make up the playing card faces. For a 4 by 4 grid, the deck holds 8 pairs of letters: 'A','A','B','B','C','C','D','D','E','E','F','F','G','G','H','H' (a total of 16 characters).

Then you declare a 2-dimensional array `grid` that will keep track of which letters are in which positions on the grid. The program loops though all grid positions and for each position it draws a random letter from the deck until the grid is filled. A card can be drawn from this virtual deck only once. You have to keep track of which positions in the deck have already been drawn by using another array `cardDrawn` so you do not draw the same card more than once.

To help with debugging and to visually inspect that the cards are correctly drawn from the deck, you print out the deck content before every attempt to draw a card. This way you can see how the deck is depleted. In order to accommodate this visualization, you will replace the drawn letter from the deck with a space . This part of the code would eventually get deleted or commented out but for now it can provide a useful view into the drawing routine.

Finally, after the grid is populated, you print it to the screen.

Requirements:

1. Create a new Netbeans project.
2. Define a constant for the size of the grid, `gridSize = 4`.
3. Declare and initialize arrays that will hold the necessary data:
    a. Declare and initialize a character array `deck` to hold the card letters. Use the {} initialization with literal values:
    char[] deck={'A','A','B','B','C','C','D','D','E','E','F','F','G','G','H','H'};

   b. Declare a boolean array `cardDrawn` that has the same size as the deck array and initialize its elements to `false`.
   c. Declare a 2-dimensional character array `grid` of appropriate size using the `gridSize` constant from step 2. The grid is a square with the side equal to `gridSize`.

4. Initialize the `grid` array by randomly choosing a card from the `deck` and assigning its value to the `grid` array elements. For each element, use a random number generator to pick a card from the deck. Then check if this card has been picked before, if so, repeat drawing the card from the deck. If the card has not been picked before, mark it in the `cardDrawn` array as taken, and assign the value of the card to the grid element.
   a. Set up an outside loop to iterate through the grid rows.
   b. Set up an inner loop to iterate through the grid columns.
   c. Set up a while loop to draw of the cards from the deck .
   d. Within the while loop, print out the content of the deck to the screen. This is only for debugging purposes and would be deleted from the production version of the program.
   e. Within the while loop, use a Random class to choose a random number.
   Hint: Here is an example of using the Random class:

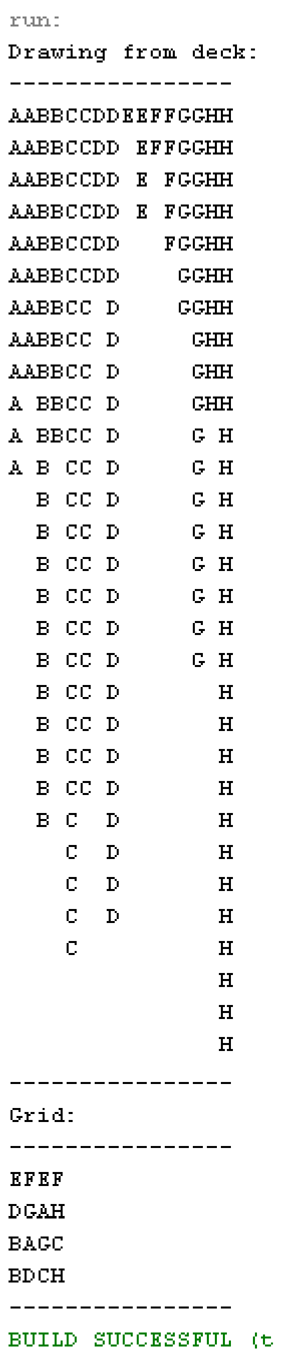
```
import java.util.Random;          //import the class as we do
                                  //with Scanner

Random myRandom=new Random();     //declare the class
                                  //variable myRandom

int x = myRandom.nextInt(max);    //use the class variable's
                                  //method nextInt(max)
                                  //where range is an integer
                                  //so that x will be between
                                  //0 inclusive and max
                                  //exclusive
```

   f. After you generate the random number, use it as an index into the `deck` and `cardDrawn` arrays. Check if the card has been drawn already before. If so, repeat the draw, otherwise assign the card value to the grid element and mark the card as already drawn. For debugging and visualization, set the card value equal to space so the print above would show which card was drawn. Finally, exit the while loop to continue with another element.

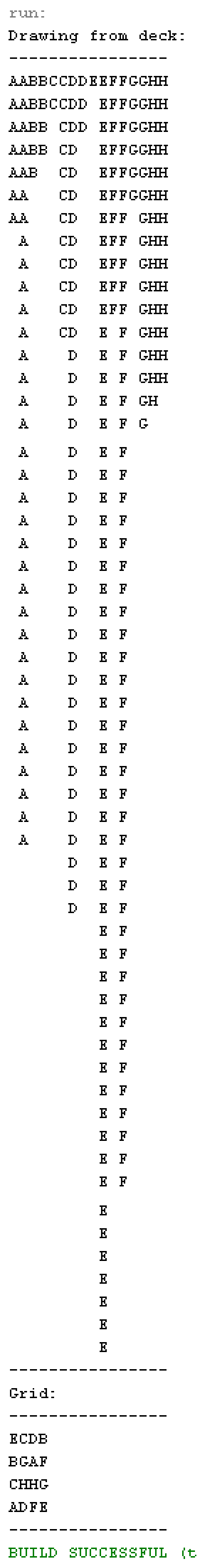
5. Print a separator line ("--------------") and print the grid to the screen.

Sample Output:

Note: This program uses a random number generator, so your results will be from those shown here. Use the sample output to get a general idea what is required and how your output should look like.



```
run:
Drawing from deck:
----------------
AABBCCDDEEFFGGHH
AABBCCDD EFFGGHH
AABBCCDD E FGGHH
AABBCCDD E FGGHH
AABBCCDD   FGGHH
AABBCCDD    GGHH
AABBCC D    GGHH
AABBCC D     GHH
AABBCC D     GHH
A BBCC D     GHH
A BBCC D     G H
A B CC D     G H
  B CC D     G H
  B CC D     G H
  B CC D     G H
  B CC D     G H
  B CC D     G H
  B CC D     G H
  B CC D       H
  B CC D       H
  B CC D       H
  B CC D       H
  B C  D       H
    C  D       H
    C  D       H
    C  D       H
    C          H
               H
               H
               H
----------------
Grid:
----------------
EFEF
DGAH
BAGC
BDCH
----------------
BUILD SUCCESSFUL (t
```



```
run:
Drawing from deck:
----------------
AABBCCDDEEFFGGHH
AABBCCDD EFFGGHH
AABB CDD EFFGGHH
AABB CD  EFFGGHH
AAB  CD  EFFGGHH
AA   CD  EFFGGHH
AA   CD  EFF GHH
 A   CD  EFF GHH
 A   CD  EFF GHH
 A   CD  EFF GHH
 A   CD  EFF GHH
 A   CD  E F GHH
 A    D  E F GHH
 A    D  E F GHH
 A    D  E F GH
 A    D  E F G
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
 A    D  E F
      D  E F
      D  E F
      D  E F
         E F
         E F
         E F
         E F
         E F
         E F
         E F
         E F
         E F
         E F
         E F
         E F
         E
         E
         E
         E
         E
         E
         E
----------------
Grid:
----------------
ECDB
BGAF
CHHG
ADFE
----------------
BUILD SUCCESSFUL (t
```