Follow the instructions:

- Design your own defined package, which holds desired abstract classes and interfaces.
- Define your own defined Exceptions that should also be keep it in the package.
- Raise suitable exceptions when particular condition occurs. Use Multi-Threading wherever it requires in the given problems.
- Usage of accessor and mutator methods must in your code. Provide appropriate documentation section
- Must and should demonstrate the program as per the given criteria's only. Moreover, make your code in the complete object-oriented model.

1. (Game: *Hangman*) Write a hangman game that randomly generates a word and prompts the user to guess one letter at a time, as shown in the sample run. Each letter in the word is displayed as an asterisk. When the user makes a correct guess, the actual letter is then displayed. When the user finishes a word, display the number of misses and ask the user whether to continue to play with another word. If user fails to guess atleast a letter of a word for first 3 attempts then appropriate Exception should be generated and also display the actual word as an output on the screen. Here is a sample run:

```
(Guess) Enter a letter in word ******* > p
(Guess) Enter a letter in word p****** > r
(Guess) Enter a letter in word pr**r** > p
     p is already in the word
(Guess) Enter a letter in word pr**r** > o
(Guess) Enter a letter in word pro*r** > g
(Guess) Enter a letter in word progr** > n
     n is not in the word
(Guess) Enter a letter in word progr** > m
(Guess) Enter a letter in word progr*m > a
The word is program. You missed 1 time
Do you want to guess another word? Enter y or n>
```

2. (*Game:* ***connect four***) Connect four is a two-player board game in which the players alternately drop colored disks into a seven-column, six-row vertically suspended grid, as shown below. The objective of the game is to connect four same-colored disks in a row, a column, or a diagonal before your opponent can do likewise. The program prompts two players to drop a red or yellow disk alternately. In the preceding figure, the red disk is shown in a dark color and the yellow in a light color. Whenever a disk is dropped, the program redisplays the board on the console and determines the status of the game (win, draw, or continue).

Screen 1

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
---------------
Drop a red disk at column (0–6): 0

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | | | | | |
---------------
```

Screen 2

```
Drop a yellow disk at column (0–6): 3

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | |Y| | | |

. . .
. . .
. . .
```

Final Screen after completion of game

```
Drop a yellow disk at column (0–6): 6

| | | | | | | |
| | | | | | | |
| | | |R| | | |
| | | |Y|R|Y| |
| | |R|Y|Y|Y|Y|
|R|Y|R|Y|R|R|R|
---------------
The yellow player won
```

3. (Game: *Craps*) Craps is a popular dice game played in casinos. Write a program to play a variation of the game, as follows: Roll two dice, each dice has six faces representing values 1, 2… and 6 respectively. Check the sum of the two dice. If the sum is two, three, or twelve (called craps), you lose; if the sum is 7 or 11 (called natural), you win; if the sum is another value (i.e., 4, 5, 6, 8, 9, or 10), a point is established. Either continue to the roll the dice until a seven or the same point value is rolled. If seven is rolled you lose, otherwise you win. Your program acts as a single player.

4. (Game: ***Dot-To-Dot***) Connect two points, pair all points and cover the entire board. In order to implement this game follow the rules as follows: Take the board, as a **grid,** which has different size. Read the board size in words, i.e., if '*n*' as input then the board size becomes *n*X*n*. Now, ask the user for the **number of points** '*p*' (use different symbols like may be an alphabet, number or any special symbol as a **point** representation and depends upon the number of points select those many special symbols like + . - * etc., to indicate the connected pairs) to keep it on the board. After, reading the points, then '*p*' points should inserted at random positions on the board. Here, **the computer acts as a single player**. It has to display, the pairing points on the board and should cover the entire board. Calculate and Display how much a pair is covering on the board. Then at last display the message as "Game over". (To calculate the **percentage** covered by a pair, use the following formulae: ((number of empty spaces covered by a pair + 2) / total board size))*100)

   Here is a sample assumed output: (Show the board with connected pairs of points)
   Enter the size of the grid: **four**
   Enter the number of points on the board**: two**
   Percentage covered by **point-1** (for X): ((8+2) / 16)*100) → **62.5**
   Percentage covered by **point-2** (for Z): ((4+2) / 16)*100) → **37.5**

```
X . . .
+ Z . .
+ + . X
Z + . .
```

5. (*Game:* ***connect-five***) Connect five is a four-player board game in which the players alternately drop colored disks into a nine-column, nine-row vertically suspended grid, as shown below. The objective of the game is to connect five same-colored disks in a row, a column, or a diagonal before your opponent can do likewise. The program prompts four players to drop a **red**, **yellow**, **blue,** or **green** disk alternately. Whenever a disk is dropped, the program re-displays the board on the console and determines the status of the game (win, draw, or continue).

   Here is a **sample assumed** output**:**

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
---------------
Drop a red disk at column (0-6): 0
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
|R| | | | | | |
---------------
```

```
Drop a yellow disk at column (0-6): 6
| | | | | | | |
| | | | | | | |
| | | |R| | | |
| | | |Y|R|Y| |
| | |R|Y|Y|Y|Y|
|R|Y|R|Y|R|R|R|
---------------
The yellow player won
```

6. (*Game: **Frame-Box***) Frame-Box is a two-player board game in which the players alternately drop colored disks into a ten-column, ten-row vertically suspended grid, as shown below. The objective of the game is to connect same-colored disks like a square-box (nXn), diamond (nXn) before your opponent can do likewise. The program prompts two players to drop a **red** or **yellow** disk alternately. Whenever a disk is dropped, the program re-displays the board on the console and determines the status of the game (win, draw, or continue).

Here is a **sample assumed** output**:**

	R	**R**	**R**	Y	
	R	Y	**R**	Y	
R	**R**	**R**	**R**	R	Y
Y	R	Y	Y	Y	Y
Y	R	R	R	R	

Player **R** has won the game

7. (*Game***: *HiDoKu***) Sometimes known as hidoku or hidato, you are given a grid with just a few numbers filled in. You must then build an ascending sequence of numbers going sideways or diagonally. In order to implement this game, the requirements as follows:

   i. The grid size should read as an input. i.e., if '*n*' is an input value, then the grid size is *n*X*n*.

   ii. Number of elements to be filled in the grid is also an input, but here it is random value and should be less than *n*X*n*, i.e., if '*k'* is an input then, from one to '*k*' are the input values should be filled in that grid. The values should up to '*k*' should be filled in that grid randomly.

   Display the array positions as an output, which shows the ascending sequence of numbers going sideways or diagonally.

8. (*Game:* ***connect four***) Connect four is a two-player board game in which the players alternately drop colored disks into a seven-column, six-row vertically suspended grid, as shown below. The objective of the game is to connect four same-colored disks in a row, a column, or a diagonal before your opponent can do likewise. The program prompts two players to drop a red or yellow disk alternately. In the preceding figure, the red disk is shown in a dark color and the yellow in a light color. Whenever a disk is dropped, the program redisplays the board on the console and determines the status of the game (win, draw, or continue).

Here player-1 is yourself, and other player is Computer. Implement this program using multithreading. Use appropriate methods of Thread class to depict this game.