Thera Bytes Exercises – Coding

Simple Questions

1. Which value has result?

```
bool result = (myBool) ? !myBool : myBool;
```

myBool	true	false
result	false	false

=> always false

2. How does the header of the class definition look like?

```
MyClass<int> myInstance = new MyClass<int>();
```

```
public class MyClass<T> {
    //....
}
```

This is C# but you may answer with any language where this kind of syntax is valid.

By the way: how is this syntax called?

myInstance is a integer Object of the generic class MyClass

3. What is the name of the following pattern:

```
public class MyClass
{
    private static MyClass instance;

    public static MyClass Instance
    {
        get
        {
            if (instance == null)
            {
                instance = new MyClass();
            }

            return instance;
        }
    }

    private MyClass() { }
}
```

It's the singleton pattern; meaning there is a single globally accessible (static) instance of the class MyClass

4. What are the results of the following operation (binary and decimal)?

I assume we should transform the decimal to binary, then use the binary operation and transform the result back to decimal:

```
12 AND 55
12 OR 55
12 XOR 55
```

```
12 AND 55:          12 OR 55:           12 XOR 55:
      0b110111            0b110111            0b110111
AND   0b001100      OR    0b001100      XOR   0b001100
------------------  ------------------  ------------------
      0b000100            0b111111            0b111011
= 4                 = 63                = 59
```

5. What is the Problem with the following code?

```
List<string> list = new List<string> { "mage", "warrior" };
for (int i = 0; i < list.Count; i++)
{
    list.Add(list[i].ToLower());
}
```

this loops infinitely since you increase the count of the list while iterating through it, thus you never reach the break condition

6. Can you find problems with this code? How to fix the problems?

```
public class MyClass
{
    public object Object;
    public event Action<object> ObjectReleasedCallback;

    public void AssignObject(object value)
    {
        Object = value;
    }

    public void ReleaseObject()
    {
        ObjectReleasedCallback(Object);
        Object = null;
    }
}
```

The Object variable isn't explicitly initialized and you could actually call ReleaseObject() before AssignObject was called. Thus the ReleaseObject() call would be done on an none-object (probably resulting in an error).

Additionally, from what I understand, Actions are containers for a function taking a single argument of type T and having no return type. The ObjectReleasedCallback never gets initialized with such a function.

Solution: Create a MyClass constructer which requires both an object and an Action<object>. Then the default constructer, which would lead to the issues above, won't be generated (source: https://stackoverflow.com/a/2875653)

7. Complete the story (two possible answers – please explain your thoughts)

Three programmers have died due to a deadlock. At heaven's gate Petrus asks: "Do all three of you want to go to heaven?"

The first programmer says: „I don't know. "

The second programmer says: „I don't know. "

The third programmer says: „<your answer> "

Okay, the overall issue with deadlocks is that multiple instances wait for each other. In this case, the first two programmers again go into a waiting state at heaven's gate even though that's the reason they died in the first place. If the third programmer now also answers "I don't know", they again are in a deadlock. As each of them waits for another programmer to decide first. If the third programmer just says "yes", no deadlock should occur and they all reach heaven.

Let's Code: Implement a Skill Tree

About Skill Trees

A skill tree is often used in role playing games or strategy games to allow the player more powerful actions after playing a while. At a certain point in the game the player can select a skill of his or her choice to unlock that skill if possible. Skills are connected like branches of a tree and a skill can only be unlocked if the previous skill on that tree is unlocked already.

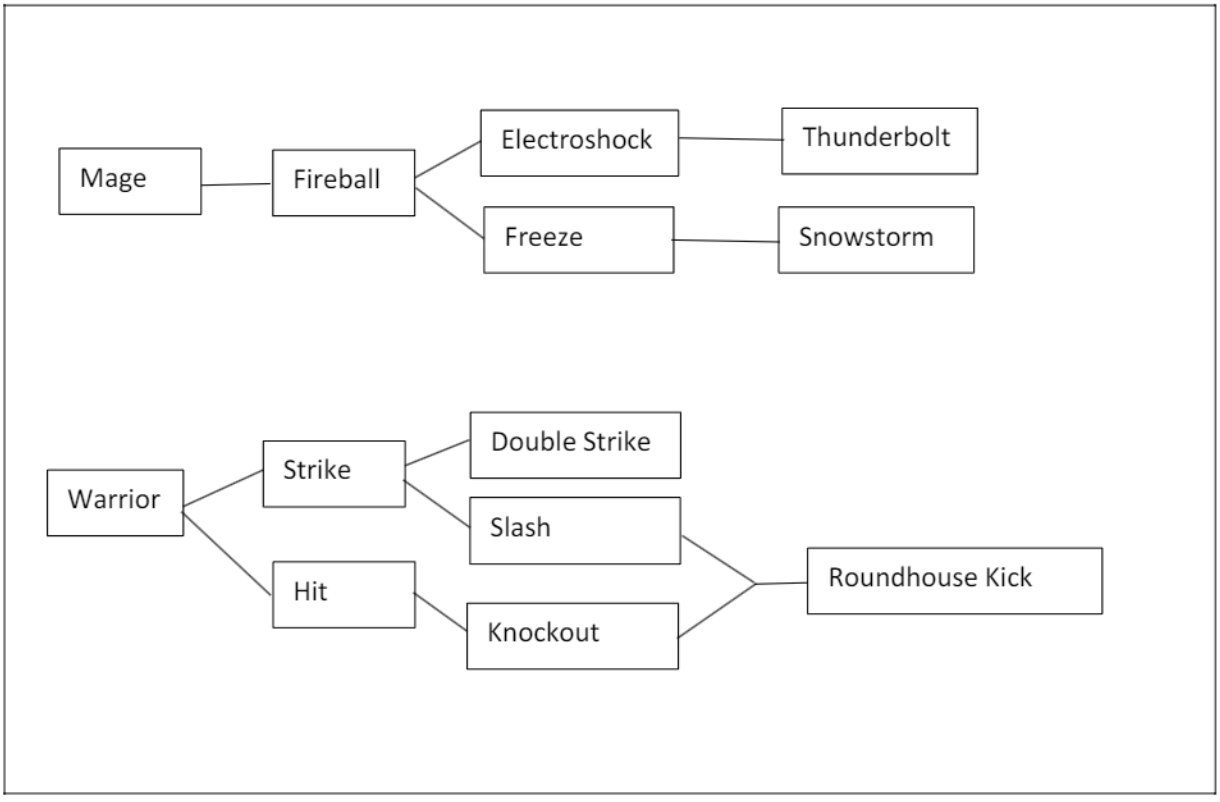
Actually, some Skill Trees are not trees but graphs. This means that some skills depend on more than one previous skill. They can only be unlocked if all of the previous skills are unlocked already.

Exercise description

Create a program which can handle a skill graph (skill tree with more than one skill dependency).

Each node of the graph should provide the following information:

- **Name**: The name of the node
- **IsLocked**: true if not available yet; false if the Node has been unlocked
- **CanBeUnlocked**: true if all linked previous nodes are unlocked and this node is still locked



Implement the skill trees of the diagram above.

There is no need for visualization of the tree. But it should be possible to see with the debugger if it works as described.

Let‘s Code: Reversi

Background: About the game Reversi

Reversi (a.k.a Othello) is a game for two players on a square tiled board.
Each player puts a disc on an empty tile in every of his / her turns. The disc is red on one side and black on the other. The players always turn their respective color facing up when placing a disc.

If a player puts a disc next to a series of opponent discs and on the other side of that series is again a disc of the active player, the enclosed tokens are turned around and taken over.

Any direction for overtaking is valid, including diagonals.
It is also possible to take over more than one series of discs with one placed disc.

This is all you have to know about Reversi for this exercise.

Exercise Description

Your solution will receive a state of a Reversi game in form of a string. You are the active player. You have to find the tile where the greatest amount of discs can be taken over, if the next disc would be placed at that tile.

Return the position of that tile.
If there is more than one position with the same amount of discs to take over, return any one of them.

The string you will receive contains the game board.
The first line of the string contains the width followed by the height of the board. These values are separated by a space character.
Each further line of the string corresponds to a line of the board.
Every second character in a line represents one tile and its content:

- An empty tile is represented by a dot: '.'
- A disc owned by the opponent is represented by an 'O'
- A disc owned by the active player (you) is represented by an 'X'
- There is a space ' ' behind every tile-character which can be ignored

Unlike in the original game, the dimension of the game board can differ:

- 3 < width <= 26
- 0 < height <= 26

The colums of the board are counted with ongoing uppercase Letters (A, B, C, ...).
The rows of the board are counted with numbers starting with 1 (1, 2, 3, ...).

For the output value you have to use this counting, specifying the column first with the letter followed by the row with a number.

Examples:

- *For the tile at the very top left, pass "A1"*
- *For the tile third from the left, second from the top, pass "C2"*

Project Setup

If you choose to use C# as programming language, add the following code to your project as entry point for your solution (if you use a different language, make it similar to that code):

```
public class Solution
{
    public static string PlaceToken(string board)
    {

    }
}
```

To test your solution call „Solution.PlaceToken()“ from the Main method of your program and pass a string representing the board you want to test.

You can copy the following code to your main function (C#) to test your program:

```
public static void Main(String[] args)
{
    // 1. Correct Answer: "E1"
    string board1 = @"5 1
X O O O . ";

    string result1 = Solution.PlaceToken(board1);
    Console.WriteLine("board 1: " + result1);

    // 2. Correct Answer: "B2"
    string board2 = @"8 7
. . . . . . . .
. . . . . . . .
. . O . . . . .
. . . O X . . .
. . . X O O . .
. . . . . X . .
. . . . . . X . ";

    string result2 = Solution.PlaceToken(board2);
    Console.WriteLine("board 2: " + result2);

    // 3. Correct Answer: "D3", "C4", "F5", "E6"
    string board3 = @"8 8
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . O X . . .
. . . X O . . .
. . . . . . . .
. . . . . . . .
. . . . . . . . ";

    string result3 = Solution.PlaceToken(board3);
    Console.WriteLine("board 3: " + result3);

    // 4. Correct Answer: "D6
    string board4 = @"7 6
. . . . . . .
. . . O . O .
X O O X O X X
. O X X X O X
. X O O O . X
. . . . . . . ";
```

```
    string result4 = Solution.PlaceToken(board4);
    Console.WriteLine("board 4: " + result4);
}
```

Please do not change the input string (board1, …, board4). There are new line characters added after each line because of the @-sign in front of the string.