Task – Tower of Hanoi

The Tower of Hanoi is a classic mathematical puzzle or game that was invented by the French mathematician Édouard Lucas in 1883. The game consists of three towers, and a number of disks of different sizes, which can slide onto any tower. The puzzle starts with the disks in a neat stack in ascending order of size on one tower, the smallest at the top, making a conical shape.

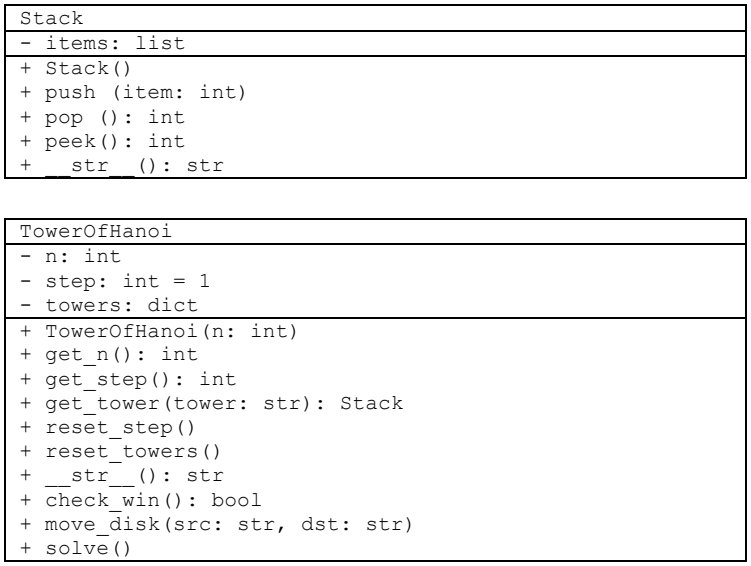
Objective:

The objective of the puzzle is to move the entire stack to another tower, following these simple rules:

- Only one disk can be moved at a time.
- Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or an empty tower.
- Disk can only be placed on top of another disk of a larger size, or an empty tower.

The challenge is to accomplish the objective in the fewest possible moves. As the number of disks increases, the problem becomes exponentially more complex.

Below is an UML class diagram to create a Tower of Hanoi game.



Implement the classes based on the following descriptions. Lastly, create a text-based user game menu to facilitate user interaction:

[You may assume that all inputs are valid]

```
Please choose one of the following options:
1. Start/Reset a new game
2. Move a disk
3. Auto-solve game
4. Quit
```

Attributes/Methods	Description
Stack Class	
- items: []	Stack class is a Stack data structure.
+ push(items) + pop(): int + peek(): int	push, pop and peek operation for the stack of items.
+ __str__(): string	String method for the class, to simply cast the list to a str. E.g: [3, 2, 1]
TowerOfHanoi Class	
- n: int	n refers to the number of disks of the first tower.
- step: int = 1	step is to track the number of steps taken by the user.
- towers = {"A": Stack(), "B": Stack(), "C": Stack()}	towers is a dict, storing 3 Stack objects, to be referenced by the strings of "A", "B", "C" respectively.
+ get_n(): int + get_step(): int	Getter for n and step.
+ get_tower(tower: str): Tower	Use the string "A", "B", "C" to access the Stack objects.
+ reset_step()	Reset the step value back to 1.
+ reset_towers()	Reset 3 towers to new Stack objects, and based on value of n, add a list of disks to Tower A. e.g. when n = 3, Tower A should have the following disks inside: [3, 2, 1]
+ __str__()	Return a string containing the 3 towers in the following format: Tower A: [3, 2, 1] Tower B: [] Tower C: []
+ check_win(): bool	Check if the players has won the game. If so, print: You win! Total steps taken: 7 Optimal steps expected (2^3 - 1): 7
move_disk(src: str, dst: str)	Move disk from src to dst tower. 1. check if src tower is empty. Print error msg if it is empty. 2. check if top disk of src can be moved to dst: - if dst is empty, it's ok to move - if top of dst is larger than top disk of src, then it's ok to move.

	- if it is not a valid move, print error msg.
	e.g:
	• `Invalid move: source tower is empty`
	• `Invalid move: disk from source is larger than top disk in dest tower`
	3. move disk from src to dst
	4. print step count and movement in the following format:
	`Step 1`
	`Moving disk from A to C`
	`Tower A: [3, 2]`
	`Tower B: []`
	`Tower C: [1]`
	5. check if game is won
	6. update the step value

`+ solve()`	1. reset the towers
	`Solving for 3 disks`
	`Starting arrangement:`
	`Tower A: [3, 2, 1]`
	`Tower B: []`
	`Tower C: []`
	
	2. solve the game automatically, by printing:
	
	`Step 1`
	`Moving disk from A to C`
	`Tower A: [3, 2]`
	`Tower B: []`
	`Tower C: [1]`
	
	`Step 2`
	`Moving disk from A to B`
	`Tower A: [3]`
	`Tower B: [2]`
	`Tower C: [1]`
	
	…
	
	`Step 7`
	`Moving disk from A to C`
	`Tower A: []`
	`Tower B: []`
	`Tower C: [3, 2, 1]`
	
	`You win!`
	`Total steps taken: 7`
	`Optimal steps expected (2^3 - 1): 7`

Bonus:

Update the user menu and implement undo and redo functions.

```
Please choose one of the following options:
1. Start/Reset a new game
2. Move a disk
3. Undo move
4. Redo move
5. Auto-solve game
6. Quit
```