Python - Classes, Objects, Constructors and Methods

1. Create a python class ATM which has a parametrised constructor (card_no, acc_balance). Create methods withdraw(amount) which should check if the amount is available on the account if yes, then deduct the amount and print the message “Amount withdrawn”, if the amount is not available then print the message “OOPS! Unable to withdraw amount, Low balance”. Create another method called, deposit, which should deposit amount if amount is positive and should print message “Amount deposited”. If not, print message “Invalid amount to deposit”. Create a method called getBalance which should print current balances at any given point of time.

   Example: atm_acc_1 = ATM(“1234”, 400)
   atm_acc_2 = ATM(“10001”, 100)

   Transactions and Expected Outputs:

Transaction	**Expected Output**	**Reason**
atm_acc_1.withdraw(300)	Amount withdrawn	400 is the balance, so 300 can be withdrawn
atm_acc_1.withdraw(300)	OOPS! Unable to withdraw amount, Low balance	Current balance is 100. Hence, low balance
atm_acc_1.deposit(300)	Amount Deposited	Positive amount deposit
atm_acc_1.getBalance()	400	400 - 300 = 100 + 300 = 400 (Refer above transactions)
atm_acc_2.getBalance()	100	For atm_acc_2 initial balance is 100
atm_acc_2.deposit(300)	Amount Deposited	Positive amount deposit
atm_acc_2.getBalance()	400	100 + 300 = 400 (Refer above transactions)

2. Create a class Book with a parameterised constructor then takes name_of_book, author, year_of_publish, price_of_book, no_of_copies_available. Create the following methods -
   - order_book(no_of_books) : This method should return price of purchase, if no of books is less than or equal to no_of_copies_available. Else, return “No stock”.
   - add_quantity(is_admin, quantity): This method should add quantity of books (2nd param) to existing no_of_copies_available if is_admin is True and return “Book quantity updated as <<count>>” . If is_admin is False, return “Unauthorised” as output.

Sample Input/Output:

book = Book(“Software Quality Assurance”, “Mr.Jochen”, 1994, 100, 50)

book.order_book(10)	1000	10*100 = 1000
book.order_book(1000)	No stock	
book.order_book(1)	100	1*100 = 100
book.order_book(43)	No stock	
book.add_quantity(False, 100)	Unauthorized	
book.add_quantity(True, 2)	Book quantity updated as 41	

3. Create a class Printer with a default constructor and a method called print_me(data), which returns the data that comes as argument.

Example: Let's say obj is the object for Printer class.
res = obj.print_me("Welcome")
print(result)

Output: Welcome

4. Create following classes
   1. Player class
   2. Board class
   3. Game class

Board Class:
1. Constructor which takes positions_of_snakes (dictionary where head is key, tail is value), positions_of_ladders (dictionary where key is start of ladder, value is end of ladder)
2. Method is_snake(position) -> Return True if it is snake box else return False.
3. Method is_ladder(position) -> Return True if it is ladder box else return False.
4. Method climb_ladder(ladder_start_box) -> Return the respective index to jump using position_of_ladders.
5. Method go_down(snake_position) -> Return the respective index to go down using positions_of_snakes.
6.

Player Class:
1. Constructor which takes name of player, initialise current position as 1 (as player need to start from 1)
2. roll_dice() method, which will roll dice, on rolling dice it should return random number between 1 to 6.

Game Class:
1. Create a method start_game() inside which create 2 player objects (say player_1 and player_2) and board class.
2. Put a while loop and check whether any of the player reaches the box 100, if not continue the loop, roll dice for player_1, player_2, player_1, player_2 and so on….(check for snake and ladders using Board class and perform ups and downs accordingly) Until any one player reaches the box 100. If reached 100, print the winner declaration. Saying "Player <Player name> wins the game!".
3. Note that: If player is in 96th box, and his roll gives 5 then 96+5 = 101 (Invalid) at this time, you should not declare him as winner! He has to exactly to go to 100th box to become winner.

Sample I/O:

```
player_1 = Player("Hari")
player_2 = Player("Ganesh")
positions_of_snakes = {5: 1, 35 : 3}
positions_of_ladders = {7 : 81, 41 : 53}
board = Board(positions_of_snakes, positions_of_ladders}

        while (# Write your conditions) :
                        ….
                        Write all you logic here
                        ….
        PRINT THE WINNER!
```

If any player reaches 5 (as per above example) he should go back to 1 (using go_down as is_snake(5) returns True. Similarly, if any player reaches 41 (as per above example) he should go to 53 as is_ladder(41) returns True and so on…. The game should continue until anyone reaches

100. Make sure to handle all possible exceptions and also print logs to understand how the game is going.

A typical sample log is like below

- Player 1 (Hari) rolled 3 and reaches 3
- Player 2 (Ganesh) rolled 4 and reaches 4
- Player 1 (Hari) rolled 2 and reaches 5
- Player 1 (Hari) Snake found and reaches 1
- Player 2 (Ganesh) rolled 3 and reaches 7
- Player 2 (Ganesh) Ladder found and reaches 81 ….. and so on.

5. Create a class Student with parameterised constructor name and gender. Create a getter which returns name in the format Mr. <<name>> if the gender is Male, Ms. <<name>> if the gender is Female.

______________.