Object Oriented Programming

Programming Day 3

Task 01:

You have been assigned the task of developing a basic calculator using Object-Oriented Programming (OOP) principles. Create a class named Calculator, and it should support common arithmetic operations such as addition, subtraction, multiplication, division, and modulo.

In addition to the arithmetic operations, the Calculator class should now include attributes to represent two numbers. These numbers will serve as the operands for the various arithmetic operations. The class should have public instance variables (number1 and number2) to store these values.

Users can give the attributes while creating the object. Users can also assign the values to the attributes after creating the object but in that case you should assign default value of 10 to both attributes. (Therefore, choose the type of constructors consciously)

The Calculator class should also include methods for each supported operation, such as add, subtract, multiply, divide, and modulo. These methods will operate on the stored numbers and return the results of the respective operations. Additionally, these methods should handle potential errors gracefully, particularly when attempting to divide by zero.

The main method should demonstrate the functionality of the Calculator class by creating an instance of it, setting values for the two numbers, and performing arithmetic operations. Test the calculator with different scenarios by making a menu with 8 options to verify the correctness of the implemented operations and the handling of attributes.

Following will be the menu options

1. Create the a Single Object of Calculator
2. Change Values of Attributes
3. Add
4. Subtract
5. Multiply
6. Divide
7. Modulo
8. Exit

Task 02:

Update the Existing Calculator class with the following behaviors of scientific calculator as well.

1. **Square Root (sqrt):**

- The method should take one integer parameter and return the square root of the number.

2. **Exponential Function (exp):**
   - The method should take one integer parameter and return the value of the exponential function, where the parameter is the exponent.
3. **Logarithm (log):**
   - The method should take one integer parameter and return the natural logarithm (base e) of the number.
4. **Trigonometric Functions (sin, cos, tan):**
   - Include methods for sine, cosine, and tangent. These methods should take one integer parameter representing an angle in degrees and return the corresponding trigonometric function value.

Update the main method accordingly.

Task 03:

This challenge is an English twist on the Japanese word game **Shiritori**. The basic premise is to follow two rules:

1. **First character** of the next **word** must match the last **character** of the previous **word**.
2. The word must not have already been said.

Below is an example of a **Shiritori** game:

```
["word", "dowry", "yodel", "leader", "righteous", "serpent"]  // valid!
["motive", "beach"]  // invalid! - beach should start with "e"
["hive", "eh", "hive"]  // invalid! - "hive" has already been said
```

Write a **Shiritori class** that has **two instance properties** and **two instance methods**::

- **words**: an array of words already said.
- **game_over**: a boolean that is true if the game is over.
- **play()**: a method that takes in a word as an argument and checks if it is valid (the word should follow rules #1 and #2 above).
  - If it is valid, it adds the word to the **words** array, and returns the **words** array.
  - If it is invalid (either rule is broken), it returns **"game over"** and sets the **game_over** boolean to **true**.
- **restart()**: a method that sets the **words** array to an empty one [ ] and sets the **game_over** boolean to **false**. It should return **"game restarted"**.

Examples

```
Shiritori my_shiritori = new Shiritori();
my_shiritori.play("apple") ➞ ["apple"]
my_shiritori.play("ear") ➞ ["apple", "ear"]
my_shiritori.play("rhino") ➞ ["apple", "ear", "rhino"]
my_shiritori.play("corn") ➞ "game over"
```

// Corn does not start with an "o".
my_shiritori.words ➞ ["apple", "ear", "rhino"]
// Words should be accessible.
my_shiritori.restart() ➞ "game restarted"
my_shiritori.words ➞ []
// Words array should be set back to empty.
my_shiritori.play("hostess") ➞ ["hostess"]
my_shiritori.play("stash") ➞ ["hostess", "stash"]
my_shiritori.play("hostess") ➞ "game over"
// Words cannot have already been said.

Notes

- The **play** method should **not** add an invalid word to the **words** array.
- You don't need to worry about capitalization or white spaces for the inputs for the **play** method.
- There will only be **single inputs** for the **play** method.

Task 04:

You are tasked with creating a versatile Book class that represents books with various attributes and behaviors. The Book class should have a constructor with the following properties:

- title: representing the title of the book.
- author: representing the author of the book.
- publicationYear: representing the year the book was published.
- price: representing the price of the book.
- quantityInStock: representing the quantity of this book available in the stock.

Additionally, the Book class should have the following methods:

1. **getTitle() method:**
    - Returns a formatted string: "Title: {title}".
2. **getAuthor() method:**
    - Returns a formatted string: "Author: {author}".
3. **getPublicationYear() method:**
    - Returns a formatted string: "Publication Year: {publicationYear}".
4. **getPrice() method:**
    - Returns a formatted string: "Price: {price}" .
5. **sellCopies(int numberOfCopies) method:**
    - Takes an integer parameter numberOfCopies and simulates selling that number of copies. Ensure to update the quantityInStock attribute accordingly. If there are not enough copies in stock, print an error message indicating the unavailability.
6. **restock(int additionalCopies) method:**
    - Takes an integer parameter additionalCopies and adds that number of copies to the stock.
7. **bookDetails() method:**
    - Return detailed information about the book, including title, author, publication year, price, and the current quantity in stock in a formatted string.

Write the main method to test your class by creating a menu driven program.
Your task is to test the Book class by creating different books like:

- Book 1: "Pride and Prejudice" by Jane Austen, published in 1813, priced at $19.99.
- Book 2: "Hamlet" by William Shakespeare, published in 1603, priced at $15.50.
- Book 3: "War and Peace" by Leo Tolstoy, published in 1869, priced at $25.99.

Create a list named bookList in the **main method** that includes all the created book instances.
Menu Options can be as follows:

1. Add Book
    a. Create the object of the book using the Class Constructor
2. View All the Books information
    a. Iterate over the list and print out the details of each book.
3. Get the Author details of a specific book
    a. First take the Title of the book and then search in your bookList.
4. Sell Copies of a Specific Book
    a. First take the Title of the book and then search in your bookList.
5. Restock a Specific Book
    a. First take the Title of the book and then search in your bookList.
6. See the count of the Books present in your bookList.
    a. Get the length of the bookList
7. Exit

Task 05:

Project Conversion:

1. Create signin and signup through **User Class** with role. Create the appropriate Behaviors of the Class
2. Create an admin menu with full CRUD for one entity (either Product, Customer, Books, etc according to your project). Add the Appropriate behaviors for that Entity in the Class.
3. Use Constructor to create a new user.
4. Save information of the users and the entity into a text file.
5. Create a user menu that utilizes the information of the entity given in step 2.

Task 06:

Game Conversion:

1. Create an Enemy class that holds information of display character, x and y. Also add appropriate behaviors to the Enemy Class.
2. Create a Player class that holds information of display character, x and y. Also add appropriate behaviors to the Player Class.
3. Use Constructor to initialize the player symbol and its position.
4. Create multiple enemies (objects), store in the List of enemies and move in the space using the behaviors of the Class.
5. Create a player (object) and move in the space using the behaviors of the Class.