Problem 1: Queue-based Task Scheduler

Problem Specification:

You are tasked with implementing a simple task scheduling system using a queue data structure. The program should allow users to interact with a menu to add tasks to a queue and execute them in the order they were added. Each task has a name and a description.

Requirements:

1. Create a class called Task that has the following attributes:
   - name: a string representing the name of the task.
   - description: a string representing the description of the task.

2. Create a class called TaskScheduler that uses a queue to manage tasks. The TaskScheduler class should have the following methods:
   - add_task(task): Add a task to the end of the queue.
   - execute_task(): Execute the task at the front of the queue (the oldest task). Remove the task from the queue after execution.
   - get_next_task(): Return the name and description of the next task to be executed without removing it from the queue.
   - is_empty(): Return True if the task queue is empty; otherwise, return False.

3. Implement a menu-driven interface that allows users to interact with the task scheduler. The menu should have the following options:
   - Add a task to the scheduler.
   - Execute the next task in the queue.
   - View the next task to be executed without removing it from the queue.
   - Check if the scheduler is empty.
   - Exit the program.

4. Ensure that the program correctly executes the selected actions based on the user's input.
5. Test your program with various tasks and use the menu to perform different actions.
6. Provide clear prompts and user-friendly messages in the menu interface to guide users in interacting with the task scheduler.

Problem 2: Browser History Using Stacks

Problem Specification:

You are tasked with creating a simple web browser history system that allows users to navigate forward and backward through their browsing history using two stacks: one for the back history and one for the forward history.

Requirements:

1. Create a class called BrowserHistory that implements the browser history system. The class should have the following methods and attributes:
   - __init__(self): Initialize the browser history with empty stacks for the back and forward history.
   - visit(self, url): Add the current URL to the back history stack and clear the forward history stack.
   - back(self): Move back to the previous page. Pop the top URL from the back history stack and push it onto the forward history stack.
   - forward(self): Move forward to the next page. Pop the top URL from the forward history stack and push it onto the back history stack.
   - current(self): Return the current URL at the top of the back history stack (the current page).
   - is_back_enabled(self): Return True if there are URLs in the back history stack, indicating that the user can navigate back. Otherwise, return False.
   - is_forward_enabled(self): Return True if there are URLs in the forward history stack, indicating that the user can navigate forward. Otherwise, return False.

2. Implement a menu-driven interface that allows users to interact with the browser history. The menu should have the following options:
   - Visit a new URL and update the history.
   - Navigate back to the previous page.
   - Navigate forward to the next page.
   - Check if back and forward navigation is enabled.
   - View the current page.
   - Quit the program.

3. Ensure that the browser history follows the rules as described in the previous specification.
4. Include error handling to handle cases where navigation is not possible (e.g., trying to go back when there's no back history).
5. Provide clear prompts and user-friendly messages in the menu interface to guide users in interacting with the browser history.

Hints on Solving the Problem:

1. You can use two stacks, one for the back history and one for the forward history, to maintain the user's browsing history.
2. When implementing the visit method, push the current URL onto the back history stack and clear the forward history stack to start fresh.

3. For the back and forward methods, consider checking if the respective history stack is empty before attempting to navigate in that direction. If it's not empty, you can pop the URL from one stack and push it onto the other.
4. When checking if back or forward navigation is enabled, simply check if the corresponding history stack has URLs.
5. Use a loop to display the menu options and take user input until they choose to quit the program.
6. Provide informative messages to guide the user, such as notifying them when they've reached the end of their history in a particular direction.

Rubric for "Queue-based Task Scheduler" and Browser History Using Stacks"

Criteria	**Excellent (4)**	**Good (3)**	**Fair (2)**	**Needs Improvement (1)**
Implementation (20)				
Class Structure	Clear class design with all required methods and attributes.	Mostly clear class design with most required methods and attributes.	Somewhat clear class design but lacks some required methods or attributes.	Unclear class design or missing multiple required methods/attributes.
Functionality (Correctness)	Functions correctly without issues.	Functions correctly with minor issues or edge cases.	Mostly functions but has several issues.	Significant issues or doesn't work as expected.
Error Handling	Comprehensive error handling for all possible user actions.	Adequate error handling for most actions.	Basic error handling but may miss some cases.	Minimal or no error handling.
User Interface (20)				
Menu Clarity	Clear, user-friendly menu with all required options.	User-friendly menu with most required options.	Somewhat user-friendly but lacks some required options.	Not user-friendly and lacks key options.
User Guidance	Clear and informative prompts/messages.	Mostly clear and informative prompts/messages.	Some unclear or less informative prompts/messages.	Confusing or missing prompts/messages.

Criteria	**Excellent (4)**	**Good (3)**	**Fair (2)**	**Needs Improvement (1)**
Project Documentation (10)				
Code Comments (Documentation)	Thorough code comments and explanations.	Adequate code comments and explanations.	Basic code comments with limited explanations.	Minimal or no code comments or explanations.
User Instructions	Clear instructions for program usage.	Adequate instructions with some details.	Basic instructions with limited details.	Unclear or incomplete instructions.
Total (50)	20	15	10	5