**PHASE 1: PROBLEM IDENTIFICATION**

1. **Identification of needs and symptoms**

* Implement a functional "Uno" card game in Java.
* Develop a program that simulates the game "Uno" using data structures such as stacks, queues, hash tables, and priority queues.
* There is no "Uno" game implemented in Java that uses the mentioned data structures.
* Users do not have the ability to play "Uno" in a digital environment using these data structures.

**Definition of the problem**

Develop a functional "Uno" card game in Java that uses data such as stacks, queues, hash tables, and priority queues to manage card distribution, the discard pile, and game rules.

**PHASE 2: COLLECTION OF THE NECESSARY INFORMATION**

Developing functional "Uno" game in Java requires a deep understanding of game rules, data structures, and user interface design. It is essential to collect detailed information on the following aspects:

**Game rules:**

**Basic rules**:

**Card Dealing:** At the beginning of the game, 7 cards are dealt to each player.

**Player Turns:** On his turn, a player must:

**Play a Card:** If you have a card that matches the color, number, or symbol of the top card in the discard pile, you may play it.

**Draw a card:** If you do not have a matching card, you must draw a card from the deck.

**Say "one":** If you have only one card in your hand, you must say "one." If you don't say it and another player finds out, you must draw two cards.

**Special Cards:** There are special cards that can change the course of the game, such as:

**Color Change:** Allows the player to change the color of the game.

**Card Draw:** Forces the next player to draw two cards and lose their turn.

**Turn Skip:** Causes the next player to lose their turn.

**Winner:** The first player to run out of cards in their hand wins the game.

**Data structures in Java:**

**Formal definitions:**

**Stacks:** A stack is a data structure that follows the LIFO (Last In, First Out) principle. This means that the last element to be added to the stack is the first to be removed.

**Queues:** A queue is a data structure that follows the FIFO (First In, First Out) principle. This means that the first element to be added to the queue is the first to be removed.

**Hash Tables:** A hash table is a data structure that maps keys to values. The key is used to calculate the location of the value in the table.

**Priority Queues:** A priority queue is a data structure that stores elements with an associated priority. Items are removed from the queue in order of priority, with the highest priority item being the first to be removed.

**Implementation in Java:**

**Stacks:** A stack can be implemented in Java using different methods, such as a linked list or an array.

**Queues:** A queue can be implemented in Java using different methods, such as a linked list or an array.

**Hash Tables:** A hash table can be implemented in Java using different methods, such as an array or a tree.

**Priority Queues:** A priority queue can be implemented in Java using different methods such as a binary tree or a heap.

**Advantages and disadvantages:**

**Batteries:**

**Advantages:**

Easy to implement.

Efficient at adding and removing elements to the beginning and end of the stack.

**Disadvantages:**

It is not efficient for accessing elements in the middle of the stack.

**Queues:**

**Advantages:**

Easy to implement.

Efficient in adding and removing elements to the beginning and end of the queue.

**Disadvantages:**

It is not efficient for accessing elements in the middle of the queue.

**Hash tables:**

**Advantages:**

Efficient for searching, inserting and deleting elements.

**Disadvantages:**

It may be less efficient if there are many collisions (two different keys mapping to the same location).

**Priority queues:**

**Advantages:**

Efficient for adding and removing elements.

Allows access to elements in priority order.

**Disadvantages:**

It can be more complex to implement than other data structures.

**Analysis in the context of the game:**

**Stacks:** Can be used to store cards that have been played in reverse order.

**Queues:** Can be used to store cards to be played in chronological order.

**Hash Tables:** Can be used to store cards in a hash table, allowing them to be quickly accessed by name.

**Priority Queues**: Can be used to store cards in a priority queue, allowing the most important cards to be played first.

Choosing the right data structure:

Choosing the appropriate data structure for each case will depend on the specific needs of the game. For example, if you need to store cards in reverse order, a stack would be the best option. If you need to store cards in chronological order, a queue would be the best option.

**PHASE 3: SEARCH FOR CREATIVE SOLUTIONS**

**Alternative 1:**

This solution uses the data structures proposed in the statement:

Stacks: for the playing deck and discard.

Queues: for the distribution and drawing of letters.

Hash tables: to store additional information about cards.

Priority queues: to determine the order of play with special cards.

**Advantages:**

Efficiency: The data structures chosen are the most efficient for the operations performed in the game.

Simplicity: The code is easy to understand and maintain.

Ease of understanding: The logic of the game is clear and transparent.

**Disadvantages:**

Less flexibility: The game structure is limited by the chosen data structures.

Difficulty to modify: Changing the behavior of data structures can be complex.

**Recommendation:**

This solution is the best option for most cases. It is the most balanced in terms of efficiency, simplicity and flexibility.

**Alternative 2: Greater Flexibility**

Description:

This solution seeks greater flexibility in the game structure, using the following data structures:

Stacks: only for the game deck.

Queues: for the distribution of letters.

Linked lists: for discard and player hands.

Hash tables: to store additional information about cards.

Binary search: to find cards in linked lists.

**Advantages:**

Greater flexibility: The structure of the game can be modified more easily.

Ability to modify: The behavior of data structures can be modified without affecting the rest of the code.

**Disadvantages:**

Lower efficiency: Linked lists are less efficient than stacks and queues for some operations.

Greater complexity: The code is more complex and difficult to understand.

Difficulty understanding: The game logic may be less clear and transparent.

**Recommendation:**

This solution is recommended if you require more flexibility in the game structure, and are willing to sacrifice a little efficiency and simplicity.

**Alternative 3: Maximum Flexibility**

**Description:**

This solution seeks maximum flexibility in the game structure, using the following data structures:

Linked lists: for the playing deck, discard, and player hands.

Binary search: to find cards in linked lists.

Binary trees: to store additional information about the cards.

Priority queues: to determine the order of play with special cards.

**Advantages:**

Maximum flexibility: The structure of the game can be modified with complete freedom.

Ability to modify: The behavior of data structures can be modified without affecting the rest of the code.

**Disadvantages:**

Less efficient: Linked lists and binary trees are less efficient than stacks, queues, and hash tables.

Greater complexity: The code is very complex and difficult to understand.

Difficulty in understanding: The logic of the game can be very complex and not very transparent.

**Recommendation:**

This solution is only recommended if you require extreme flexibility in the game structure, and have the experience and resources necessary to deal with the complexity of development.

**PHASE 4: TRANSITION FROM IDEA FORMULATION TO PRELIMINARY DESIGNS**

In this phase, the transition from general ideas to feasible preliminary designs is carried out. A critical analysis of the ideas is carried out to discard those that are not viable and the most promising are selected to turn them into preliminary projects.

**Idea Evaluation**

The following aspects are considered:

Feasibility: Is it possible to implement the idea with the available technology and resources?

Economic viability: Is the idea profitable and sustainable in the long term?

Impact on the user: Does the idea meet the needs of the user and improve their experience?

Technical aspects: Do you have the necessary knowledge and skills to develop the idea?

Risks: What are the possible risks associated with the implementation of the idea?

**Tools for Evaluation**

Pugh Matrix: Allows you to compare different ideas based on predefined criteria.

SWOT Analysis: The strengths, opportunities, weaknesses and threats of each idea are analyzed.

Brainstorming: Creative ideas are generated in groups.

**Selection of Ideas**

Based on the evaluation carried out, the most promising ideas are selected to be developed into preliminary projects. The following characteristics are considered:

Potential for success: Does the idea have a high potential for success in the market?

Differentiation: Is the idea different from existing solutions?

Scalability: Is the idea likely to be scalable in the future?

**Preliminary design**

Once the ideas are selected, a preliminary design begins to be developed. The following aspects are defined:

System architecture: The general structure of the system and how its components are interconnected is defined.

Functionalities: The functionalities that the system will have are defined.

User interface: It defines how users will interact with the system.

Technical requirements: The hardware and software requirements of the system are defined.

**Alternatives Evaluation**

**Alternative 1: Efficiency and Simplicity**

Feasibility: High

Economic viability: High

Impact on the user: Medium

Technical aspects: Low

Risks: Low

**Alternative 2: Greater Flexibility**

Feasibility: Medium

Economic viability: Medium

Impact on the user: High

Technical aspects: Medium

Risks: Medium

**Alternative 3: Maximum Flexibility**

Feasibility: Low

Economic viability: Low

Impact on the user: High

Technical aspects: High

Risks: High

**Selection of Ideas**

Alternatives 1 and 2 are selected to be developed into preliminary projects. Alternative 3 is discarded due to its high complexity and cost.

**Preliminary design**

**Alternative 1**

System architecture: Simple, using the proposed data structures.

Features: Basic rules of the Uno game.

User interface: Simple and intuitive.

Technical requirements: Standard Java technologies.

**Alternative 2**

System architecture: Flexible, allowing customization.

Features: Basic rules of the Uno game and additional features.

User interface: Flexible, adaptable to different needs.

Technical requirements: Standard Java technologies and some additional libraries.

**Conclusion**

Alternatives 1 and 2 have been selected to be developed in preliminary projects. A critical analysis of the ideas has been carried out and the basic aspects of the preliminary design have been defined.

**PHASE 5: EVALUATION AND SELECTION OF THE BEST SOLUTION**

**Criteria:**

**Definition of Criteria for Selection of the Best Solution:**

**1. Accuracy:**

Alternative 1: Exact (an exact solution is preferred)

Alternative 2: Approximate

**2. Efficiency:**

Alternative 1: High

Alternative 2: High

**3. Ease of Implementation:**

Alternative 1: High (an easy to implement solution is preferred)

Alternative 2: Medium

**4. Risks:**

Alternative 1: Low (a solution with low risk is preferred)

Alternative 2: Medium

**5. Flexibility:**

Alternative 1: Low

Alternative 2: High

**6. Economic Viability:**

Alternative 1: High

Alternative 2: Medium

**7. Impact on the User:**

Alternative 1: Medium

Alternative 2: Stop

**8. Scalability:**

Alternative 1: High

Alternative 2: High

**9. Maintainability:**

Alternative 1: High

Alternative 2: Medium

**10. Compatibility:**

Alternative 1: High

Alternative 2: Medium

**Weighting of Criteria:**

Accuracy: 0.4 (most important factor)

Efficiency: 0.2

Ease of Implementation: 0.2

Risks: 0.2

Note: The weighting of the criteria can be adjusted according to the needs of the project.

**Solutions Evaluation:**

Each solution will be evaluated based on the defined criteria, using a scale from 1 to 5 (5 being the best rating).

| **Criteria** | **Weighing** | **Alternative 1** | **Punctuation** | **Alternative 2** | **Punctuation** |
| --- | --- | --- | --- | --- | --- |
| **Precision** | 0.4 | Exact | 5 | Approximate | 3 |
| **Efficiency** | 0.2 | high | 5 | high | 5 |
| **Ease of implementation** | 0.2 | high | 5 | average | 3 |
| **Risks** | 0.2 | Low | 5 | average | 4 |
| **Total** | 1.0 |  | 5 |  | 4 |

**Selection:**

Based on the evaluation carried out, Alternative 1 is selected as the best option due to its greater accuracy, ease of implementation and lower risk.

**Additional considerations:**

The efficiency of both solutions is comparable.

Alternative 2 offers greater flexibility, but this factor is not as important in this case.

Development time and cost are important factors to consider, especially if you have limited resources.

Based on these considerations, Solution 1 is presented as the most viable and effective option for the development of the Uno game.