CARD GAME WITH GRAPHICAL USER INTERFACE (GUI)

alexandrubrabete@yahoo.com

TEHNICAL UNIVERSITY OF CLUJ- NAPOCA, SOFTWARE ENGINEERING DEPARTMENT

Project Documentation: Card Game with Graphical User Interface (GUI)

1. Project Overview:

This project is a simple card game developed in **Java** where cards are randomly drawn from a shuffled deck, and the game is visually represented using **Java Swing** for the graphical user interface (GUI). The goal is to draw cards from the deck until a **club (♣)** card with a value greater than **8** is drawn, at which point the game ends. The game includes a security feature to validate the integrity of the deck, preventing tampering. The user can interact with the game by clicking a button to draw cards, with each card draw being animated on a **green table background**. Additionally, when the player wins the game, a message is displayed on the screen.

2. Main Features:

- **Deck Initialization and Shuffling:**
  - The deck consists of **52 cards**, including 13 values for each of the 4 suits: ♠, ♥, ♦, ♣.
  - The deck is shuffled at the start to ensure randomness.
- **Card Draw:**
  - Cards are drawn randomly when the user clicks the **"Draw Card"** button.
  - The cards are represented by **strings** (e.g., "10♠", "K♦", "7♥").
  - Each drawn card is animated as it moves from the deck to the "extracted cards" section of the green table.
- **Winning Condition:**
  - The game ends when a **club (♣)** card with a value greater than **8** is drawn.
  - Once this occurs, a **"You Win!"** message is displayed on the screen.
- **Card Animation:**
  - Each drawn card moves smoothly across the screen from the deck to the "extracted cards" area.
  - The animation enhances the visual experience, making the game more engaging.
- **Security Check:**
  - The integrity of the deck is validated using **SHA-256** hashing.
  - After every card draw, the deck's hash is recalculated and compared with the initial hash.
  - If the deck has been tampered with (i.e., the hash has changed), the game stops.
- **User Interface (UI):**
  - The main UI contains a **green table background** with the **deck** of cards displayed in the center.
  - The **Draw Card** button allows the user to draw cards from the deck.
  - The **status label** shows the number of cards drawn.
  - The **"You Win!"** message appears when the game ends.

3. Technical Components Used:

- **Java Swing**: The primary library used for creating the graphical user interface, enabling components like **JPanel**, **JButton**, and **JLabel**.
- **Java AWT**: Used for event handling and drawing custom graphics (such as card animation).
- **Cryptography**: **SHA-256 hashing** ensures the integrity of the deck throughout the game.
- **Randomization**: Used to shuffle the deck and draw cards randomly.
- **Animation**: Implemented through **Java Graphics** and **Timer** classes to animate the drawn cards moving from one position to another.

4. Project Flow:

1. **Deck Initialization and Shuffling:**
    - At the start of the game, a deck with 52 cards is created, and the cards are shuffled.
    - Each card is represented by a string (e.g., "A♠", "K♦", "5♥").
2. **Card Drawing:**
    - The user clicks the **"Draw Card"** button to draw a card from the deck.
    - The drawn card is animated, moving from the deck's location to the "extracted cards" section on the table.
3. **Game Termination and Winning Condition:**
    - If a **club (♣)** card with a value greater than **8** is drawn, the game stops, and a **"You Win!"** message is displayed.
    - If the deck is exhausted before a winning card is drawn, the game ends with a **"Game Over"** message.
4. **Security Check (SHA-256):**
    - The integrity of the deck is validated using a **SHA-256** cryptographic hash.
    - After each card is drawn, the deck's hash is recalculated and compared to the initial hash. If there is a mismatch, it indicates that the deck has been altered, and the game stops.
5. **User Interface:**
    - The UI features the deck, the **draw button**, and the **extracted cards** section.
    - The number of drawn cards is displayed on a **label**.
    - Upon winning, the UI will display a **"You Win!"** message at the center of the screen.

5. User Interface Layout:

- **Main Panel:**
    - The **deck** is displayed at the center of the screen.
    - A **button** allows the user to draw cards from the deck.
    - **Extracted cards** are shown in the area to the right or below the deck.
    - A **status label** shows the number of cards drawn so far.
- **Card Representation:**
    - Cards are shown as text (e.g., "A♠", "10♦", "2♥").

    - The **green background** simulates a playing table, where cards are displayed and animated.
- **Winning Message:**
    - When the game ends, a **"You Win!"** message appears, signaling that the player has successfully drawn a **club (♣)** card with a value greater than **8**.

6. Security and Integrity Features:

- **Deck Integrity Check (SHA-256):**
    - The deck's integrity is verified using **SHA-256 hashing**. At the start of the game, a hash of the deck is generated.
    - After each card draw, the deck's hash is recalculated and compared to the original hash. If they do not match, it indicates that the deck has been tampered with, and the game stops.
- **Prevention of Tampering:**
    - If the deck's hash changes, the game halts to prevent further manipulation of the deck.

7. Skills Acquired from this Project (CV Format):

Java Programming Skills:

- Practical experience with **Java programming** and **object-oriented principles**.
- Hands-on experience with **Swing** to create graphical user interfaces with interactive elements like buttons and labels.

Graphical User Interface (GUI) Development:

- Developed skills in using **Swing components** such as **JPanel**, **JButton**, and **JLabel** to design and implement a functional and visually appealing game.
- Learned how to create smooth animations in **Java** using **Graphics** and **Timer** classes.

Cryptography and Security:

- Implemented **SHA-256 hashing** to maintain the integrity of the deck and ensure that the deck's state has not been altered.
- Applied **security principles** to prevent data manipulation, ensuring the game runs as intended.

Event-Driven Programming:

- Gained experience in **event-driven programming**, handling user interactions such as clicks and input with **ActionListener** and **MouseListener**.

Game Logic and Randomization:

- Implemented **randomization techniques** to shuffle and draw cards, ensuring fairness in gameplay.
- Developed logic to determine the win condition based on card values and suits.

Animation Techniques:

- Enhanced the user experience by implementing **animations** to make the card drawing process more visually engaging.

Software Integrity Best Practices:

- Gained practical experience in **cryptographic validation** to prevent tampering with the game data, ensuring the integrity of the game.

Problem-Solving and Debugging:

- Applied problem-solving skills to handle complex scenarios such as detecting tampered decks and ensuring the smooth operation of the animation and game mechanics.

Conclusion:

This project has provided a comprehensive learning experience in Java development, particularly in **GUI programming**, **animation**, **cryptography**, and **event handling**. By building this card game, I not only gained technical proficiency in Java but also learned how to integrate advanced concepts such as **data integrity validation** and **randomization**. The game's interactive and animated nature, combined with its secure features, showcases my ability to create engaging and reliable applications. This project is a solid example of my capability to build secure, interactive, and visually appealing software.