|  |
| --- |
| C:\Users\NewGabriel\Desktop\cart.jpg |
|  |
| Solitaire Encryption Testing and Design |
|  |
|  |
|  |

|  |
| --- |
|  |

Contents

[Addition Method 2](#_Toc308776011)

[Encryption and Decryption 3](#_Toc308776012)

[Testing case: Keygen 4](#_Toc308776013)

[Testing case: Encryption and Decryption 6](#_Toc308776014)

[Testing case: Deck.txt exception 9](#_Toc308776015)

[Other Exception 12](#_Toc308776016)

Addition Method

**Class Solitaire**

* **addToTail**

Use for add the deck to the list node

* **makeDeck**

Make the deck (throw the number of deck.txt into deck list)

* **Joker A**

Execute Joker A

* **Joker B**

Execute Joker B

* **Triple Cut**Execute Triple cut
* **Count Cut**Execute Count cut
* **Encryption**

Execute encryption

* **Decryption**Execute decryption
* **ToString**

Print out the deck number

* **GotTheKey**

Get the key

**CardNode**

* **CardNode ( Constructor )**

Use for input to deck list

Encryption and Decryption

In this exercise, you are required to store the deck in a **LinkedList**. A search method should be added so that you can find the position of your jokers. The method should return an integer value indicating the position of your search item.

* A method that moves an element in the list down ***n*** positions (for steps 1 and 2 of your key generation process). You may also consider writing a swapping method to facilitate the move.
* Methods that will remove ***n*** items from the head or tail from your list (for the triple cut and count cut). These methods should return a **LinkedList** storing the removed items. An empty list should be returned if nothing is removed.
* Methods that will insert a **LinkedList** to the head or at the tail of your existing **LinkedList** (for the triple cut). Take extra care to check if the **LinkedList** is ***null*** or ***empty***.
* You may want to add another method that handles the insertion that occurs in the middle of the list (for count cut), but you can always use some other tricks to obtain the result.
* You can use the ***get()*** method in the **LinkedList** I have worked on during the Lab to get the face value of the top card, the bottom card and the (N+1)-th card, i.e., no method needs to be added for this purpose.

**Example of Encryption and decryption:**

Take the first Solitaire message mentioned in novel as an example - “Do not use PC.”:

1. Removing the non-letters and capitalizing the word gives the plaintext: DONOTUSEPC.
2. Convert the message to numbers, we will have: 4 15 14 15 20 21 19 5 16 3
3. The message contains 10 letters. So, we need to generate ten keystream values using the given deck on Page 1. The first keystream value should be 21 as shown before. All ten values are: 21 26 13 1 18 20 25 24 9 7.
4. Keystream values are added to corresponding numbers and the numbers are then converted back to letters as follows:

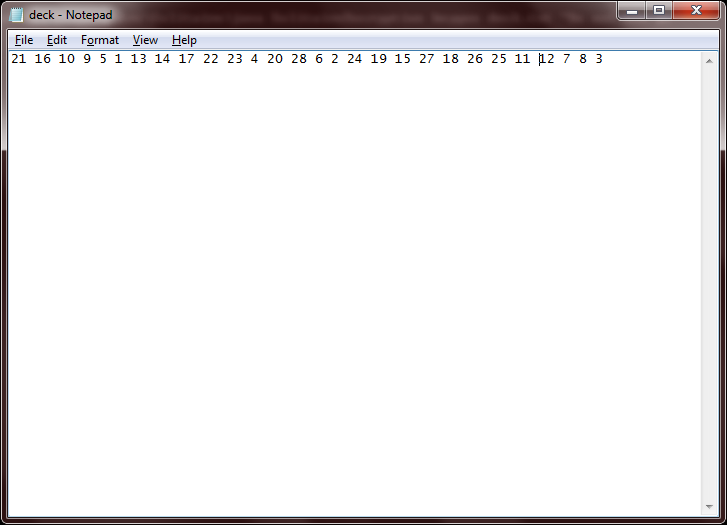
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Plaintext | D | O | N | O | T | U | S | E | P | C |
| (number representation) | 4 | 15 | 14 | 15 | 20 | 21 | 19 | 5 | 16 | 3 |
| Keystream values | 21 | 26 | 13 | 1 | 18 | 20 | 25 | 24 | 9 | 7 |
| (encoded numbers) | 25 | 15 | 1 | 16 | 12 | 15 | 18 | 3 | 25 | 10 |
| Ciphertext | Y | O | A | P | L | O | R | C | Y | J |

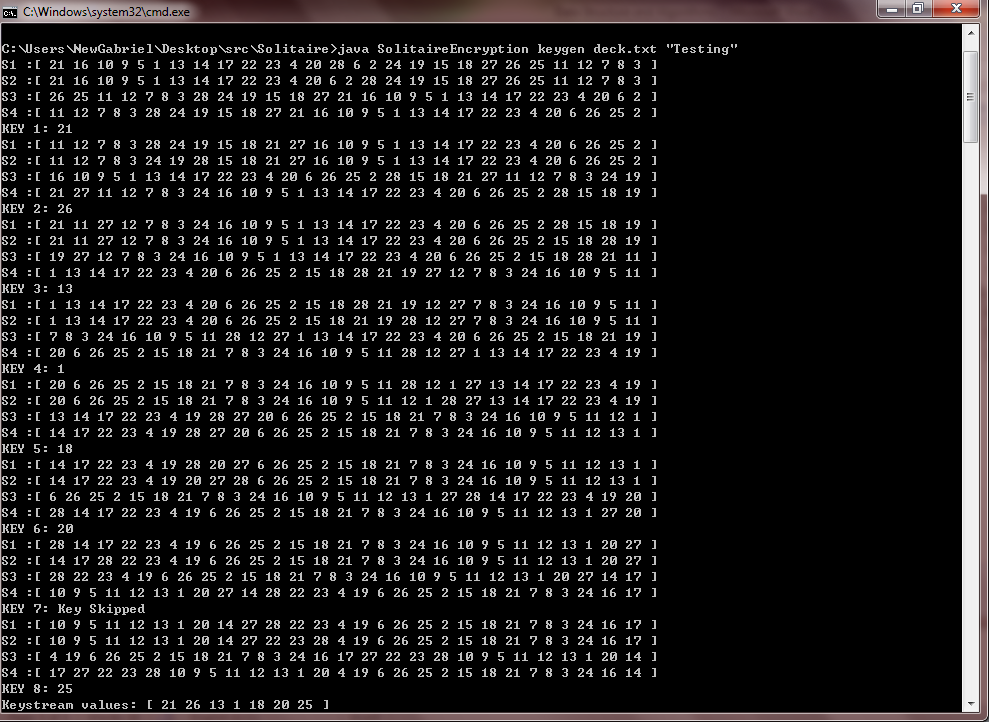
1. Decryption is just the reverse process of encryption.

Testing case: Keygen

**Case 1: General case**

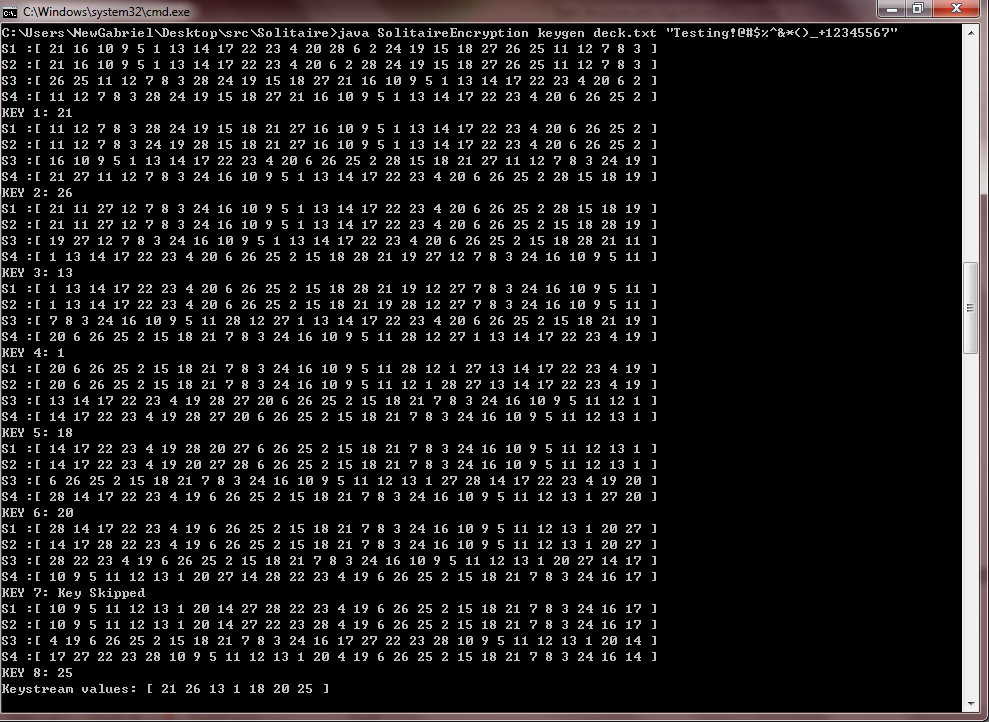
java SolitaireEncryption keygen deck.txt “Testing”:





**Case 2: Special words and number**

java SolitaireEncryption keygen deck.txt “Testing!@#$%^&\*()\_+12345567”

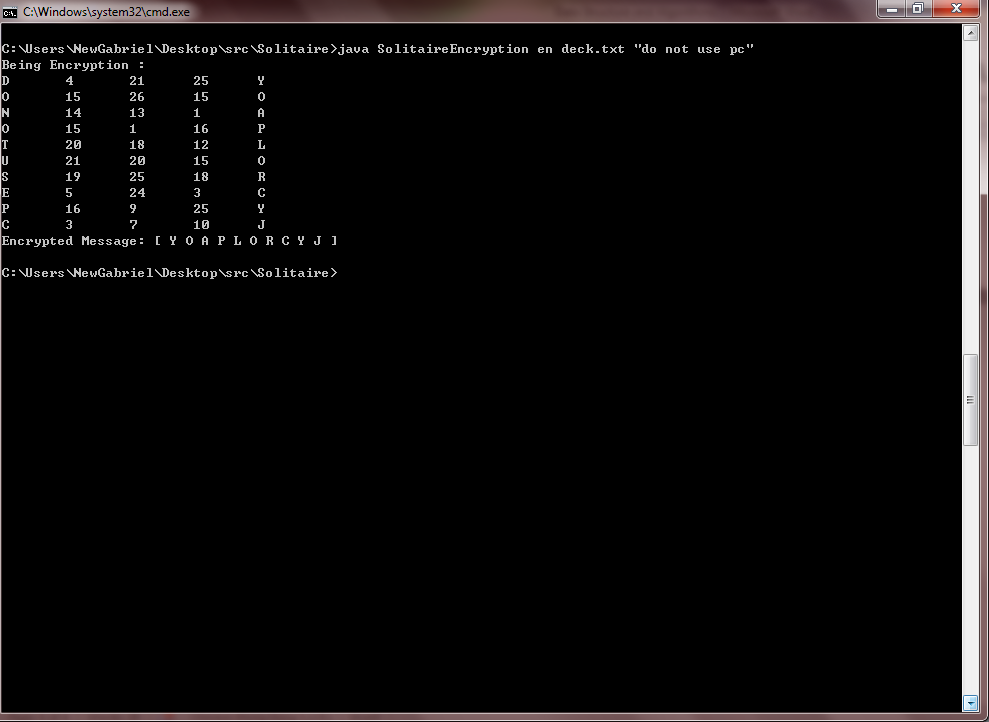


Testing case: Encryption and Decryption

**Case 1: General case**

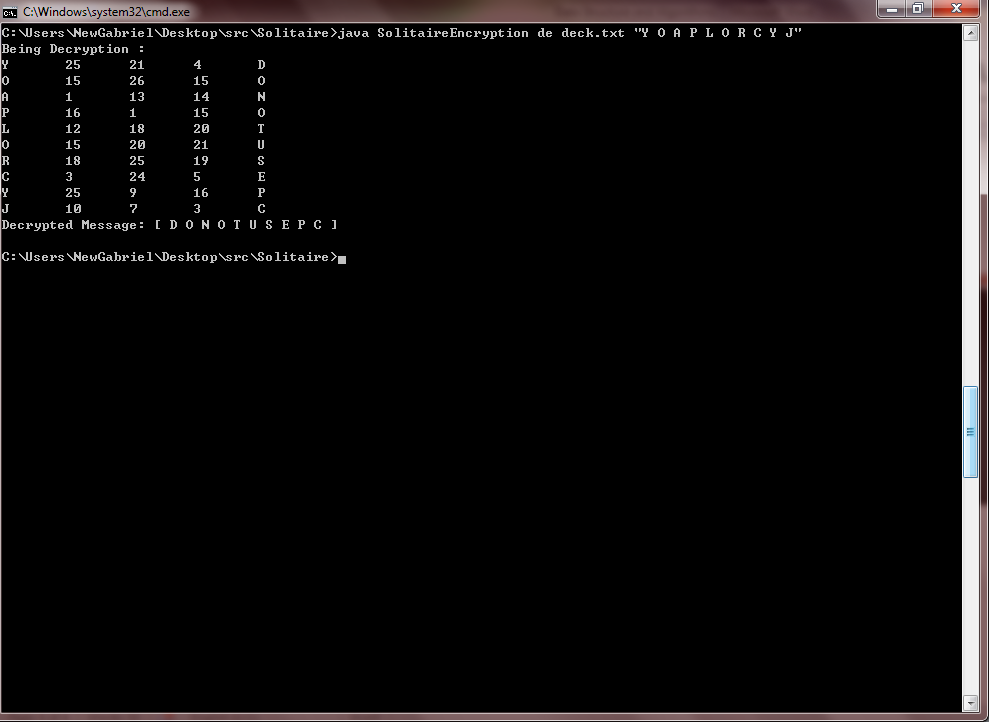
**Encryption:**

java SolitaireEncryption en deck.txt “do not use pc”



**Decryption:**

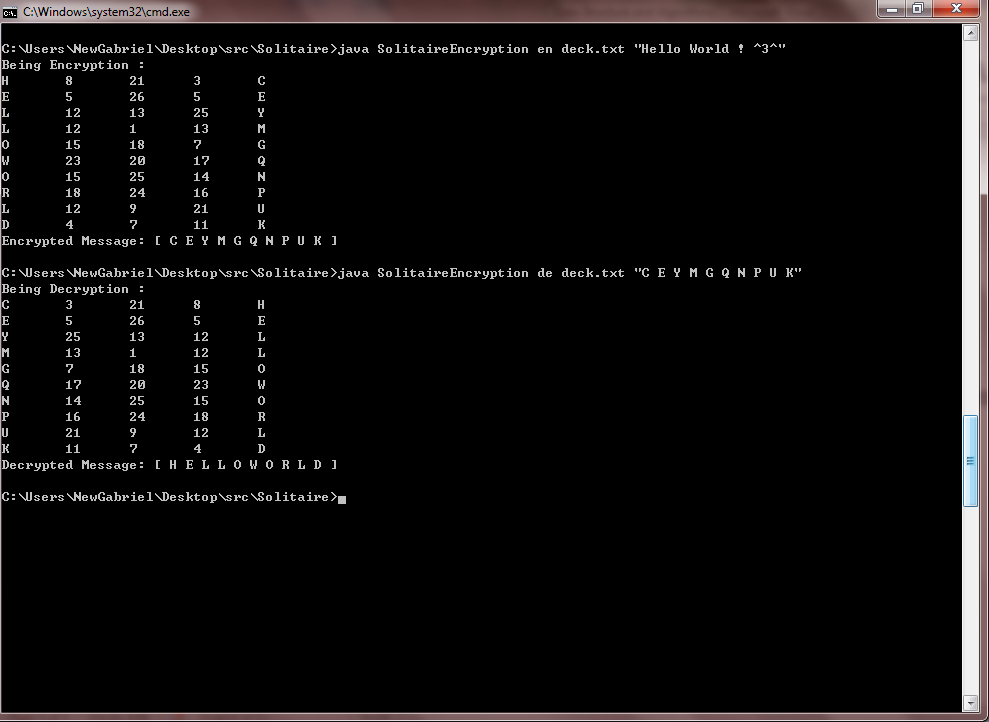
Java SolitaireEncryption de deck.txt “Y O A P L O R C V J”



**Case 2: Special words and number**

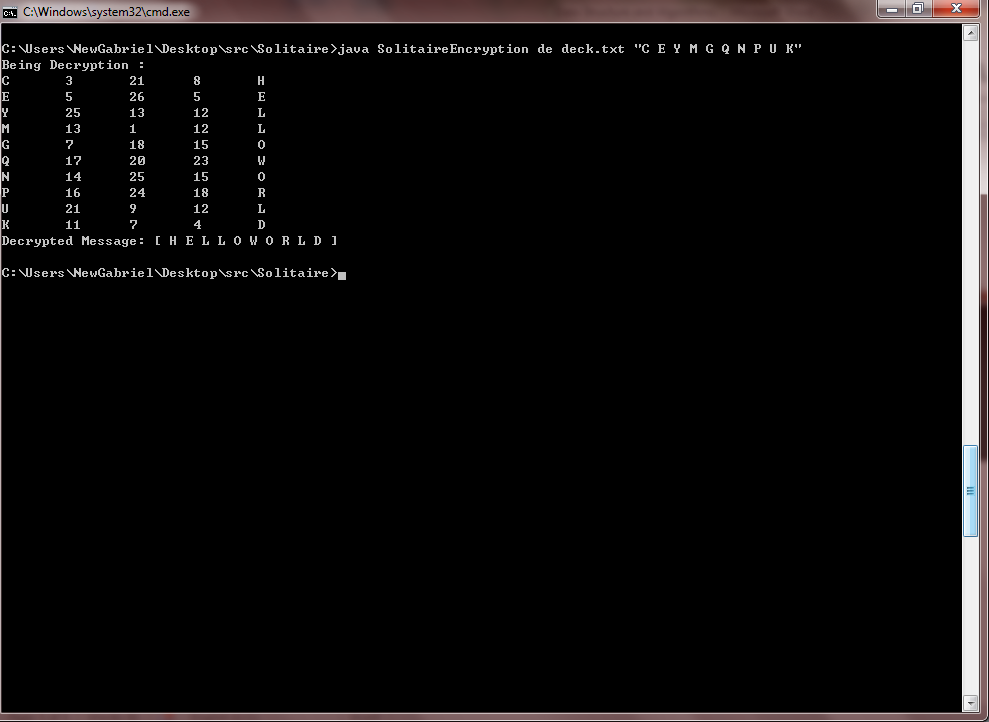
**Encryption:**

Java SolitaireEncryption en deck.txt “Hello World ! ^3^”



**Decryption:**

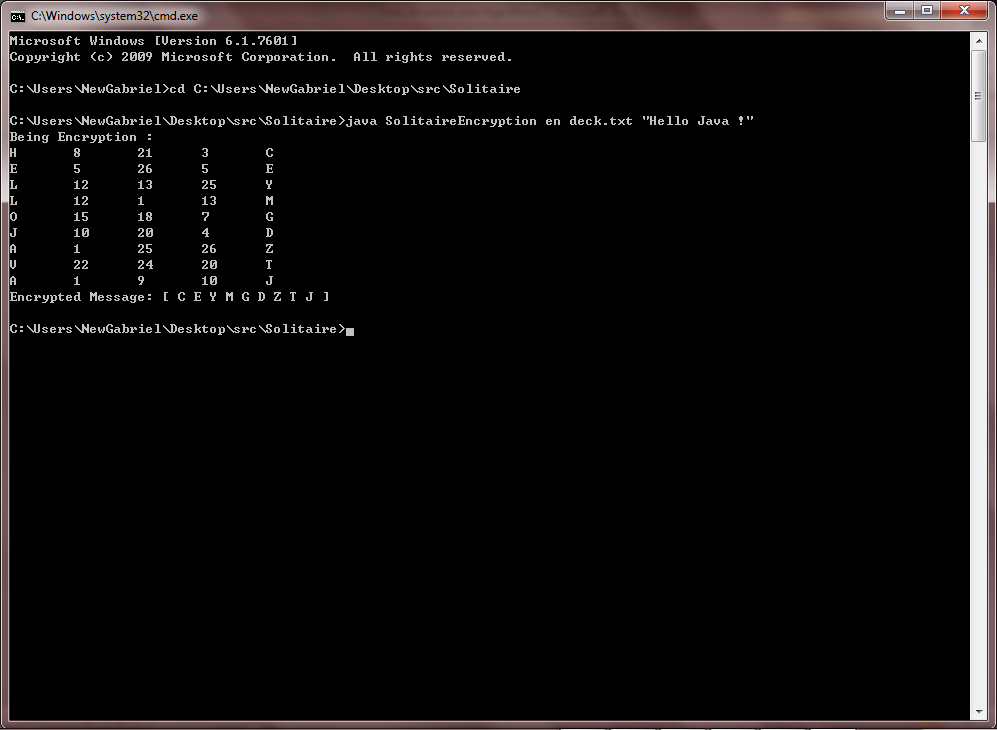
Java SolitaireEncrytion de deck.txt “C E Y M G Q N P U K”



**Case 3: Not Upper case decryption**

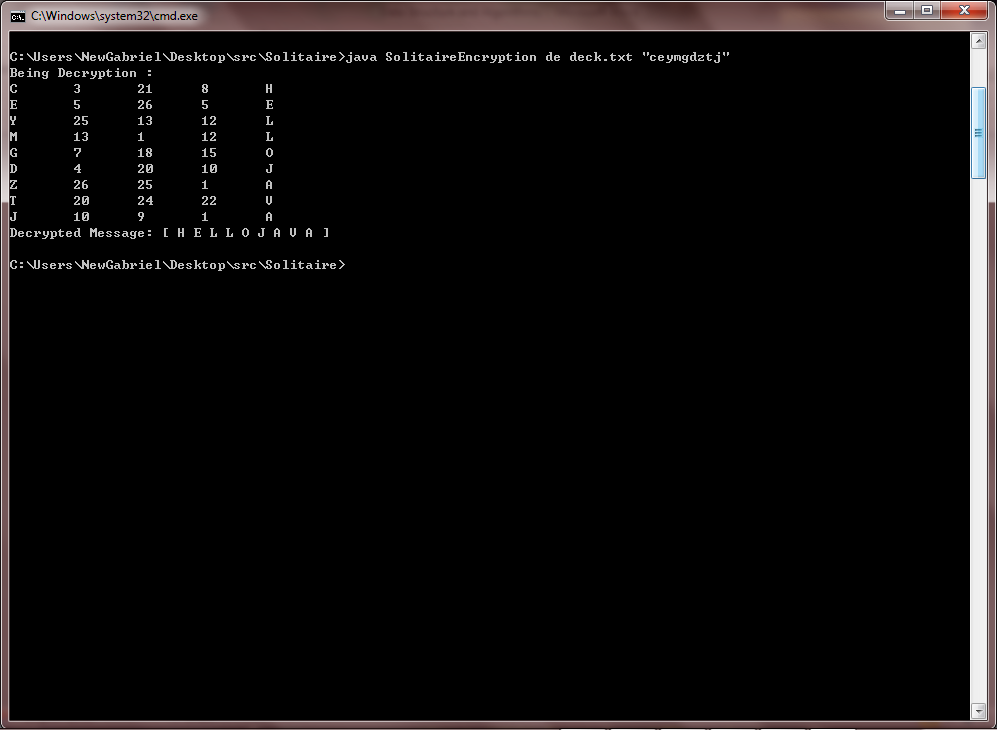
**Encryption:**

Java SolitaireEncryption en deck.txt “Hello Java !”



**Decryption:**

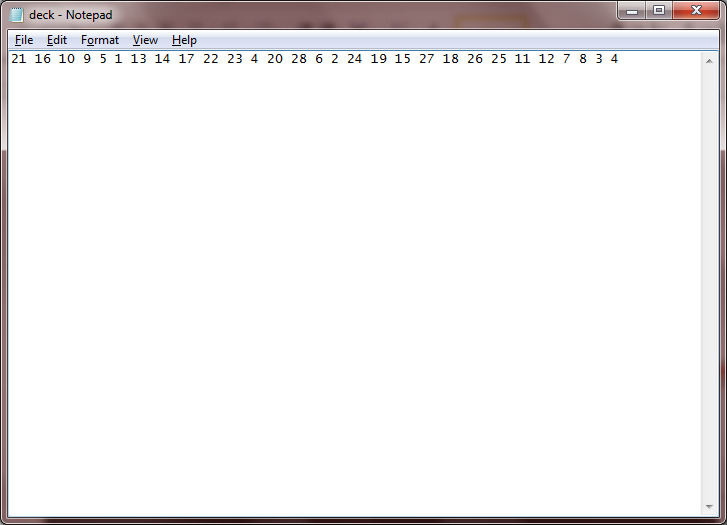
Java SolitaireEncryption de deck.txt “ceymgdztj”

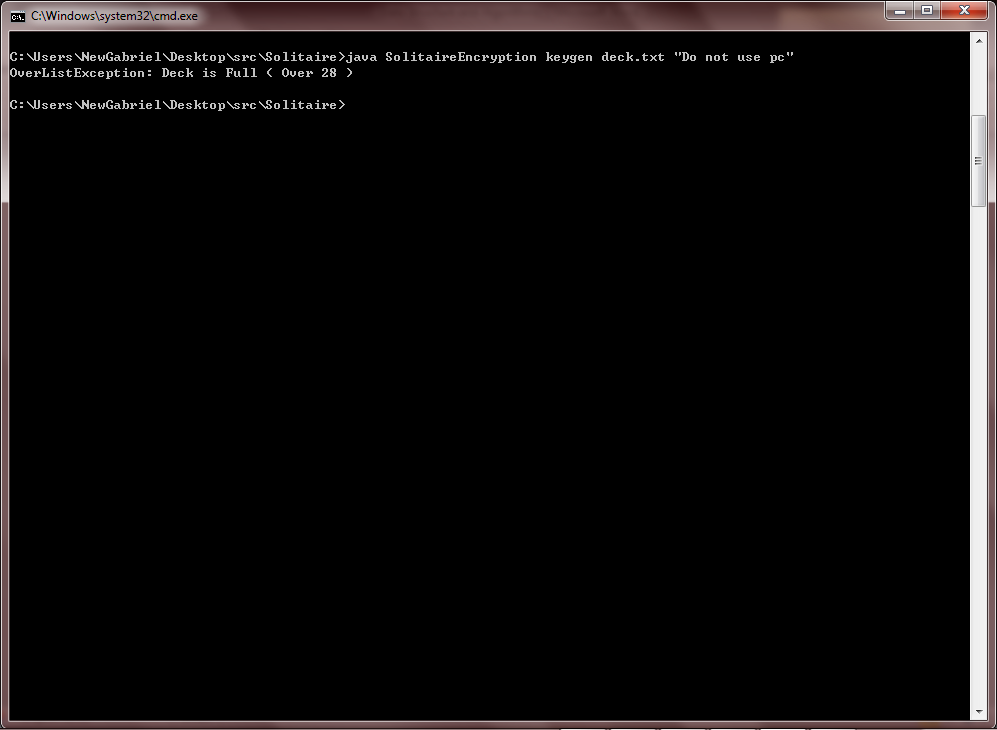


Testing case: Deck.txt exception

**Case 1: OverListException - More than 28 deck number**

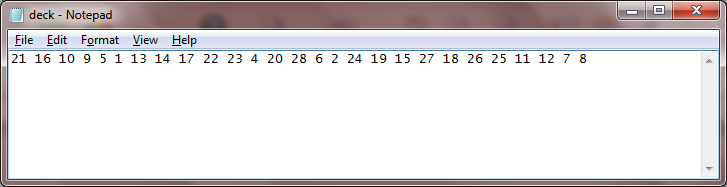
Java SolitaireEncryption keygen deck.txt “Do not use pc”

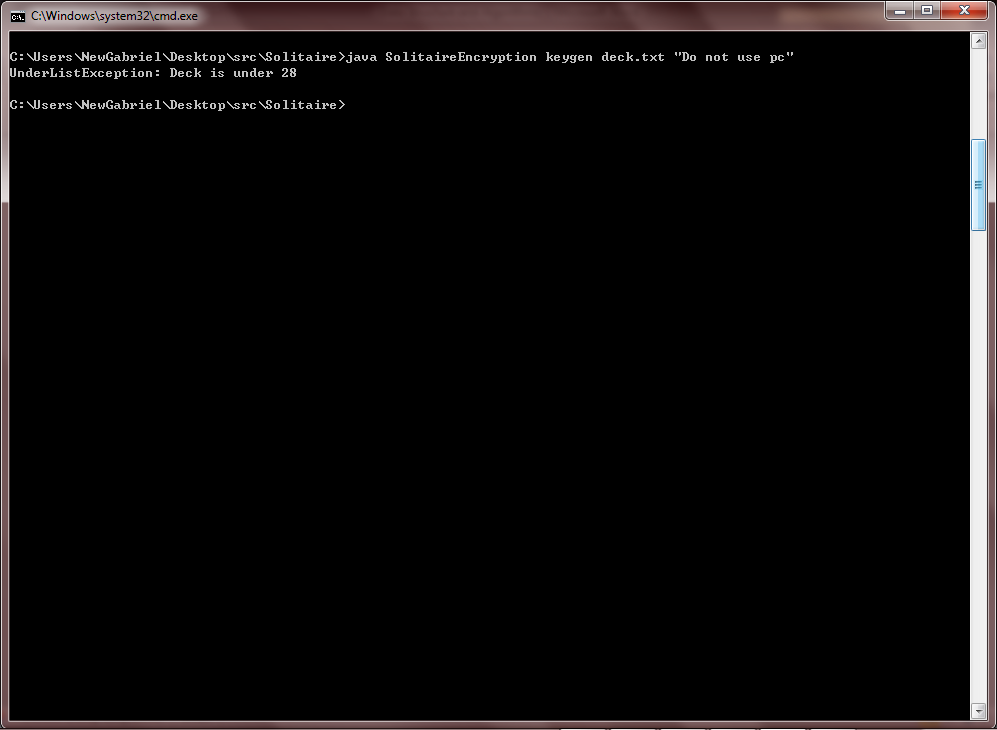




**Case 2: UndeListException - Less than 28 deck number**

java SolitaireEncryption keygen deck.txt “Do not use pc”

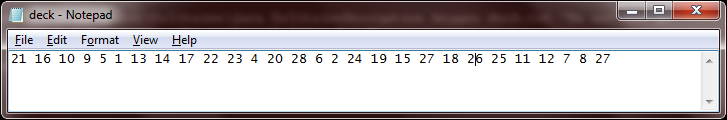




**Case 3: Joker Exception - More than one Joker**

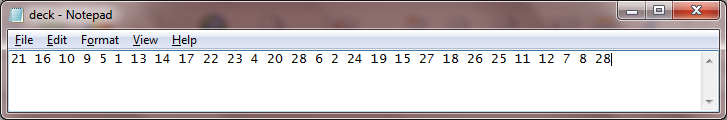
java SolitaireEncryption keygen deck.txt “Do not use pc”

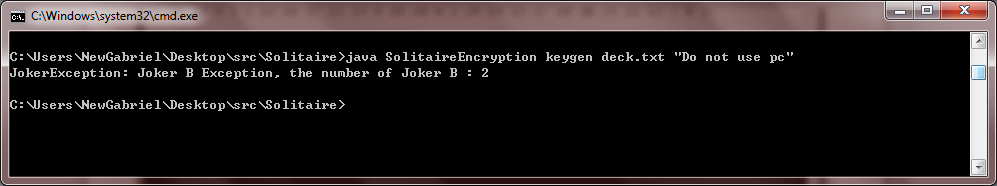
**More than one Joker A:**





**More than one Joker B:**

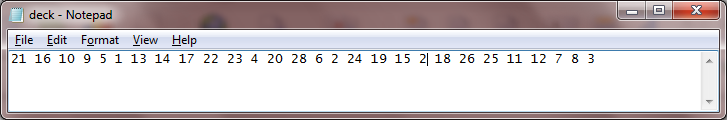




**Case 4: Joker Exception - Less than one joker**

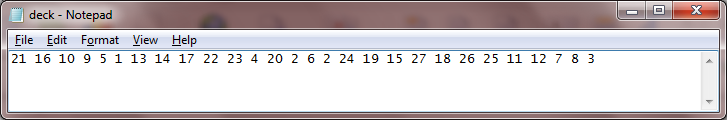
Java SolitaireEncryption keygen deck.txt “Do not use pc”

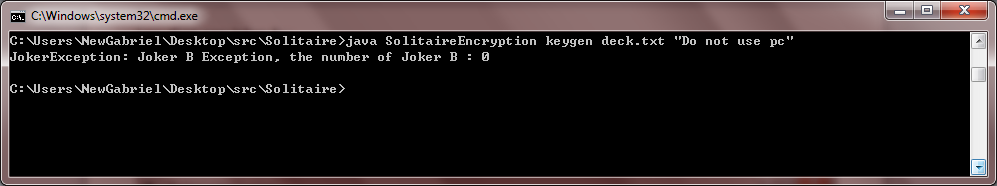
**Less than one Joker A:**





**Less than one Joker B:**





Other Exception

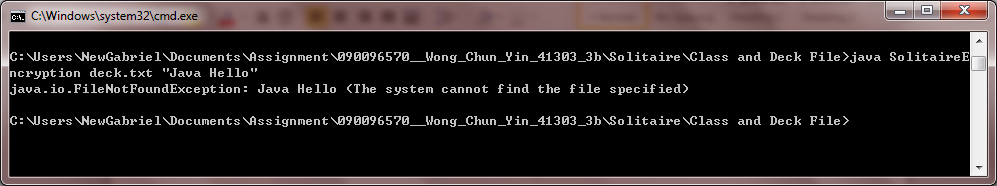
**Case 1:**

Java SolitaireEncryption abc deck.txt “Java Hello”



**Case 2:**

Java SolitaireEncryption deck.txt “Java Hello”



**Case 3:**

Java SolitaireEncryption

