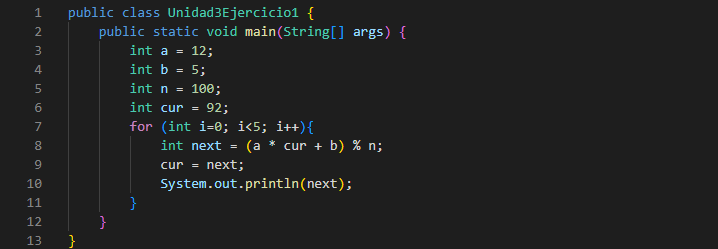
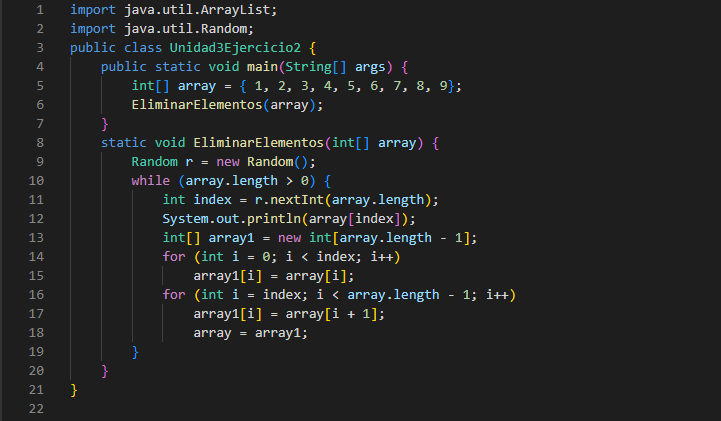
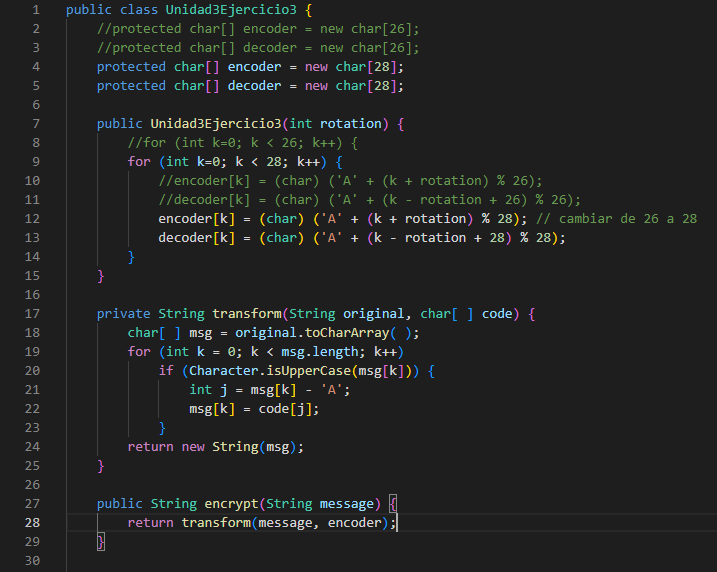
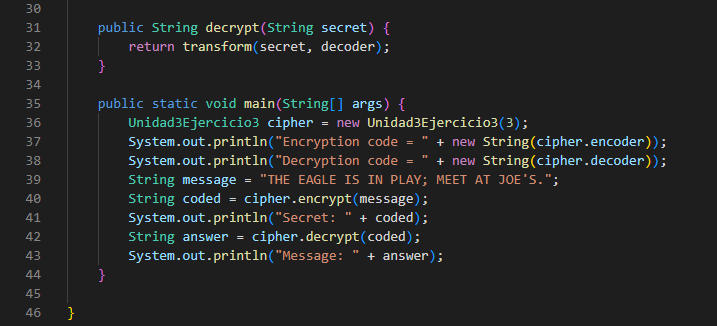
1. Give the next five pseudorandom numbers generated by the process described on page 113, with a = 12, b = 5, and n = 100, and 92 as the seed for cur.

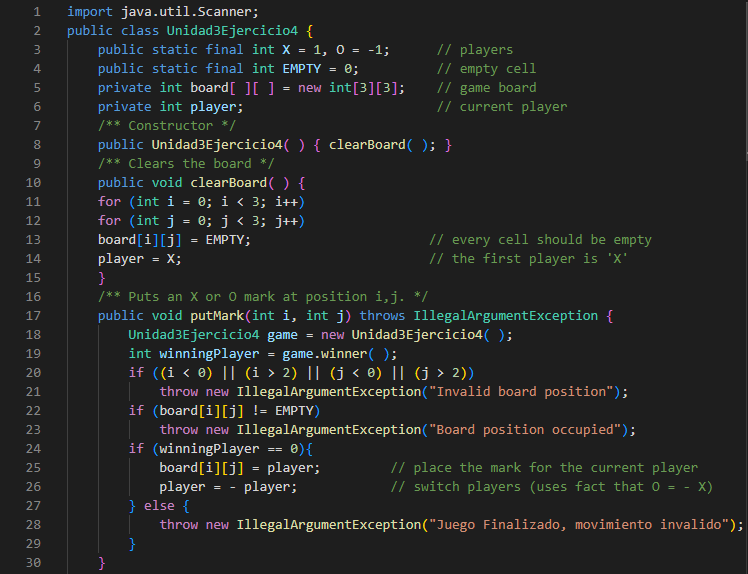


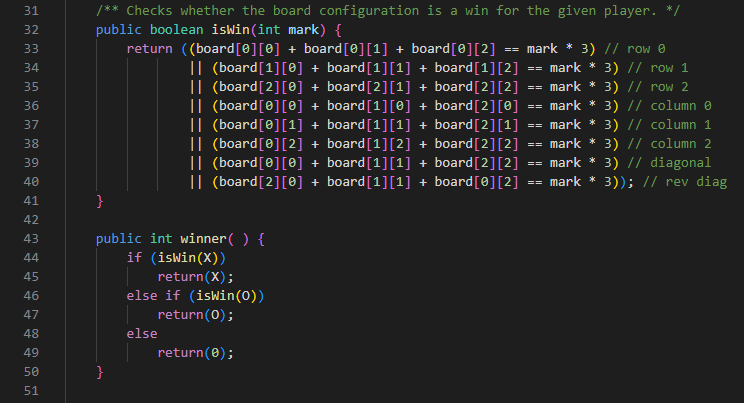
1. Write a Java method that repeatedly selects and removes a random entry from an array until the array holds no more entries.

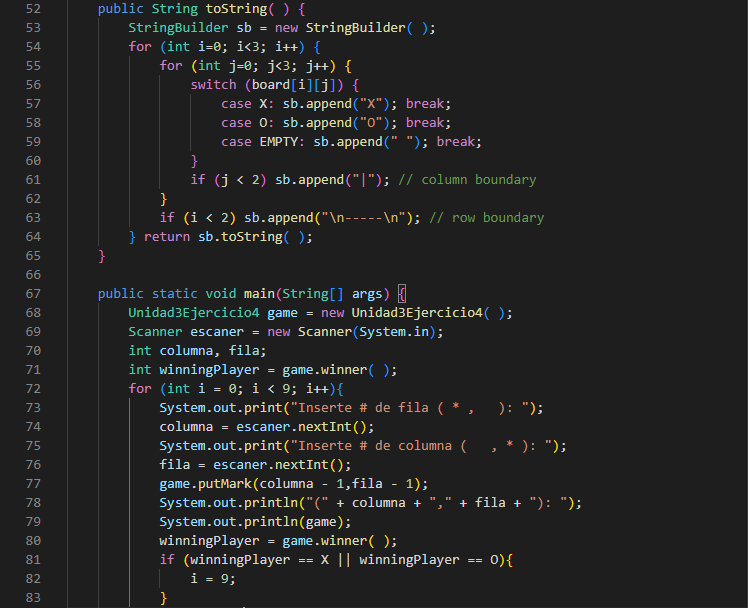
1. Explain the changes that would have to be made to the program of Code Fragment 3.8 so that it could perform the Caesar cipher for messages that are written in an alphabet-based language other than English, such as Greek, Russian, or Hebrew.

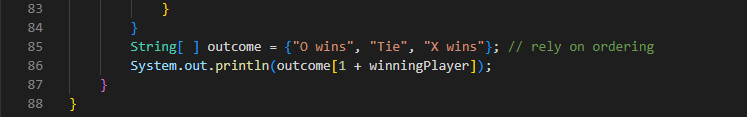




1. The TicTacToe class of Code Fragments 3.9 and 3.10 has a flaw, in that it allows a player to place a mark even after the game has already been won by someone. Modify the class so that the putMark method throws an IllegalStateException in that case.







1. The removeFirst method of the SinglyLinkedList class includes a special case to reset the tail field to null when deleting the last node of a list (see lines 51 and 52 of Code Fragment 3.15). What are the consequences if we were to remove those two lines from the code? Explain why the class would or would not work with such a modification.

De acuerdo con la pregunta, en la lista enlazada individualmente tenemos un método **removeFirst** que incluye un caso especial para restablecer un campo de cola a **null** cuando se elimina el último nodo de la lista, es decir

Si eliminamos estas dos líneas de código, entonces no hay un impacto tan malo en el código, pero todavía nos enfrentamos a problemas como:

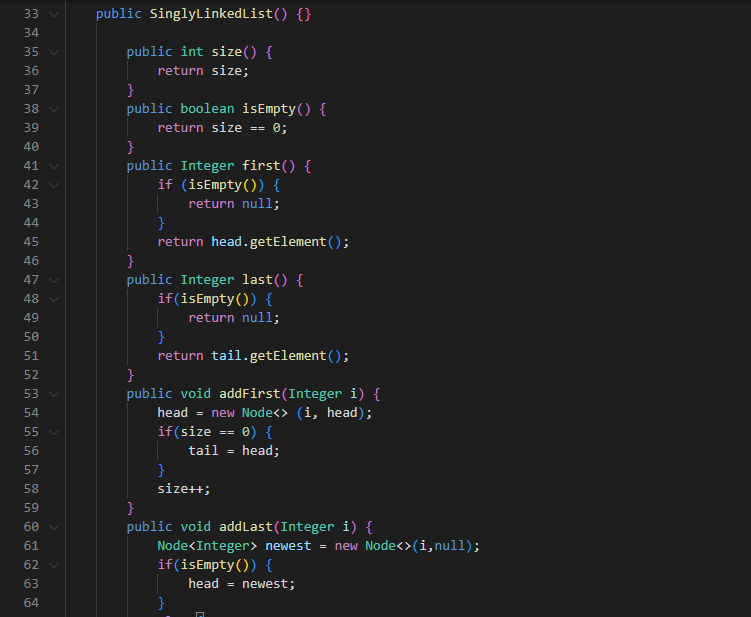
* Como hemos eliminado la parte de la cola de la lista enlazada que almacena los resultados de la referencia de memoria del siguiente elemento. Es una lista vacía hasta que se añade un nuevo elemento al último nodo.
* Como el último nodo es eliminado, no recuperará su memoria hasta que un nuevo nodo sea insertado en la misma posición.

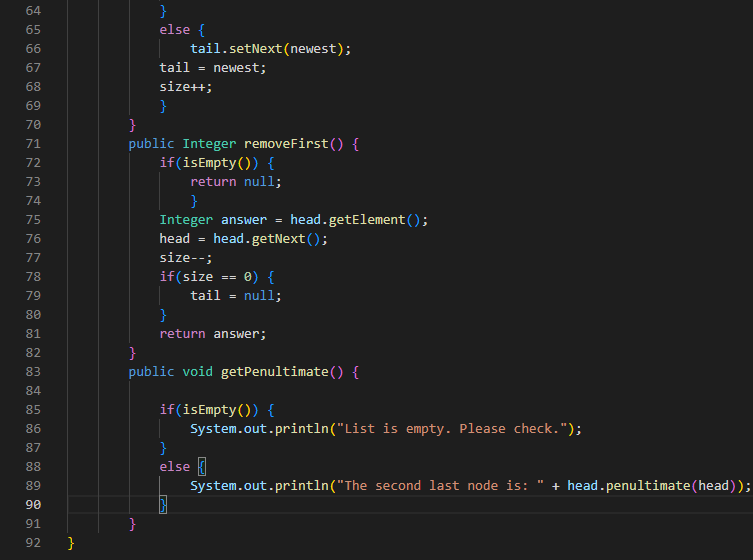
Por lo tanto, hay alguna referencia colgante de la cola, es decir, como no hay tal último nodo que tiene una dirección de memoria. por lo tanto, la parte de la cola de la lista no almacenará ningún tipo de referencia de memoria específica. la referencia colgante también se denomina como referencia de basura por eso la clase no funcionará con dicha modificación.

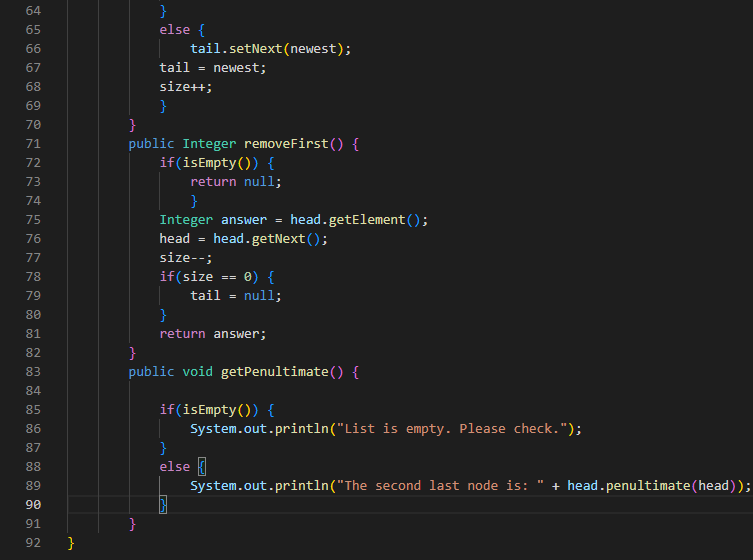
1. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.



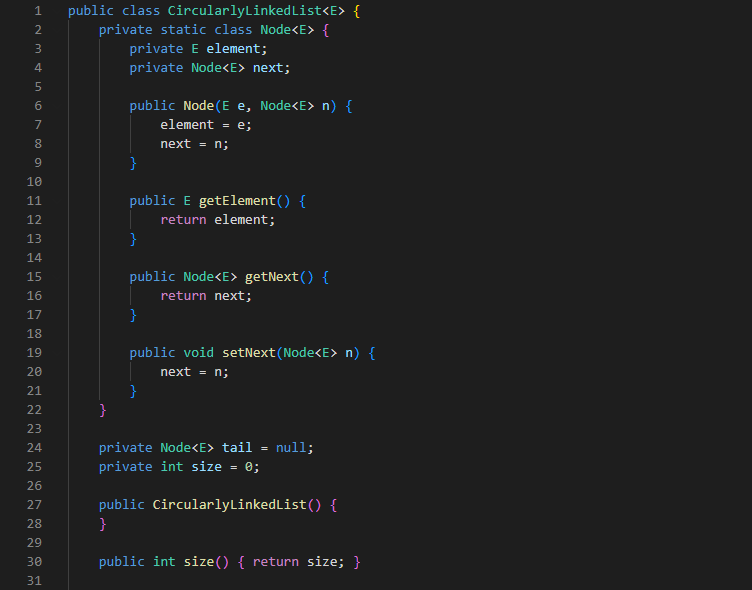


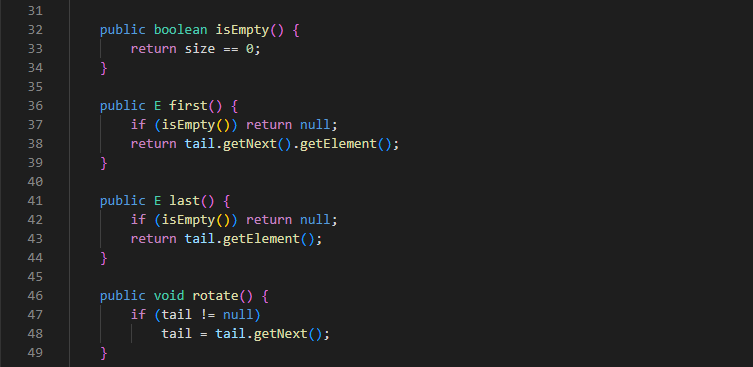


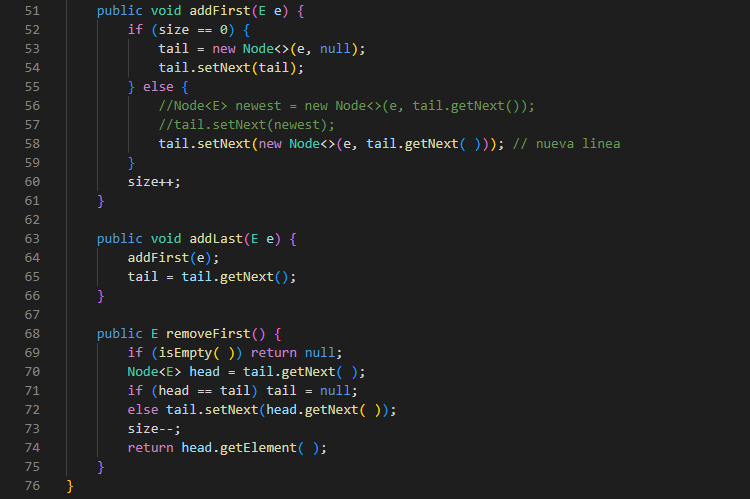




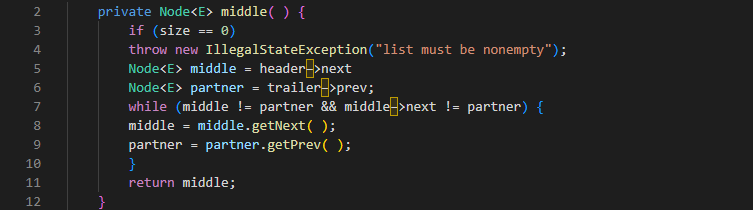
1. Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable.





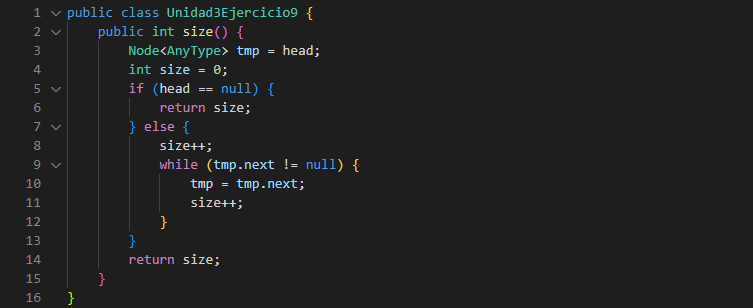


1. Describe a method for finding the middle node of a doubly linked list with header and trailer sentinels by “link hopping,” and without relying on explicit knowledge of the size of the list. In the case of an even number of nodes, report the node slightly left of center as the “middle.” What is the running time of this method?

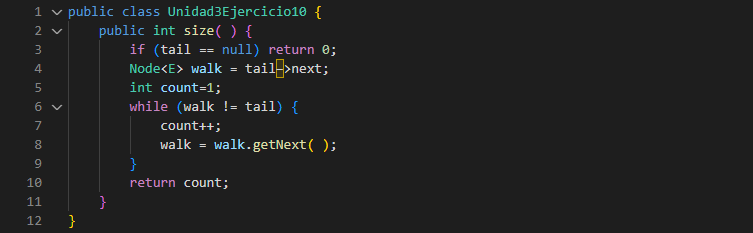


El método anterior se ejecuta en tiempo O(n)

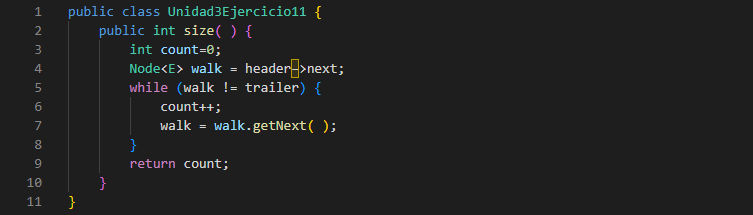
1. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.



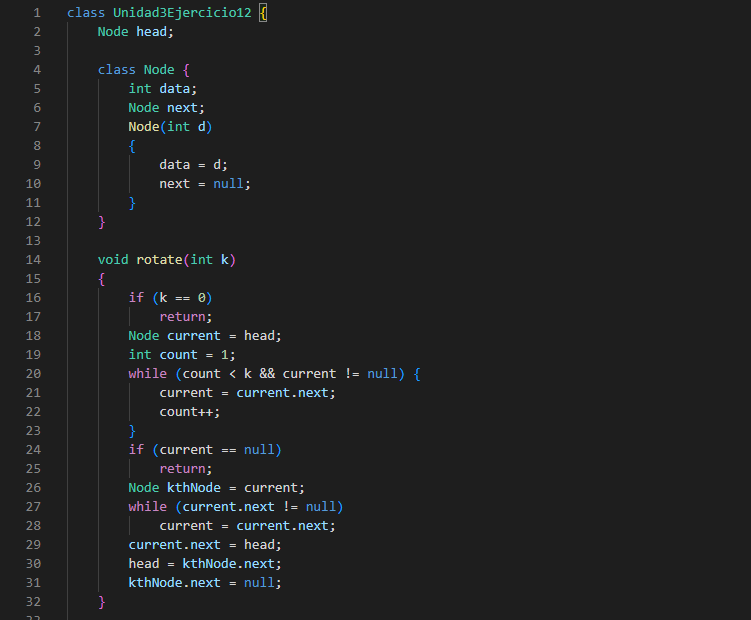
1. Give an implementation of the size( ) method for the CircularlyLinkedList class, assuming that we did not maintain size as an instance variable.

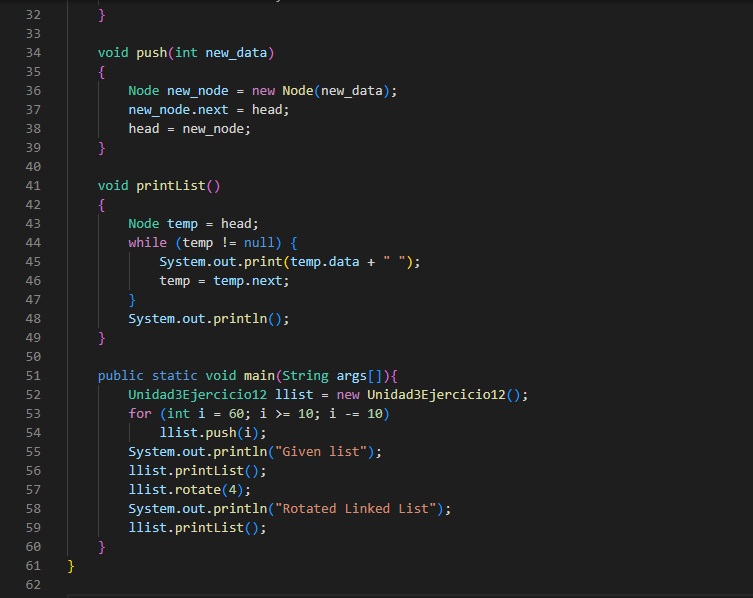


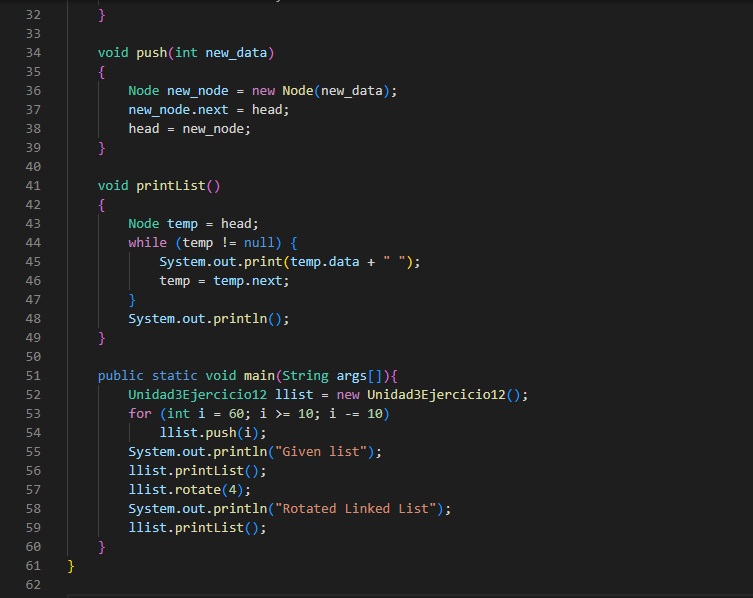
1. Give an implementation of the size() method for the DoublyLinkedList class, assuming that we did not maintain size as an instance variable.



1. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.







1. What is the difference between a shallow equality test and a deep equality test between two Java arrays, A and B, if they are one-dimensional arrays of type int? What if the arrays are two-dimensional arrays of type int?

Comparar (==) la identidad de los objetos, es decir, se comprueba si los operandos son iguales de hecho (ocupando la misma área de memoria), mientras que equals compara la equivalencia de los objetos, es decir, el valor "lógico" de dos objetos, posiblemente no idénticos, es el mismo. Si para dos objetos

a == b

Entonces es cierto que: a.equals(b) // si a!= null

Pero lo contrario no es cierto en todos los casos

La distinción superficial/profunda sólo tiene sentido para la comparación de iguales.

Superficial significa que comparas sólo los contenidos inmediatos de dos objetos para encontrar si son "iguales" en tu sentido, mientras que profundo significa que comparas los contenidos de tus objetos recursivamente hasta que todo lo que necesitas comparar son campos primitivos.

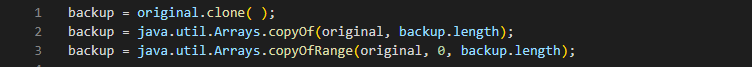
Si defines el método equals de tus objetos como una secuencia de llamadas a equals en los campos de instancia de estos objetos, utilizas la comparación profunda.

Si defines equals usando el operador == para comparar tipos compuestos, como Strings, entonces usas una comparación superficial -- y eso es incorrecto en Java.

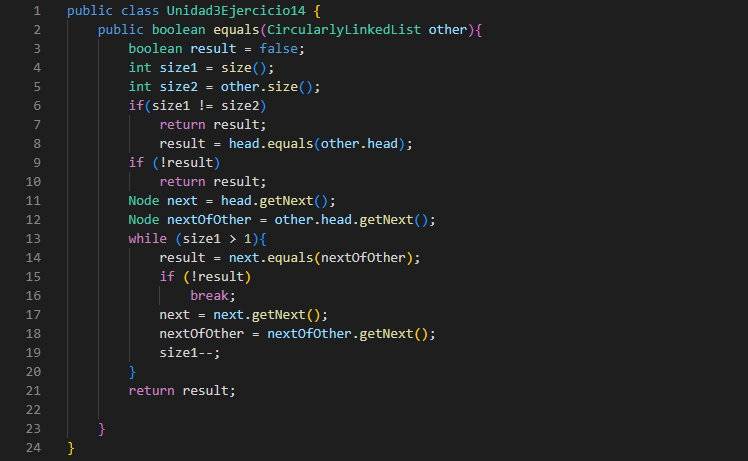
Además, nunca debes usar == para comparar dos objetos compuestos, a menos que los consideres iguales sólo si son iguales.

Una matriz bidimensional en Java es en realidad una matriz unidimensional de tal manera que cada entrada es a su vez una referencia a una matriz unidimensional.

1. 4 Give three different examples of a single Java statement that assigns variable, backup, to a new array with copies of all int entries of an existing array, original.



1. Implement the equals() method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list.



1. Implement the equals() method for the DoublyLinkedList class.

