1.Password

import java.nio.file.Files;  
import java.nio.file.Paths;  
  
public class DataReader {  
  
 public static String readFileAsString(String fileName)  
 throws Exception  
 {  
 String data = "";  
 data = new String(  
 Files.*readAllBytes*(Paths.*get*(fileName)));  
 return data;  
 }  
}

import java.util.Arrays;  
  
public class Main {  
 public static void main(String[] args) throws Exception {  
 int flag = 0;  
 String data = DataReader.*readFileAsString*(  
 "C:\\Users\\rmahto\\Documents\\passwords list.txt");  
 // System.out.println(data);  
  
 String[] individualConditionals\_passwords = data.split("\n");  
 //System.out.println("individual\_passwords::" +individual\_passwords);  
 for (int i = 0; i < individualConditionals\_passwords.length; i++) {  
 //System.out.println("individual\_passwords::" + individualConditionals\_passwords[i]);  
 String[] required\_count = individualConditionals\_passwords[i].split(":");  
 //System.out.println("required\_count::" + required\_count[0] + required\_count[1]);  
  
 String[] counter = (required\_count[0].split(" "));  
 String token=counter[1];  
 //System.out.println("token::" + counter[1]);  
 String[] counts = (counter[0].split("-"));  
 String minimumCount = counts[0];  
 String maximumCount = counts[1];  
 String password = required\_count[1];  
 System.*out*.println("minimumCount::" + minimumCount + " maxcount::" + maximumCount + " password::" + password);  
 if (password.contains(token)  
 && *countLettersInPass*(password, token.charAt(0)) < Integer.*valueOf*(minimumCount)  
 && *countLettersInPass*(password, token.charAt(0)) < Integer.*valueOf*(maximumCount)) {  
 System.*out*.println("Password is valid::");  
 flag++;  
 }  
 }  
 System.*out*.println("Number of Valid Passwords:" + flag);  
 }  
  
 public static int countLettersInPass(String password, char token) {  
 int count = 0;  
  
 for (int i = 0; i < password.length(); i++) {  
 if (password.charAt(i) == token) {  
 count++;  
 }  
 }  
 return count;  
 }  
}

2.DoublyLinkedList

public interface MyDoublyLinkedList<E> {  
 public void addFirstElement(E data);  
 public void addLastElement(E data);  
 public void moveForward();  
 public E removeFirstElement();  
 public E removeLastElement();  
 public void moveBackward();  
}

import java.util.NoSuchElementException;  
  
public class MyDoublyLinkedListImpl<E> implements MyDoublyLinkedList<E> {  
  
 private Node head;  
 private Node tail;  
 private int size;  
  
 private class Node {  
 E element;  
 Node next;  
 Node prev;  
  
 public Node(E element, Node next, Node prev) {  
 this.element = element;  
 this.next = next;  
 this.prev = prev;  
 }  
 }  
  
 */\*\*  
 \* adds 1st element to the doubly linked list  
 \* @param element  
 \*/* public void addFirstElement(E element) {  
 Node tmp = new Node(element, head, null);  
 if(head != null ) {head.prev = tmp;}  
 head = tmp;  
 if(tail == null) { tail = tmp;}  
 size++;  
 System.*out*.println("adding 1st element: "+element);  
 }  
  
 */\*\*  
 \* Adds last element to the doubly linked list  
 \* @param element  
 \*/* public void addLastElement(E element) {  
  
 Node tmp = new Node(element, null, tail);  
 if(tail != null) {tail.next = tmp;}  
 tail = tmp;  
 if(head == null) { head = tmp;}  
 size++;  
 System.*out*.println("adding: "+element);  
 }  
  
 */\*\*  
 \* Moving the cursor head  
 \*/* public void moveForward(){  
  
 System.*out*.println("moving forward:");  
 Node tmp = head;  
 while(tmp != null){  
 System.*out*.println(tmp.element);  
 tmp = tmp.next;  
 }  
 }  
  
 */\*\*  
 \* this method moves cursor tail  
 \*/* public void moveBackward(){  
  
 System.*out*.println("moving backward tail");  
 Node tmp = tail;  
 while(tmp != null){  
 System.*out*.println(tmp.element);  
 tmp = tmp.prev;  
 }  
 }  
  
 */\*\*  
 \* This method removes the first element from the doubly linked list  
 \* @return  
 \*/* public E removeFirstElement() {  
 if (size == 0) throw new NoSuchElementException();  
 Node tmp = head;  
 head = head.next;  
 head.prev = null;  
 size--;  
 System.*out*.println("deleted: "+tmp.element);  
 return tmp.element;  
 }  
  
 */\*\*  
 \* This method removes the first element from the doubly linked list  
 \* @return  
 \*/* public E removeLastElement() {  
 if (size == 0) throw new NoSuchElementException();  
 Node tmp = tail;  
 tail = tail.prev;  
 tail.next = null;  
 size--;  
 System.*out*.println("deleted: "+tmp.element);  
 return tmp.element;  
 }  
  
 }

public class Animal {  
}  
class Cat extends Animal{  
}  
class Dog extends Animal{  
}  
class Mouse extends Animal{  
}  
class Horse extends Animal{  
}  
class Bird extends Animal{  
}

// Press Shift twice to open the Search Everywhere dialog and type `show whitespaces`,  
// then press Enter. You can now see whitespace characters in your code.  
public class Main {  
 public static void main(String[] args) {  
 MyDoublyLinkedListImpl<Integer> myDoublyLinkedListImpl = new MyDoublyLinkedListImpl<Integer>();  
 myDoublyLinkedListImpl.addFirstElement(10);  
 myDoublyLinkedListImpl.addFirstElement(34);  
 myDoublyLinkedListImpl.addLastElement(56);  
 myDoublyLinkedListImpl.addLastElement(364);  
 myDoublyLinkedListImpl.moveForward();  
 myDoublyLinkedListImpl.removeFirstElement();  
 myDoublyLinkedListImpl.removeLastElement();  
 myDoublyLinkedListImpl.moveBackward();  
  
 MyDoublyLinkedListImpl<Animal> myDoublyLinkedListImpl1 = new MyDoublyLinkedListImpl<Animal>();  
 Dog d=new Dog();;  
 Cat c= new Cat();  
 Mouse m= new Mouse();  
 Horse h= new Horse();  
 Bird b=new Bird();  
 myDoublyLinkedListImpl1.addFirstElement(d);  
 myDoublyLinkedListImpl1.addFirstElement(c);  
 myDoublyLinkedListImpl1.addLastElement(m);  
 myDoublyLinkedListImpl1.addLastElement(h);  
 myDoublyLinkedListImpl1.addFirstElement(d);  
 myDoublyLinkedListImpl1.addFirstElement(b);  
 myDoublyLinkedListImpl1.moveForward();  
 myDoublyLinkedListImpl1.removeFirstElement();  
 myDoublyLinkedListImpl1.removeLastElement();  
 myDoublyLinkedListImpl1.moveBackward();  
  
 }  
}