

Avance de proyecto

Estructuras de datos

Adalberto Emmanuel
Rojas Perea

Documentación del código

Node.java

La clase **Node** toma como atributos el dato que va a contener **String**, y el puntero que apunta hacia adelante.

También cuenta con el constructor del Nodo, en el que se le asigna el dato que se le vaya a dar, y el puntero por defecto apunta hacia **null**.

```
Class > J Node.java > ...
1  package Class;
2
3  public class Node {
4
5      String data;
6      Node forward;
7
8      public Node(String data) {
9          this.data = data;
10         this.forward = null;
11     }
12 }
13
```

LinkedList.java

La clase **LinkedList** toma como atributos los Nodos cabeza y cola. Cuenta con el constructor en que el que asigna por defecto los Nodos cabeza y cola como **null**.

La clase cuenta con todos los métodos que manejan las operaciones de las listas, desde insertar, eliminar, mostrar y buscar tareas por departamento.

```

1 package Class;
2
3 public class LinkedList {
4
5     private static final String RESET = "\u001B[0m";
6     private static final String RED = "\u001B[31m";
7     private static final String BLUE = "\u001B[34m";
8     private static final String YELLOW = "\u001B[33m";
9
10    private Node head;
11    private Node tail;
12
13    > public LinkedList() { ...
17
18    > public boolean isEmpty() { ...
21
22    > public void insertFirst(String task) { ...
32
33    > public void insertLast(String task) { ...
43
44    > public void deleteFirst() { ...
57
58    > public void deleteLast() { ...
78
79    > public void findByDepartment(String department) { ...
101
102    > public void show() { ...
119 }

```

Stack.java

La clase **Stack** se inicializa con una capacidad máxima de 5 incidentes críticos, debido a que un negocio no debería de manejar con demasiadas a la vez, y esta cuenta con sus métodos de **push**, **pop**, **peek**, y **show**. Todo siguiendo con la lógica de LIFO

```

1 package Class;
2
3 public class Stack {
4
5     private static final String RESET = "\u001B[0m";
6     private static final String RED = "\u001B[31m";
7     private static final String CYAN = "\u001B[36m";
8     private static final String YELLOW = "\u001B[33m";
9
10    private int top;
11    private final int capacity = 5;
12    private String[] data;
13
14    > public Stack() { ...
18
19    > public boolean isEmpty() { ...
22
23    > public boolean isFull() { ...
26
27    > public void push(String urg_task) { ...
37
38    > public String pop() { ...
46
47    > public String peek() { ...
55
56    > public void show() { ...
66 }

```

Queue.java

La clase **Queue**, similar a Stack, se inicializa con una capacidad máxima para programar tareas, y siguiendo la lógica FIFO con sus métodos **enqueue**, **dequeue**, **peek** y **display**.

```
1 package Class;
2
3 public class Queue {
4     private static final String RESET = "\u001B[0m";
5     private static final String RED = "\u001B[31m";
6     private static final String CYAN = "\u001B[36m";
7     private static final String YELLOW = "\u001B[33m";
8
9     private int front;
10    private int rear;
11    private final int capacity = 40;
12    private String[] data;
13
14    > public Queue() { ...
19
20    > public boolean isEmpty() { ...
23
24    > public boolean isFull() { ...
27
28    > public void enqueue(String norm_task) { ...
44
45    > public String dequeue() { ...
63
64    > public String peek() { ...
72
73    > public void display() { ...
84    }
```

Main.java

La clase Main está estructurada donde el usuario puede gestionar diferentes tareas para la empresa **IT SOLUTIONS CUU**, donde puede gestionar incidentes críticos de clientes, programar tareas y gestionar tareas por departamentos dentro de la empresa.

La estructura del código está compuesta principalmente por **do while**, **switch**, **while** y **try_catch**, con el fin de que el programa sea resiliente ante diferentes errores de usuario y no termine la ejecución de este de manera abrupta; al igual que el programa muestre los menús de manera organizada.

```

15
16
17 Run | Debug
18 public static void main(String[] args) {
19
20     Queue queue = new Queue();
21     Stack stack = new Stack();
22     LinkedList linkedList = new LinkedList();
23     Scanner scanner = new Scanner(System.in);
24
25     int option = 0;
26
27     do {
28         System.out.println(YELLOW + "**** IT SOLUTIONS CUU ****" + RESET);
29         System.out.println(x:"1- Gestionar incidentes criticos\n2- Programar tareas\n3- Gestionar tareas por departamento\n4- Salir del programa");
30
31         try {
32             System.out.print(BLUE + "\nIngresa una opción: " + RESET);
33             option = scanner.nextInt();
34             scanner.nextLine();
35         } catch (InputMismatchException e) {
36             System.out.println(RED + "Error. Opción inválida." + RESET);
37             scanner.nextLine();
38         }
39
40         switch (option) {
41             case 1: // Tareas urgentes (Stack)
42                 int urgOption = 0;
43
44                 do {
45                     System.out.println(YELLOW + "\n*** Incidentes criticos ***" + RESET);
46                     System.out.println(x:"1- Agregar incidente critico\n2- Ver último incidente\n3- Marcar como resuelto el último incidente\n4- Ver todas los incidentes");
47
48                     try {
49                         System.out.print(BLUE + "\nIngresa una opción: " + RESET);
50                         urgOption = scanner.nextInt();
51                     } catch (InputMismatchException e) {
52                         System.out.println(RED + "Error. Opción inválida." + RESET);
53                         scanner.nextLine();
54                     }
55
56                     switch (urgOption) {
57                         case 1:
58                             System.out.print(CYAN + "\nDetalles del incidente (Ej. Servidor caído, BDD corrompida): " + RESET);
59                             String urgTask = scanner.nextLine();
60                             stack.push(urgTask);
61                             break;
62                         case 2:
63                             System.out.print(CYAN + "\nÚltimo incidente agregado: " + RESET + YELLOW + stack.peek() + RESET + "\n");
64                             break;
65                         case 3:
66                             String solvedStack = stack.pop();
67                             System.out.println(CYAN + "\nIncidente " + RESET + YELLOW + solvedStack + RESET + CYAN + " marcado como resuelto." + RESET);
68                             break;
69                         case 4:
70                             stack.show();
71                             break;
72                     }
73                 } while (urgOption != 4);
74             }
75         } while (option != 4);
76     }
77 }

```

```

39
40
41     switch (option) {
42         case 1: // Tareas urgentes (Stack)
43             int urgOption = 0;
44
45             do {
46                 System.out.println(YELLOW + "\n*** Incidentes criticos ***" + RESET);
47                 System.out.println(x:"1- Agregar incidente critico\n2- Ver último incidente\n3- Marcar como resuelto el último incidente\n4- Ver todas los incidentes");
48
49                 try {
50                     System.out.print(BLUE + "\nIngresa una opción: " + RESET);
51                     urgOption = scanner.nextInt();
52                     scanner.nextLine();
53                 } catch (InputMismatchException e) {
54                     System.out.println(RED + "Error." + RESET);
55                     scanner.nextLine();
56                 }
57
58                 switch (urgOption) {
59                     case 1:
60                         System.out.print(CYAN + "\nDetalles del incidente (Ej. Servidor caído, BDD corrompida): " + RESET);
61                         String urgTask = scanner.nextLine();
62                         stack.push(urgTask);
63                         break;
64                     case 2:
65                         System.out.print(CYAN + "\nÚltimo incidente agregado: " + RESET + YELLOW + stack.peek() + RESET + "\n");
66                         break;
67                     case 3:
68                         String solvedStack = stack.pop();
69                         System.out.println(CYAN + "\nIncidente " + RESET + YELLOW + solvedStack + RESET + CYAN + " marcado como resuelto." + RESET);
70                         break;
71                     case 4:
72                         stack.show();
73                         break;
74                 }
75             } while (urgOption != 4);
76         } while (option != 4);
77     }
78 }

```

Evidencias de funcionamiento

```
PS C:\Users\leona\Desktop\avance-de-proyecto-edd> & 'C:\Users\leona\AppData\Local\leona\AppData\Roaming\Code\User\workspaceStorage\96310efcdac087fb3189c5dd8f22
*** IT SOLUTIONS CUU ***
1- Gestionar incidentes críticos
2- Programar tareas
3- Gestionar tareas por departamento
4- Salir del programa

Ingresa una opción: 1

*** Incidentes críticos ***
1- Agregar incidente crítico
2- Ver último incidente
3- Marcar como resuelto el último incidente
4- Ver todas los incidentes críticos
5- Regresar al gestor de tareas

Ingresa una opción: 1

Detalles del incidente (Ej. Servidor caído, BDD corrompida): Servidor caído

*** Incidentes críticos ***
1- Agregar incidente crítico
2- Ver último incidente
3- Marcar como resuelto el último incidente
4- Ver todas los incidentes críticos
5- Regresar al gestor de tareas

Ingresa una opción: 2

Última incidente agregado: Servidor caído

*** Incidentes críticos ***
1- Agregar incidente crítico
2- Ver último incidente
3- Marcar como resuelto el último incidente
4- Ver todas los incidentes críticos
```