Colas en Java:

Una cola es una estructura de datos que sigue el principio de "primero en entrar, primero en salir" (FIFO, por sus siglas en inglés). En Java, puedes implementar una cola utilizando la interfaz Queue de la biblioteca estándar, o también puedes crear tu propia implementación utilizando listas enlazadas u otras estructuras de datos.

Aquí hay un ejemplo de cómo crear y utilizar una cola en Java utilizando la interfaz Queue:

- **offer(E elemento):** Este método permite agregar un elemento al final de la cola. Esencialmente, añade un nuevo elemento al final de la estructura de datos de la cola.
- **poll():** Este método elimina y devuelve el elemento que está en el frente de la cola. Es decir, quita el elemento más antiguo de la cola y lo devuelve. La cola se modifica después de esta operación, ya que el elemento eliminado ya no está presente en la cola.
- **peek():** Este método devuelve el elemento que está en el frente de la cola sin eliminarlo. Es útil para inspeccionar el elemento más antiguo de la cola sin modificar la estructura de la misma.
- **isEmpty():** Este método verifica si la cola está vacía o no. Devuelve true si la cola no contiene ningún elemento, y false en caso contrario. Es una operación comúnmente utilizada para determinar si se deben realizar más operaciones en la cola.
- **size():** Este método devuelve el número de elementos presentes en la cola. Proporciona información sobre la cantidad de elementos almacenados en la cola en un momento dado. Es útil para monitorear la capacidad y la carga de la cola.

Este ejercicio muestra cómo trabajar con una cola en Java. Primero, se agregan algunos números a la cola. Luego, se sacan dos números de la cola y se muestran. Después, se revisa el primer número de la cola sin sacarlo y se verifica si la cola está vacía. Es un ejemplo sencillo para entender cómo usar una cola en Java.

Ejemplo 1: Este ejercicio muestra cómo trabajar con una cola en Java. Primero, se agregan algunos números a la cola. Luego, se sacan dos números de la cola y se muestran. Después, se revisa el primer número de la cola sin sacarlo y se verifica si la cola está vacía. Es un ejemplo sencillo para entender cómo usar una cola en Java.

```
package View;

import java.util.Queue;
import java.util.LinkedList;

public class Ejemplo1Cola {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();

        // Agregar elementos a la cola
        queue.offer(10);
        queue.offer(20);
        queue.offer(30);
        queue.offer(19);
```

```
        int topElement;

        // Obtener y eliminar el primer elemento de la cola
        topElement = queue.poll();
        System.out.println("Primer elemento de la cola: " + topElement);

        // Obtener y eliminar el primer elemento de la cola
        topElement = queue.poll();
        System.out.println("Primer elemento de la cola: " + topElement);

        // Obtener el primer elemento de la cola sin eliminarlo
        int peekElement = queue.peek();
        System.out.println("Primer elemento de la cola (sin eliminar): " +
peekElement);

        // Verificar si la cola está vacía
        boolean isEmpty = queue.isEmpty();
        System.out.println("La cola está vacía: " + isEmpty);
    }
}
```

Ejemplo 2: Este programa ejemplifica el uso de una cola en Java mediante el empleo de métodos como offer(), poll(), peek() y isEmpty(). A través de un menú interactivo, permite al usuario agregar elementos a la cola, eliminar el primer elemento, obtener el primer elemento sin eliminarlo y verificar si la cola está vacía. Además de proporcionar una comprensión práctica de cómo trabajar con colas en Java, el programa busca ofrecer una experiencia de usuario intuitiva y amigable mediante el uso de una interfaz gráfica sencilla.

```
package View;

import javax.swing.JOptionPane;
import java.util.Queue;
import java.util.LinkedList;

public class Ejemplo2Cola {

    public static void main(String[] args) {
        Queue<Integer> cola = new LinkedList<>();

        while (true) {
            int opcion = mostrarMenu();
            try {
                switch (opcion) {
                    case 1:
                        int elemento =
Integer.parseInt(JOptionPane.showInputDialog("Ingrese el elemento a agregar:"));
                        agregarElemento(cola, elemento);
                        break;
                    case 2:
                        eliminarElemento(cola);
                        break;
                    case 3:
```

```
                        int frenteCola = obtenerElementoFrente(cola);
                        JOptionPane.showMessageDialog(null, "Elemento en el frente
de la cola: " + frenteCola);
                        break;
                    case 4:
                        boolean estaVacia = estaVacia(cola);
                        JOptionPane.showMessageDialog(null, "La cola está vacía: "
+ estaVacia);
                        break;
                    case 5:
                        JOptionPane.showMessageDialog(null, "Saliendo del
programa...");
                        System.exit(0);
                        break;
                    default:
                        JOptionPane.showMessageDialog(null, "Opción inválida. Por
favor, seleccione nuevamente.");
                }
            } catch (Exception e) {
                JOptionPane.showMessageDialog(null, "La cola está vacía. No se
puede realizar esta operación.");
            }
        }
    }

    public static int mostrarMenu() {
        String[] opciones = {"Agregar elemento", "Eliminar elemento", "Obtener
elemento en el frente",
                "Verificar si la cola está vacía", "Salir del programa"};
        return JOptionPane.showOptionDialog(null, "Seleccione una opción:",
"Menú", JOptionPane.DEFAULT_OPTION,
                JOptionPane.INFORMATION_MESSAGE, null, opciones, opciones[0]) + 1;
    }

    public static void agregarElemento(Queue<Integer> cola, int elemento) {
        cola.offer(elemento);
        JOptionPane.showMessageDialog(null, "Elemento agregado correctamente.");
    }

    public static int obtenerElementoFrente(Queue<Integer> cola) {
        return cola.peek();
    }

    public static void eliminarElemento(Queue<Integer> cola) {
        cola.poll();
        JOptionPane.showMessageDialog(null, "Elemento en el frente eliminado
correctamente.");
    }

    public static boolean estaVacia(Queue<Integer> cola) {
        return cola.isEmpty();
    }
}
```

Ejemplo 3: Comienza inicializando la cola con tres elementos predefinidos. Luego, se llena la cola con tres números aleatorios generados por la clase Random. Después de añadir dos elementos adicionales a la cola, el programa muestra todos los elementos de la cola uno por uno hasta que esté vacía. Este ejemplo demuestra cómo trabajar con colas en Java, agregando elementos, llenándola con datos aleatorios y luego vaciándola para mostrar su contenido de manera secuencial.

```
package View;

import javax.swing.JOptionPane;
import java.util.Queue;
import java.util.LinkedList;
import java.util.Random;

public class Ejemplo3Cola {

    public static void main(String[] args) {
        Queue<Integer> cola = new LinkedList<>();

        //Quemar tres datos
        cola.offer(10);
        cola.offer(20);
        cola.offer(30);

        // Llenar la cola con números aleatorios
        llenarCola(cola);

        // Luego quemar dos más:
        cola.offer(12);
        cola.offer(9);

        // Mostrar los datos de la cola hasta que esté vacía
        mostrarDatosCola(cola);
    }

    public static void llenarCola(Queue<Integer> cola) {
        Random rand = new Random();
        int numElementos = 3;
        for (int i = 0; i < numElementos; i++) {
            cola.offer(rand.nextInt(100)); // Generar números aleatorios entre 0 y
99
        }
    }

    public static void mostrarDatosCola(Queue<Integer> cola) {
        JOptionPane.showMessageDialog(null, "Mostrando los datos de la cola:");
        while (!cola.isEmpty()) {
            int elemento = cola.poll();
            JOptionPane.showMessageDialog(null, "Elemento de la cola: " +
elemento);
        }
        JOptionPane.showMessageDialog(null, "La cola está vacía.");
```

```
    }
}
```

Ejemplo 4: Este código demuestra una situación en la que se manipula una cola en Java, mientras se mantiene un registro de los elementos eliminados y su sucesor inmediato en la cola. Comienza agregando varios números a la cola. Luego, utiliza un ciclo para eliminar los elementos de la cola uno por uno y guardar cada elemento eliminado junto con su sucesor en una lista. Esta lista se utiliza para mostrar los elementos eliminados y su sucesor correspondiente en la cola. Es un ejemplo práctico que ilustra cómo trabajar con colas en Java y cómo mantener un registro de las operaciones realizadas en la cola.

```
package View;

import java.util.LinkedList;
import java.util.ArrayList;
import java.util.EmptyStackException;

public class Ejemplo4Cola {
    public static void main(String[] args) {
        LinkedList<Integer> cola = new LinkedList<>();
        ArrayList<ArrayList<Integer>> listaEliminados = new ArrayList<>();

        // Agregar elementos a la cola
        cola.offer(10);
        cola.offer(20);
        cola.offer(30);
        cola.offer(19);
        cola.offer(21);
        cola.offer(15);

        // Ciclo para eliminar elementos de la cola uno por uno
        while (!cola.isEmpty()) {
            try {
                // Eliminar elemento de la cola y guardar en lista
                int eliminado = cola.poll();
                ArrayList<Integer> pareja = new ArrayList<>();
                pareja.add(eliminado);
                if (!cola.isEmpty()) {
                    pareja.add(cola.peek());
                } else {
                    pareja.add(null);
                }
                listaEliminados.add(pareja);
            } catch (EmptyStackException e) {
                System.out.println("Se han eliminado todos los elementos de la
cola.");
                break;
            }
        }

        // Mostrar lista con las parejas de elementos
        System.out.println("Elemento Eliminado | Siguiente Elemento");
```

```
        for (ArrayList<Integer> pareja : listaEliminados) {
            System.out.println(pareja.get(0) + "                | " +
(pareja.get(1) != null ? pareja.get(1) : "N/A"));
        }
    }
}
```

Ejemplo 5: Este código Java crea una cola de tamaño 5 y la llena con números del 1 al 5. Luego, crea objetos Libro utilizando los elementos de la cola como IDs y generando títulos y autores dinámicamente. Estos libros se agregan a una lista y se imprimen sus detalles, como ID, título y autor. Este ejemplo ilustra cómo trabajar con colas, crear objetos dinámicamente y usar listas en Java.

Entity

```
package Model;

public class Libro {
    private int id;
    private String titulo;
    private String autor;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }
}
```

View

Ejercicios de Evaluación sobre Colas en Java

1. **Implementación de una Cola Personalizada:**
    - Propón desarrollar una versión personalizada de la clase Cola en Java, utilizando una lista enlazada como base. Esta implementación debe incluir los métodos `offer()`, `poll()`, `peek()`, `isEmpty()` y `size()` para manipular la cola. Además, considera la adición de cualquier otro método útil que creas necesario.
2. **Aplicación de una Cola en un Sistema de Mensajería:**
    - Diseña una aplicación de mensajería que utilice una cola para gestionar los mensajes entrantes de los usuarios. Cada vez que un usuario envíe un mensaje, este se agregará a la cola de mensajes pendientes. El sistema deberá procesar los mensajes en el orden en que se recibieron, es decir, siguiendo el principio de "primero en entrar, primero en salir". Implementa una función que permita enviar mensajes y otra para recibirlos de la cola.
3. **Sistema de Gestión de Turnos en un Consultorio Dental:**
    - Desarrolla un sistema de gestión de turnos para un consultorio dental que utilice una cola para organizar las citas de los pacientes. Al iniciar el programa, el usuario podrá asignar una cantidad de turnos aleatoria, que determinará el tamaño inicial de la cola de turnos. Una vez asignados los turnos, el programa permitirá atender a los pacientes, extrayendo turnos de la cola.