Pilas en Java:

Una pila es una estructura de datos que sigue el principio de "último en entrar, primero en salir" (LIFO, por sus siglas en inglés). En Java, puedes implementar una pila utilizando la clase Stack de la biblioteca estándar, o también puedes crear tu propia implementación utilizando arreglos o listas enlazadas.

Aquí hay un ejemplo de cómo crear y utilizar una pila en Java utilizando la clase Stack:

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

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

```java
package View;

import java.util.Stack;
public class Ejemplo1Pila {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        // Agregar elementos a la pila
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(19);

        int topElement;

        // Obtener el elemento superior de la pila y eliminarlo
        topElement = stack.pop();
        System.out.println("Elemento superior de la pila: " + topElement);
```

```
        // Obtener el elemento superior de la pila y eliminarlo
        topElement = stack.pop();
        System.out.println("Elemento superior de la pila: " + topElement);

        // Obtener el elemento superior de la pila sin eliminarlo
        int peekElement = stack.peek();
        System.out.println("Elemento superior de la pila (sin eliminar): " +
peekElement);

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

Ejemplo 2: Claro, aquí tienes: Este programa ejemplifica el uso de una pila en Java mediante el empleo de métodos como push(), pop(), peek() y isEmpty(). A través de un menú interactivo, permite al usuario agregar elementos a la pila, eliminar el elemento superior, obtener el elemento en la cima sin eliminarlo y verificar si la pila está vacía. Además de proporcionar una comprensión práctica de cómo trabajar con pilas 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.EmptyStackException;
import java.util.Stack;

public class Ejemplo2Pila {

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        while (true) {
            int opcion = mostrarMenu();
            try {
                switch (opcion) {
                    case 1:
                        int elemento =
Integer.parseInt(JOptionPane.showInputDialog("Ingrese el elemento a agregar:"));
                        agregarElemento(stack, elemento);
                        break;
                    case 2:
                        eliminarElemento(stack);
                        break;
                    case 3:
                        int topElement = obtenerElementoSuperior(stack);
                        JOptionPane.showMessageDialog(null, "Elemento superior de
la pila: " + topElement);
                        break;
                    case 4:
```

```
                        boolean isEmpty = estaVacia(stack);
                        JOptionPane.showMessageDialog(null, "La pila está vacía: "
+ isEmpty);
                        break;
                    case 5:
                        // Salir del programa
                        JOptionPane.showMessageDialog(null, "Saliendo del
programa...");
                        System.exit(0);
                        break;
                    default:
                        JOptionPane.showMessageDialog(null, "Opción inválida. Por
favor, seleccione nuevamente.");
                }
            } catch (EmptyStackException e) {
                JOptionPane.showMessageDialog(null, "La pila está vacía. No se
puede realizar esta operación.");
            }
        }
    }

    public static int mostrarMenu() {
        String[] opciones = {"Agregar elemento", "Eliminar elemento", "Obtener
elemento superior",
                "Verificar si la pila 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(Stack<Integer> stack, int elemento) {
        stack.push(elemento);
        JOptionPane.showMessageDialog(null, "Elemento agregado correctamente.");
    }

    public static int obtenerElementoSuperior(Stack<Integer> stack) {
        return stack.peek();
    }

    public static void eliminarElemento(Stack<Integer> stack) {
        stack.pop();
        JOptionPane.showMessageDialog(null, "Elemento eliminado correctamente.");
    }

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

Ejemplo 3: Comienza inicializando la pila con tres elementos predefinidos. Luego, se llena la pila con tres números aleatorios generados por la clase Random. Después de añadir dos elementos adicionales a la pila, el programa muestra todos los elementos de la pila uno por uno hasta que esté vacía. Este ejemplo demuestra

cómo trabajar con pilas 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.Stack;
import java.util.Random;

public class Ejemplo3Pila {

    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();

        //Quemar tres datos
        stack.push(10);
        stack.push(20);
        stack.push(30);

        // Llenar la pila con números aleatorios
        llenarPila(stack);

        // Luego quemar dos más:
        stack.push(12);
        stack.push(9);

        // Mostrar los datos de la pila hasta que esté vacía
        mostrarDatosPila(stack);
    }

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

    public static void mostrarDatosPila(Stack<Integer> stack) {
        JOptionPane.showMessageDialog(null, "Mostrando los datos de la pila:");
        while (!stack.isEmpty()) {
            int elemento = stack.pop();
            JOptionPane.showMessageDialog(null, "Elemento de la pila: " +
elemento);
        }
        JOptionPane.showMessageDialog(null, "La pila está vacía.");
    }
}
```

Ejemplo 4: Este código demuestra una situación en la que se manipula una pila en Java, mientras se mantiene un registro de los elementos eliminados y su sucesor inmediato en la pila. Comienza agregando varios números a la pila. Luego, utiliza un ciclo para eliminar los elementos de la pila 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 pila. Es un ejemplo práctico que ilustra cómo trabajar con pilas en Java y cómo mantener un registro de las operaciones realizadas en la pila.

```
package View;

import java.util.Stack;
import java.util.ArrayList;
import java.util.EmptyStackException;

public class Ejemplo4Pila {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>();
        ArrayList<ArrayList<Integer>> listaEliminados = new ArrayList<>();

        // Agregar elementos a la pila
        stack.push(10);
        stack.push(20);
        stack.push(30);
        stack.push(19);
        stack.push(21);
        stack.push(15);

        // Ciclo para eliminar elementos de la pila uno por uno
        while (!stack.isEmpty()) {
            try {
                // Eliminar elemento de la pila y guardar en lista
                int eliminado = stack.pop();
                ArrayList<Integer> pareja = new ArrayList<>();
                pareja.add(eliminado);
                if (!stack.isEmpty()) {
                    pareja.add(stack.peek());
                } else {
                    pareja.add(null);
                }
                listaEliminados.add(pareja);
            } catch (EmptyStackException e) {
                System.out.println("Se han eliminado todos los elementos de la
pila.");
                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 pila de tamaño 5 y la llena con números del 1 al 5. Luego, crea objetos Libro usando los elementos de la pila 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 pilas, 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

```
package View;

import java.util.ArrayList;
import java.util.Random;
import java.util.Stack;
```

```
import Model.Libro;

public class LibroPila {

    public static void main(String[] args) {
        Stack<Integer> pila = new Stack<>();
        int n = 5;

        llenarPila(pila, n);

        ArrayList<Libro> libros = new ArrayList<>(); // Se crea la lista de libros

        // Iterar sobre la pila y crear libros con los datos correspondientes
        for (int i = 0; i < n; i++) {
            Libro libro = new Libro();
            libro.setId(pila.peek());
            libro.setTitulo("Título del Libro " + (i + 1));
            libro.setAutor("Autor del Libro " + (i + 1));
            libros.add(libro);
            pila.pop();
        }

        // Aquí mostrar los libros de la lista
        for (Libro libro : libros) {
            System.out.println("ID: " + libro.getId() + ", Título: " +
libro.getTitulo() + ", Autor: " + libro.getAutor());
        }
    }

    public static void llenarPila(Stack<Integer> pila, int n) {
        Random rand = new Random();
        for (int i = 1; i <= n; i++) {
            pila.push(i);
        }
    }
}
```

Ejercicios de Evaluación sobre Pilas en Java

1. **Implementación de una Pila Personalizada:**
   - Implementar una versión personalizada de la clase Pila en Java, utilizando una lista enlazada como base. Deberá incluir los métodos `push()`, `pop()`, `peek()`, `isEmpty()` y `size()` para manipular la pila, así como cualquier otro método adicional que considere útil.
2. **Aplicación de una Pila en una Aplicación de Edición de Texto:**
   - Desarrollar una aplicación de edición de texto que utilice una pila para implementar la funcionalidad "Deshacer". Cada vez que el usuario realice una acción (como escribir, eliminar texto, etc.), se deberá almacenar la acción en una pila. Cuando el usuario desee deshacer una acción, se recuperará la última acción de la pila y se revertirá la edición.

3. **Sistema de Gestión de Historiales Médicos:**

    - Desarrollar un sistema de gestión de citas médicas que utilice una pila para almacenar las citas pendientes. El programa permitirá al usuario asignar una cantidad aleatoria de citas médicas, donde la cantidad exacta será redondeada al entero más cercano. Esta cantidad determinará el tamaño inicial de la pila de citas.

    - Una vez asignadas las citas, el programa permitirá atender a los pacientes, extrayendo citas de la pila. A medida que se atiendan pacientes, las citas se irán agotando hasta que la pila quede vacía. En caso de que no haya más citas disponibles, se mostrará un mensaje indicando que se han agotado las citas.

    - Cada cita médica estará representada por un objeto de la clase `Cita`, que contendrá información relevante como la fecha, el diagnóstico, el tratamiento, etc. Estos datos se guardarán en una lista de citas, lo que permitirá la posterior búsqueda y visualización de los historiales médicos de los pacientes.