UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

ALUMNO:

PAZ MALDONADO CARLOS SAÚL

NOMBRE DEL PROFESOR:

HERNANDEZ CABRERA JESUS

NOMBRE DE LA MATERIA:

ESTRUCTURA DE DATOS

FECHA DE ENTREGA:

22 de octubre de 2024

TAREA NO. 11

INSTUCCIONES DE TAREA

Resolver con recursividad 2 problemas:

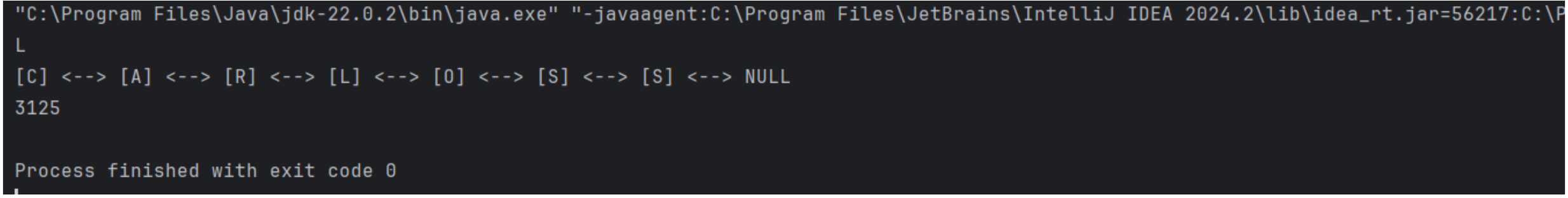
- - El primero es el problema de sacar de un ADT pila el valor en la posición media con recursión.

Y el segundo deben elegir uno de los siguientes.

- - Crear una
  lista de enteros en Python y realizar la suma con recursividad, el caso base es
  cuando la lista esta vacia.
- - Hacer un
  contador regresivo con recursión.
- - Dado un número
  entero positivo, escribe un método recursivo que calcule la suma de sus
  dígitos.
- - Escribe un método recursivo para calcular la potencia de un número a elevado a b. a^b.

Seleccione el metodo para calcular la potencia de un numero elevado a otro.

CAPTURA DE EJECUCIÓN DEL PROGRAMA



CÓDIGO

Clase **Recursividad:**

```java
package unam.fesaragon.estructuradatos.recursividad;

import unam.fesaragon.estructuradatos.adts.ADTStack;

public class Recursividad {
    public static void main(String[] args) {
        ADTStack<String> enteros = new ADTStack<>();
        enteros.push("C");
        enteros.push("A");
        enteros.push("R");
        enteros.push("L");
        enteros.push("O");
        enteros.push("S");
        enteros.push("S");
        System.out.println(posicionMediaPila(enteros, enteros.length()));
        enteros.imprimirStack();
        int resultado = potencia(5,5);
        System.out.println(resultado);
    }

    public static Object posicionMediaPila(ADTStack pila, int
tamanioOriginal) {
        if (tamanioOriginal % 2 == 0) {
            System.out.println("Es pila par");
            return null;
        }
        if (pila.length() == tamanioOriginal / 2 + 1) {
            return pila.peek();
        }
        Object elementoQuitado = pila.pop();
        Object medio = posicionMediaPila(pila, tamanioOriginal);
        pila.push(elementoQuitado);
        return medio;
    }

    public static int potencia(int base, int exponente) {
        if (exponente == 0) {
            return 1;
        } else {
            return base * potencia(base, exponente - 1);
        }
    }

}
```

Clase **NodoDoble**

```java
package unam.fesaragon.estructuradatos.adts;

public class NodoDoble<T> {
    private T dato;
    private NodoDoble<T> anterior;
    private NodoDoble<T> siguiente;

    // Constructores
    public NodoDoble() {
    }

    public NodoDoble(T dato) {
        this.dato = dato;
        this.anterior = null;
        this.siguiente = null;
```

```
    }

    public NodoDoble(T dato, NodoDoble<T> anterior, NodoDoble<T>
siguiente) {
        this.dato = dato;
        this.anterior = anterior;
        this.siguiente = siguiente;
    }

    // Getters y Setters
    public T getDato() {
        return dato;
    }

    public void setDato(T dato) {
        this.dato = dato;
    }

    public NodoDoble<T> getAnterior() {
        return anterior;
    }

    public void setAnterior(NodoDoble<T> anterior) {
        this.anterior = anterior;
    }

    public NodoDoble<T> getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoDoble<T> siguiente) {
        this.siguiente = siguiente;
    }
}
```

Clase **ListaDoblementeLigada**

```
package unam.fesaragon.estructuradatos.adts;

public class ListaDoblementeLigada<T> {
    private NodoDoble<T> head;
    private NodoDoble<T> tail;
    private int tamanio;

    // Constructor
    public ListaDoblementeLigada() {
        this.head = null;
        this.tail = null;
        this.tamanio = 0;
    }

    public ListaDoblementeLigada(NodoDoble<T> head, NodoDoble<T> tail) {
        this.head = head;
        this.tail = tail;
    }

    // Comprobar si está vacía
    public boolean esta_vacia() {
        return this.tamanio == 0;
    }

    // Agregar al inicio de la lista
    public void agregar_al_inicio(T valor) {
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        if (esta_vacia()) {
            this.head = nuevo;
```

```
            this.tail = nuevo;
        } else {
            nuevo.setSiguiente(this.head);
            this.head.setAnterior(nuevo);
            this.head = nuevo;
        }
        tamanio++;
    }

    // Agregar al final de la lista
    public void agregar_al_final(T valor) {
        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        if (esta_vacia()) {
            this.head = nuevo;
            this.tail = nuevo;
        } else {
            this.tail.setSiguiente(nuevo);
            nuevo.setAnterior(this.tail);
            this.tail = nuevo;
        }
        tamanio++;
    }

    // Agregar después de un nodo de referencia
    public void agregar_después_de(T referencia, T valor) {
        if (esta_vacia()) {
            System.out.println("La lista esta vacia y por lo tanto no
existe la referencia");
            return;
        }

        NodoDoble<T> aux = this.head;
        while (aux != null) {
            if (aux.getDato().equals(referencia)) {
                break;
            }
            aux = aux.getSiguiente();
        }

        if (aux == null) {
            System.out.println("El nodo de referencia no existe");
            return;
        }

        NodoDoble<T> nuevo = new NodoDoble<>(valor);
        nuevo.setSiguiente(aux.getSiguiente());
        nuevo.setAnterior(aux);
        //Nodo de referencia, no es la ultima en la lista.
        if (aux.getSiguiente() != null) {
            aux.getSiguiente().setAnterior(nuevo);
        } else {
            this.tail = nuevo;
        }
        aux.setSiguiente(nuevo);
        this.tamanio++;
    }

    // Obtener el elemento en una posición específica
    public T obtener(int posicion) {
        //Condiciones
        if (esta_vacia()) {
            System.out.println("La lista está vacía");
            return null;
        }
        if (posicion > this.tamanio || posicion<0) {
            System.out.println("Posición fuera de rango");
```

```
            return null;
        }

        NodoDoble<T> aux = this.head;
        for (int i = 0; i < posicion; i++) {
            aux = aux.getSiguiente();
        }

        return aux.getDato();
    }

    // Eliminar el primer elemento
    public void eliminar_el_primero() {
        if (esta_vacia()) {
            System.out.println("La lista está vacía");
            return;
        }
        if (this.head == this.tail) { // Solo un elemento
            this.head = null;
            this.tail = null;
        } else {
            this.head = this.head.getSiguiente();
            this.head.setAnterior(null);
        }
        tamanio--;
    }

    // Eliminar el último elemento
    public void eliminar_el_final() {
        if (esta_vacia()) {
            System.out.println("La lista está vacía");
            return;
        }

        if (this.head == this.tail) { // Solo un elemento
            this.head = null;
            this.tail = null;
        } else {
            this.tail = this.tail.getAnterior();
            this.tail.setSiguiente(null);
        }
        tamanio--;
    }

    // Eliminar un elemento en una posición específica
    public void eliminar(int posicion) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }
        if (posicion >= this.tamanio || posicion < 0) {
            System.out.println("Posición fuera de rango");
            return;
        }

        if (posicion == 0) {
            eliminar_el_primero();
            return;
        }
        if (posicion == this.tamanio - 1) {
            eliminar_el_final();
            return;
        }

        NodoDoble<T> aux = this.head;
        int i = 0;
```

```
        while (i < posicion) {
            aux = aux.getSiguiente();
            i++;
        }

        aux.getAnterior().setSiguiente(aux.getSiguiente());
        aux.getSiguiente().setAnterior(aux.getAnterior());
        this.tamanio--;
    }


    // Buscar un elemento y retornar su posición
    public int buscar(T valor) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía");
            return 0;
        }

        NodoDoble<T> aux = this.head;
        int posicion = 0;
        while (aux != null) {
            if (aux.getDato().equals(valor)) {
                return posicion;
            }
            aux = aux.getSiguiente();
            posicion++;
        }
        System.out.println("No se encontro el elemento");
        return 0;
    }

    // Actualizar un elemento
    public void actualizar(T a_buscar, T valor) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        NodoDoble<T> aux = this.head;
        while (aux != null) {
            if (aux.getDato().equals(a_buscar)) {
                aux.setDato(valor);
                return;
            }
            aux = aux.getSiguiente();
        }
        System.out.println("El elemento valor o elemento no esta en la
lista");
    }

    // Recorrido transversal en una dirección específica
    // Verdadero es valor por defecto y con el falso es de derecha a
izquierda
    public void transversal(boolean porDefecto) {
        if (esta_vacia()) {
            System.out.println("La lista está vacía.");
            return;
        }

        if (porDefecto) {
            NodoDoble<T> aux = this.head;
            while (aux != null) {
                System.out.print("[" + aux.getDato() + "] <--> ");
                aux = aux.getSiguiente();
            }
        } else {
```

```
            NodoDoble<T> aux = this.tail;
            while (aux != null) {
                System.out.print("[" + aux.getDato() + "] <--> ");
                aux = aux.getAnterior();
            }
        }
        System.out.println("NULL");
    }
    //Sobrecargue el metodo para poner por default el recorrido de
izquierda a derecha
    public void transversal() {
        transversal(true);
    }

    //GETTERS Y SETTERS
    // Obtener el tamaño de la lista
    public int get_tamanio() {
        return this.tamanio;
    }

    //ToString
    @Override
    public String toString() {
        StringBuilder estado = new StringBuilder();
        NodoDoble<T> aux = this.head;
        while (aux != null) {
            estado.append("[").append(aux.getDato()).append("] <--> ");
            aux = aux.getSiguiente();
        }
        return estado +" Tamaño: "+ this.tamanio;
    }
}
```

Clase **ADTStack**

```
package unam.fesaragon.estructuradatos.adts;

public class ADTStack<T> {
    private ListaDoblementeLigada<T> datos;

    public ADTStack() {
        datos = new ListaDoblementeLigada<T>();
    }
    public boolean isEmpty(){
        return datos.esta_vacia();
    }
    public int length(){
        return datos.get_tamanio();
    }
    public T pop(){
        T datoASacar = datos.obtener(datos.get_tamanio()-1);
        datos.eliminar_el_final();
        return datoASacar;
    }
    public T peek(){
        return datos.obtener(datos.get_tamanio()-1);
    }
    public void push(T datoAInsertar){
        datos.agregar_al_final(datoAInsertar);
    }

    public void imprimirStack(){
        datos.transversal();
    }
}
```