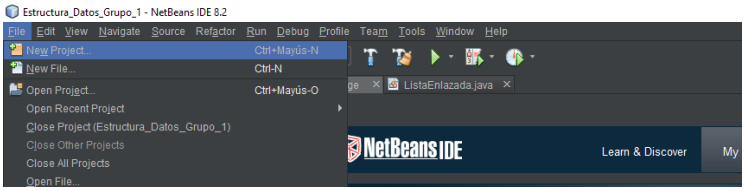
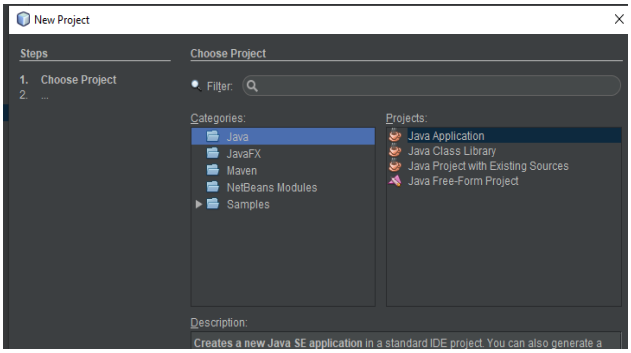
ESTRUCTURA DE DATOS - LISTAS SIMPLES

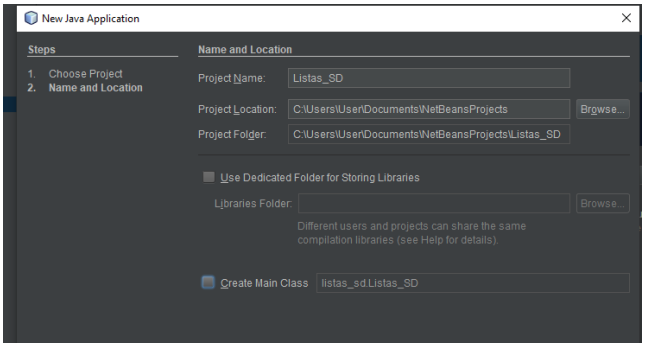
1. Iniciamos creando un proyecto



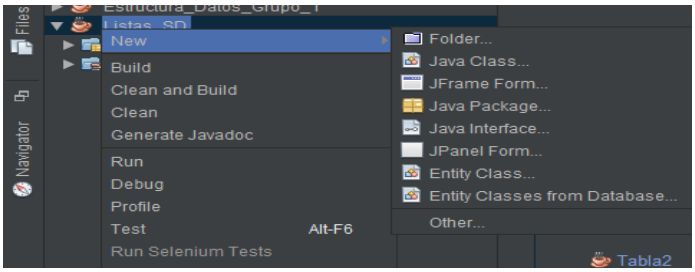
2. Escogemos la categoría



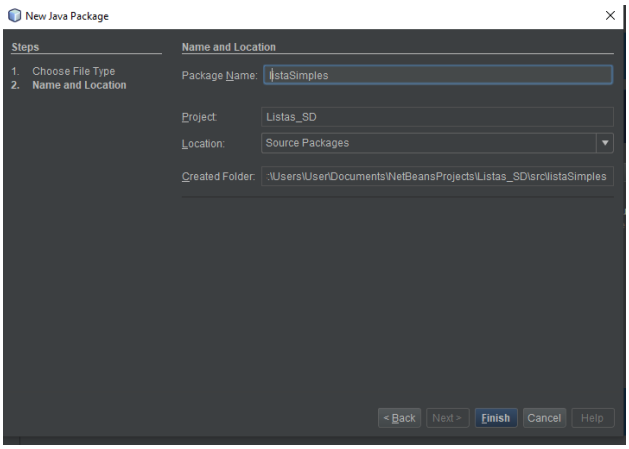
3. Le indicamos un nombre, este archivo se llamará Listas SD, no seleccionar la clase main



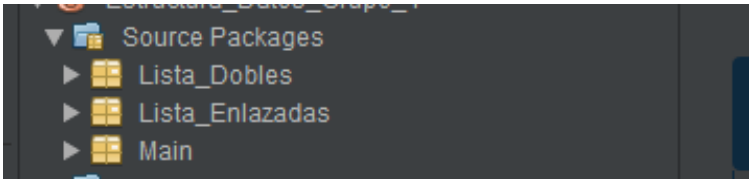
4. Se inicia creando la paquetería



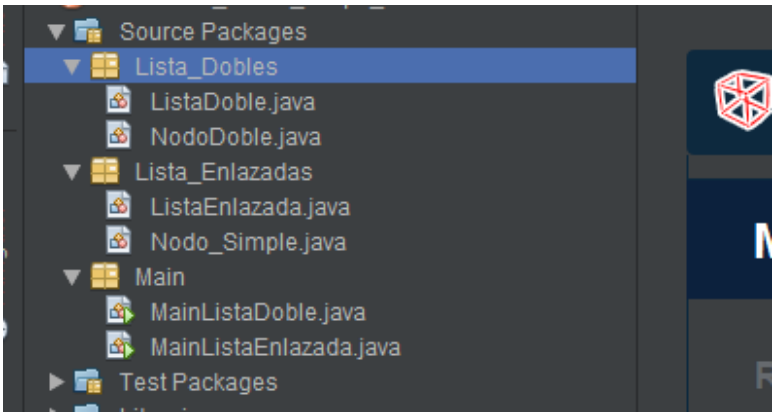
5. Paquetes con sus respectivas clases



6. La idea es tener 3 paqueterías la de listas simples, listas dobles y el main



7. Las cuales contienen las siguientes clases



8. Empezamos con la clase nodos simples

```
*/
public class Nodo_Simple {

    private int dato;
    private Nodo_Simple siguiente;

    public Nodo_Simple(int dato) {
        this.dato = dato;
        this.siguiente = null;
    }

    public int getDato() {
        return dato;
    }

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

    public Nodo_Simple getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(Nodo_Simple siguiente) {
        this.siguiente = siguiente;
    }

}
```

private int dato: Esto define un atributo privado llamado dato de tipo int. Este atributo almacena el valor del nodo

```
public class Nodo_Simple {

    private int dato;
    private Nodo_Simple siguiente;
```

private Nodo _Simple: Este es el constructor de la clase. Recibe un parámetro dato y lo asigna al atributo correspondiente. Además, inicializa el atributo siguiente como null

```
public Nodo_Simple(int dato) {
    this.dato = dato;
    this.siguiente = null;
}
```

public int getDato. Este es un método público que devuelve el valor del atributo dato

```
public int getDato() {
    return dato;
}
```

Public void setDato: Este es un método público que establece el valor del atributo dato

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

public Nodo_Simple getSiguiente: Este es un método público que devuelve el siguiente nodo de la lista.

```
public Nodo_Simple getSiguiente() {
    return siguiente;
}
```

public void set Siguiente: Este es un método público que establece el siguiente nodo de la lista.

```
}

public void setSiguiente(Nodo_Simple siguiente) {
    this.siguiente = siguiente;
}
```

9. Siguiente a esto continuamos con la clase lista enlazada se crearán los siguientes métodos.

- **Metodo lista enlazada**

```
    private Nodo_Simple inicio;

    public ListaEnlazada() {
        this.inicio = null;
    }
```

En esta sección, se define la clase Lista Enlazada y se crea un miembro de datos privado llamado inicio. Este miembro de datos es un puntero al primer nodo de la lista. El constructor de la clase inicializa el puntero a nulo, lo que indica que la lista está vacía.

- **Metodo estaVacia**

```
    }

    public boolean estaVacia() {
        return inicio == null;
    }
```

Este método devuelve un valor booleano que indica si la lista está vacía o no. Si el puntero inicio está apuntando a nulo, entonces la lista está vacía y el método devuelve verdadero. De lo contrario, el método devuelve falso.

- **Metodo InsertaAlInicio**

```
// Método para insertar un nodo al inicio de la lista
public void insertarAlInicio(int dato) {
    Nodo_Simple nuevo = new Nodo_Simple(dato);

    if (estaVacia()) {
        inicio = nuevo;
    } else {
        nuevo.setSiguiente(inicio);
        inicio = nuevo;
    }
}
```

Este método inserta un nuevo nodo al principio de la lista. Primero, se crea un nuevo nodo con el dato especificado. Luego, si la lista está vacía, se establece el puntero inicio para que apunte al nuevo nodo. De lo contrario, el puntero siguiente del nuevo nodo se establece en el nodo actualmente en el inicio de la lista y luego el puntero inicio se establece en el nuevo nodo.

- **Metodo insertarAlFinal**

```java
// Método para insertar un nodo al final de la lista
public void insertarAlFinal(int dato) {
    Nodo_Simple nuevo = new Nodo_Simple(dato);

    if (estaVacia()) {
        inicio = nuevo;
    } else {
        Nodo_Simple actual = inicio;
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();
        }
        actual.setSiguiente(nuevo);
    }
}
```

Este método inserta un nuevo nodo al final de la lista. Primero, se crea un nuevo nodo con el dato especificado. Si la lista está vacía, se establece el puntero inicio para que apunte al nuevo nodo. De lo contrario, se recorre la lista hasta llegar al último nodo y se establece el puntero siguiente del último nodo para que apunte al nuevo nodo.

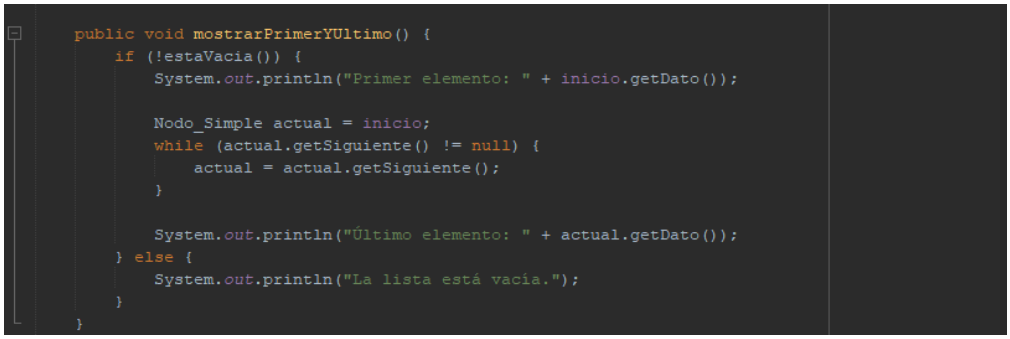
- **Método eliminar**

```java
public void eliminar(int dato) {
    if (!estaVacia()) {
        if (inicio.getDato() == dato) {
            inicio = inicio.getSiguiente();
        } else {
            Nodo_Simple actual = inicio;
            while (actual.getSiguiente() != null && actual.getSiguiente().getDato() != dato) {
                actual = actual.getSiguiente();
            }
            if (actual.getSiguiente() != null) {
                actual.setSiguiente(actual.getSiguiente().getSiguiente());
            }
        }
    } else {
        System.out.println("No se elimino no datos !!!");
    }
```

Este método elimina un nodo con el dato especificado de la lista. Si la lista no está vacía, primero verifica si el nodo a eliminar es el primer nodo de la lista. Si es así, se establece el puntero inicio para que apunte al siguiente nodo en la lista. De lo contrario, se recorre la

lista hasta encontrar el nodo anterior al que se desea eliminar y se actualiza su puntero siguiente para saltar sobre el nodo que se desea eliminar.

- **Método mostrarPrimerYUltimo**



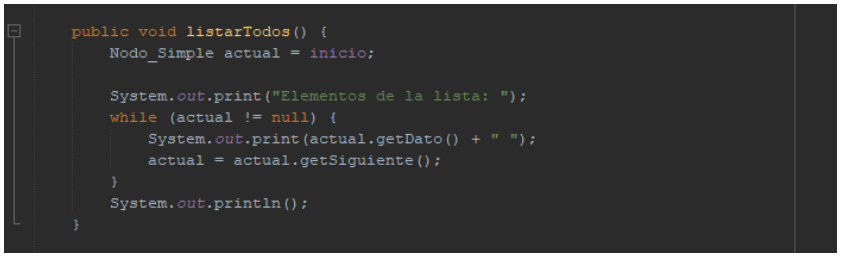
```
public void mostrarPrimerYUltimo() {
    if (!estaVacia()) {
        System.out.println("Primer elemento: " + inicio.getDato());

        Nodo_Simple actual = inicio;
        while (actual.getSiguiente() != null) {
            actual = actual.getSiguiente();
        }

        System.out.println("Último elemento: " + actual.getDato());
    } else {
        System.out.println("La lista está vacía.");
    }
}
```

Este método muestra el primer y último elemento de la lista, si existe alguno. Si la lista no está vacía, primero muestra el dato del primer nodo de la lista y luego recorre la lista hasta encontrar el último nodo y muestra su dato. Si la lista está vacía, muestra un mensaje indicando que la lista está vacía.

- **Método listarTodos**

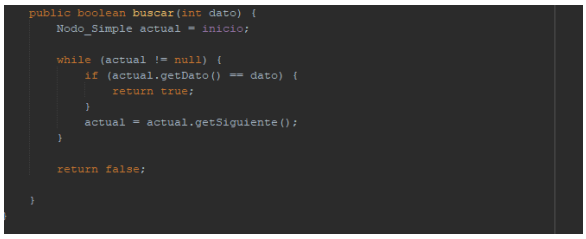


```
public void listarTodos() {
    Nodo_Simple actual = inicio;

    System.out.print("Elementos de la lista: ");
    while (actual != null) {
        System.out.print(actual.getDato() + " ");
        actual = actual.getSiguiente();
    }
    System.out.println();
}
```

Este método muestra todos los elementos de la lista. Primero, establece un puntero actual en el primer nodo de la lista. Luego, recorre la lista y muestra los datos de cada nodo.

- **Método buscar**



```
public boolean buscar(int dato) {
    Nodo_Simple actual = inicio;

    while (actual != null) {
        if (actual.getDato() == dato) {
            return true;
        }
        actual = actual.getSiguiente();
    }

    return false;

}
```

Este método busca un nodo en la lista que tenga el valor dato. Si lo encuentra, devuelve true. Si no lo encuentra, devuelve false.

Los anteriores métodos nos muestran un funcionamiento básico, donde se encuentran temas como condicionales, ciclos y booleanos.

10. Para finalizar listas simples nos vamos al main e importamos la clase "ListaEnlazada" y la clase "Scanner".

```
*/
import Lista_Enlazadas.ListaEnlazada;
import java.util.Scanner;
```

Aquí se define la clase "MainListaEnlazada" que contiene el método "main" que es el punto de entrada del programa, y en el método "main" se crea una instancia de la clase "Scanner" y otra instancia de la clase "ListaEnlazada".

La instancia de "Scanner" se utiliza para leer la entrada del usuario y la instancia de "ListaEnlazada" es la lista enlazada que se va a utilizar en el programa.

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    ListaEnlazada lista = new ListaEnlazada();
```

Se define la variable "opción" para almacenar la opción elegida por el usuario, y se utiliza un bucle "do-while" para mostrar el menú y permitir al usuario elegir una opción.

```
int opcion = 0;
do {
    System.out.println("---------- Menú_Lista_Enlazadas----------");
    System.out.println("¿Qué acción deseas realizar?");
    System.out.println("1. Insertar al inicio");
    System.out.println("2. Insertar al final");
    System.out.println("3. Eliminar");
    System.out.println("4. Mostrar primer y último elemento");
    System.out.println("5. Listar todos los elementos");
    System.out.println("6. Buscar un elemento");
    System.out.println("7. Salir");
```

En esta parte del código se utiliza un bucle "do-while" para mostrar el menú y permitir al usuario elegir una opción. Se muestra el menú en pantalla, se lee la opción elegida por el usuario con el método "nextInt()" de la instancia de "Scanner" y se utiliza un switch para realizar una acción dependiendo de la opción elegida por el usuario.

```
switch (opcion) {
    case 1:
        System.out.println("Ingresa el dato a insertar:");
        int datoInicio = scanner.nextInt();
        lista.insertarAlInicio(datoInicio);
        break;
    case 2:
        System.out.println("Ingresa el dato a insertar:");
        int datoFinal = scanner.nextInt();
        lista.insertarAlFinal(datoFinal);
        break;
    case 3:
        System.out.println("Ingresa el dato a eliminar:");
        int datoEliminar = scanner.nextInt();
        lista.eliminar(datoEliminar);

        break;
    case 4:
        lista.mostrarPrimerYUltimo();
        break;
    case 5:
        lista.listarTodos();
        break;
    case 6:
        System.out.println("Ingresa el dato a buscar:");
        int datoBuscar = scanner.nextInt();
        boolean encontrado = lista.buscar(datoBuscar);
        if (encontrado) {
            System.out.println("El elemento " + datoBuscar + " se encuentra en
        } else {
```

Finalizando así con la creación de una lista simple java

ESTRUCTURA DE DATOS - LISTAS DOBLES

1. Para las listas dobles ya teníamos la paquetería creada. A diferencia de la anterior la creación de nodos serán de dos por eso reciben el nombre de listas dobles.

- Como primer paso crear la clase nodo Doble

```java
public class NodoDoble {
private int dato;
private NodoDoble siguiente;
private NodoDoble anterior;

public NodoDoble(int dato) {
    this.dato = dato;
    this.siguiente = null;
    this.anterior = null;
}

public int getDato() {
    return dato;
}

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

public NodoDoble getSiguiente() {
    return siguiente;
}

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

Donde se contará con los siguientes métodos y atributos

- Aquí se define la clase NodoDoble con tres atributos privados: dato de tipo int, siguiente de tipo NodoDoble y anterior de tipo NodoDoble. Estos atributos representan el valor del nodo y las referencias al siguiente y anterior nodo de la lista.

```
* @author Rafael Corredor
*/

   public class NodoDoble {
   private int dato;
   private NodoDoble siguiente;
   private NodoDoble anterior;
```

- Este es el constructor de la clase NodoDoble. Recibe un parámetro dato y lo asigna al atributo correspondiente. Además, inicializa los atributos siguiente y anterior como null.

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

- Estos son cuatro métodos públicos para acceder y modificar la referencia al nodo anterior de la lista y para el siguiente nodo en la lista.

```
    public NodoDoble getSiguiente() {
        return siguiente;
    }

    public void setSiguiente(NodoDoble siguiente) {
        this.siguiente = siguiente;
    }

    public NodoDoble getAnterior() {
        return anterior;
    }

    public void setAnterior(NodoDoble anterior) {
        this.anterior = anterior;
    }
}
```

- En la siguiente clase ListaDoble se define con dos atributos privados: inicio y fin, ambos de tipo “NodoDoble”. Estos atributos representan el primer y último nodo de la lista, respectivamente. El constructor de la clase inicializa ambos atributos como null.

```
public class ListaDoble {

    private NodoDoble inicio;
    private NodoDoble fin;

    public ListaDoble() {
        this.inicio = null;
        this.fin = null;
    }
```

Este es un método público que devuelve “true” si la lista está vacía, es decir, si el atributo” inicio” es null.

```
    }

    public boolean estaVacia() {
        return inicio == null;
    }
```

De aquí en adelante se realice el mismo proceso de la creación de métodos.

```
    // Método para insertar un nodo al inicio de la lista
    public void insertarAlInicio(int dato) {
        NodoDoble nuevo = new NodoDoble(dato);

        if (estaVacia()) {
            inicio = nuevo;
            fin = nuevo;
        } else {
            nuevo.setSiguiente(inicio);
            inicio.setAnterior(nuevo);
            inicio = nuevo;
        }
    }

    // Método para insertar un nodo al final de la lista
    public void insertarAlFinal(int dato) {
        NodoDoble nuevo = new NodoDoble(dato);

        if (estaVacia()) {
            inicio = nuevo;
            fin = nuevo;
        } else {
            fin.setSiguiente(nuevo);
            nuevo.setAnterior(fin);
            fin = nuevo;
        }
    }
```

Ya solo es correr el proyecto y verificar su funcionalidad.