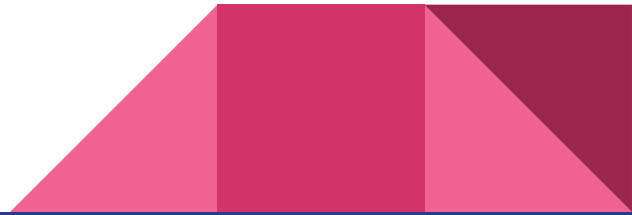
CLASE 2

Ing. SILVESTRE ALEJANDRO
INFORMÁTICA III
IUA - 2023

DEVOLUCIÓN ACTIVIDAD

REVISIÓN ACTIVIDAD

- Los nombre de las clases comienzan con mayúscula.
- Los nombre de los métodos comienzan con minúscula.
- La calculadora tiene que seguir funcionando luego de hacer una operación.
- Si separan en una clase “Calculadora” o “Calc” lo ideal es que lo método de dividir, suma, etc retorne el valor y que el main si quiere lo imprima o no, pero que sea una decisión del main.
- Pongan nombres representativos en los nombres de variables.



JAVA

ARRAY Y LIST

Array

- Los arrays son estructuras de datos que almacenan elementos del mismo tipo.
- **Tamaño fijo:** Se especifica el tamaño en la creación y no cambia después.
- Sintaxis: tipo[] nombreArray = new tipo[tamaño];
- Acceso a elementos por índice: nombreArray[indice]

```
int[] numeros = new int[5];

numeros[0] = 10;

numeros[1] = 20;

// ...
```

Array - Características

- Rendimiento eficiente para acceso directo a elementos.
- Declaración y creación en una sola línea.
- No puede cambiar su tamaño después de la creación.
- Requiere un conocimiento previo del tamaño necesario.

```
int[] numeros = new int[5];

numeros[0] = 10;

numeros[1] = 20;

// ...
```

List

- Las listas son estructuras de datos que almacenan elementos y pueden cambiar de tamaño dinámicamente.
- Implementadas por la interfaz List en Java.
- Utilizan clases concretas como ArrayList y LinkedList.

```
import java.util.ArrayList;

import java.util.List;


List<String> nombres = new
ArrayList<>();

nombres.add("Juan");

nombres.add("María");

// ...
```

List - Características

- Tamaño dinámico: Pueden crecer o disminuir según sea necesario.
- Mayor flexibilidad en comparación con arrays.
- Ofrece métodos para agregar, eliminar y buscar elementos.

```
import java.util.ArrayList;

import java.util.List;


List<String> nombres = new ArrayList<>();

nombres.add("Juan");

nombres.add("María");

// ...
```

PROGRAMACIÓN ORIENTADA A OBJETOS

Conceptos Clave de la POO

1. Clases y Objetos.
2. Atributos y Métodos.
3. Encapsulación.
4. Herencia.
5. Polimorfismo.

Clases y Objetos

En Java, una clase es una plantilla que define la estructura y el comportamiento de los objetos. Un objeto es una instancia de una clase. Por ejemplo, si tienes una clase "Vehiculo", un objeto sería un auto específico.

```
class Vehiculo {
        public String precio;
        public String color;
        //Constructor
}

class Ejemplo {
        public static main(String args []){
                Vehiculo ferrari = new Vehiculo();
                Vehiculo porche = new Vehiculo();
        }
}
```

Atributos y Métodos

Las clases contienen atributos (variables) que describen las características del objeto y métodos (funciones) que definen su comportamiento. Por ejemplo, un Vehiculo podría tener atributos como "precio" y "color", y métodos como "arrancar" y "detener".

```
class Vehiculo {
        private String precio;
        private String color;
        //Constructor

        public arrancar(){
        };
        public detener(){
        };
}

class Ejemplo {
        public static main(String args []){
                Vehiculo ferrari = new Vehiculo();
                ferrari.arrancar();
                Vehiculo porche = new Vehiculo();
        }
}
```

Encapsulación

La encapsulación es el concepto de ocultar los detalles internos de una clase y exponer solo lo necesario para interactuar con ella. Se logra utilizando modificadores de acceso (public, private, protected) para controlar la visibilidad de atributos y métodos.

```
class Vehiculo {
        private String precio;
        private String color;
        public String estado;
        //Constructor

        public arrancar(){};
        public detener(){};
        private lavar(){};
}

class Ejemplo {
        public static main(String args []){
                Vehiculo ferrari = new Vehiculo();
                ferrari.precio = “150000”;
                ferrari.color = “rojo”;
                ferrari.estado = “sucio”;
                ferrari.arrancar();
                ferrari.lavar();
                ferrari.detener();
        }
}
```

Encapsulación

La encapsulación es el concepto de ocultar los detalles internos de una clase y exponer solo lo necesario para interactuar con ella. Se logra utilizando modificadores de acceso (public, private, protected) para controlar la visibilidad de atributos y métodos.

```
class Vehiculo {
        private String precio;
        private String color;
        public String estado;
        //Constructor

        public arrancar(){};
        public detener(){};
        private lavar(){};
}

class Ejemplo {
        public static main(String args []){
                Vehiculo ferrari = new Vehiculo();
                ferrari.precio = “150000”;
                ferrari.color = “rojo”;
                ferrari.estado = “sucio”;
                ferrari.arrancar();
                ferrari.lavar();
                ferrari.detener();
        }
}
```

Herencia

La herencia permite que una clase herede los atributos y métodos de otra clase. Esto fomenta la reutilización de código y la creación de jerarquías de clases. La clase que hereda se llama "subclase" y la clase de la que hereda se llama "superclase".

```
class Vehiculo {
	private String precio;
	private String color;
	public String estado;
	//Constructor
	//Metodos
}

class Auto extends Vehiculo {
	private Int cantidadPersonas;
}

class Camion extends Vehiculo {
	private Float capacidadMaxima;
}


class Ejemplo {
	public static main(String args []){
		Auto ferrari = new Vehiculo();
		Camion scania = new Vehiculo();
		Vehiculo vehiculo = new Vehiculo();

	}
}
```

Polimorfismo

El polimorfismo permite que distintas clases compartan un mismo nombre de método, pero cada clase puede implementar ese método de manera diferente. Esto brinda flexibilidad y permite escribir código más genérico y reutilizable.

```
class Vehiculo {
	private String precio;
	private String color;
	public String estado;
	//Constructor
	//Metodos
	public int velocidadMaximaPermitida(){
		return 0;
	}
}

class Auto extends Vehiculo {
	private Int cantidadPersonas;

	@Override
	public int velocidadMaximaPermitida(){
		return 110;
	}
}

class Camion extends Vehiculo {
	private Float capacidadMaxima;
	@Override
	public int velocidadMaximaPermitida(){
		return 80;
	}
}


class Ejemplo {
	public static main(String args []){
		Auto ferrari = new Vehiculo();
		ferrari.velocidadMaximaPermitida();
		Camion scania = new Vehiculo();
		scania.velocidadMaximaPermitida();
	}
}
```

Tratamiento de Excepciones

Las excepciones son situaciones inesperadas que pueden ocurrir durante la ejecución de un programa y que interrumpen el flujo normal del mismo.

Pueden ser causadas por diversos motivos, como errores de entrada del usuario, problemas de red, división por cero, acceso a archivos inexistentes, entre otros.

Java proporciona un sistema de manejo de excepciones que permite identificar y manejar estas situaciones de manera controlada, evitando que el programa se bloquee o cierre de manera abrupta.

Excepciones try-catch

Son aquellas excepciones que se capturar en bloque de try-catch.

```
class Vehiculo {
	public String precio;
	public String color;
	//Constructor
	public Float convertirPrecioEnFloat(){
		return Float.parseFloat(precio);
	}
}
class Ejemplo {
	public static main(String args []){
		try{
			Vehiculo ferrari = new Vehiculo();
			ferrari.convertirPrecioEnFloat();
			//Bloque de código.
		}catch(Exception e){
			e.printStackTrace();
		}catch(FileException e){
			e.printStackTrace();
		}catch(HttpConectionException e){
			e.printStackTrace();
		}
	}
}
```

Excepciones throws

Son aquellas excepciones definidas en los métodos.

Cuando está definido en los métodos cualquier instancia que lo llame debería capturarlo obligatoriamente.

```
class Vehiculo {
	public String precio;
	public String color;
	//Constructor
	public Float convertirPrecioEnFloat() throws Exception{
		return Float.parseFloat(precio);
	}

	public Float convertirPrecioEnFloat(){
		try{
		    return Float.parseFloat(precio);
		}catch(Exception e){
		    //print error
		}
	}
}
class Ejemplo {
	public static main(String args []){
		try{
			Vehiculo ferrari = new Vehiculo();
			ferrari.convertirPrecioEnFloat();
			//Bloque de código.
		}catch(Exception e){
			e.printStackTrace();
		}
	}
}
```

¿DUDAS?

BREAK

MANOS A LA OBRA

Actividad

En el mismo proyecto de la calculadora de la clase pasada, agregar:

- Manejo de excepciones modificando un mensaje de error para el usuario final.
- Guardar en un array o list el histórico de operaciones y resultados.
- Poder ver el histórico desde el menú de opciones.