PROGRAMACION ORIENTADA A OBJETOS

La programación orientada a objetos es un paradigma que consiste en crear objetos que contienen atributos y métodos.

Clase: Plantilla para instanciar objetos.

```
public class Area {

	int base = 5;
	int altura = 3;

	public void calcular() {
		System.out.println("El area es: " + base*altura);
	}

}
```

Donde el conjunto de variables declaradas se le conocen como **atributos**.

Los bloques de código definidos para realizar una determinada acción se le conocen como **métodos**.

Objeto: Instancia de una clase.

```
public class Main {
	public static void main(String[] args) {
		Area area = new Area();
		area.calcular();
	}
}
```

Constructor: es un tipo de método que es usado para inicializar objetos, el constructor es invocado cuando se instancia un objeto.

```
public class Area {

	int base = 5;
```

```
        int altura = 3;

        public Area() {
        }

        public Area(int base, int altura) {
                this.base = base;
                this.altura = altura;
        }

        public void calcular() {
                System.out.println("El area es: " + base*altura);
        }

}
```

Modificadores de acceso: Un modificador de acceso especifica la accesibilidad de un método, atributo, constructor o clase.

private	Solo se puede acceder dentro de la clase, no se puede acceder desde clases externas
default	Solo pueden acceder clases que están dentro de un paquete
protected	El acceso puede ser dentro de un paquete o a través de clases hijas
public	Es de libre acceso, tanto clases y paquetes externos pueden acceder

Principios de programación orientada a objetos

Encapsulamiento: Restringe el acceso de ciertas partes del código desde otra clases, esto se logra declarando las variables como privadas y proporcionando métodos públicos (getters/setters) para acceder y asignarle valores a una variable privada.

```
public class Area {

        private int area;

        public int getArea() {
                return area;
        }
```

```
	public void setArea(int area) {
		this.area = area;
	}
}
```

El método **get** retorna el valor de la variable, el método **set** toma un parámetro y asigna esta valor a la variable.

Herencia: proceso mediante el cual un objeto adquiere las propiedades de otro objeto, para heredad de una clase, se debe usar la palabra reservada **extends**.

```
public class Area {

	public void imprimir(int area) {
		System.out.println("El area es: " + area);
	}

}

public class Cuadrado extends Area {
}
```

```
public class Main {
	public static void main(String[] args) {
		Cuadrado cuadrado = new Cuadrado();
		cuadrado.imprimir(30);
	}
}
```

Polimorfismo: Ocurre cuando tenemos varias clases que están relacionadas a una clase padre por medio de herencia, estas heredan los métodos y atributos que pueden realizar diferentes tareas dependiendo la clase que los herede.

```
public class Area {
	public void imprimir(int area) {
		System.out.println("Areas de figuras");
	}
}

public class Cuadrado extends Area {
	public void imprimir(int area) {
		System.out.println("Area del cuadrado: " + area);
	}
}

public class Circulo extends Area {
```

```
        public void imprimir(int area) {
                System.out.println("Area del circulo: " + area);
        }
}
```

```
public class Main {
        public static void main(String[] args) {
                Area cuadrado = new Cuadrado();
                Area circulo = new Circulo();
                cuadrado.imprimir(30);
                circulo.imprimir(20);
        }
}
```

Composición: Estrategia mediante la cual un objeto puede contener otros objetos, esto se logra instanciando variables que se refieren a otros objetos.

```
public class Trabajo {

        private String cargo;
        private Double salario;

        public String getCargo() {
                return cargo;
        }

        public void setCargo(String cargo) {
                this.cargo = cargo;
        }

        public Double getSalario() {
                return salario;
        }

        public void setSalario(Double salario) {
                this.salario = salario;
        }
}

public class Persona {

        private Trabajo trabajo;
        private String nombre;

        public Trabajo getTrabajo() {
                return trabajo;
        }

        public void setTrabajo(Trabajo trabajo) {
```

```
            this.trabajo = trabajo;
        }

        public String getNombre() {
            return nombre;
        }

        public void setNombre(String nombre) {
            this.nombre = nombre;
        }
}
```

Interfaces: Conjunto de métodos abstractos donde se especifica que se debe hacer más no su implementación.

```
public interface Empleado {
   void imprimirNombre();
}

public class Gerente implements Empleado{
   @Override
   public void imprimirNombre() {
      System.out.println("Jose Silva");
   }
}
```

```
public class Main {
   public static void main(String[] args) {
      Empleado gerente = new Gerente();
      gerente.imprimirNombre();
   }
}
```

Fuentes:

https://www.w3schools.com/java/java_oop.asp

https://www.javatpoint.com/java-oops-concepts

https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/