PRACTICA II PROGRAMACION II
NOMBRE: LUIS ALEJANDRO ZEBALLOS QUIROZ
CI: 12896709
RU: 1886052

HERENCIA:
1.

```
class Producto {
    protected:
        String nombre;
        double precio;

    public:
        Producto(String nombre, double precio) {
            this.nombre <- nombre;
            this.precio <- precio;
        }

        String getNombre() {
            return this.nombre;
        }

        double getPrecio() {
            return this.precio;
        }

        void setNombre(String nombre) {
            this.nombre <- nombre;
        }
}
class ProductoAlimenticio extends Producto {
    private:
        String fechaVencimiento;
        double calorias;

    public:
        ProductoAlimenticio(String nombre, double precio, String
fechaVencimiento, double calorias) {
            super(nombre, precio);
            this.fechaVencimiento <- fechaVencimiento;
            this.calorias <- calorias;
        }

        void diasRestantesVencimiento(String fechaActual) {
            String[] fechaActualPartes <- fechaActual.split("-");
            String[] fechaVencimientoPartes <- this.fechaVencimiento.split("-");
            int diaActual <- Integer.parseInt(fechaActualPartes[0]);
            int mesActual <- Integer.parseInt(fechaActualPartes[1]);
            int anioActual <- Integer.parseInt(fechaActualPartes[2]);
            int diaVencimiento <- Integer.parseInt(fechaVencimientoPartes[0]);
            int mesVencimiento <- Integer.parseInt(fechaVencimientoPartes[1]);
            int anioVencimiento <- Integer.parseInt(fechaVencimientoPartes[2]);
            int diasRestantes <- (anioVencimiento - anioActual) * 365 +
(mesVencimiento - mesActual) * 30 + (diaVencimiento - diaActual);
            print(diasRestantes + " días restantes para el vencimiento del producto "
+ this.nombre);
        }

        void caloriasInfo() {
            if (this.calorias < 100) {
```

```
            print("El producto " + this.nombre + " es saludable con " +
this.calorias + " calorias");
        } else {
            print("El producto " + this.nombre + " no es saludable con " +
this.calorias + " calorias");
        }
    }

    double getCalorias() {
        return this.calorias;
    }
}
class ProductoElectrodomestico extends Producto {
  protected:
    double potenciaW;
    String marca;

  public:
    ProductoElectrodomestico(String nombre, double precio, double
potenciaW, String marca) {
        super(nombre, precio);
        this.potenciaW <- potenciaW;
        this.marca <- marca;
    }

    double getPotenciaW() {
        return this.potenciaW;
    }

    String getMarca() {
        return this.marca;
    }

    void setPotenciaW(double potenciaW) {
        this.potenciaW <- potenciaW;
    }
}
class Refrigerador extends ProductoElectrodomestico {
  private:
    double capacidadLitros;
    int claseEnergetica;

  public:
    Refrigerador(String nombre, double precio, double potenciaW, String
marca, double capacidadLitros, int claseEnergetica) {
        super(nombre, precio, potenciaW, marca);
        this.capacidadLitros <- capacidadLitros;
        this.claseEnergetica <- claseEnergetica;
    }

    void compararCapacidad(Refrigerador otro) {
        if (this.capacidadLitros > otro.capacidadLitros) {
            print("El refrigerador " + this.nombre + " tiene mayor capacidad que "
+ otro.nombre);
        } else if (this.capacidadLitros < otro.capacidadLitros) {
            print("El refrigerador " + otro.nombre + " tiene mayor capacidad que "
+ this.nombre);
        } else {
            print("Ambos refrigeradores tienen la misma capacidad");
        }
```

```
        }

        double getCapacidadLitros() {
            return this.capacidadLitros;
        }

        int getClaseEnergetica() {
            return this.claseEnergetica;
        }
}
class mainProductos {
    ProductoAlimenticio producto <- new ProductoAlimenticio("Manzana", 1.5,
"2025-12-31", 52);
    producto.diasRestantesVencimiento("2025-07-15");
    producto.caloriasInfo();
    Refrigerador refrigerador1 <- new Refrigerador("Refrigerador A", 500.0,
150.0, "Samsung", 300.0, 1);
    Refrigerador refrigerador2 <- new Refrigerador("Refrigerador B", 600.0,
200.0, "LG", 350.0, 2);
    refrigerador1.compararCapacidad(refrigerador2);
}
```

2.

```
abstract class Animal {
    protected:
        String nombre;
        int edad;
        String especie;

    public:
        Animal(String nombre, int edad, String especie) {
            this.nombre <- nombre;
            this.edad <- edad;
            this.especie <- especie;
        }

        String getNombre() {
            return this.nombre;
        }

        int getEdad() {
            return this.edad;
        }

        abstract String hacerSonido();
}
abstract class Mamifero extends Animal {
    protected:
        String tipoPelaje;
        double peso;

    public:
        Mamifero(String nombre, int edad, String especie, String tipoPelaje, double
peso) {
            super(nombre, edad, especie);
            this.tipoPelaje <- tipoPelaje;
            this.peso <- peso;
        }

        String getTipoPelaje() {
            return this.tipoPelaje;
        }

        double getPeso() {
            return this.peso;
        }

        abstract String hacerSonido();
}
class Murcielago extends Mamifero {
    private:
        int alcanceVueloKm;
        boolean activoNoche;

    public:
        Murcielago(String nombre, int edad, String especie, String tipoPelaje,
double peso, int alcanceVueloKm, boolean activoNoche) {
```

```
        super(nombre, edad, especie, tipoPelaje, peso);
        this.alcanceVueloKm <- alcanceVueloKm;
        this.activoNoche <- activoNoche;
    }

    String hacerSonido() {
        return "Chillido";
    }

    void comparaAlcanceVuelo(Murcielago otro) {
        if (this.alcanceVueloKm > otro.alcanceVueloKm) {
            print(this.nombre + " vuela más lejos que " + otro.nombre);
        } else if (this.alcanceVueloKm < otro.alcanceVueloKm) {
            print(this.nombre + " vuela menos lejos que " + otro.nombre);
        } else {
            print(this.nombre + " y " + otro.nombre + " tienen el mismo alcance de
vuelo");
        }
    }

    int getAlcanceVueloKm() {
        return this.alcanceVueloKm;
    }
}
class Perro extends Mamifero {
  private:
    String raza;

  public:
    Perro(String nombre, int edad, String especie, String tipoPelaje, double
peso, String raza) {
        super(nombre, edad, especie, tipoPelaje, peso);
        this.raza <- raza;
    }

    String hacerSonido() {
        return "Ladrar";
    }

    String getRaza() {
        return this.raza;
    }

    void setRaza(String raza) {
        this.raza <- raza;
    }
}
class mainAnimales {
  Perro perro1 <- new Perro("Max", 4, "Canino", "Largo", 22.0, "Golden
Retriever");
  Murcielago murcielago1 <- new Murcielago("Batman", 2, "Quiróptero",
"Corto", 0.5, 50, true);
  Murcielago murcielago2 <- new Murcielago("Dracula", 3, "Quiróptero",
"Corto", 0.6, 70, true);

  Animal[] animales <- new Animal[12];
  animales[0] <- perro1;
  animales[1] <- murcielago1;
  animales[2] <- murcielago2;
```

```
    for (int i <- 0; i < animales.length; i++) {
        print("Nombre: " + animales[i].getNombre() + ", Especie: " +
animales[i].getEspecie() + ", Sonido: " + animales[i].hacerSonido());
    }

    murcielago1.comparaAlcanceVuelo(murcielago2);
}
```

3.

```
abstract class Vehiculo {
    protected:
        String color;
        int velocidadMaxima;

    public:
        Vehiculo(String color, int velocidadMaxima) {
            this.color <- color;
            this.velocidadMaxima <- velocidadMaxima;
        }

        String getColor() {
            return this.color;
        }

        int getVelocidadMaxima() {
            return this.velocidadMaxima;
        }

        abstract void mostrarDatos();
}
abstract class Motorizado extends Vehiculo {
    protected:
        String tipoCombustible;

    public:
        Motorizado(String color, int velocidadMaxima, String tipoCombustible) {
            super(color, velocidadMaxima);
            this.tipoCombustible <- tipoCombustible;
        }

        String getTipoCombustible() {
            return this.tipoCombustible;
        }

        void setTipoCombustible(String tipoCombustible) {
            this.tipoCombustible <- tipoCombustible;
        }

        abstract void mostrarDatos();
}
class Auto extends Motorizado {
    private:
        int numeroPuertas;

    public:
        Auto(String color, int velocidadMaxima, String tipoCombustible, int
numeroPuertas) {
            super(color, velocidadMaxima, tipoCombustible);
            this.numeroPuertas <- numeroPuertas;
        }

        void mostrarDatos() {
```

```
            print("Auto - Color: " + this.color + ", Velocidad Máxima: " +
this.velocidadMaxima + ", Combustible: " + this.tipoCombustible + ", Puertas: "
+ this.numeroPuertas);
        }

        int getNumeroPuertas() {
            return this.numeroPuertas;
        }

        void setNumeroPuertas(int numeroPuertas) {
            this.numeroPuertas <- numeroPuertas;
        }
}
class Bicicleta extends NoMotorizado {
    private:
        int numeroMarchas;

    public:
        Bicicleta(String color, int velocidadMaxima, String tipoFrenos, int
numeroMarchas) {
            super(color, velocidadMaxima, tipoFrenos);
            this.numeroMarchas <- numeroMarchas;
        }

        void mostrarDatos() {
            print("Bicicleta - Color: " + this.color + ", Velocidad Máxima: " +
this.velocidadMaxima + ", Frenos: " + this.tipoFrenos + ", Marchas: " +
this.numeroMarchas);
        }

        boolean mismoTipoFrenos(Bicicleta otra) {
            return this.tipoFrenos.equals(otra.tipoFrenos);
        }

        int getNumeroMarchas() {
            return this.numeroMarchas;
        }
}
class mainVehiculos {
    Auto auto1 <- new Auto("Rojo", 180, "Gasolina", 4);
    Auto auto2 <- new Auto("Azul", 200, "Diesel", 2);
    Bicicleta bici1 <- new Bicicleta("Amarillo", 40, "Disco", 21);
    Bicicleta bici2 <- new Bicicleta("Negro", 35, "Disco", 18);

    Vehiculo[] vehiculos <- new Vehiculo[4];
    vehiculos[0] <- auto1;
    vehiculos[1] <- auto2;
    vehiculos[2] <- bici1;
    vehiculos[3] <- bici2;

    for (int i <- 0; i < vehiculos.length; i++) {
        vehiculos[i].mostrarDatos();
    }

    if (bici1.mismoTipoFrenos(bici2)) {
        print("Ambas bicicletas tienen el mismo tipo de frenos: " +
bici1.getTipoFrenos());
    }
}
```

Composicion:
1.

```
class Universidad {
    private:
        String nombre;
        Estudiante[] estudiantes;
        int nEstudiantes;
        final int MAX_ESTUDIANTES = 100;

    public:
        Universidad(String nombre) {
            this.nombre <- nombre;
            this.estudiantes <- new Estudiante[MAX_ESTUDIANTES];
            this.nEstudiantes <- 0;
        }

        agregarEstudiante(Estudiante estudiante) {
            if (this.nEstudiantes < MAX_ESTUDIANTES) {
                this.estudiantes[this.nEstudiantes] <- estudiante;
                this.nEstudiantes <- this.nEstudiantes + 1;
            }
        }

        mostrarEstudiantes() {
            print("Estudiantes de " + this.nombre + ":");
            for (int i <- 0; i < this.nEstudiantes; i++) {
                print("- " + this.estudiantes[i].getNombre() + " (" +
this.estudiantes[i].getCodigo() + ")");
            }
        }

        int contarEstudiantes() {
            return this.nEstudiantes;
        }
}
class Estudiante {
    private:
        String nombre;
        String codigo;

    public:
        Estudiante(String nombre, String codigo) {
            this.nombre <- nombre;
            this.codigo <- codigo;
        }

        String getNombre() {
            return this.nombre;
        }

        String getCodigo() {
            return this.codigo;
        }

        actualizarCodigo(String nuevoCodigo) {
```

```
            this.codigo <- nuevoCodigo;
        }
}
class mainUniversidad {
    Universidad universidad <- new Universidad("Universidad Central");

    Estudiante est1 <- new Estudiante("Juan Pérez", "2021001");
    Estudiante est2 <- new Estudiante("María García", "2021002");
    Estudiante est3 <- new Estudiante("Carlos López", "2021003");

    universidad.agregarEstudiante(est1);
    universidad.agregarEstudiante(est2);
    universidad.agregarEstudiante(est3);

    universidad.mostrarEstudiantes();
    print("Total estudiantes: " + universidad.contarEstudiantes());
}
```

2.

```
class Casa {
    private:
        String direccion;
        Habitacion[] habitaciones;
        int nHabitaciones;
        final int MAX_HABITACIONES = 10;

    public:
        Casa(String direccion) {
            this.direccion <- direccion;
            this.habitaciones <- new Habitacion[MAX_HABITACIONES];
            this.nHabitaciones <- 0;

            // Habitaciones por defecto
            this.habitaciones[this.nHabitaciones] <- new Habitacion("Sala", 20.0);
            this.nHabitaciones <- this.nHabitaciones + 1;
            this.habitaciones[this.nHabitaciones] <- new Habitacion("Cocina", 15.0);
            this.nHabitaciones <- this.nHabitaciones + 1;
            this.habitaciones[this.nHabitaciones] <- new Habitacion("Dormitorio",
18.0);
            this.nHabitaciones <- this.nHabitaciones + 1;
        }

        mostrarHabitaciones() {
            print("Habitaciones de la casa en " + this.direccion + ":");
            for (int i <- 0; i < this.nHabitaciones; i++) {
                print("- " + this.habitaciones[i].getTipo() + ": " +
this.habitaciones[i].getArea() + " m²");
            }
        }

        double calcularAreaTotal() {
            double total <- 0;
            for (int i <- 0; i < this.nHabitaciones; i++) {
                total <- total + this.habitaciones[i].getArea();
            }
            return total;
        }

        agregarHabitacion(Habitacion habitacion) {
            if (this.nHabitaciones < MAX_HABITACIONES) {
                this.habitaciones[this.nHabitaciones] <- habitacion;
                this.nHabitaciones <- this.nHabitaciones + 1;
            }
        }
}
class Habitacion {
    private:
        String tipo;
        double area;

    public:
        Habitacion(String tipo, double area) {
            this.tipo <- tipo;
```

```
            this.area <- area;
        }

        String getTipo() {
            return this.tipo;
        }

        double getArea() {
            return this.area;
        }

        modificarArea(double nuevaArea) {
            this.area <- nuevaArea;
        }
}
class mainCasa {
    Casa casa <- new Casa("Av. 6 de Agosto #123");
    casa.mostrarHabitaciones();
    print("Área total: " + casa.calcularAreaTotal() + " m²");

    Habitacion bano <- new Habitacion("Baño", 8.0);
    casa.agregarHabitacion(bano);
    casa.mostrarHabitaciones();
}
```

3.

```
class Computadora {
    private:
        String marca;
        Cpu[] cpu;
        int nroCpu;
        final int MAX_CPU = 10;

    public:
        Computadora(String marca) {
            this.marca <- marca;
            this.cpu <- new Cpu[MAX_CPU];
            this.nroCpu <- 0;
        }

        agregarCpu(Cpu procesador) {
            if (this.nroCpu < MAX_CPU) {
                this.cpu[this.nroCpu] <- procesador;
                this.nroCpu <- this.nroCpu + 1;
            }
        }

        double calcularPotenciaTotal() {
            double total <- 0;
            for (int i <- 0; i < this.nroCpu; i++) {
                total <- total + this.cpu[i].calcularVelocidadTotal();
            }
            return total;
        }

        boolean esGamer() {
            for (int i <- 0; i < this.nroCpu; i++) {
                if (this.cpu[i].esGamer()) {
                    return true;
                }
            }
            return false;
        }
}
class Cpu {
    private:
        String modelo;
        Nucleo[] nuc;
        int nroNucleo;
        final int MAX_NUCLEOS = 10;

    public:
        Cpu(String modelo) {
            this.modelo <- modelo;
            this.nuc <- new Nucleo[MAX_NUCLEOS];
            this.nroNucleo <- 0;
        }

        agregarNucleo(Nucleo nucleo) {
            if (this.nroNucleo < MAX_NUCLEOS) {
```

```
            this.nuc[this.nroNucleo] <- nucleo;
            this.nroNucleo <- this.nroNucleo + 1;
        }
    }

    double calcularVelocidadTotal() {
        double total <- 0;
        for (int i <- 0; i < this.nroNucleo; i++) {
            total <- total + this.nuc[i].getVelocidad();
        }
        return total;
    }

    boolean esGamer() {
        if (this.nroNucleo < 4) return false;
        for (int i <- 0; i < this.nroNucleo; i++) {
            if (this.nuc[i].getVelocidad() < 2.5) {
                return false;
            }
        }
        return true;
    }
}
class Nucleo {
    private:
        double velocidad;

    public:
        Nucleo(double velocidad) {
            this.velocidad <- velocidad;
        }

        double getVelocidad() {
            return this.velocidad;
        }

        modificarVelocidad(double nuevaVelocidad) {
            this.velocidad <- nuevaVelocidad;
        }

        int compararVelocidad(Nucleo otro) {
            if (this.velocidad > otro.getVelocidad()) return 1;
            if (this.velocidad < otro.getVelocidad()) return -1;
            return 0;
        }
}
class mainComputadora {
    Computadora comp1 <- new Computadora("Dell");

    Cpu cpu1 <- new Cpu("Intel i7-10700K");
    Nucleo nucleo1 <- new Nucleo(3.8);
    Nucleo nucleo2 <- new Nucleo(3.8);
    Nucleo nucleo3 <- new Nucleo(3.8);
    Nucleo nucleo4 <- new Nucleo(3.8);

    cpu1.agregarNucleo(nucleo1);
    cpu1.agregarNucleo(nucleo2);
    cpu1.agregarNucleo(nucleo3);
    cpu1.agregarNucleo(nucleo4);
```

```
    comp1.agregarCpu(cpu1);

    print("Potencia total: " + comp1.calcularPotenciaTotal() + " GHz");
    print("Es gamer: " + comp1.esGamer());
}
```

4.

```
class Persona {
    protected:
        String nombre;
        String ci;
        String telefono;

    public:
        Persona(String nombre, String ci, String telefono) {
            this.nombre <- nombre;
            this.ci <- ci;
            this.telefono <- telefono;
        }

        String getNombre() {
            return this.nombre;
        }

        String getCi() {
            return this.ci;
        }

        actualizarTelefono(String nuevoTelefono) {
            this.telefono <- nuevoTelefono;
        }
}
class Cliente extends Persona {
    private:
        String tipoCliente;
        double montoCredito;

    public:
        Cliente(String nombre, String ci, String telefono, String tipoCliente, double
montoCredito) {
            super(nombre, ci, telefono);
            this.tipoCliente <- tipoCliente;
            this.montoCredito <- montoCredito;
        }

        mostrarCliente() {
            print("Cliente: " + this.nombre + " - CI: " + this.ci + " - Teléfono: " +
this.telefono +
                    " - Tipo: " + this.tipoCliente + " - Crédito: " + this.montoCredito);
        }

        actualizarCredito(double nuevoCredito) {
            this.montoCredito <- nuevoCredito;
        }

        boolean esClientePremium() {
            return this.tipoCliente.equals("Premium") ||
this.tipoCliente.equals("VIP");
        }
}
class Empleado extends Persona {
```

```
    private:
        String cargo;
        double salario;

    public:
        Empleado(String nombre, String ci, String telefono, String cargo, double
salario) {
            super(nombre, ci, telefono);
            this.cargo <- cargo;
            this.salario <- salario;
        }

        mostrarEmpleado() {
            print("Empleado: " + this.nombre + " - CI: " + this.ci + " - Teléfono: " +
this.telefono +
                  " - Cargo: " + this.cargo + " - Salario: " + this.salario);
        }

        aumentarSalario(double porcentaje) {
            this.salario <- this.salario * (1 + porcentaje/100);
        }

        boolean esGerente() {
            return this.cargo.equals("Gerente");
        }
}
class Empresa {
    private:
        String nombre;
        String ruc;
        Empleado[] empleados;
        int nEmpleados;
        Cliente[] clientes;
        int nClientes;
        final int MAX_EMPLEADOS = 50;
        final int MAX_CLIENTES = 50;

    public:
        Empresa(String nombre, String ruc) {
            this.nombre <- nombre;
            this.ruc <- ruc;
            this.empleados <- new Empleado[MAX_EMPLEADOS];
            this.nEmpleados <- 0;
            this.clientes <- new Cliente[MAX_CLIENTES];
            this.nClientes <- 0;
        }

        agregarEmpleado(Empleado empleado) {
            if (this.nEmpleados < MAX_EMPLEADOS) {
                this.empleados[this.nEmpleados] <- empleado;
                this.nEmpleados <- this.nEmpleados + 1;
            } else {
                print("No se puede agregar más empleados");
            }
        }

        agregarCliente(Cliente cliente) {
            if (this.nClientes < MAX_CLIENTES) {
                this.clientes[this.nClientes] <- cliente;
                this.nClientes <- this.nClientes + 1;
```

```
            } else {
                print("No se puede agregar más clientes");
            }
        }

        eliminarEmpleadosQueEsCliente() {
            Empleado[] tempEmpleados <- new Empleado[MAX_EMPLEADOS];
            int newNEmpleados <- 0;

            for (int i <- 0; i < this.nEmpleados; i++) {
                Empleado actualEmpleado <- this.empleados[i];
                boolean esCliente <- false;
                for (int j <- 0; j < this.nClientes; j++) {
                    if (actualEmpleado.getCi().equals(this.clientes[j].getCi())) {
                        esCliente <- true;
                        print("Eliminando empleado que también es cliente: " +
actualEmpleado.getNombre());
                        break;
                    }
                }
                if (!esCliente) {
                    tempEmpleados[newNEmpleados] <- actualEmpleado;
                    newNEmpleados <- newNEmpleados + 1;
                }
            }
            this.empleados <- tempEmpleados;
            this.nEmpleados <- newNEmpleados;
        }
}
class mainEmpresa {
    Empresa empresa <- new Empresa("TechCorp", "12345678901");

    Empleado emp1 <- new Empleado("Juan Pérez", "12345678", "70123456",
"Gerente", 8000.0);
    Empleado emp2 <- new Empleado("María García", "87654321", "70654321",
"Desarrollador", 5000.0);

    Cliente cli1 <- new Cliente("Pedro Rodríguez", "99887766", "70998877",
"Premium", 50000.0);
    Cliente cli2 <- new Cliente("María García", "87654321", "70654321", "VIP",
100000.0);

    empresa.agregarEmpleado(emp1);
    empresa.agregarEmpleado(emp2);
    empresa.agregarCliente(cli1);
    empresa.agregarCliente(cli2);

    print("Estado inicial:");
    emp1.mostrarEmpleado();
    emp2.mostrarEmpleado();
    cli1.mostrarCliente();
    cli2.mostrarCliente();

    empresa.eliminarEmpleadosQueEsCliente();

    print("Después de eliminar empleados que son clientes:");
}
```

Genericidad
1.

```
class Par<T1, T2>{
    private:
        T1 primero;
        T2 segundo;
    public:
        Par(T1 primero, T2 segundo){
            this.primero <- primero;
            this.segundo <- segundo;
        }

        boolean sonIguales(){
            if (this.primero == null AND this.segundo == null) return true;
            if (this.primero == null OR this.segundo == null) return false;
            return this.primero.equals(this.segundo);
        }

        boolean tienenTiposIguales(){
            if (this.primero == null AND this.segundo == null) return true;
            if (this.primero == null OR this.segundo == null) return false;
            return this.primero.getClass().equals(this.segundo.getClass());
        }

        mostrar(){
            print("Par: [" + this.primero + ", " + this.segundo + "]");
        }
}
class MainPar{
    Par<String, Integer> par1 <- new Par<>("Hola", 42);
    Par<Integer, Integer> par2 <- new Par<>(10, 10);
    Par<String, String> par3 <- new Par<>("Mundo", "Mundo");

    print("=== MOSTRANDO PARES ===");
    par1.mostrar();
    par2.mostrar();
    par3.mostrar();

    print("=== VERIFICANDO SI SON IGUALES ===");
    print("Par1 son iguales: " + par1.sonIguales());
    print("Par2 son iguales: " + par2.sonIguales());
    print("Par3 son iguales: " + par3.sonIguales());

    print("=== VERIFICANDO SI TIENEN TIPOS IGUALES ===");
    print("Par1 tipos iguales: " + par1.tienenTiposIguales());
    print("Par2 tipos iguales: " + par2.tienenTiposIguales());
    print("Par3 tipos iguales: " + par3.tienenTiposIguales());
}
class UtilPar{
    private:
        Par<String, String>[] pares;
        int totalPares;
    public:
        UtilPar(){
            this.pares <- new Par[20];
```

```
        this.totalPares <- 0;
    }

    agregarPar(Par<String, String> par){
        if(this.totalPares < 20){
            this.pares[this.totalPares] <- par;
            this.totalPares <- this.totalPares + 1;
        }
    }

    int contarParesIguales(){
        int contador <- 0;
        for(int i <- 0; i < this.totalPares; i++){
            if(this.pares[i].sonIguales()){
                contador <- contador + 1;
            }
        }
        return contador;
    }

    mostrarTodosPares(){
        print("=== TODOS LOS PARES ===");
        for(int i <- 0; i < this.totalPares; i++){
            this.pares[i].mostrar();
        }
    }
}
class GestorPares{
    private:
        UtilPar utilPares;
        int paresCreados;
    public:
        GestorPares(){
            this.utilPares <- new UtilPar();
            this.paresCreados <- 0;
        }

        crearParAleatorio(){
            String[] palabras <- {"casa", "perro", "gato", "mesa"};
            int indice1 <- random(0, 3);
            int indice2 <- random(0, 3);
            Par<String, String> nuevoPar <- new Par<>(palabras[indice1],
palabras[indice2]);
            this.utilPares.agregarPar(nuevoPar);
            this.paresCreados <- this.paresCreados + 1;
        }

        generarEstadisticas(){
            int iguales <- this.utilPares.contarParesIguales();
            print("Total pares creados: " + this.paresCreados);
            print("Pares con elementos iguales: " + iguales);
        }

        mostrarTodo(){
            this.utilPares.mostrarTodosPares();
            this.generarEstadisticas();
        }
}
```

2.

```
class MaximoGenerico{
    private:
        static final int CAPACIDAD <- 100;
    public:
        MaximoGenerico(){
            // Constructor vacío
        }

        static <T extends Comparable<T>> T encontrarMaximo(T a, T b, T c){
            T max <- a;
            if (b.compareTo(max) > 0) {
                max <- b;
            }
            if (c.compareTo(max) > 0) {
                max <- c;
            }
            return max;
        }

        static String encontrarMaximoPorLongitud(String a, String b, String c){
            String max <- a;
            if (b.length() > max.length()) {
                max <- b;
            }
            if (c.length() > max.length()) {
                max <- c;
            }
            return max;
        }

        static <T extends Comparable<T>> T encontrarMaximoEnArray(T[] array){
            if(array.length == 0) return null;
            T max <- array[0];
            for(int i <- 1; i < array.length; i++){
                if(array[i].compareTo(max) > 0){
                    max <- array[i];
                }
            }
            return max;
        }
}
class MainMaximo{
    print("=== MÁXIMO DE ENTEROS ===");
    Integer maxInt <- MaximoGenerico.encontrarMaximo(5, 12, 8);
    print("Máximo entre 5, 12, 8: " + maxInt);

    print("=== MÁXIMO DE DECIMALES ===");
    Double maxDouble <- MaximoGenerico.encontrarMaximo(3.14, 2.71, 1.41);
    print("Máximo entre 3.14, 2.71, 1.41: " + maxDouble);

    print("=== MÁXIMO DE STRINGS ===");
    String maxString <- MaximoGenerico.encontrarMaximo("banana",
"manzana", "pera");
    print("Máximo entre banana, manzana, pera: " + maxString);
```

```
    print("=== MÁXIMO POR LONGITUD ===");
    String maxPorLongitud <-
MaximoGenerico.encontrarMaximoPorLongitud("manzana", "pera", "limón");
    print("Máximo por longitud: " + maxPorLongitud);
}
class ComparadorNumerico{
    private:
        double precision;
        String tipoComparacion;
    public:
        ComparadorNumerico(double precision, String tipo){
            this.precision <- precision;
            this.tipoComparacion <- tipo;
        }

        boolean esIgual(double a, double b){
            return Math.abs(a - b) < this.precision;
        }

        double encontrarMaximo(double a, double b, double c){
            double max <- a;
            if(b > max) max <- b;
            if(c > max) max <- c;
            return max;
        }

        mostrarComparacion(double a, double b, double c){
            print("Comparando números con precisión: " + this.precision);
            print("Valores: " + a + ", " + b + ", " + c);
            print("Máximo encontrado: " + this.encontrarMaximo(a, b, c));
        }
}
class AnalizadorTexto{
    private:
        String[] textos;
        int totalTextos;
    public:
        AnalizadorTexto(){
            this.textos <- new String[50];
            this.totalTextos <- 0;
        }

        agregarTexto(String texto){
            if(this.totalTextos < 50){
                this.textos[this.totalTextos] <- texto;
                this.totalTextos <- this.totalTextos + 1;
            }
        }

        String encontrarTextoMasLargo(){
            if(this.totalTextos == 0) return null;
            String masLargo <- this.textos[0];
            for(int i <- 1; i < this.totalTextos; i++){
                if(this.textos[i].length() > masLargo.length()){
                    masLargo <- this.textos[i];
                }
            }
            return masLargo;
        }
```

```
    mostrarEstadisticas(){
        print("Total de textos: " + this.totalTextos);
        print("Texto más largo: " + this.encontrarTextoMasLargo());
        print("Longitud máxima: " + this.encontrarTextoMasLargo().length()); }}
```

3.

```
class ComparadorGenerico<T extends Comparable<T>>{
    private:
        String criterio;
    public:
        ComparadorGenerico(){
            this.criterio <- "natural";
        }

        T determinarMayor(T objeto1, T objeto2){
            int resultado <- objeto1.compareTo(objeto2);
            if (resultado > 0) {
                return objeto1;
            } else if (resultado < 0) {
                return objeto2;
            } else {
                return objeto1;
            }
        }

        boolean sonIguales(T objeto1, T objeto2){
            return objeto1.compareTo(objeto2) == 0;
        }

        void compararObjetos(T objeto1, T objeto2){
            print("Comparando: " + objeto1 + " vs " + objeto2);
            T mayor <- this.determinarMayor(objeto1, objeto2);
            print("Mayor: " + mayor);
            boolean iguales <- this.sonIguales(objeto1, objeto2);
            print("Son iguales: " + iguales);
            print("---");
        }
}
class Persona implements Comparable<Persona>{
    private:
        String nombre;
        int edad;
    public:
        Persona(String nombre, int edad){
            this.nombre <- nombre;
            this.edad <- edad;
        }

        int compareTo(Persona otra){
            return Integer.compare(this.edad, otra.edad);
        }

        String toString(){
            return this.nombre + " (" + this.edad + " años)";
        }

        boolean esMayorDeEdad(){
            return this.edad >= 18;
        }
```

```
}
class MainComparador{
    ComparadorGenerico<Integer> comparadorInt <- new
ComparadorGenerico<>();

    print("=== COMPARACIÓN DE ENTEROS ===");
    comparadorInt.compararObjetos(15, 25);
    comparadorInt.compararObjetos(100, 50);
    comparadorInt.compararObjetos(7, 7);

    ComparadorGenerico<String> comparadorString <- new
ComparadorGenerico<>();

    print("=== COMPARACIÓN DE STRINGS ===");
    comparadorString.compararObjetos("banana", "manzana");
    comparadorString.compararObjetos("zebra", "abeja");
    comparadorString.compararObjetos("igual", "igual");

    ComparadorGenerico<Persona> comparadorPersona <- new
ComparadorGenerico<>();

    Persona persona1 <- new Persona("Juan", 25);
    Persona persona2 <- new Persona("María", 30);

    print("=== COMPARACIÓN DE PERSONAS ===");
    comparadorPersona.compararObjetos(persona1, persona2);
}
class GestorComparaciones<T extends Comparable<T>>{
    private:
        T[] elementos;
        int totalComparaciones;
    public:
        GestorComparaciones(){
            this.elementos <- (T[]) new Comparable[20];
            this.totalComparaciones <- 0;
        }

        agregarElemento(T elemento){
            if(this.totalComparaciones < 20){
                this.elementos[this.totalComparaciones] <- elemento;
                this.totalComparaciones <- this.totalComparaciones + 1;
            }
        }

        T encontrarMaximo(){
            if(this.totalComparaciones == 0) return null;
            T max <- this.elementos[0];
            for(int i <- 1; i < this.totalComparaciones; i++){
                if(this.elementos[i].compareTo(max) > 0){
                    max <- this.elementos[i];
                }
            }
            return max;
        }

        void mostrarTodosElementos(){
            print("=== ELEMENTOS ALMACENADOS ===");
            for(int i <- 0; i < this.totalComparaciones; i++){
                print("Elemento " + i + ": " + this.elementos[i]);
            }
```

```
        print("Máximo encontrado: " + this.encontrarMaximo());
    }
}


4.
abstract class Vehiculo{
    protected:
        String marca;
        String modelo;
        double precio;
    public:
        Vehiculo(String marca, String modelo, double precio){
            this.marca <- marca;
            this.modelo <- modelo;
            this.precio <- precio;
        }

        abstract void mostrarInfo();

        String toString(){
            return this.marca + " " + this.modelo + " - Bs. " + this.precio;
        }

        boolean esCostoso(){
            return this.precio > 50000;
        }
}
class Auto extends Vehiculo{
    private:
        int numeroPuertas;
        String tipoCombustible;
    public:
        Auto(String marca, String modelo, double precio, int numeroPuertas,
String tipoCombustible){
            super(marca, modelo, precio);
            this.numeroPuertas <- numeroPuertas;
            this.tipoCombustible <- tipoCombustible;
        }

        void mostrarInfo(){
            print("Auto: " + this.marca + " " + this.modelo + " - " +
this.numeroPuertas + " puertas - " +
                    this.tipoCombustible + " - Bs. " + this.precio);
        }

        boolean esSedan(){
            return this.numeroPuertas == 4;
        }

        void cambiarTipoCombustible(String nuevoTipo){
            this.tipoCombustible <- nuevoTipo;
        }
}
class Moto extends Vehiculo{
    private:
        int cilindrada;
        String tipoMoto;
    public:
```

```
        Moto(String marca, String modelo, double precio, int cilindrada, String
tipoMoto){
            super(marca, modelo, precio);
            this.cilindrada <- cilindrada;
            this.tipoMoto <- tipoMoto;
        }

        void mostrarInfo(){
            print("Moto: " + this.marca + " " + this.modelo + " - " + this.cilindrada +
"cc - " +
                    this.tipoMoto + " - Bs. " + this.precio);
        }

        boolean esDeportiva(){
            return this.tipoMoto.equals("Deportiva");
        }

        void aumentarCilindrada(int incremento){
            this.cilindrada <- this.cilindrada + incremento;
        }
}
class Inventario<T extends Vehiculo>{
    private:
        T[] vehiculos;
        int count;
    public:
        Inventario(){
            this.vehiculos <- (T[]) new Vehiculo[100];
            this.count <- 0;
        }

        void agregar(T vehiculo){
            if (this.count < this.vehiculos.length) {
                this.vehiculos[this.count] <- vehiculo;
                this.count <- this.count + 1;
                print("Vehículo agregado: " + vehiculo.getClass().getSimpleName());
            } else {
                print("Inventario lleno, no se puede agregar más vehículos.");
            }
        }

        int contarVehiculosSuperioresA(double precio){
            int contador <- 0;
            for (int i <- 0; i < this.count; i++) {
                if (this.vehiculos[i].getPrecio() > precio) {
                    contador <- contador + 1;
                }
            }
            return contador;
        }

        void mostrarTodos(){
            print("=== INVENTARIO DE VEHÍCULOS ===");
            for (int i <- 0; i < this.count; i++) {
                T vehiculo <- this.vehiculos[i];
                print("Tipo: " + vehiculo.getClass().getSimpleName());
                print("Marca: " + vehiculo.getMarca());
                print("Modelo: " + vehiculo.getModelo());
                print("Precio: Bs. " + vehiculo.getPrecio());
            }
```

```
        }
}
```

5.

```
abstract class Publicacion {
    protected:
        String titulo;
        int anioPublicacion;

    public:
        Publicacion(String titulo, int anioPublicacion) {
            this.titulo <- titulo;
            this.anioPublicacion <- anioPublicacion;
        }

        String getTitulo() {
            return this.titulo;
        }

        setTitulo(String titulo) {
            this.titulo <- titulo;
        }

        abstract mostrarInfo();
}
class Autor {
    private:
        String nombre;
        String correo;

    public:
        Autor(String nombre, String correo) {
            this.nombre <- nombre;
            this.correo <- correo;
        }

        String getNombre() {
            return this.nombre;
        }

        setNombre(String nombre) {
            this.nombre <- nombre;
        }

        mostrarAutor() {
            print("Autor: " + this.nombre + " (Email: " + this.correo + ")");
        }
}
class Editorial {
    private:
        String nombre;
        String pais;

    public:
        Editorial(String nombre, String pais) {
            this.nombre <- nombre;
            this.pais <- pais;
```

```
        }

        String getNombre() {
            return this.nombre;
        }

        setNombre(String nombre) {
            this.nombre <- nombre;
        }

        mostrarEditorial() {
            print("Editorial: " + this.nombre + " - " + this.pais);
        }
}
class Asesor {
    private:
        String nombre;
        String especialidad;

    public:
        Asesor(String nombre, String especialidad) {
            this.nombre <- nombre;
            this.especialidad <- especialidad;
        }

        String getNombre() {
            return this.nombre;
        }

        setNombre(String nombre) {
            this.nombre <- nombre;
        }

        mostrarAsesor() {
            print("Asesor: " + this.nombre + " - Especialidad: " + this.especialidad);
        }
}
class Libro extends Publicacion {
    private:
        String isbn;
        Editorial editorial;
        Autor[] autores;
        int nAutores;
        static final int MAX_AUTORES = 10;

    public:
        Libro(String titulo, int anioPublicacion, String isbn, Editorial editorial) {
            super(titulo, anioPublicacion);
            this.isbn <- isbn;
            this.editorial <- editorial;
            this.autores <- new Autor[MAX_AUTORES];
            this.nAutores <- 0;
        }

        agregarAutor(Autor autor) {
            if (this.nAutores < MAX_AUTORES) {
                this.autores[this.nAutores] <- autor;
                this.nAutores <- this.nAutores + 1;
            } else {
                print("No se pueden agregar más autores, límite alcanzado.");
```

```
        }
    }

    mostrarInfo() {
        print("=== LIBRO ===");
        print("Título: " + this.titulo);
        print("Año de publicación: " + this.anioPublicacion);
        print("ISBN: " + this.isbn);
        if (this.editorial != null) {
            this.editorial.mostrarEditorial();
        }
        print("Autores:");
        for (int i <- 0; i < this.nAutores; i++) {
            this.autores[i].mostrarAutor();
        }
        print("===============");
    }
}
class Articulo extends Publicacion {
    private:
        String revista;
        int volumen;
        Autor[] autores;
        int nAutores;
        static final int MAX_AUTORES = 10;

    public:
        Articulo(String titulo, int anioPublicacion, String revista, int volumen) {
            super(titulo, anioPublicacion);
            this.revista <- revista;
            this.volumen <- volumen;
            this.autores <- new Autor[MAX_AUTORES];
            this.nAutores <- 0;
        }

        agregarAutor(Autor autor) {
            if (this.nAutores < MAX_AUTORES) {
                this.autores[this.nAutores] <- autor;
                this.nAutores <- this.nAutores + 1;
            } else {
                print("No se pueden agregar más autores, límite alcanzado.");
            }
        }

        mostrarInfo() {
            print("=== ARTÍCULO ===");
            print("Título: " + this.titulo);
            print("Año de publicación: " + this.anioPublicacion);
            print("Revista: " + this.revista);
            print("Volumen: " + this.volumen);
            print("Autores:");
            for (int i <- 0; i < this.nAutores; i++) {
                this.autores[i].mostrarAutor();
            }
            print("================");
        }
}
class Tesis extends Publicacion {
    private:
        String grado;
```

```
    Asesor asesor;
    Autor[] autores;
    int nAutores;
    static final int MAX_AUTORES = 10;

  public:
    Tesis(String titulo, int anioPublicacion, String grado, String nombreAsesor,
String especialidadAsesor) {
        super(titulo, anioPublicacion);
        this.grado <- grado;
        this.asesor <- new Asesor(nombreAsesor, especialidadAsesor);
        this.autores <- new Autor[MAX_AUTORES];
        this.nAutores <- 0;
    }

    agregarAutor(Autor autor) {
        if (this.nAutores < MAX_AUTORES) {
            this.autores[this.nAutores] <- autor;
            this.nAutores <- this.nAutores + 1;
        } else {
            print("No se pueden agregar más autores, límite alcanzado.");
        }
    }

    mostrarInfo() {
        print("=== TESIS ===");
        print("Título: " + this.titulo);
        print("Año de publicación: " + this.anioPublicacion);
        print("Grado: " + this.grado);
        if (this.asesor != null) {
            this.asesor.mostrarAsesor();
        }
        print("Autores:");
        for (int i <- 0; i < this.nAutores; i++) {
            this.autores[i].mostrarAutor();
        }
        print("==============");
    }
}
class Repositorio<T> {
  private:
    T[] items;
    int nItems;
    static final int CAPACITY = 100;

  public:
    Repositorio() {
        this.items <- new T[CAPACITY];
        this.nItems <- 0;
    }

    agregar(T item) {
        if (this.nItems < CAPACITY) {
            this.items[this.nItems] <- item;
            this.nItems <- this.nItems + 1;
            print("Item agregado al repositorio: " +
item.getClass().getSimpleName());
        } else {
            print("Repositorio lleno, no se puede agregar: " +
item.getClass().getSimpleName());
```

```
        }
    }

    boolean eliminar(T item) {
        for (int i <- 0; i < this.nItems; i++) {
            if (this.items[i] != null && this.items[i].equals(item)) {
                for (int j <- i; j < this.nItems - 1; j++) {
                    this.items[j] <- this.items[j + 1];
                }
                this.items[this.nItems - 1] <- null;
                this.nItems <- this.nItems - 1;
                print("Item eliminado del repositorio: " +
item.getClass().getSimpleName());
                return true;
            }
        }
        print("Item no encontrado en el repositorio");
        return false;
    }

    mostrarTodo() {
        print("=== CONTENIDO DEL REPOSITORIO ===");
        if (this.nItems == 0) {
            print("El repositorio está vacío");
        } else {
            for (int i <- 0; i < this.nItems; i++) {
                if (this.items[i] instanceof Publicacion) {
                    ((Publicacion) this.items[i]).mostrarInfo();
                } else if (this.items[i] instanceof Autor) {
                    ((Autor) this.items[i]).mostrarAutor();
                } else {
                    print(this.items[i].toString());
                }
            }
        }
    }
}
class Main {
    public static main() {
        Repositorio<Publicacion> repositorioPublicaciones <- new
Repositorio<>();
        Repositorio<Autor> repositorioAutores <- new Repositorio<>();

        Autor autor1 <- new Autor("Gabriel García Márquez",
"ggarcia@email.com");
        Autor autor2 <- new Autor("Mario Vargas Llosa", "mvargas@email.com");
        Autor autor3 <- new Autor("Carlos Fuentes", "cfuentes@email.com");
        Autor autor4 <- new Autor("Dr. Ana López", "alopez@university.edu");

        repositorioAutores.agregar(autor1);
        repositorioAutores.agregar(autor2);
        repositorioAutores.agregar(autor3);
        repositorioAutores.agregar(autor4);

        Editorial editorial1 <- new Editorial("Sudamericana", "Argentina");

        Libro libro1 <- new Libro("Cien años de soledad", 1967, "978-84-376-0494-
7", editorial1);
        libro1.agregarAutor(autor1);
```

```
        Articulo articulo1 <- new Articulo("Literatura Latinoamericana
Contemporánea", 2020, "Revista de Letras", 15);
        articulo1.agregarAutor(autor2);
        articulo1.agregarAutor(autor3);

        Tesis tesis1 <- new Tesis("Impacto de la Globalización en la Literatura
Contemporánea", 2021, "Doctorado en Literatura", "Dr. Pedro Martínez",
"Literatura Latinoamericana");
        tesis1.agregarAutor(autor4);

        repositorioPublicaciones.agregar(libro1);
        repositorioPublicaciones.agregar(articulo1);
        repositorioPublicaciones.agregar(tesis1);

        print("=== REPOSITORIO DE AUTORES ===");
        repositorioAutores.mostrarTodo();

        print("\n=== REPOSITORIO DE PUBLICACIONES ===");
        repositorioPublicaciones.mostrarTodo();

        print("\n=== DEMOSTRACIÓN DE COMPOSICIÓN ===");
        print("Información del asesor de la tesis:");
        tesis1.getAsesor().mostrarAsesor();

        print("\n=== EJEMPLO DE ELIMINACIÓN ===");
        print("Eliminando el artículo del repositorio...");
        repositorioPublicaciones.eliminar(articulo1);

        print("\nRepositorio después de la eliminación:");
        repositorioPublicaciones.mostrarTodo();

        print("\nTamaño del repositorio de publicaciones: " +
repositorioPublicaciones.size());
        print("Tamaño del repositorio de autores: " + repositorioAutores.size());
    }
}
```