CENTRO UNIVERSITARIO UAEM ATLACOMULCO

MATERIA

PARADIGMAS DE LA PROGRAMACIÓN

DOCENTE

JULIO ALBERTO DE LA TEJA LÓPEZ

ALUMNO

ALEXIS VALENCIA MARTÍNEZ

CARRERA

LICENCIATURA EN INGENIERÍA EN SISTEMAS COMPUTACIONALES

ICO-27

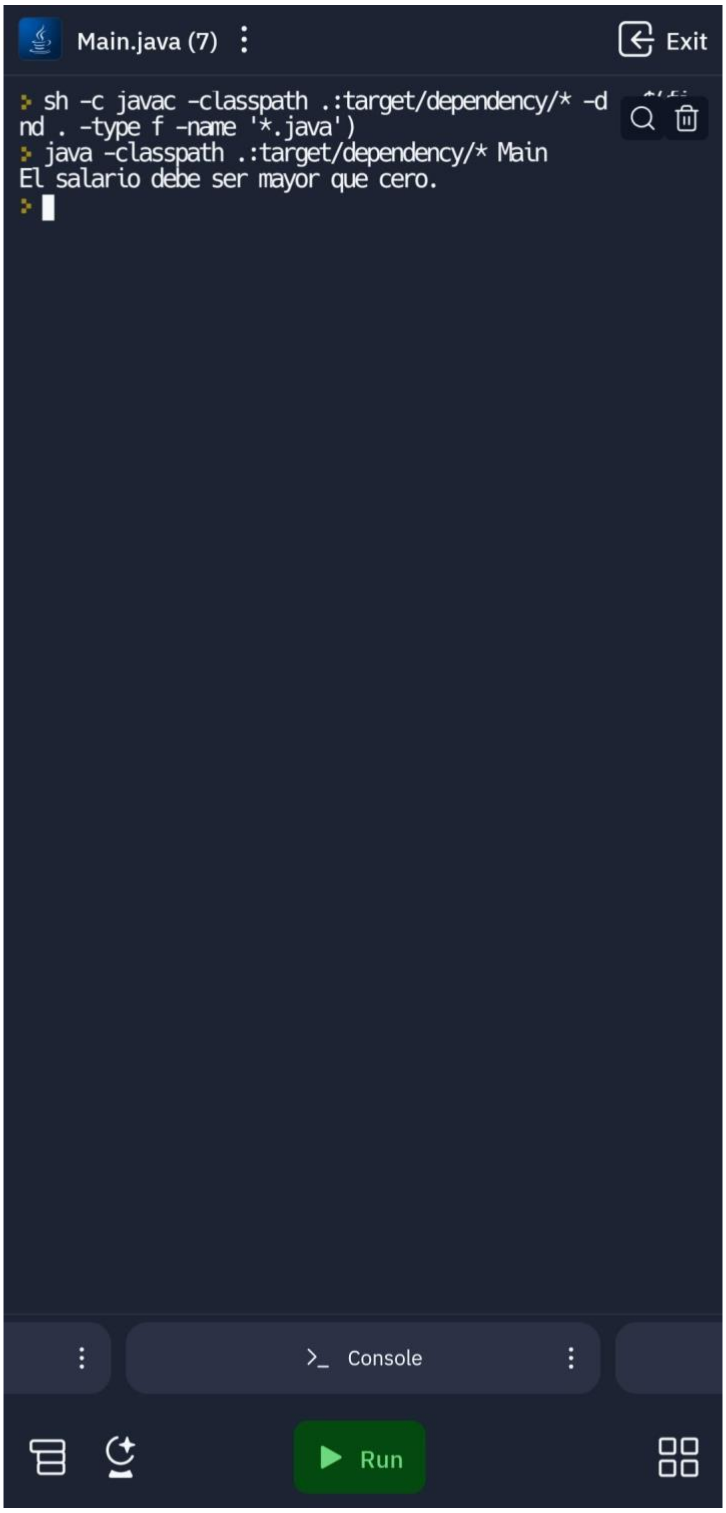
FECHA DE ENTREGA

12/SEPTIEMBRE/2023

```
import java.util.Scanner;

abstract class Empleado {
   private String nombre;
   private double salario;

   public Empleado(String nombre, double salario) throws SalarioInvalidoException {
      if (salario <= 0) {
         throw new SalarioInvalidoException("El salario debe ser mayor que cero.");
      }
      this.nombre = nombre;
      this.salario = salario;
   }

   public String getNombre() {
      return nombre;
   }

   public double getSalario() {
      return salario;
   }
}

class EmpleadoPorHora extends Empleado {
   private int horasTrabajadas;
   private double valorHora;

   public EmpleadoPorHora(String nombre, double salario, int horasTrabajadas, double valorHora) throws
SalarioInvalidoException {
      super(nombre, salario);
```

```
        if (salario <= 0) {
            throw new SalarioInvalidoException("El salario debe ser mayor que cero.");
        }
        this.horasTrabajadas = horasTrabajadas;
        this.valorHora = valorHora;
    }

    public double calcularSalario() {
        return horasTrabajadas * valorHora;
    }
}

class EmpleadoAsalariado extends Empleado implements Bonificable {
    private int diasTrabajados;
    private double sueldoMensual;

    public EmpleadoAsalariado(String nombre, double salario, int diasTrabajados, double sueldoMensual)
throws SalarioInvalidoException {
        super(nombre, salario);
        if (salario <= 0) {
            throw new SalarioInvalidoException("El salario debe ser mayor que cero.");
        }
        this.diasTrabajados = diasTrabajados;
        this.sueldoMensual = sueldoMensual;
    }

    public double calcularSalario() {
        return (sueldoMensual / 30) * diasTrabajados;
    }
    public double calcularBonificacion() {
        return calcularSalario() * 0.1; // bonificación del 10%
    }
}

interface Bonificable {
    public double calcularBonificacion();
}

class SalarioInvalidoException extends Exception {
    public SalarioInvalidoException(String mensaje) {
        super(mensaje);
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        try {
            EmpleadoPorHora emp1 = new EmpleadoPorHora("Juan", 0, 8, 10);
            System.out.println("El salario de " + emp1.getNombre() + " es: " + emp1.calcularSalario());

            EmpleadoAsalariado emp2 = new EmpleadoAsalariado("Pedro", 10000, 20, 30000);
            System.out.println("El salario de " + emp2.getNombre() + " es: " + emp2.calcularSalario());
            if (emp2 instanceof Bonificable) { // si la instancia es de tipo 'EmpleadoAsalariado'
                System.out.println("La bonificación de " + emp2.getNombre() + " es: " + ((Bonificable)
emp2).calcularBonificacion());
            }
            EmpleadoAsalariado emp3 = new EmpleadoAsalariado("María", -10000, 25, 40000); // intenta crear un empleado con salario negativo
        } catch (SalarioInvalidoException e) { // captura la excepción 'SalarioInvalidoException'
            System.out.println(e.getMessage()); // muestra un mensaje de error
        }
    }
}
```

- SALIDA



```
> sh -c javac -classpath .:target/dependency/* -d
nd . -type f -name '*.java')
> java -classpath .:target/dependency/* Main
El salario debe ser mayor que cero.
> 
```

• What is the difference between abstract classes and interfaces?

The main difference between the two is that abstract classes can contain both concrete and abstract methods, while interfaces can only contain abstract methods.

• What is the purpose of using abstract classes and interfaces?

The use of abstract classes and interfaces allows for the creation of reusable and modular code. By defining common behavior in an abstract class or interface, it can be ensured that all subclasses implement that behavior. This makes the code easier to maintain and update.

• What are the advantages of using your own exceptions?

The use of custom exceptions allows developers to create exceptions specific to their application. This can make the code easier to understand and debug, as custom exceptions can provide detailed information about the error. Additionally, custom exceptions can be caught and handled specifically rather than simply being propagated up the call stack.

• What is the advantage of polymorphism?

Polymorphism allows developers to write code that can work with objects of different types. This makes the code more flexible and reusable, as a single piece of code can be written that works with several different types of objects.

• What improvement would you make to this exercise?

Although this thing about abstract classes and interfaces has gotten a little complicated for me, I wouldn’t change anything to the code, since it seems to me that thanks to this it is easier to understand and order the code