Nombre de la práctica	UNIDAD 1. INTRODUCCIÓN AL PARADIGMA DE LA POO			No.	1
Signature:	Programación Orientada a Objetos	Carrera:	Ingeniería en Sistemas Computacionales	Duración de la práctica (Hrs)	8 horas

Nombre: Vanesa Hernández Martínez
Grupo: 3203

I. Competencia(s) específica(s):

Comprende y aplica los conceptos del paradigma de programación orientada a objetos para modelar Situaciones de la vida real.

Encuadre con CACEI: Registra el (los) atributo(s) de egreso y los criterios de desempeño que se evaluarán en esta práctica.

No. atributo	Atributos de egreso del PE que impactan en la asignatura	Criterios de desempeño	
2	El estudiante diseñará esquemas de trabajo y procesos, usando metodologías congruentes en la resolución de problemas de ingeniería en sistemas computacionales	1	Identifica metodologías y procesos empleados en la resolución de problemas
		2	Diseña soluciones a problemas, empleando metodologías apropiadas al área
3	El estudiante plantea soluciones basadas En tecnologías empleando su juicio ingenieril para valorar necesidades, recursos y resultados esperados.	1	Emplea los conocimientos adquiridos para el desarrollar soluciones
		2	Analiza y comprueba resultados

II. Lugar de realización de la práctica (laboratorio, taller, aula u otro):

Actividades en aula de clases y en equipo personal

III. Material empleado:

Laptop
Software: starUML
Libreta

*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

IV. Desarrollo de la práctica:

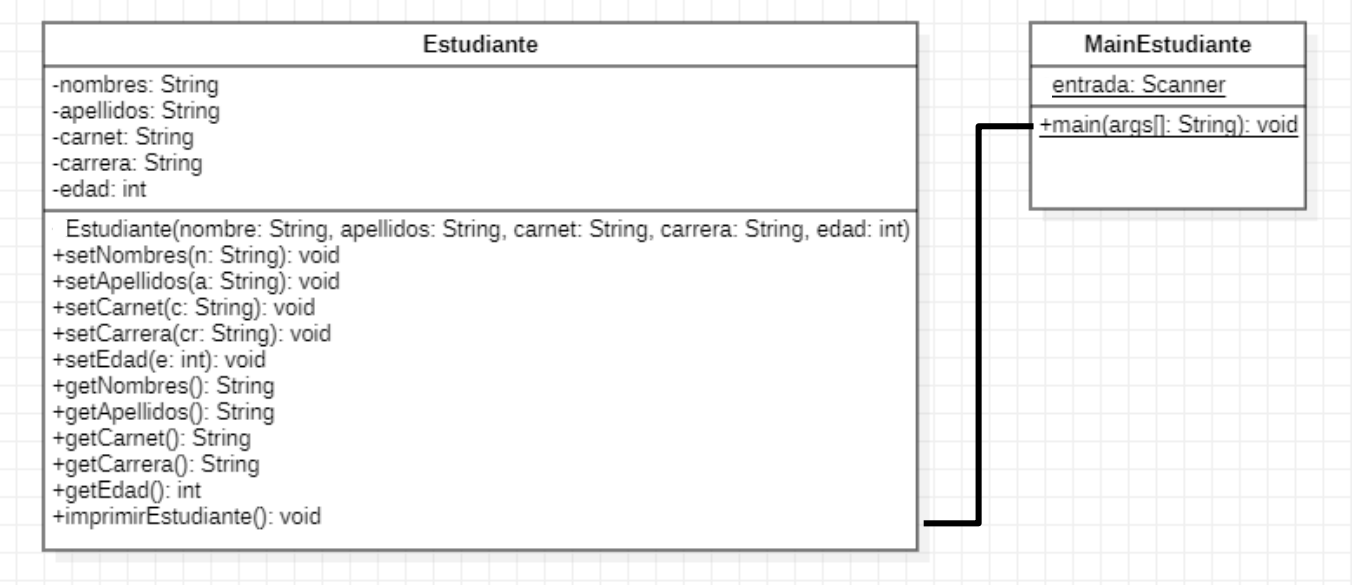
UNIDAD 1. INTRODUCCIÓN AL PARADIGMA DE LA POO

A. CÓDIGO 1

```java
import java.util.Scanner;

class Estudiante{

    private String nombres, apellidos, carnet, carrera;
    private int edad;

    Estudiante( String nombres, String apellidos, String carnet, String carrera, int edad){
        setNombres(nombres);
        setApellidos(apellidos);
        setCarnet(carnet);
        setCarrera(carrera);
        setEdad(edad);
    }
    /* Metodos Modificadores */
    public void setNombres(String n){ nombres = n; }
    public void setApellidos(String a){ apellidos = a; }
    public void setCarnet(String c){ carnet = c; }
    public void setCarrera(String cr){ carrera = cr; }
    public void setEdad(int e){ edad = e; }
    /* Metodos Accesores */
    public String getNombres(){ return nombres; }
    public String getApellidos(){ return apellidos; }
    public String getCarnet(){ return carnet; }
    public String getCarrera(){ return carrera; }
    public int getEdad(){ return edad; }

    public void imprimirEstudiante(){
        System.out.print("\nNombres: " +getNombres()+"\nApellidos: "+getApellidos()+"\nCarnet: "+getCarnet()+
                         "\nCarrera: " +getCarrera()+"\nEdad: "+getEdad() );
    }
}
public class MainEstudiante{

    static Scanner entrada = new Scanner(System.in);

    public static void main(String[] args) {
        // TODO, add your application code
        String nombres, apellidos, carnet, carrera;
        int edad;
        System.out.println("Favor ingresar los nombres: ");
        nombres = entrada.nextLine();
        System.out.println("Favor ingresar los apellidos: ");
        apellidos = entrada.nextLine();
        System.out.println("Favor ingresar el numero de carnet: ");
        carnet = entrada.nextLine();
        System.out.println("Favor ingresar nombre de carrera: ");
        carrera = entrada.nextLine();
        System.out.println("Favor ingresar edad: ");
        edad = entrada.nextInt();
        Estudiante e;
        e = new Estudiante(nombres,apellidos,carnet,carrera,edad);
        e.imprimirEstudiante();
    }
}
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

B. DIAGRAMA 1


Estudiante
-nombres: String
-apellidos: String
-carnet: String
-carrera: String
-edad: int
Estudiante(nombre: String, apellidos: String, carnet: String, carrera: String, edad: int)
+setNombres(n: String): void
+setApellidos(a: String): void
+setCarnet(c: String): void
+setCarrera(cr: String): void
+setEdad(e: int): void
+getNombres(): String
+getApellidos(): String
+getCarnet(): String
+getCarrera(): String
+getEdad(): int
+imprimirEstudiante(): void
MainEstudiante
entrada: Scanner
+main(args[]: String): void

C. CÓDIGO 2

```
1  // Fig. 9.4: EmpleadoPorComision.java
2  // La clase EmpleadoPorComision representa a un empleado que
3  // recibe como sueldo un porcentaje de las ventas brutas.
4  public class EmpleadoPorComision extends Object
5  {
6     private final String primerNombre;
7     private final String apellidoPaterno;
8     private final String numeroSeguroSocial;
9     private double ventasBrutas; // ventas totales por semana
10    private double tarifaComision; // porcentaje de comisión
11
12    // constructor con cinco argumentos
13    public EmpleadoPorComision(String primerNombre, String apellidoPaterno,
14       String numeroSeguroSocial, double ventasBrutas,
15       double tarifaComision)
16    {
17       // la llamada implícita al constructor predeterminado de Object ocurre aquí
18
19       // si ventasBrutas no es válida, lanza excepción
20       if (ventasBrutas < 0.0)
21          throw new IllegalArgumentException(
22             "Las ventas brutas deben ser >= 0.0");
23
24       // si tarifaComision no es válida, lanza excepción
25       if (tarifaComision <= 0.0 || tarifaComision >= 1.0)
26          throw new IllegalArgumentException(
27             "La tarifa de comision debe ser > 0.0 y < 1.0");
28
29       this.primerNombre = primerNombre;
30       this.apellidoPaterno = apellidoPaterno;
31       this.numeroSeguroSocial = numeroSeguroSocial;
32       this.ventasBrutas = ventasBrutas;
33       this.tarifaComision = tarifaComision;
34    } // fin del constructor
35
36    // devuelve el primer nombre
37    public String obtenerPrimerNombre()
38    {
39       return primerNombre;
40    }
41
42    // devuelve el apellido paterno
43    public String obtenerApellidoPaterno()
44    {
45       return apellidoPaterno;
46    }
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

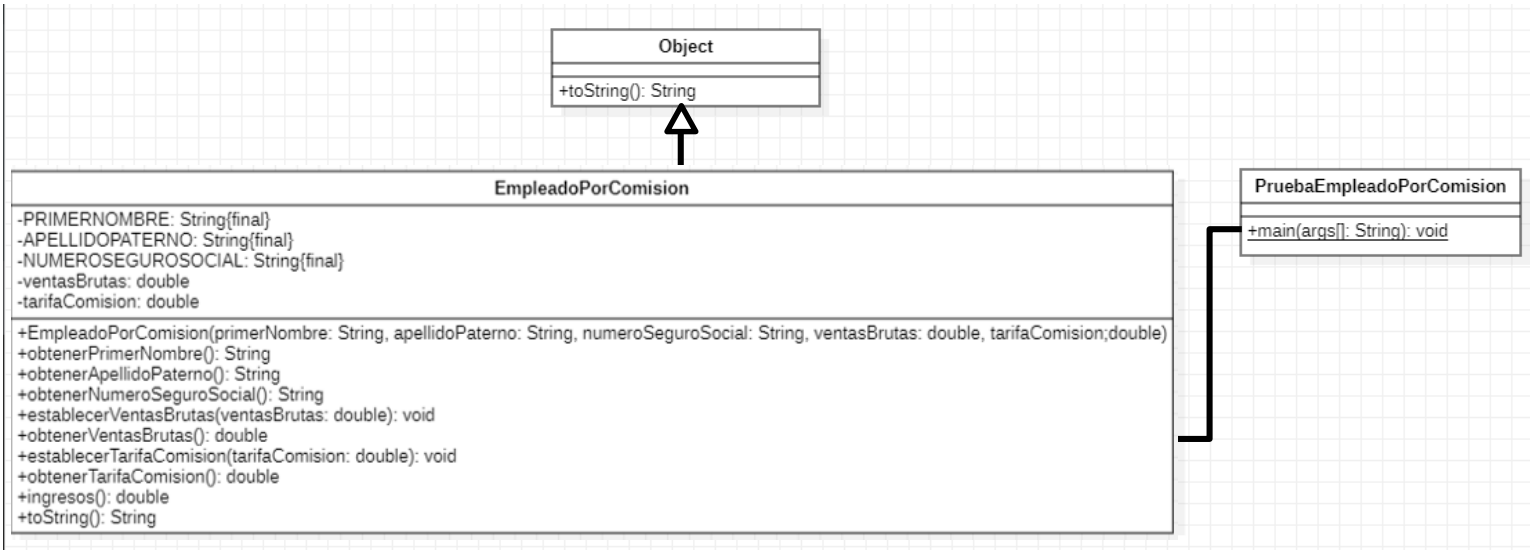
```
47
48      // devuelve el número de seguro social
49      public String obtenerNumeroSeguroSocial()
50      {
51         return numeroSeguroSocial;
52      } // fin del método obtenerNumeroSeguroSocial
53
54      // establece el monto de ventas brutas
55      public void establecerVentasBrutas(double ventasBrutas)
56      {
57         if (ventasBrutas >= 0.0)
58            throw new IllegalArgumentException(
59               "Las ventas brutas deben ser >= 0.0");
60
61         this.ventasBrutas = ventasBrutas;
62      }
63
64      // devuelve el monto de ventas brutas
65      public double obtenerVentasBrutas()
66       {
67         return ventasBrutas;
68      }
69
70      // establece la tarifa de comisión
71      public void establecerTarifaComision(double tarifaComision)
72      {
73         if (tarifaComision <= 0.0 || tarifaComision >= 1.0)
74            throw new IllegalArgumentException(
75               "La tarifa de comisión debe ser > 0.0 y < 1.0");
76
77         this.tarifaComision = tarifaComision;
78      }
79
80      // devuelve la tarifa de comisión
81      public double obtenerTarifaComision()
82      {
83         return tarifaComision;
84      }
85
86      // calcula los ingresos
87      public double ingresos()
88      {
89         return tarifaComision * ventasBrutas;
90      }
91
92      // devuelve representación String del objeto EmpleadoPorComision
93      @Override // indica que este método sobrescribe el método de una superclase
94      public String toString()
95      {
96         return String.format("%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
97            "empleado por comision", primerNombre, apellidoPaterno,
98            "numero de seguro social", numeroSeguroSocial,
99            "ventas brutas", ventasBrutas,
100           "tarifa de comision", tarifaComision);
101     }
102  } // fin de la clase EmpleadoPorComision
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```
1  // Fig. 9.5: PruebaEmpleadoPorComision.java
2  // Programa de prueba de la clase EmpleadoPorComision.
3
4  public class PruebaEmpleadoPorComision
5  {
6     public static void main(String[] args)
7     {
8        // crea instancia de objeto EmpleadoPorComision
9        EmpleadoPorComision empleado = new EmpleadoPorComision(
10          "Sue", "Jones", "222-22-2222", 10000, .06);
11
12       // obtiene datos del empleado por comisión
13       System.out.println(
14          "Informacion del empleado obtenida por los metodos establecer:");
15       System.out.printf("%n%s %s%n", "El primer nombre es",
16          empleado.obtenerPrimerNombre());
17       System.out.printf("%s %s%n", "El apellido paterno es",
18          empleado.obtenerApellidoPaterno());
19       System.out.printf("%s %s%n", "El numero de seguro social es",
20          empleado.obtenerNumeroSeguroSocial());
21       System.out.printf("%s %.2f%n", "Las ventas brutas son",
22          empleado.obtenerVentasBrutas());
23       System.out.printf("%s %.2f%n", "La tarifa de comision es",
24          empleado.obtenerTarifaComision());
25
26       empleado.establecerVentasBrutas(500);
27       empleado.establecerTarifaComision(.1);
28
29       System.out.printf("%n%s:%n%n%s%n",
30          "Informacion actualizada del empleado, obtenida mediante toString",
            empleado);
31    } // fin de main
32 } // fin de la clase PruebaEmpleadoPorComision
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

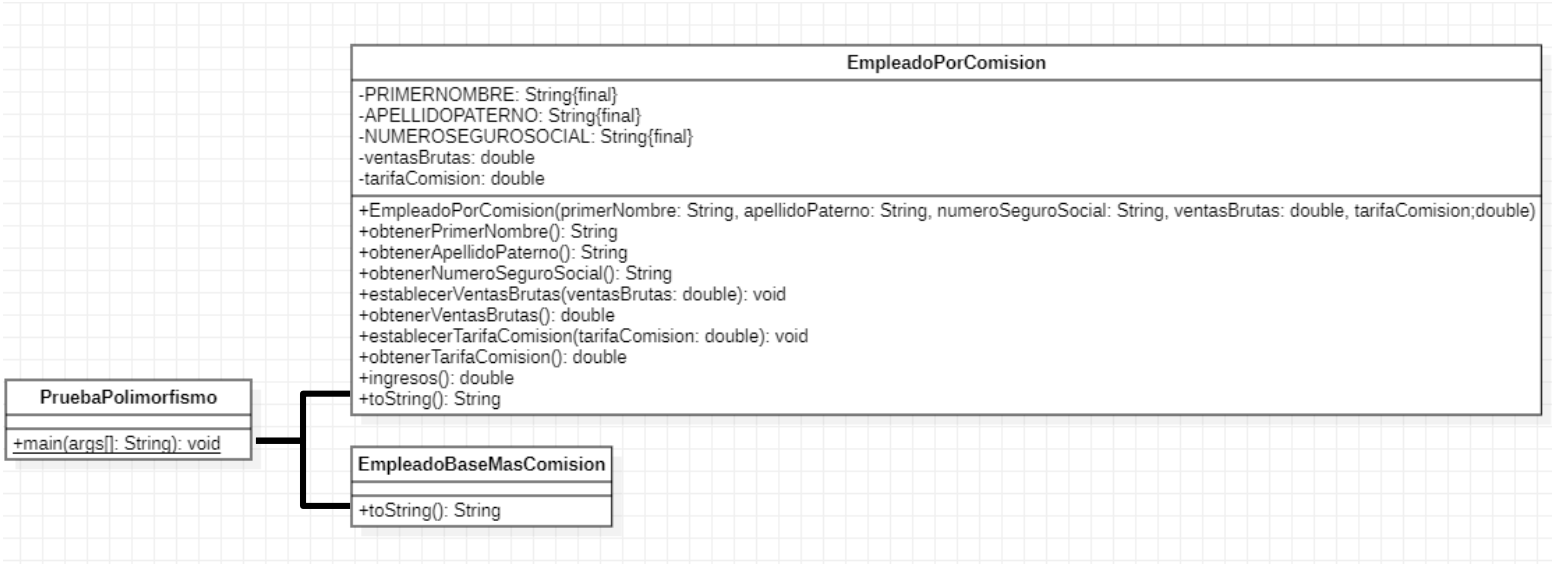
D. DIAGRAMA 2


Object
+toString(): String
EmpleadoPorComision
-PRIMERNOMBRE: String{final}
-APELLIDOPATERNO: String{final}
-NUMEROSEGUROSOCIAL: String{final}
-ventasBrutas: double
-tarifaComision: double
+EmpleadoPorComision(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String, ventasBrutas: double, tarifaComision;double)
+obtenerPrimerNombre(): String
+obtenerApellidoPaterno(): String
+obtenerNumeroSeguroSocial(): String
+establecerVentasBrutas(ventasBrutas: double): void
+obtenerVentasBrutas(): double
+establecerTarifaComision(tarifaComision: double): void
+obtenerTarifaComision(): double
+ingresos(): double
+toString(): String
PruebaEmpleadoPorComision
+main(args[]: String): void

E. CÓDIGO 3

```
1  // Fig. 10.1: PruebaPolimorfismo.java
2  // Asignación de referencias a la superclase y la subclase, a
3  // variables de la superclase y la subclase.
4
5  public class PruebaPolimorfismo
6  {
7     public static void main(String[] args)
8     {
9        // asigna la referencia a la superclase a una variable de la superclase
10       EmpleadoPorComision empleadoPorComision = new EmpleadoPorComision(
11          "Sue", "Jones", "222-22-2222", 10000, .06);
12
13       // asigna la referencia a la subclase a una variable de la subclase
14       EmpleadoBaseMasComision empleadoBaseMasComision =
15          new EmpleadoBaseMasComision(
16          "Bob", "Lewis", "333-33-3333", 5000, .04, 300);
17
18       // invoca a toString en un objeto de la superclase, usando una variable de
            la superclase
19       System.out.printf("%s %s:\n\n%s\n\n%s\n\n%s\n",
20          "Llamada a toString de EmpleadoPorComision con referencia de superclase ",
21          "a un objeto de la superclase", empleadoPorComision.toString());
22
23       // invoca a toString en un objeto de la subclase, usando una variable de
            la subclase
24       System.out.printf("%s %s:\n\n%s\n\n%s\n\n%s\n",
25          "Llamada a toString de EmpleadoBaseMasComision con referencia",
26          "de subclase a un objeto de la subclase",
27          empleadoBaseMasComision.toString());
28
29       // invoca a toString en un objeto de la subclase, usando una variable de
            la superclase
30       EmpleadoPorComision empleadoPorComision2 =
31          empleadoBaseMasComision;
32       System.out.printf("%s %s:\n\n%s\n\n%s\n",
33          "Llamada a toString de EmpleadoBaseMasComision con referencia de
             superclase",
34          "a un objeto de la subclase", empleadoPorComision2.toString());
35    } // fin de main
36 } // fin de la clase PruebaPolimorfismo
```

F. DIAGRAMA 3



*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

G. CÓDIGO 4

```
1  // Fig. 10.4: Empleado.java
2  // La superclase abstracta Empleado.
3
4  public abstract class Empleado
5  {
6     private final String primerNombre;
7     private final String apellidoPaterno;
8     private final String numeroSeguroSocial;
9
10    // constructor
11    public Empleado(String primerNombre, String apellidoPaterno,
12       String numeroSeguroSocial)
13    {
14       this.primerNombre = primerNombre;
15       this.apellidoPaterno = apellidoPaterno;
16       this.numeroSeguroSocial = numeroSeguroSocial;
17    }
18
19    // devuelve el primer nombre
20    public String obtenerPrimerNombre()
21    {
22       return primerNombre;
23    }
24
25    // devuelve el apellido paterno
26    public String obtenerApellidoPaterno()
27    {
28       return apellidoPaterno;
29    }
30
31    // devuelve el número de seguro social
32    public String obtenerNumeroSeguroSocial()
33    {
34       return numeroSeguroSocial;
35    }
36
37    // devuelve representación String de un objeto Empleado
38    @Override
39    public String toString()
40    {
41       return String.format("%s %s%nnumero de seguro social: %s",
42          obtenerPrimerNombre(), obtenerApellidoPaterno(),
43          obtenerNumeroSeguroSocial());
44    }
45
46    // método abstracto sobrescrito por las subclases concretas
47    public abstract double ingresos(); // aquí no hay implementación
48 } // fin de la clase abstracta Empleado
```

Fig. 10.4 | La superclase abstracta `Empleado` (parte 2 de 2).

*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```java
// Fig. 10.5: EmpleadoAsalariado.java
// La clase concreta EmpleadoAsalariado extiende a la clase abstracta Empleado.

public class EmpleadoAsalariado extends Empleado
{
   private double salarioSemanal;

   // constructor
   public EmpleadoAsalariado(String primerNombre, String apellidoPaterno,
      String numeroSeguroSocial, double salarioSemanal)
   {
      super(primerNombre, apellidoPaterno, numeroSeguroSocial);

      if (salarioSemanal < 0.0)
         throw new IllegalArgumentException(
            "El salario semanal debe ser >= 0.0");

      this.salarioSemanal = salarioSemanal;
   }

   // establece el salario
   public void establecerSalarioSemanal(double salarioSemanal)
   {
      if (salarioSemanal < 0.0)
         throw new IllegalArgumentException(
            "El salario semanal debe ser >= 0.0");

      this.salarioSemanal = salarioSemanal;
   }

   // devuelve el salario
   public double obtenerSalarioSemanal()
   {
      return salarioSemanal;
   }

   // calcula los ingresos; sobrescribe el método abstracto ingresos en Empleado
   @Override
   public double ingresos()
   {
      return obtenerSalarioSemanal();
   }

   // devuelve representación String de un objeto EmpleadoAsalariado
   @Override
   public String toString()
   {
      return String.format("empleado asalariado: %s%n%s: $%,.2f",
         super.toString(), "salario semanal", obtenerSalarioSemanal());
   }
} // fin de la clase EmpleadoAsalariado
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```
1  // Fig. 10.6: EmpleadoPorHoras.java
2  // La clase EmpleadoPorHoras extiende a Empleado.
3
4  public class EmpleadoPorHoras extends Empleado
5  {
6     private double sueldo; // sueldo por hora
7     private double horas; // horas trabajadas por semana
8
9     // constructor
10    public EmpleadoPorHoras(String primerNombre, String apellidoPaterno,
11       String numeroSeguroSocial, double sueldo, double horas)
12    {
13       super(primerNombre, apellidoPaterno, numeroSeguroSocial);
14
```

```
15       if (sueldo < 0.0) // valida sueldo
16          throw new IllegalArgumentException(
17             "El sueldo por horas debe ser >= 0.0");
18
19       if ((horas < 0.0) || (horas > 168.0)) // valida horas
20          throw new IllegalArgumentException(
21             "Las horas trabajadas deben ser >= 0.0 y <= 168.0");
22
23       this.sueldo = sueldo;
24       this.horas = horas;
25    }
26
27    // establece el sueldo
28    public void establecerSueldo(double sueldo)
29    {
30       if (sueldo < 0.0) // valida sueldo
31          throw new IllegalArgumentException(
32             "El sueldo por horas debe ser >= 0.0");
33
34       this.sueldo = sueldo;
35    }
36
37    // devuelve el sueldo
38    public double obtenerSueldo()
39    {
40       return sueldo;
41    }
42
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```
43     // establece las horas trabajadas
44     public void establecerHoras(double horas)
45     {
46        if ((horas < 0.0) || (horas > 168.0)) // valida horas
47           throw new IllegalArgumentException(
48              "Las horas trabajadas deben ser >= 0.0 y <= 168.0");
49
50        this.horas = horas;
51     }
52
53     // devuelve las horas trabajadas
54     public double obtenerHoras()
55     {
56        return horas;
57     }
58
59     // calcula los ingresos; sobrescribe el método abstracto ingresos en Empleado
60     @Override
61     public double ingresos()
62     {
63        if (obtenerHoras() <= 40) // no hay tiempo extra
64           return obtenerSueldo() * obtenerHoras();
65        else
66           return 40 * obtenerSueldo() + (obtenerHoras() - 40) * obtenerSueldo() * 1.5;
67     }
```

```
68
69     // devuelve representación String de un objeto EmpleadoPorHoras
70     @Override
71     public String toString()
72     {
73        return String.format("empleado por horas: %s%n%s: $%,.2f; %s: %,.2f",
74           super.toString(), "sueldo por hora", obtenerSueldo(),
75           "horas trabajadas", obtenerHoras());
76     }
77  } // fin de la clase EmpleadoPorHoras
```

Fig. 10.6 | La clase EmpleadoPorHoras extiende a Empleado (parte 3 de 3).

```java
// Fig. 10.7: EmpleadoPorComision.java
// La clase EmpleadoPorComision extiende a Empleado.

public class EmpleadoPorComision extends Empleado
{
   private double ventasBrutas; // ventas totales por semana
   private double tarifaComision; // porcentaje de comisión

   // constructor
   public EmpleadoPorComision(String primerNombre, String apellidoPaterno,
      String numeroSeguroSocial, double ventas,
      double tarifaComision)
   {
      super(primerNombre, apellidoPaterno, numeroSeguroSocial);

      if (tarifaComision <= 0.0 || tarifaComision >= 1.0) // valida
         throw new IllegalArgumentException(
            "La tarifa de comision debe ser > 0.0 y < 1.0");

      if (ventasBrutas < 0.0)
         throw new IllegalArgumentException("Las ventas brutas deben ser >= 0.0");

      this.ventasBrutas = ventasBrutas;
      this.tarifaComision = tarifaComision;
   }

   // establece el monto de ventas brutas
   public void establecerVentasBrutas(double ventasBrutas)
   {
      if (ventasBrutas < 0.0)
         throw new IllegalArgumentException("Las ventas brutas deben ser >= 0.0");

      this.ventasBrutas = ventasBrutas;
   }

   // devuelve el monto de ventas brutas
   public double obtenerVentasBrutas()
   {
      return ventasBrutas;
   }
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```
41
42     // establece la tarifa de comisión
43     public void establecerTarifaComision(double tarifaComision)
44     {
45        if (tarifaComision <= 0.0 || tarifaComision >= 1.0) // valida
46           throw new IllegalArgumentException(
47              "La tarifa de comision debe ser > 0.0 y < 1.0");
48
49        this.tarifaComision = tarifaComision;
50     }
51
52     // devuelve la tarifa de comisión
53     public double obtenerTarifaComision()
54     {
55        return tarifaComision;
56     }
57
58     // calcula los ingresos; sobrescribe el método abstracto ingresos en Empleado
59     @Override
60     public double ingresos()
61     {
62        return obtenerTarifaComision() * obtenerVentasBrutas();
63     }
64
65     // devuelve representación String de un objeto EmpleadoPorComision
66     @Override
67     public String toString()
68     {
69        return String.format("%s: %s\n%s: $%,.2f; %s: %.2f",
70           "empleado por comision", super.toString(),
71           "ventas brutas", obtenerVentasBrutas(),
72           "tarifa de comision", obtenerTarifaComision());
73     }
74  } // fin de la clase EmpleadoPorComision
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```java
// Fig. 10.8: EmpleadoBaseMasComision.java
// La clase EmpleadoBaseMasComision extiende a EmpleadoPorComision.

public class EmpleadoBaseMasComision extends EmpleadoPorComision
{
   private double salarioBase; // salario base por semana

   // constructor
   public EmpleadoBaseMasComision(String primerNombre, String apellidoPaterno,
      String numeroSeguroSocial, double ventasBrutas,
      double tarifaComision, double salarioBase)
   {
     super(primerNombre, apellidoPaterno, numeroSeguroSocial,
        ventasBrutas, tarifaComision);

      if (salarioBase < 0.0) // valida el salarioBase
         throw new IllegalArgumentException("El salario base debe ser >= 0.0");

      this.salarioBase = salarioBase;
   }

   // establece el salario base
   public void establecerSalarioBase(double salarioBase)
   {
      if (salarioBase < 0.0) // valida el salarioBase
         throw new IllegalArgumentException("El salario base debe ser >= 0.0");

      this.salarioBase = salarioBase;
   }

   // devuelve el salario base
   public double obtenerSalarioBase()
   {
      return salarioBase;
   }

   // calcula los ingresos; sobrescribe el método ingresos en EmpleadoPorComision
   @Override
   public double ingresos()
   {
      return obtenerSalarioBase() + super.ingresos();
   }

   // devuelve representación String de un objeto EmpleadoBaseMasComision
   @Override
   public String toString()
   {
      return String.format("%s %s; %s: $%,.2f",
         "con salario base", super.toString(),
         "salario base", obtenerSalarioBase());
   }
} // fin de la clase EmpleadoBaseMasComision
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

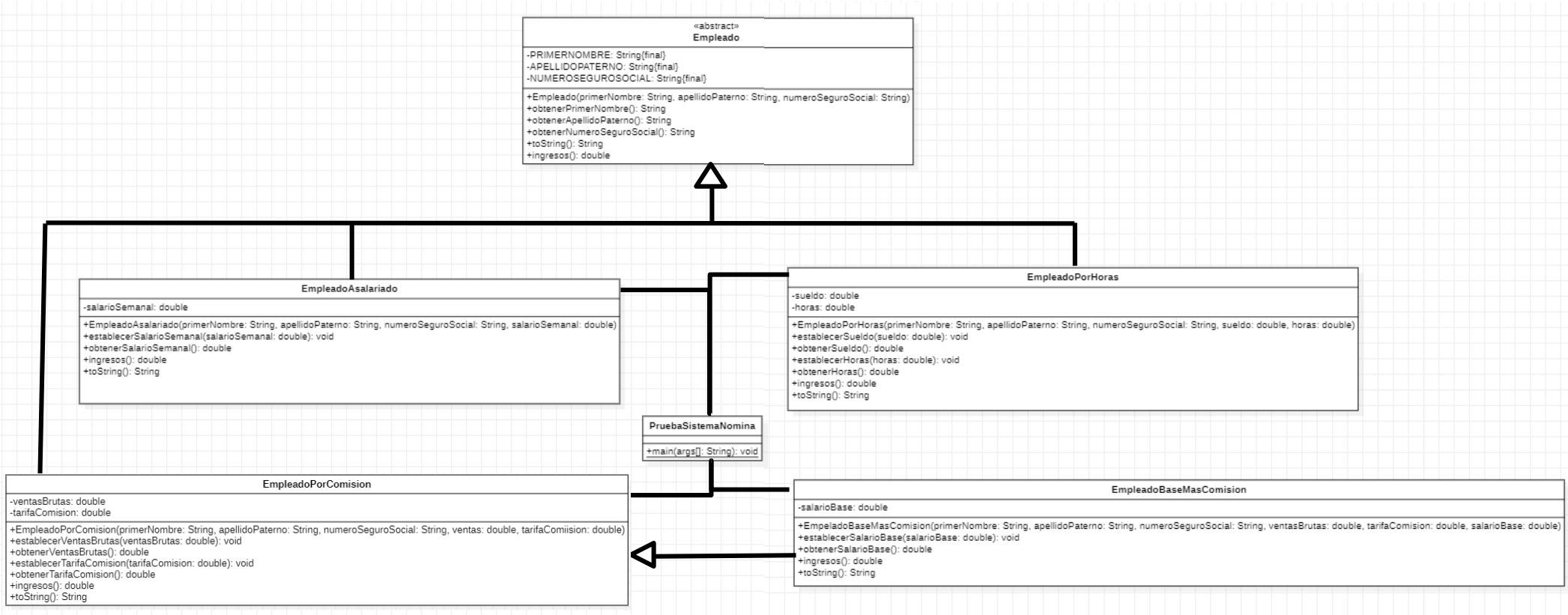
```
1  // Fig. 10.9: PruebaSistemaNomina.java
2  // Programa de prueba para la jerarquia de Empleado.
3
4  public class PruebaSistemaNomina
5  {
6     public static void main(String[] args)
7     {
8        // crea objetos de las subclases
9        EmpleadoAsalariado empleadoAsalariado =
10          new EmpleadoAsalariado("John", "Smith", "111-11-1111", 800.00);
11       EmpleadoPorHoras empleadoPorHoras =
12          new EmpleadoPorHoras("Karen", "Price", "222-22-2222", 16.75, 40);
13       EmpleadoPorComision empleadoPorComision =
14          new EmpleadoPorComision(
15          "Sue", "Jones", "333-33-3333", 10000, .06);
16       EmpleadoBaseMasComision empleadoBaseMasComision =
17          new EmpleadoBaseMasComision(
18          "Bob", "Lewis", "444-44-4444", 5000, .04, 300);
19
20       System.out.println("Empleados procesados por separado:");
21
22       System.out.printf("%n%s%n%s: $%,.2f%n%n",
23          empleadoAsalariado, "ingresos", empleadoAsalariado.ingresos());
24       System.out.printf("%s%n%s: $%,.2f%n%n",
25          empleadoPorHoras, "ingresos", empleadoPorHoras.ingresos());
26       System.out.printf("%s%n%s: $%,.2f%n%n",
27          empleadoPorComision, "ingresos", empleadoPorComision.ingresos());
28       System.out.printf("%s%n%s: $%,.2f%n%n",
29          empleadoBaseMasComision,
30          "ingresos", empleadoBaseMasComision.ingresos());
31
32       // crea un arreglo Empleado de cuatro elementos
33       Empleado[] empleados = new Empleado[4];
34
35       // inicializa el arreglo con objetos Empleado
36       empleados[0] = empleadoAsalariado;
37       empleados[1] = empleadoPorHoras;
38       empleados[2] = empleadoPorComision;
39       empleados[3] = empleadoBaseMasComision;
40
41       System.out.println("Empleados procesados en forma polimurfica:%n%n");
42
43       // procesa en forma genérica a cada elemento en el arreglo de empleados
44       for (Empleado empleadoActual : empleados)
45       {
46          System.out.println(empleadoActual); // invoca a toString
47
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```
46            System.out.println(empleadoActual); // invoca a toString
47
48            // determina si el elemento es un EmpleadoBaseMasComision
49            if (empleadoActual instanceof EmpleadoBaseMasComision)
50            {
51               // conversión descendente de la referencia de Empleado
52               // a una referencia de EmpleadoBaseMasComision
53               EmpleadoBaseMasComision empleado =
54                  (EmpleadoBaseMasComision) empleadoActual;
55
56               empleado.establecerSalarioBase(1.10 * empleado.obtenerSalarioBase());
57
```

```
58               System.out.printf(
59                  "el nuevo salario base con 10%% de aumento es: $%,.2f%n",
60                  empleado.obtenerSalarioBase());
61            } // fin de if
62
63            System.out.printf(
64               "ingresos $%,.2f%n%n", empleadoActual.ingresos());
65         } // fin de for
66
67         // obtiene el nombre del tipo de cada objeto en el arreglo de empleados
68         for (int j = 0; j < empleados.length; j++)
69            System.out.printf("El empleado %d es un %s%n", j,
70               empleados[j].getClass().getName());
71      } // fin de main
72   } // fin de la clase PruebaSistemaNomina
```

A. DIAGRAMA 4


«abstract»
Empleado
-PRIMERNOMBRE: String{final}
-APELLIDOPATERNO: String{final}
-NUMEROSEGUROSOCIAL: String{final}
+Empleado(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String)
+obtenerPrimerNombre(): String
+obtenerApellidoPaterno(): String
+obtenerNumeroSeguroSocial(): String
+toString(): String
+ingresos(): double
EmpleadoAsalariado
-salarioSemanal: double
+EmpleadoAsalariado(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String, salarioSemanal: double)
+establecerSalarioSemanal(salarioSemanal: double): void
+obtenerSalarioSemanal(): double
+ingresos(): double
+toString(): String
EmpleadoPorHoras
-sueldo: double
-horas: double
+EmpleadoPorHoras(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String, sueldo: double, horas: double)
+establecerSueldo(sueldo: double): void
+obtenerSueldo(): double
+establecerHoras(horas: double): void
+obtenerHoras(): double
+ingresos(): double
+toString(): String
PruebaSistemaNomina
+main(args[]: String): void
EmpleadoPorComision
-ventasBrutas: double
-tarifaComision: double
+EmpleadoPorComision(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String, ventas: double, tarifaComiision: double)
+establecerVentasBrutas(ventasBrutas: double): void
+obtenerVentasBrutas(): double
+establecerTarifaComision(tarifaComision: double): void
+obtenerTarifaComision(): double
+ingresos(): double
+toString(): String
EmpleadoBaseMasComision
-salarioBase: double
+EmpeladoBaseMasComision(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String, ventasBrutas: double, tarifaComision: double, salarioBase: double)
+establecerSalarioBase(salarioBase: double): void
+obtenerSalarioBase(): double
+ingresos(): double
+toString(): String

B. CÓDIGO 5

```
1  // Fig. 10.12: Factura.java
2  // La clase Factura implementa a PorPagar.
3
4  public class Factura implements PorPagar
5  {
6     private final String numeroPieza;
7     private final String descripcionPieza;
8     private int cantidad;
9     private double precioPorArticulo;
10
11    // constructor
12    public Factura(String numeroPieza, String descripcionPieza, int cantidad,
13       double precioPorArticulo)
14    {
15       if (cantidad < 0) // valida la cantidad
16          throw new IllegalArgumentException ("Cantidad debe ser >= 0");
17
18       if (precioPorArticulo < 0.0) // valida el precioPorArticulo
19          throw new IllegalArgumentException(
20             "El precio por articulo debe ser >= 0");
21
22       this.cantidad = cantidad;
23       this.numeroPieza = numeroPieza;
24       this.descripcionPieza = descripcionPieza;
25       this.precioPorArticulo = precioPorArticulo;
26    } // fin del constructor
27
```

```
80      // método requerido para realizar el contrato con la interfaz PorPagar
81      @Override
82      public double obtenerMontoPago()
83      {
84         return obtenerCantidad() * obtenerPrecioPorArticulo(); // calcula el costo
                                                                   total
85      }
86   } // fin de la clase Factura
```

```
1   // Fig. 10.13: Empleado.java
2   // La superclase abstracta Empleado que implementa a PorPagar.
3
4   public abstract class Empleado implements PorPagar
5   {
6      private final String primerNombre;
7      private final String apellidoPaterno;
8      private final String numeroSeguroSocial;
9
10     // constructor
11     public Empleado(String primerNombre, String apellidoPaterno,
12        String numeroSeguroSocial)
13     {
14        this.primerNombre = primerNombre;
15        this.apellidoPaterno = apellidoPaterno;
16        this.numeroSeguroSocial = numeroSeguroSocial;
17     }
18
19     // devuelve el primer nombre
20     public String obtenerPrimerNombre()
21     {
22        return primerNombre;
23     }
24
25     // devuelve el apellido paterno
26     public String obtenerApellidoPaterno()
27     {
28        return apellidoPaterno;
29     }
30
31     // devuelve el número de seguro social
32     public String obtenerNumeroSeguroSocial()
33     {
34        return numeroSeguroSocial;
35     }
36
37     // devuelve representación String de un objeto Empleado
38     @Override
39     public String toString()
40     {
41        return String.format("%s %s%nnumero de seguro social: %s",
42           obtenerPrimerNombre(), obtenerApellidoPaterno(),
           obtenerNumeroSeguroSocial());
43     }
44
45     // Nota: Aqui no implementamos el método obtenerMontoPago de PorPagar, así que
46     // esta clase debe declararse como abstract para evitar un error de compilación.
47  } // fin de la clase abstracta Empleado
```

```
1  // Fig. 10.14: EmpleadoAsalariado.java
2  // La clase EmpleadoAsalariado que implementa la interfaz PorPagar.
3  // método obtenerMontoPago
4  public class EmpleadoAsalariado extends Empleado
5  {
6     private double salarioSemanal;
7
8     // constructor
9     public EmpleadoAsalariado(String primerNombre, String apellidoPaterno,
10       String numeroSeguroSocial,  double salarioSemanal)
11    {
12       super(primerNombre, apellidoPaterno, numeroSeguroSocial);
13
14       if (salarioSemanal < 0.0)
15          throw new IllegalArgumentException(
16             "El salario semanal debe ser >= 0.0");
17
18       this.salarioSemanal = salarioSemanal;
19    }
20
21    // establece el salario
22    public void establecerSalarioSemanal(double salarioSemanal)
23    {
24       if (salariosemanal < 0.0)
25          throw new IllegalArgumentException(
26             "El salario semanal debe ser >= 0.0");
```

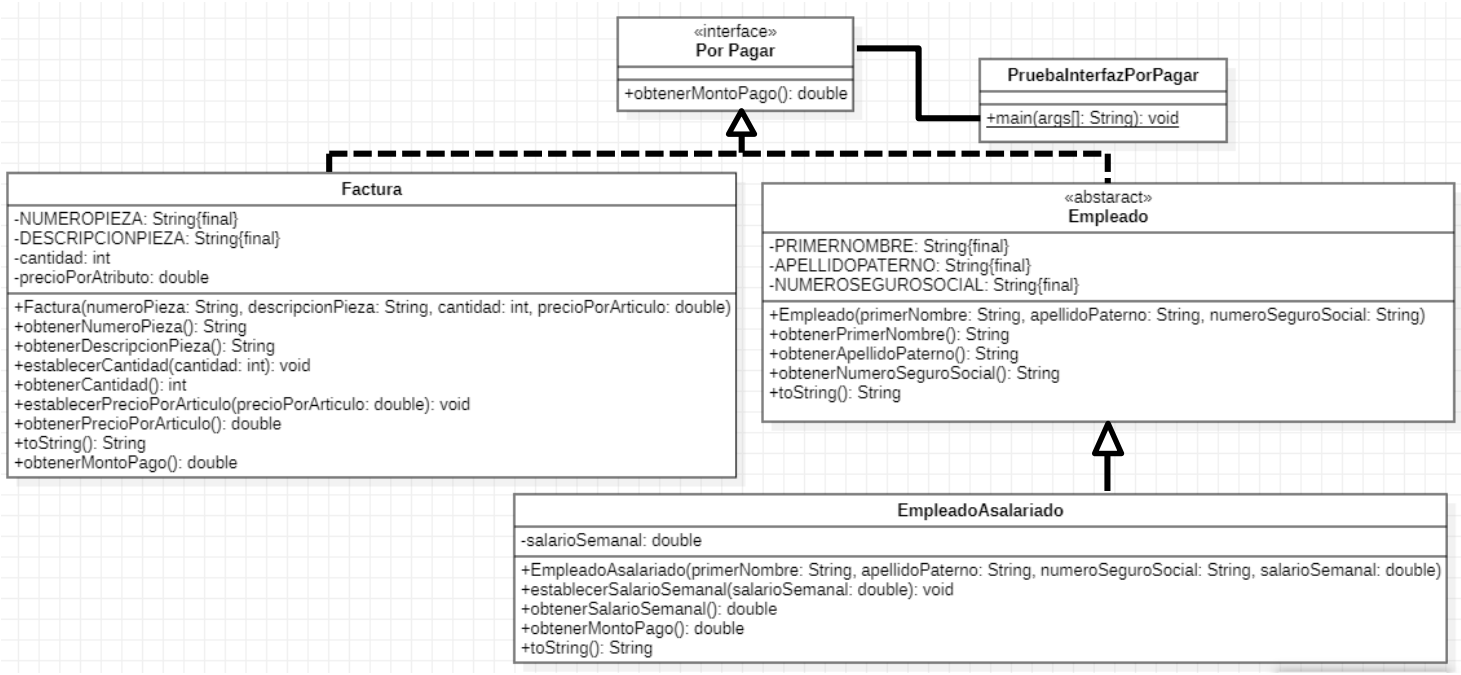
```
27
28       this.salarioSemanal = salarioSemanal;
29    }
30
31    // devuelve el salario
32    public double obtenerSalarioSemanal()
33    {
34       return salarioSemanal;
35    } // fin del método obtenerSalarioSemanal
36
37    // calcula los ingresos; implementa el método de la interfaz PorPagar
38    // que era abstracto en la superclase Empleado
39    @Override
40    public double obtenerMontoPago()
41    {
42       return obtenerSalarioSemanal();
43    }
44
45    // devuelve representación String de un objeto EmpleadoAsalariado
46    @Override
47    public String toString()
48    {
49       return String.format("empleado asalariado: %s%n%s: $%,.2f",
50          super.toString(), "salario semanal", obtenerSalarioSemanal());
51    }
52 } // fin de la clase EmpleadoAsalariado
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

```
1   // Fig. 10.15: PruebaInterfazPorPagar.java
2   // Programa de prueba de la interfaz PorPagar que procesa objetos
3   // Factura y Empleado mediante el polimorfismo.
4   public class PruebaInterfazPorPagar
5   {
6      public static void main(String[] args)
7      {
8         // crea arreglo PorPagar con cuatro elementos
9         PorPagar[] objetosPorPagar = new PorPagar[4];
10
11        // llena el arreglo con objetos que implementan la interfaz PorPagar
12        objetosPorPagar[0] = new Factura("01234", "asiento", 2, 375.00);
13        objetosPorPagar[1] = new Factura("56789", "llanta", 4, 79.95);
14        objetosPorPagar[2] =
15           new EmpleadoAsalariado("John", "Smith", "111-11-1111", 800.00);
16        objetosPorPagar[3] =
17           new EmpleadoAsalariado("Lisa", "Barnes", "888-88-8888", 1200.00);
18
19        System.out.println(
20           "Facturas y Empleados procesados en forma polimorfica:");
21
22        // procesa en forma genérica cada elemento en el arreglo objetosPorPagar
23        for (PorPagar porPagarActual : objetosPorPagar)
24        {
```

```
25           // imprime porPagarActual y su monto de pago apropiado
26           System.out.printf("%n%s %n%s: $%,.2f%n",
27              porPagarActual.toString(), // se podria invocar de manera implicita
28              "pago vencido", porPagarActual.obtenerMontoPago());
29        }
30     } // fin de main
31  } // fin de la clase PruebaInterfazPorPagar
```

C. DIAGRAMA 5


«interface»
Por Pagar
+obtenerMontoPago(): double
PruebaInterfazPorPagar
+main(args[]: String): void
Factura
-NUMEROPIEZA: String{final}
-DESCRIPCIONPIEZA: String{final}
-cantidad: int
-precioPorAtributo: double
+Factura(numeroPieza: String, descripcionPieza: String, cantidad: int, precioPorArticulo: double)
+obtenerNumeroPieza(): String
+obtenerDescripcionPieza(): String
+establecerCantidad(cantidad: int): void
+obtenerCantidad(): int
+establecerPrecioPorArticulo(precioPorArticulo: double): void
+obtenerPrecioPorArticulo(): double
+toString(): String
+obtenerMontoPago(): double
«abstaract»
Empleado
-PRIMERNOMBRE: String{final}
-APELLIDOPATERNO: String{final}
-NUMEROSEGUROSOCIAL: String{final}
+Empleado(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String)
+obtenerPrimerNombre(): String
+obtenerApellidoPaterno(): String
+obtenerNumeroSeguroSocial(): String
+toString(): String
EmpleadoAsalariado
-salarioSemanal: double
+EmpleadoAsalariado(primerNombre: String, apellidoPaterno: String, numeroSeguroSocial: String, salarioSemanal: double)
+establecerSalarioSemanal(salarioSemanal: double): void
+obtenerSalarioSemanal(): double
+obtenerMontoPago(): double
+toString(): String

D. CÓDIGO 6

```
1  // Fig. 11.2: DivisionEntreCeroSinManejoDeExcepciones.java
2  // División de enteros sin manejo de excepciones.
3  import java.util.Scanner;
4
5  public class DivisionEntreCeroSinManejoDeExcepciones
6  {
7     // demuestra el lanzamiento de una excepción cuando ocurre una división entre
       cero
8     public static int cociente(int numerador, int denominador)
9     {
10       return numerador / denominador; // posible división entre cero
11    }
12
13    public static void main(String[] args)
14    {
15       Scanner explorador = new Scanner(System.in);
16
17       System.out.print("Introduzca un numerador entero: ");
18       int numerador = explorador.nextInt();
19       System.out.print("Introduzca un denominador entero: ");
20       int denominador = explorador.nextInt();
21
22       int resultado = cociente(numerador, denominador);
23       System.out.printf(
24          "%nResultado: %d / %d = %d%n", numerador, denominador, resultado);
25    }
26 } // fin de la clase DivisionEntreCeroSinManejoDeExcepciones
```


*Cualquier documento no identificado como **Controlado** se considera **COPIA NO CONTROLADA** y no es auditable.*

E. DIAGRAMA 6


DivisonEntreCerosSinManejoDeExepciones
+cociente(numerador: int, denominador: int): int
+main(args[]: String): void
Scanner

V. Conclusiones

Los diagramas de clases son diagramas de estructura y se utiliza para representar los elementos que componen un sistema de información desde un punto de vista estático. Es un diagrama del modelo de programación orientado a objetos, ya que define las clases que se utilizarán cuando se pase a la fase de construcción y la manera en que se relacionan las mismas.

Una clase está compuesta por tres elementos: nombre de la clase, atributos y métodos. Para representar la clase con estos elementos se utiliza un rectángulo que es dividido en tres zonas: La primera de las zonas se utiliza para el nombre de la clase, la segunda de las zonas se utiliza para escribir los atributos de la clase y la tercera para los métodos. Tanto los atributos como los métodos presentaran al principio su modificador de acceso, el cual puede ser publico (+), privado (-), protegido (#) y amigable, que es cuando no se especifica ninguno de los anteriores.

La importancia de los diagramas de clase, es poder tener una major visualización de las clases y de esta manera poder entender el cómo se comunican los objetos, atributos y los métodos de las clases entre sí,y comprender la interacción del sistema a partir de las relaciones existentes entre las diversas clases para que de esta manera sea mas sencillo entender el funcionamiento del código.