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

NOMBRE DEL ALUMNO: Jesus Navarrete 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:

- Equipo de cómputo
- StarUML: Programa para generar los diagramas.
- Libreta
- Libro de como programar en java de Paul Dietel, décima edición

IV. Desarrollo de la práctica:

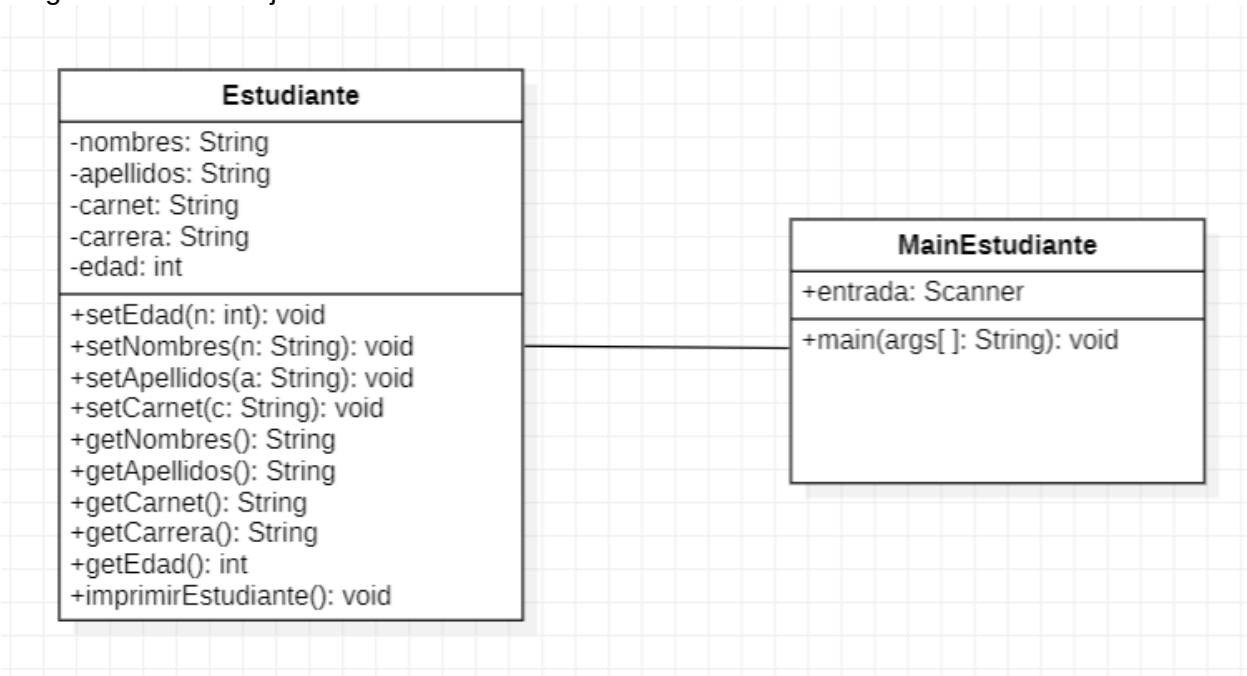
Unidad 1. Introducción al paradigma de la POO

a) Ejercicio 1

Código en java:

```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();
    }
}
```

Diagrama UML del ejercicio 1:



b) Ejercicio 2

Código en java

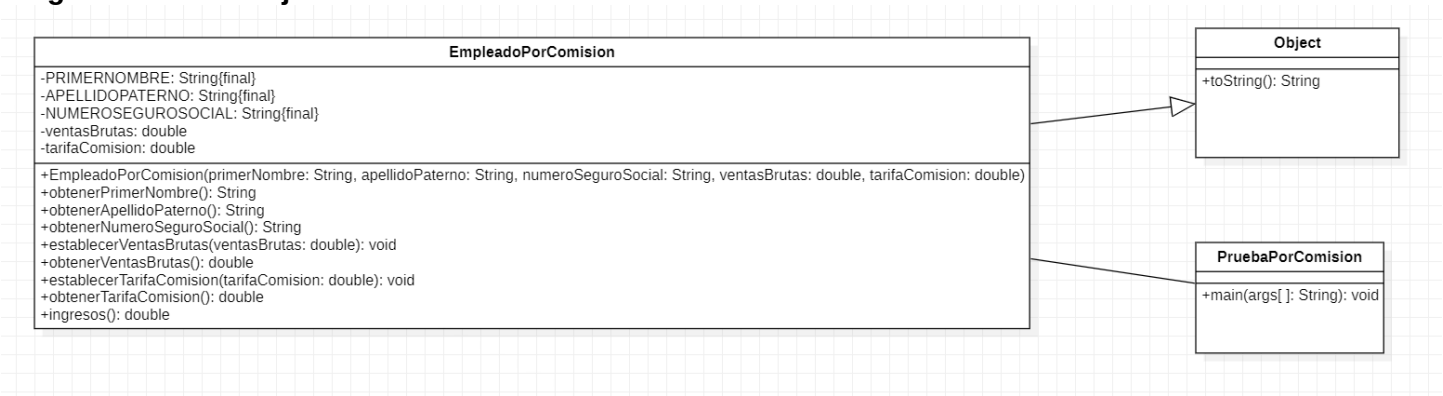
```java
// Fig. 9.4: EmpleadoPorComision.java
// La clase EmpleadoPorComision representa a un empleado que
// recibe como sueldo un porcentaje de las ventas brutas.
public class EmpleadoPorComision extends Object
{
   private final String primerNombre;
   private final String apellidoPaterno;
   private final String numeroSeguroSocial;
   private double ventasBrutas; // ventas totales por semana
   private double tarifaComision; // porcentaje de comisión

   // constructor con cinco argumentos
   public EmpleadoPorComision(String primerNombre, String apellidoPaterno,
      String numeroSeguroSocial, double ventasBrutas,
      double tarifaComision)
   {
      // la llamada implícita al constructor predeterminado de Object ocurre aquí

      // si ventasBrutas no es válida, lanza excepción
      if (ventasBrutas < 0.0)
         throw new IllegalArgumentException(
            "Las ventas brutas deben ser >= 0.0");

      // si tarifaComision no es válida, lanza excepción
      if (tarifaComision <= 0.0 || tarifaComision >= 1.0)
         throw new IllegalArgumentException(
            "La tarifa de comision debe ser > 0.0 y < 1.0");

      this.primerNombre = primerNombre;
      this.apellidoPaterno = apellidoPaterno;
      this.numeroSeguroSocial = numeroSeguroSocial;
      this.ventasBrutas = ventasBrutas;
      this.tarifaComision = tarifaComision;
   } // fin del constructor

   // devuelve el primer nombre
   public String obtenerPrimerNombre()
   {
      return primerNombre;
   }

   // devuelve el apellido paterno
   public String obtenerApellidoPaterno()
   {
      return apellidoPaterno;
   }
```

```
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
```

```
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",
31          empleado);
32    } // fin de main
33 } // fin de la clase PruebaEmpleadoPorComision
```

```
Informacion del empleado obtenida por los metodos establecer:

El primer nombre es Sue
El apellido paterno es Jones
El numero de seguro social es 222-22-2222
Las ventas brutas son 10000.00
La tarifa de comision es 0.06

Informacion actualizada del empleado, obtenida mediante toString:

empleado por comision: Sue Jones
numero de seguro social: 222-22-2222
ventas brutas: 500.00
tarifa de comision: 0.10
```

Diagrama UML del ejercicio 2:



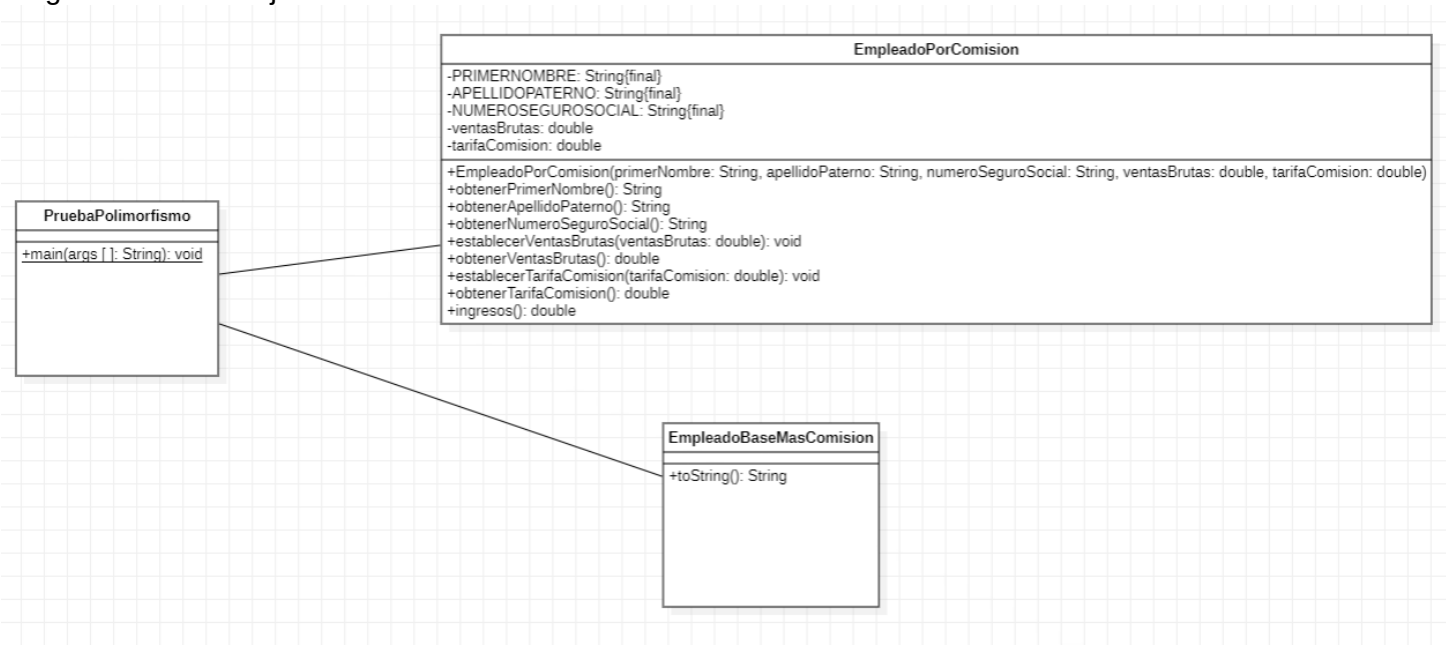
c)Ejercicio 3:

Código en java

```
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",
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",
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",
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
```

```
Llamada a toString de EmpleadoPorComision con referencia de superclase a un objeto
de la superclase:

empleado por comision: Sue Jones
numero de seguro social: 222-22-2222
ventas brutas: 10000.00
tarifa de comision: 0.06

Llamada a toString de EmpleadoBaseMasComision con referencia de subclase a un objeto
de la subclase:

con sueldo base empleado por comision: Bob Lewis
numero de seguro social: 333-33-3333
ventas brutas: 5000.00
tarifa de comision: 0.04
sueldo base: 300.00
```

Diagrama UML del ejercicio 3



d)Ejercicio 4

código en java

```
// Fig. 10.4: Empleado.java
// La superclase abstracta Empleado.

public abstract class Empleado
{
   private final String primerNombre;
   private final String apellidoPaterno;
   private final String numeroSeguroSocial;

   // constructor
   public Empleado(String primerNombre, String apellidoPaterno,
      String numeroSeguroSocial)
   {
      this.primerNombre = primerNombre;
      this.apellidoPaterno = apellidoPaterno;
      this.numeroSeguroSocial = numeroSeguroSocial;
   }

   // devuelve el primer nombre
   public String obtenerPrimerNombre()
   {
      return primerNombre;
   }

   // devuelve el apellido paterno
   public String obtenerApellidoPaterno()
   {
      return apellidoPaterno;
   }

   // devuelve el número de seguro social
   public String obtenerNumeroSeguroSocial()
   {
```

```
      return numeroSeguroSocial;
   }

   // devuelve representación String de un objeto Empleado
   @Override
   public String toString()
   {
      return String.format("%s %s%nnumero de seguro social: %s",
         obtenerPrimerNombre(), obtenerApellidoPaterno(),
         obtenerNumeroSeguroSocial());
   }

   // método abstracto sobrescrito por las subclases concretas
   public abstract double ingresos(); // aquí no hay implementación
} // fin de la clase abstracta Empleado
```

```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
```

```java
// Fig. 10.6: EmpleadoPorHoras.java
// La clase EmpleadoPorHoras extiende a Empleado.

public class EmpleadoPorHoras extends Empleado
{
   private double sueldo; // sueldo por hora
   private double horas; // horas trabajadas por semana

   // constructor
   public EmpleadoPorHoras(String primerNombre, String apellidoPaterno,
      String numeroSeguroSocial, double sueldo, double horas)
   {
      super(primerNombre, apellidoPaterno, numeroSeguroSocial);

```

```
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
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
```

```
1  // Fig. 10.7: EmpleadoPorComision.java
2  // La clase EmpleadoPorComision extiende a Empleado.
3
4  public class EmpleadoPorComision extends Empleado
5  {
6     private double ventasBrutas; // ventas totales por semana
7     private double tarifaComision; // porcentaje de comisión
8
9     // constructor
10    public EmpleadoPorComision(String primerNombre, String apellidoPaterno,
11       String numeroSeguroSocial, double ventas,
12       double tarifaComision)
13    {
14       super(primerNombre, apellidoPaterno, numeroSeguroSocial);
15
16       if (tarifaComision <= 0.0 || tarifaComision >= 1.0) // valida
17          throw new IllegalArgumentException(
18             "La tarifa de comision debe ser > 0.0 y < 1.0");
19
20       if (ventasBrutas < 0.0)
21          throw new IllegalArgumentException("Las ventas brutas deben ser >= 0.0");
22
```

```
23        this.ventasBrutas = ventasBrutas;
24        this.tarifaComision = tarifaComision;
25     }
26
27     // establece el monto de ventas brutas
28     public void establecerVentasBrutas(double ventasBrutas)
29     {
30        if (ventasBrutas < 0.0)
31           throw new IllegalArgumentException("Las ventas brutas deben ser >= 0.0");
32
33        this.ventasBrutas = ventasBrutas;
34     }
35
36     // devuelve el monto de ventas brutas
37     public double obtenerVentasBrutas()
38     {
39        return ventasBrutas;
40     }
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
```

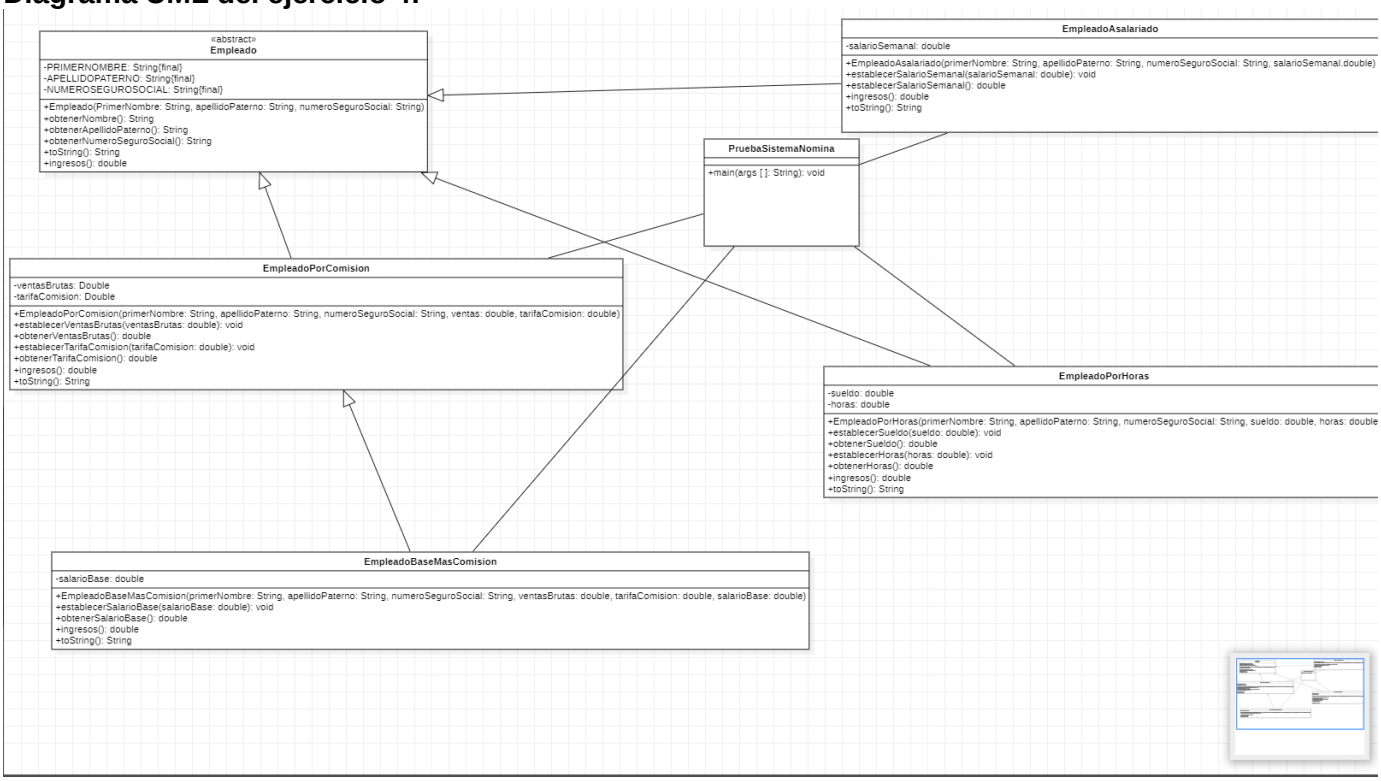
```
1  // Fig. 10.8: EmpleadoBaseMasComision.java
2  // La clase EmpleadoBaseMasComision extiende a EmpleadoPorComision.
3
4  public class EmpleadoBaseMasComision extends EmpleadoPorComision
5  {
6     private double salarioBase; // salario base por semana
7
8     // constructor
9     public EmpleadoBaseMasComision(String primerNombre, String apellidoPaterno,
10       String numeroSeguroSocial, double ventasBrutas,
11       double tarifaComision, double salarioBase)
12    {
13      super(primerNombre, apellidoPaterno, numeroSeguroSocial,
14        ventasBrutas, tarifaComision);
15
16       if (salarioBase < 0.0) // valida el salarioBase
17          throw new IllegalArgumentException("El salario base debe ser >= 0.0");
18
19       this.salarioBase = salarioBase;
20    }
21
22    // establece el salario base
23    public void establecerSalarioBase(double salarioBase)
24    {
25       if (salarioBase < 0.0) // valida el salarioBase
26          throw new IllegalArgumentException("El salario base debe ser >= 0.0");
```

```
27
28       this.salarioBase = salarioBase;
29    }
30
31    // devuelve el salario base
32    public double obtenerSalarioBase()
33    {
34       return salarioBase;
35    }
36
37    // calcula los ingresos; sobrescribe el método ingresos en EmpleadoPorComision
38    @Override
39    public double ingresos()
40    {
41       return obtenerSalarioBase() + super.ingresos();
42    }
43
44    // devuelve representación String de un objeto EmpleadoBaseMasComision
45    @Override
46    public String toString()
47    {
48       return String.format("%s %s; %s: $%,.2f",
49          "con salario base", super.toString(),
50          "salario base", obtenerSalarioBase());
51    }
52 } // fin de la clase EmpleadoBaseMasComision
```

```
// Fig. 10.9: PruebaSistemaNomina.java
// Programa de prueba para la jerarquía de Empleado.

public class PruebaSistemaNomina
{
   public static void main(String[] args)
   {
      // crea objetos de las subclases
      EmpleadoAsalariado empleadoAsalariado =
         new EmpleadoAsalariado("John", "Smith", "111-11-1111", 800.00);
      EmpleadoPorHoras empleadoPorHoras =
         new EmpleadoPorHoras("Karen", "Price", "222-22-2222", 16.75, 40);
      EmpleadoPorComision empleadoPorComision =
         new EmpleadoPorComision(
         "Sue", "Jones", "333-33-3333", 10000, .06);
      EmpleadoBaseMasComision empleadoBaseMasComision =
         new EmpleadoBaseMasComision(
         "Bob", "Lewis", "444-44-4444", 5000, .04, 300);

      System.out.println("Empleados procesados por separado:");

      System.out.printf("%n%s%n%s: $%,.2f%n%n",
         empleadoAsalariado, "ingresos", empleadoAsalariado.ingresos());
      System.out.printf("%s%n%s: $%,.2f%n%n",
         empleadoPorHoras, "ingresos", empleadoPorHoras.ingresos());
      System.out.printf("%s%n%s: $%,.2f%n%n",
         empleadoPorComision, "ingresos", empleadoPorComision.ingresos());
      System.out.printf("%s%n%s: $%,.2f%n%n",
         empleadoBaseMasComision,
         "ingresos", empleadoBaseMasComision.ingresos());

      // crea un arreglo Empleado de cuatro elementos
      Empleado[] empleados = new Empleado[4];

      // inicializa el arreglo con objetos Empleado
      empleados[0] = empleadoAsalariado;
      empleados[1] = empleadoPorHoras;
      empleados[2] = empleadoPorComision;
      empleados[3] = empleadoBaseMasComision;

      System.out.println("Empleados procesados en forma polimorfica:%n%n");

      // procesa en forma genérica a cada elemento en el arreglo de empleados
      for (Empleado empleadoActual : empleados)
      {
         System.out.println(empleadoActual); // invoca a toString

         // determina si el elemento es un EmpleadoBaseMasComision
         if (empleadoActual instanceof EmpleadoBaseMasComision)
         {
            // conversión descendente de la referencia de Empleado
            // a una referencia de EmpleadoBaseMasComision
            EmpleadoBaseMasComision empleado =
               (EmpleadoBaseMasComision) empleadoActual;

            empleado.establecerSalarioBase(1.10 * empleado.obtenerSalarioBase());

            System.out.printf(
               "el nuevo salario base con 10%% de aumento es: $%,.2f%n",
               empleado.obtenerSalarioBase());
         } // fin de if

         System.out.printf(
            "ingresos $%,.2f%n%n", empleadoActual.ingresos());
      } // fin de for

      // obtiene el nombre del tipo de cada objeto en el arreglo de empleados
      for (int j = 0; j < empleados.length; j++)
         System.out.printf("El empleado %d es un %s%n", j,
            empleados[j].getClass().getName());
   } // fin de main
} // fin de la clase PruebaSistemaNomina
```

```
Empleados procesados por separado:

empleado asalariado: John Smith
numero de seguro social: 111-11-1111
salario semanal: $800.00
ingresos: $800.00

empleado por horas: Karen Price
numero de seguro social: 222-22-2222
sueldo por hora: $16.75; horas trabajadas: 40.00
ingresos: $670.00

empleado por comision: Sue Jones
numero de seguro social: 333-33-3333
ventas brutas: $10,000.00; tarifa de comision: 0.06
ingresos: $600.00

con salario base empleado por comision: Bob Lewis
numero de seguro social: 444-44-4444
ventas brutas: $5,000.00; tarifa de comision: 0.04; salario base: $300.00
ingresos: $500.00

Empleados procesados en forma polimorfica:

empleado asalariado: John Smith
numero de seguro social: 111-11-1111
salario semanal: $800.00
ingresos $800.00

empleado por horas: Karen Price
numero de seguro social: 222-22-2222
sueldo por hora: $16.75; horas trabajadas: 40.00
ingresos $670.00

empleado por comision: Sue Jones
numero de seguro social: 333-33-3333
ventas brutas: $10,000.00; tarifa de comision: 0.06
ingresos $600.00
```

Diagrama UML del ejercicio 4:



E) Ejercicio 5

Código en java

```java
// Fig. 10.11: PorPagar.java
// Declaración de la interfaz PorPagar.

public interface PorPagar
{
   double obtenerMontoPago(); // calcula el pago; no hay implementación
} // fin de la interfaz PorPagar
```

```java
// Fig. 10.12: Factura.java
// La clase Factura implementa a PorPagar.

public class Factura implements PorPagar
{
   private final String numeroPieza;
   private final String descripcionPieza;
   private int cantidad;
   private double precioPorArticulo;

   // constructor
   public Factura(String numeroPieza, String descripcionPieza, int cantidad,
      double precioPorArticulo)
   {
      if (cantidad < 0) // valida la cantidad
         throw new IllegalArgumentException ("Cantidad debe ser >= 0");

      if (precioPorArticulo < 0.0) // valida el precioPorArticulo
         throw new IllegalArgumentException(
            "El precio por articulo debe ser >= 0");

      this.cantidad = cantidad;
      this.numeroPieza = numeroPieza;
      this.descripcionPieza = descripcionPieza;
      this.precioPorArticulo = precioPorArticulo;
   } // fin del constructor

```

```
28     // obtiene el número de pieza
29     public String obtenerNumeroPieza()
30     {
31        return numeroPieza; // debe validar
32     }
33
34     // obtiene la descripción
35     public String obtenerDescripcionPieza()
36     {
37        return descripcionPieza;
38     }
39
40     // establece la cantidad
41     public void establecerCantidad(int cantidad)
42     {
43        if (cantidad < 0) // valida la cantidad
44           throw new IllegalArgumentException ("Cantidad debe ser >= 0");
45
46        this.cantidad = cantidad;
47     }
48
49     // obtener cantidad
50     public int obtenerCantidad()
51     {
52        return cantidad;
53     }
54
55     // establece el precio por articulo
56     public void establecerPrecioPorArticulo(double precioPorArticulo)
57     {
58        if (precioPorArticulo < 0.0) // valida el precioPorArticulo
59           throw new IllegalArgumentException(
60              "El precio por articulo debe ser >= 0");
61
62        this.precioPorArtculo = precioPorArticulo;
63     }
64
65     // obtiene el precio por articulo
66     public double obtenerPrecioPorArticulo()
67     {
68        return precioPorArticulo;
69     }
70
71     // devuelve representación String de un objeto Factura
72     @Override
73     public String toString()
74     {
75        return String.format("%s: %n%s: %s (%s) %n%s: %d %n%s: $%,.2f",
76           "factura", "numero de pieza", obtenerNumeroPieza(),
                                          obtenerDescripcionPieza(),
77           "cantidad", obtenerCantidad(), "precio por articulo", obtenerPrecioPor-
                                                                  Articulo());
78     }
79
```

```
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
```

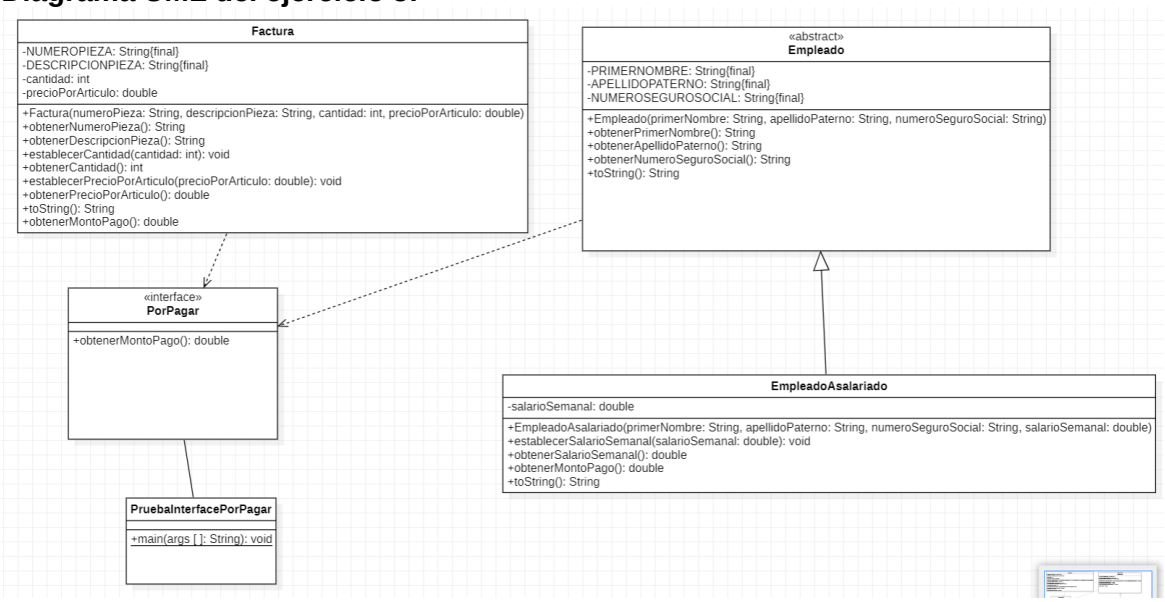
```
4  public abstract class Empleado implements PorPagar
5  {
6     private final String primerNombre;
7     private final String apellidoPaterno;
8     private final String numeroSeguroSocial;
9
10    // constructor
11    public Empleado(String primerNombre, String apellidoPaterno,
12       String 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: Aquí 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
```

```java
// Fig. 10.14: EmpleadoAsalariado.java
// La clase EmpleadoAsalariado que implementa la interfaz PorPagar.
// método obtenerMontoPago
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;
   } // fin del método obtenerSalarioSemanal

   // calcula los ingresos; implementa el método de la interfaz PorPagar
   // que era abstracto en la superclase Empleado
   @Override
   public double obtenerMontoPago()
   {
      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
```

```java
// Fig. 10.15: PruebaInterfazPorPagar.java
// Programa de prueba de la interfaz PorPagar que procesa objetos
// Factura y Empleado mediante el polimorfismo.
public class PruebaInterfazPorPagar
{
   public static void main(String[] args)
   {
      // crea arreglo PorPagar con cuatro elementos
      PorPagar[] objetosPorPagar = new PorPagar[4];

      // llena el arreglo con objetos que implementan la interfaz PorPagar
      objetosPorPagar[0] = new Factura("01234", "asiento", 2, 375.00);
      objetosPorPagar[1] = new Factura("56789", "llanta", 4, 79.95);
      objetosPorPagar[2] =
         new EmpleadoAsalariado("John", "Smith", "111-11-1111", 800.00);
      objetosPorPagar[3] =
         new EmpleadoAsalariado("Lisa", "Barnes", "888-88-8888", 1200.00);

      System.out.println(
         "Facturas y Empleados procesados en forma polimorfica:");

      // procesa en forma genérica cada elemento en el arreglo objetosPorPagar
      for (PorPagar porPagarActual : objetosPorPagar)
      {
         // imprime porPagarActual y su monto de pago apropiado
         System.out.printf("%n%s %n%s: $%,.2f%n",
            porPagarActual.toString(), // se podría invocar de manera implícita
            "pago vencido", porPagarActual.obtenerMontoPago());
      }
   } // fin de main
} // fin de la clase PruebaInterfazPorPagar
```

```
Facturas y Empleados procesados en forma polimorfica:

factura:
numero de pieza: 01234 (asiento)
cantidad: 2
precio por articulo: $375.00
pago vencido: $750.00

factura:
numero de pieza: 56789 (llanta)
cantidad: 4
precio por articulo: $79.95
pago vencido: $319.80

empleado asalariado: John Smith
numero de seguro social: 111-11-1111
salario semanal: $800.00
pago vencido: $800.00

empleado asalariado: Lisa Barnes
numero de seguro social: 888-88-8888
salario semanal: $1,200.00
pago vencido: $1,200.00
```

Diagrama UML del ejercicio 5:



F) Ejercicio 6:

Código en java

```
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);
```

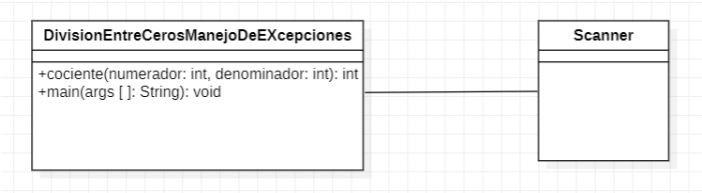
```
16
17       System.out.print("Introduzca un numerador entero: ");
18       int numerador = explorador.nextInt();
19       System.out.print("Introduzca un denominador entero: ");
20       int denominador = explorador.nextInt();
21
22       int resultado = cociente(numerador, denominador);
23       System.out.printf(
24          "%nResultado: %d / %d = %d%n", numerador, denominador, resultado);
25    }
26 } // fin de la clase DivisionEntreCeroSinManejoDeExcepciones
```

```
Introduzca un numerador entero: 100
Introduzca un denominador entero: 7

Resultado: 100 / 7 = 14
```

```
Introduzca un numerador entero: 100
Introduzca un denominador entero: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at  DivisionEntreCeroSinManejoDeExcepciones.cociente(
            DivisionEntreCeroSinManejoDeExcepciones.java:10)
        at  DivisionEntreCeroSinManejoDeExcepciones.main(
            DivisionEntreCeroSinManejoDeExcepciones.java:22)
```

```
Introduzca un numerador entero: 100
Introduzca un denominador entero: hola
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at DivisionEntreCeroSinManejoDeExcepciones.main(
            DivisionEntreCeroSinManejoDeExcepciones.java:20)
```

Diagrama UML del ejercicio 6:



V. Conclusiones:

En conclusión los diagramas UML son de gran ayuda para entender mejor la estructura de un sistema o proceso así como, poder identificar cada uno de los componentes que conforman su código y la funcionalidad de cada uno de ellos, estos diagramas de cierta forma nos permiten ir analizando y separando las distintas partes elementales del código, desde mi punto de vista esto es muy útil para llegar a identificar posibles errores o fallas en el código fuente o inclusive pueden ser de utilidad para comprobar la eficiencia y rendimiento del programa, claro todo esto puede identificarse si los diagramas se realizan adecuadamente siguiendo un determinado orden, estructura, funcionamiento, simbología e inclusive la incorrecta sintaxis de algún algoritmo podría cambiar el entendimiento que deseamos alcanzar, por ello la precisión al momento de realizar estos diagramas, estos diagramas los considero una herramienta fundamental para la representación y documentalizacion de las funcionalidades de un código fuente.