

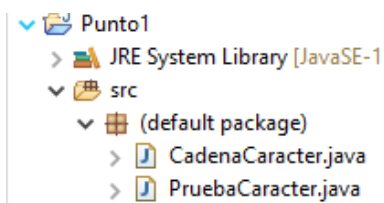
 UNIVERSIDAD CENTRAL		Facultad de Ingeniería y Ciencias Básicas Programa de Ingeniería Electrónica	POO – G04
			Portafolio
Cód: 1006825494	Est: Santiago Steven Reyes Naranjo		Tercer corte: 2022-2S

Retroalimentación – Segundo parcial

π Punto 1

Programa que encuentre en número de veces que se encuentra el caracter 'e' o 'E' en una cadena dada, haciendo uso de los elementos proporcionados por la POO.

Proyecto



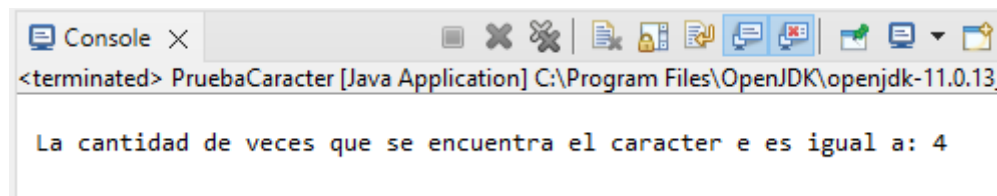
Código fuente

```

CadenaCaracter.java  PruebaCaracter.java
1 // Santiago Steven Reyes Naranjo
2
3 public class CadenaCaracter {
4
5     private String cadena;
6
7     public CadenaCaracter() {
8         this.cadena = "una frase diferente";
9     }
10
11     public void contarCaracter() {
12         char e = 'e'; int ne = 0; char l = ' ';
13         for (int i = 0; i < cadena.length(); i++) {
14             l = cadena.charAt(i);
15             if (l == e) {
16                 ne ++;
17             }
18         }
19         System.out.print("\n La cantidad de veces que se encuentra el caracter " + e + " es igual a: " + ne);
20     }
21 }
22
23
CadenaCaracter.java  PruebaCaracter.java
1 // Santiago Steven Reyes Naranjo
2
3
4 public class PruebaCaracter {
5     public static void main(String[] args) {
6         CadenaCaracter caracter = new CadenaCaracter();
7
8         caracter.contarCaracter();
9     }
10 }
11
12

```

Pruebas

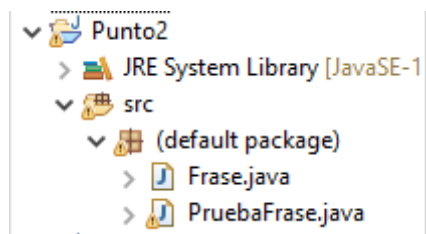


```
<terminated> PruebaCaracter [Java Application] C:\Program Files\OpenJDK\openjdk-11.0.13_
La cantidad de veces que se encuentra el caracter e es igual a: 4
```

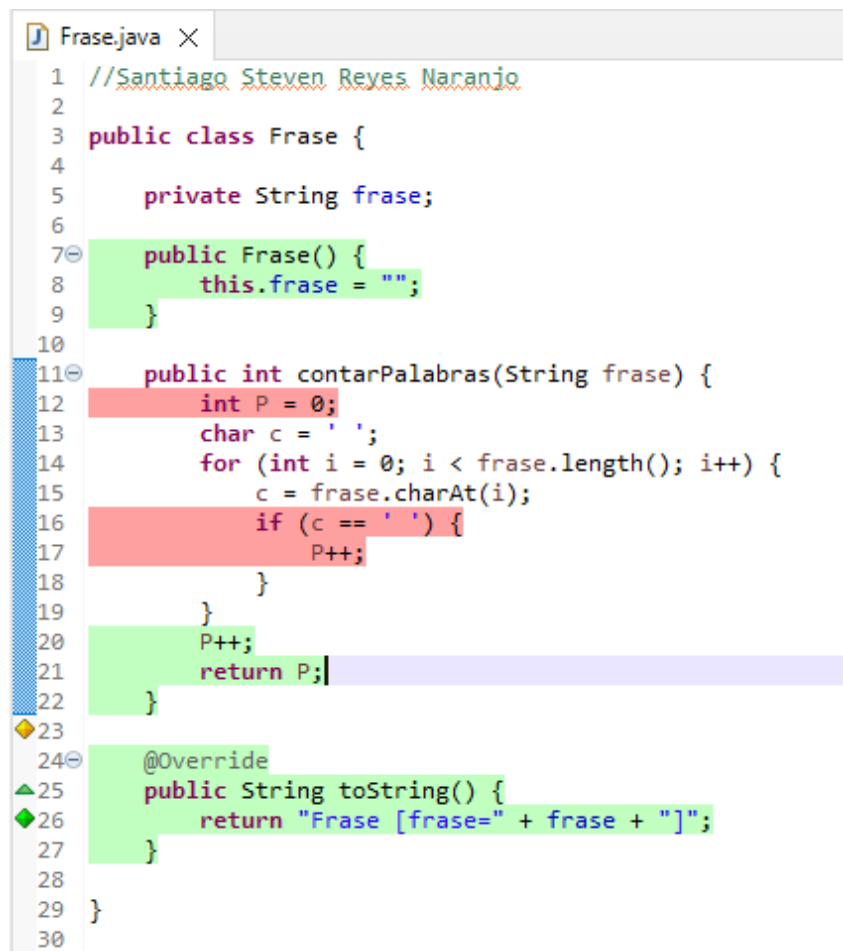
π Punto 2

Programa que encuentre en número de palabras que conforman una cadena dada (frase), haciendo uso de los elementos proporcionados por la POO.

Proyecto



Código fuente



```
1 //Santiago Steven Reyes Naranjo
2
3 public class Frase {
4
5     private String frase;
6
7     public Frase() {
8         this.frase = "";
9     }
10
11     public int contarPalabras(String frase) {
12         int P = 0;
13         char c = ' ';
14         for (int i = 0; i < frase.length(); i++) {
15             c = frase.charAt(i);
16             if (c == ' ') {
17                 P++;
18             }
19         }
20         P++;
21         return P;
22     }
23
24     @Override
25     public String toString() {
26         return "Frase [frase=" + frase + "];
27     }
28
29 }
30
```

```
Frse.java PruebaFrse.java X
1 import java.util.Scanner;
2
3 //Santiago Steven Reyes Naranjo
4
5 public class PruebaFrse {
6
7     public static void main(String[] args) {
8         String Frase = "";
9         System.out.println("Digite la frase para contar su cantidad de palabras: ");
10        Scanner lea = new Scanner(System.in);
11        Frase unaFrase = new Frase();
12        Frase = lea.nextLine();
13        System.out.println("La cantidad de palabras que tiene la frase digitada es de: " + unaFrase.contarPalabras(Frase));
14    }
15
16 }
17
```

Pruebas

```
Console X
<terminated> PruebaFrse (1) [Java Application] C:\Program Files\OpenJDK\openjdk-11.0.13_8\bin\jav
Digite la frase para contar su cantidad de palabras:
haber haber a mover la colita
La cantidad de palabras que tiene la frase digitada es de: 6
```

π Punto 3

Construir un programa que encuentre el número de veces y el carácter que más se repite en una cadena dada, haciendo uso de los elementos proporcionados por la POO.

Proyecto

```
▼ Punto3
  > JRE System Library [JavaSE-
  ▼ src
    ▼ (default package)
      > Cadena.java
      > PruebaCadena.java
```

Código fuente

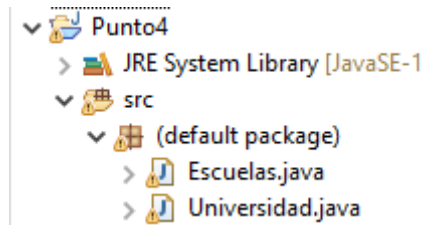
```
Cadena.java X *PruebaCadena.java
1 //Santiago Reyes
2
3 import java.util.HashMap;
4 import java.util.Map;
5
6 public class Cadena {
7     String cadena;
8
9     public Cadena() {
10    }
11
12     public Cadena(String cadena) {
13         this.cadena = cadena;
14     }
15
16     public String getCadena() {
17         return cadena;
18     }
19
20     public void setCadena(String cadena) {
21         this.cadena = cadena;
22     }
23
24     @Override
25
26     public String toString() {
27         return "Cadena (" + "cadena=" + cadena + ')';
28     }
29
30     public void contarCaracteres(String cadena) {
31         Map<Character, Integer> m = new HashMap();
32         for (char c : cadena.toCharArray()) {
33             m.put(c, m.containsKey(c) ? m.get(c) + 1 : 1);
34         }
35         m.forEach((c, v) -> {
36             System.out.println(c + " : " + v);
37         });
38     }
39 }
```

```
Cadena.java *PruebaCadena.java X
1 import java.util.Scanner;
2
3 //Santiago Reyes
4 public class PruebaCadena {
5
6     public static void main(String[] args) {
7         Scanner sc = new Scanner(System.in);
8         System.out.print("Introduzca una cadena: ");
9         String cadena = sc.nextLine();
10
11         Cadena miCadena = new Cadena();
12         miCadena.contarCaracteres(cadena);
13     }
14 }
```

π Punto 4

Construir una relación de clases que permita modelar la relación dada por la frase "la universidad tiene varias escuelas".

Proyecto

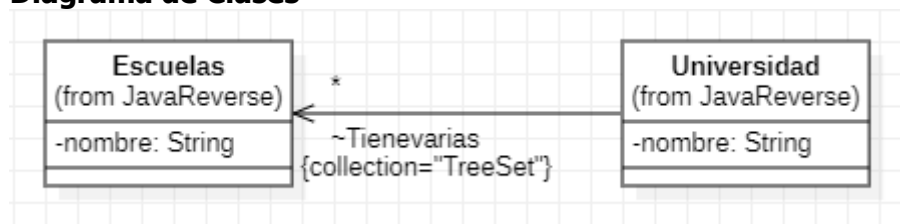


Código fuente

```
Universidad.java X Escuelas.java
1 //Santiago Reyes
2
3 import java.util.TreeSet;
4
5 public class Universidad {
6
7     private String nombre;
8     TreeSet<Escuelas> Tienevarias = new TreeSet<>();
9
10 }
11
```

```
Universidad.java Escuelas.java X
1 //Santiago Reyes
2
3 public class Escuelas {
4
5     private String nombre;
6
7 }
8
```

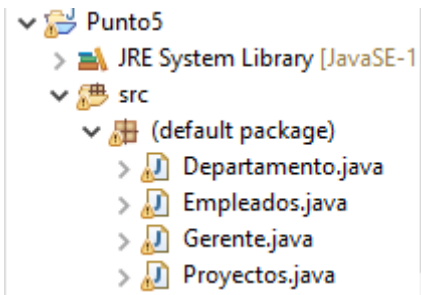
Diagrama de Clases



π Punto 5

Elaborar el diagrama de clases donde un gerente dirige un departamento, y un departamento tiene varios empleados y varios empleados trabajan en varios proyectos.

Proyecto



Código Fuente

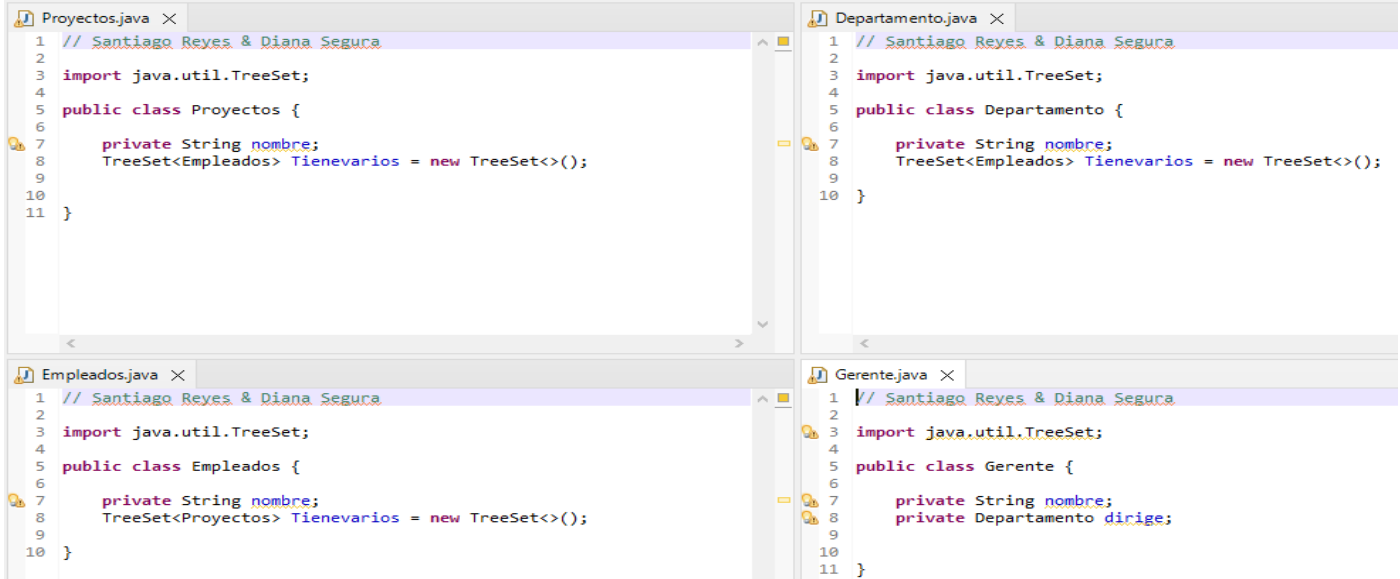
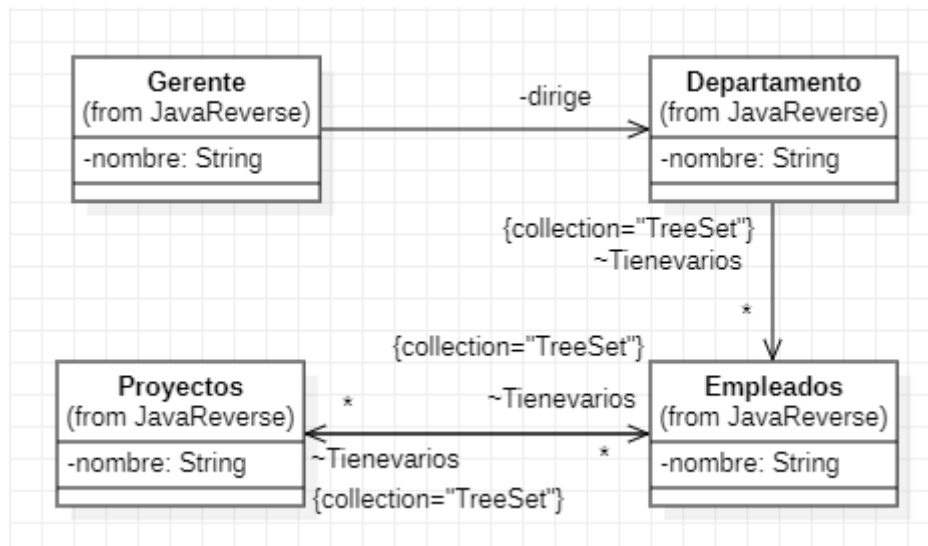


Diagrama de Clases



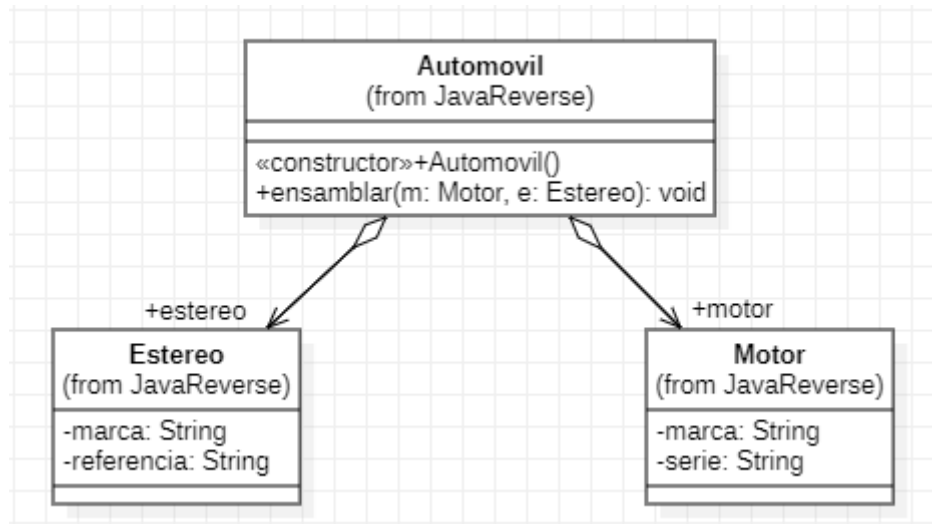
Tema – Relaciones entre clase: Agregación

π Enunciado #1

Creamos un programa llamado Agregacion, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Agregación.

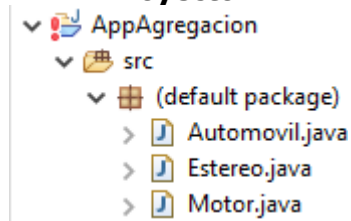
π

Diagrama de Clases



π

Proyecto



π

Código Fuente

```
Automovil.java X
1 //Santiago Reyes
2
3 public class Automovil {
4
5     public Motor motor;
6     public Estereo estereo;
7
8     public Automovil() {
9     }
10
11    public void ensamblar(Motor m, Estereo e) {
12        motor = m;
13        estereo = e;
14    }
15 }
16
```

```
Motor.java X
1 //Santiago Reyes
2
3 public class Motor {
4
5     private String marca;
6     private String serie;
7
8 }
9
Estereo.java X
1 //Santiago Reyes
2
3 public class Estereo {
4
5     private String marca;
6     private String referencia;
7
8 }
9
```

π **Análisis de resultados**

Se creó un proyecto java llamado **AppAgregacion**, en la cual se crearon tres clases una llamada **Automóvil**, **Motor** y otra clase llamada **Estéreo**.

En la clase **Automóvil** se declaró un método público, en el método se llaman **motor y estéreo** para poder crear el método **ensamblar**, el cual recibe los parámetros **motor y estéreo**. Esta clase es una relación de agregación ya que este objeto (Automóvil) está compuesta por otros objetos de otras clases.

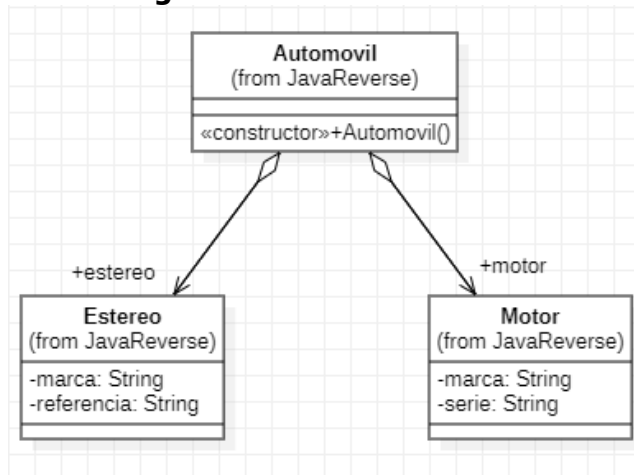
En la clase **Motor** se declaró dos atributos privados, el primer atributo es de tipo **String** llamado marca y el segundo atributo de tipo **String** llamado serie.

En la clase **Estéreo** se declaró dos atributos privados, el primer atributo es de tipo **String** llamado marca y el segundo atributo de tipo **String** llamado referencia.

π **Enunciado #2**

Creamos un programa llamado **Agregacion2**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Agregación.

π **Diagrama de Clases**



π **Proyecto**

```
AppAgregacion2
├── src
│   └── (default package)
└── *Automovil.java X
    1 //Santiago Reyes
    2 public class Automovil {
    3
    4     public Motor motor;
    5     public Estereo estereo;
    6
    7     public Automovil() {
    8
    9         estereo = new Estereo();
    10        motor = new Motor();
    11
    12    }
    13
    14 }
    15

    *Motor.java X
    1 //Santiago Reyes
    2
    3 public class Motor {
    4
    5     private String marca;
    6     private String serie;
    7
    8 }
    9

    *Estereo.java X
    1 //Santiago Reyes
    2
    3 public class Estereo {
    4
    5     private String marca;
    6     private String referencia;
    7
    8 }
    9
```


π

Análisis de resultados

Se creó un proyecto java llamado **AppAgregacion2**, en la cual se crearon tres clases una llamada **Automóvil**, **Motor** y otra clase llamada **Estéreo**.

En la clase **Automóvil** se declaró un método público, en el método se declara **motor y estéreo** para poder crear el método constructor, el cual crea los objetos de tipo **Motor y Estéreo**. Esta clase es una relación de agregación ya que este objeto (Automovil) está compuesta por otros objetos de otras clases.

En la clase **Motor** se declaró dos atributos privados, el primer atributo es de tipo **String** llamado marca y el segundo atributo de tipo **String** llamado serie.

En la clase **Estéreo** se declaró dos atributos privados, el primer atributo es de tipo **String** llamado marca y el segundo atributo de tipo **String** llamado referencia.

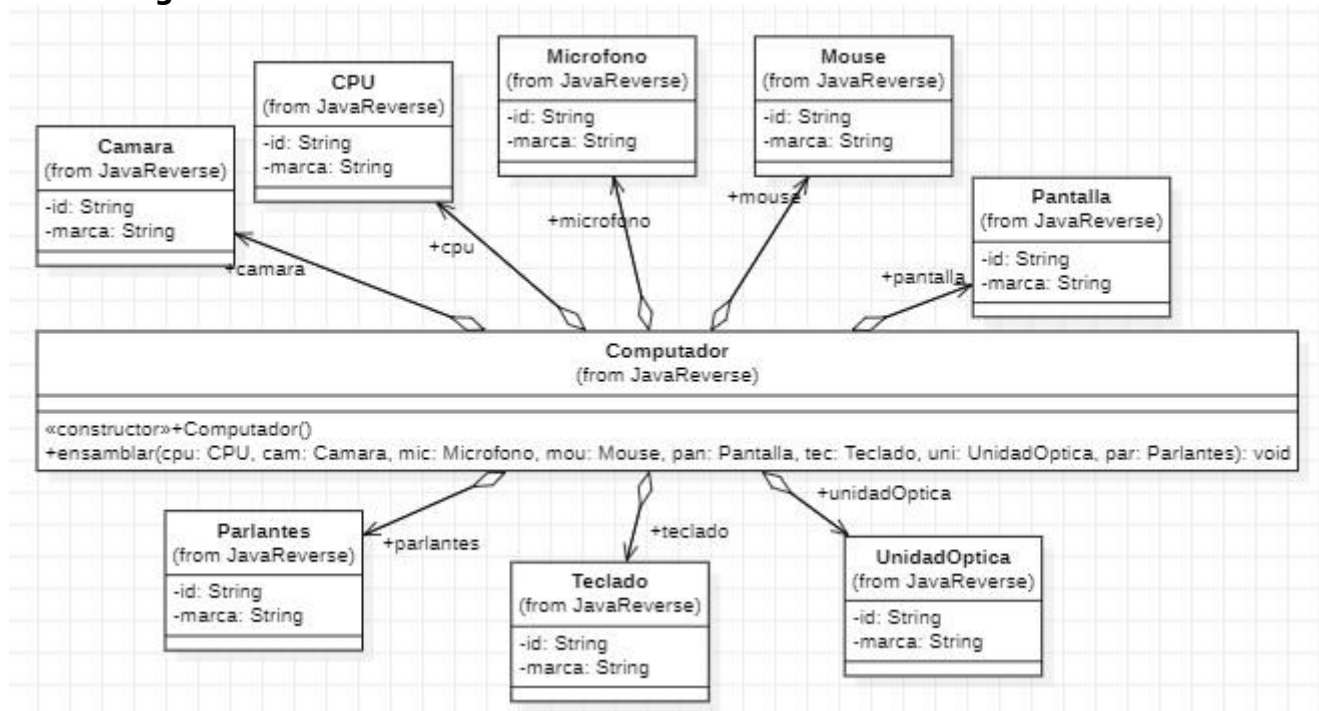
π

Enunciado #3

Creamos un programa llamado **Agregacion3**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Agregación.

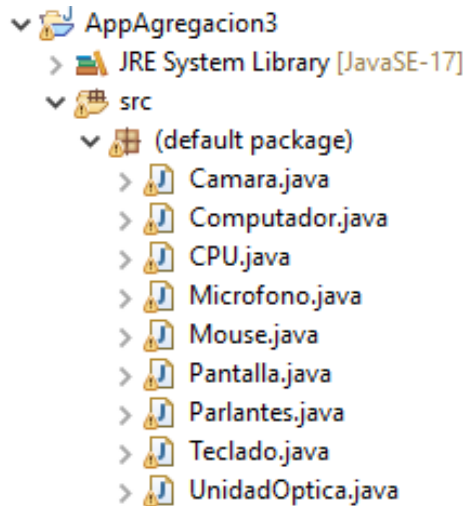
π

Diagrama de Clases



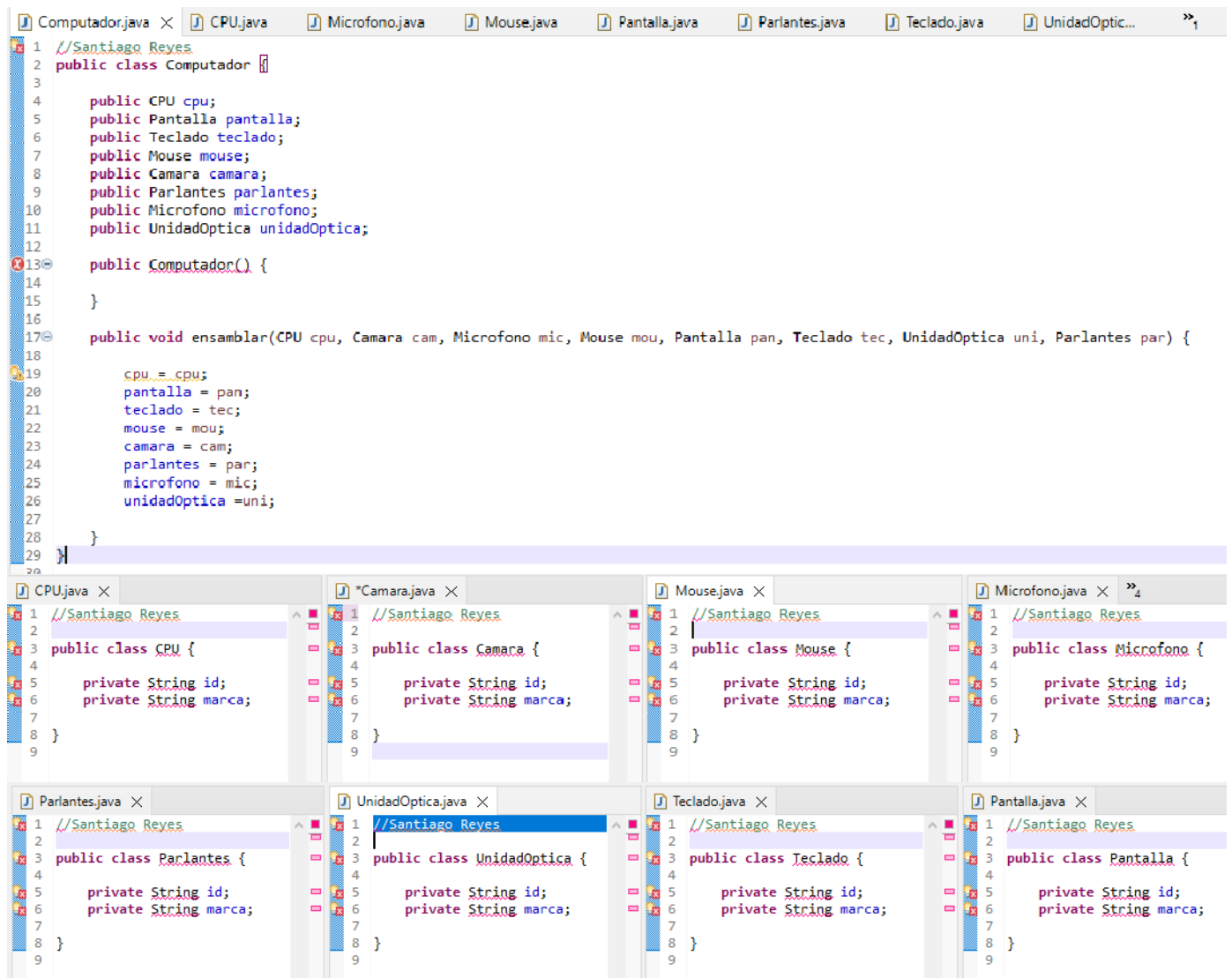
π

Proyecto



π

Código Fuente.



Análisis de resultados

Se creó un proyecto java llamado **AppAgregacion3**, en la cual se crearon nueve clases llamadas **Computador**, **CPU**, **Pantalla**, **Teclado**, **Mouse**, **Cámara**, **Parlantes**, **Micrófono** y **UnidadOptica**.

En la clase **Computador** se declaró un atributo público para cada una de las clases, en los atributos se declaran, **CPU**, **pantalla**, **teclado**, **mouse**, **cámara**, **parlantes**, **microfono** y **unidadOptica** para poder crear el método **ensamblar**, el cual recibe los parámetros **CPU**, **pantalla**, **teclado**, **mouse**, **cámara**, **parlantes**, **microfono** y **unidadOptica**. Los cuales quedan convertidos en **cpu**, **pan**, **tec**, **mou**, **cam**, **par**, **mic** y **uni**. Esta clase es una relación de agregación ya que este objeto (Computador) está compuesto por otros objetos de otras clases.

En la clase **CPU** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

.

En la clase **Pantalla** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

En la clase **Teclado** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

En la clase **Mouse** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

En la clase **Cámara** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

En la clase **Parlantes** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

En la clase **Micrófono** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

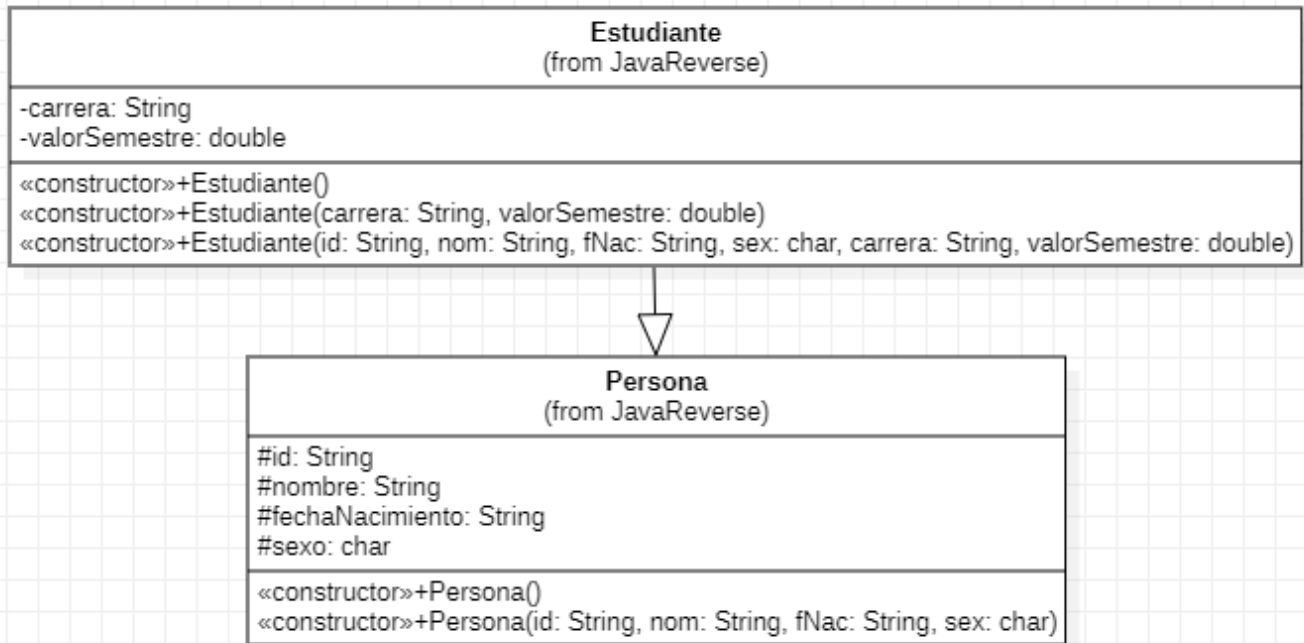
En la clase **UnidadOptica** se declaró dos atributos privados tipo **String** llamados **id** y **marca**.

Tema – Relaciones entre clase: Especialización/Generalización [Herencia]

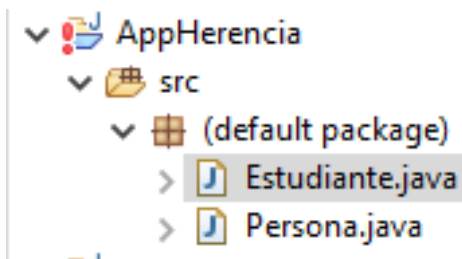
π Enunciado #1

Creemos un programa llamado **Herencia1**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Herencia.

π Diagrama de Clases



π Proyecto



```

Estudiante.java Persona.java X
1 // Santiago Reyes
2 public class Persona {
3
4     protected String id;
5     protected String nombre;
6     protected String fechaNacimiento;
7     protected char sexo;
8
9     public Persona() {
10         this.id = "1006825494";
11         this.nombre = "Santiago Reyes";
12         this.fechaNacimiento = "02/04/03";
13         this.sexo = "M";
14     }
15
16     public Persona (String id, String nom, String fNac, char sex) {
17
18         this.id = id;
19         this.nombre = nom;
20         this.fechaNacimiento = fNac;
21         this.sexo = sex;
22     }
23 }
24
Estudiante.java X Persona.java
1 // Santiago Reyes
2
3 public class Estudiante extends Persona{
4
5     private String carrera;
6     private double valorSemestre;
7
8     public Estudiante() {
9
10         this.carrera = "Ingeniería electrónica";
11         this.valorSemestre = 12345678;
12     }
13
14     public Estudiante(String carrera, double valorSemestre) {
15
16         this.carrera = carrera;
17         valorSemestre = valorSemestre;
18     }
19
20     public Estudiante(String id, String nom, String fNac, char sex, String carrera, double valorSemestre) {
21
22         super(id, nom, fNac, sex);
23         this.carrera = carrera;
24         valorSemestre = valorSemestre;
25     }
26
27 }
28

```

Análisis de resultados

Se creó un proyecto java llamado **Herencia1**, en la cual se crearon dos clases una llamada **Persona** y otra clase llamada **Estudiante**.

En la clase **Persona** se declararon cuatro atributos protegidos(**protected**), tres atributos de tipo **String** llamado **nombre**, **id**, **fechaNacimiento**, el otro atributo de tipo **char** llamado **sexo**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **id**, **nombre** **fechaNacimiento** y **sexo**. Usamos el método constructor para crear nuevos atributos.

En la clase **Estudiante** se declararon dos atributos privados, uno de los atributos es de tipo **String** llamado **carrera**, el otro atributo de tipo **double** llamado **valorSemestre**.

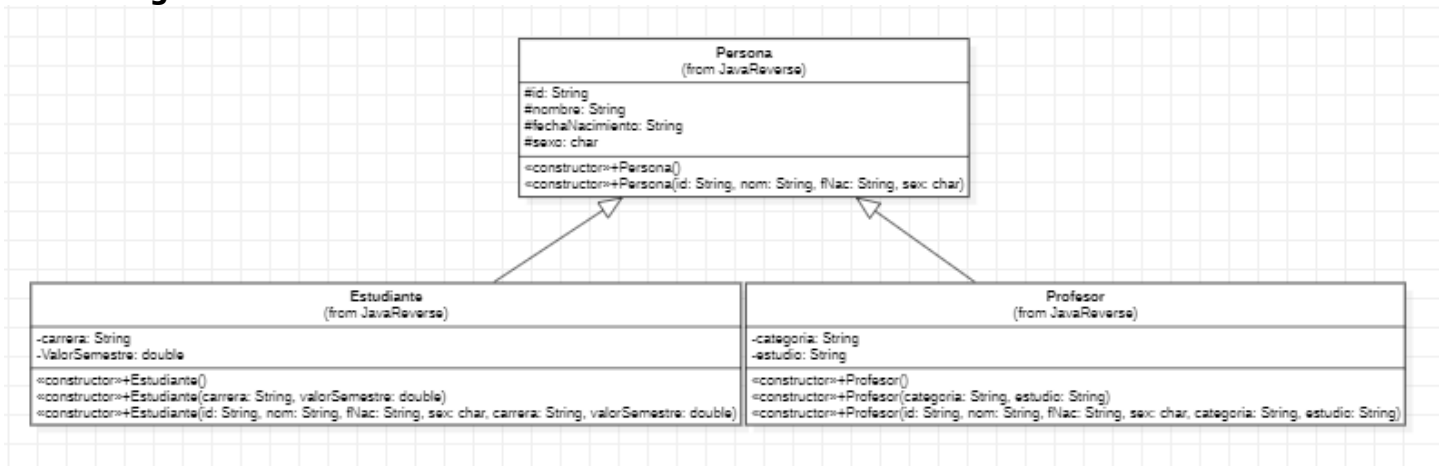
Haciendo uso de **this** declaramos nuevos parámetros para los atributos **carrera y valorSemestre**. Usamos el método constructor para crear nuevos atributos. Usamos el método constructor **super** para usar lo atributos de la clase **Persona**, y generar los atributos **carrera y valorSemestre** haciendo uso de **this**.

En la clase **Estudiante** hacemos uso de **extends** la cual cumple con la función de llamar a la clase llamada **Persona** para poder hacer uso de la relación entre clases (**herencia**), que permite definir nuevas clases a partir de otras ya definidas para crear clases basadas en clases ya existentes, de forma que heredan las propiedades de la clase padre.

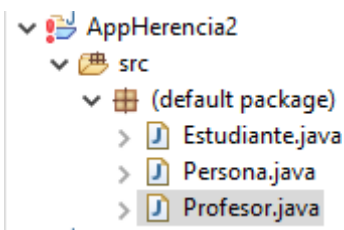
π Enunciado #2

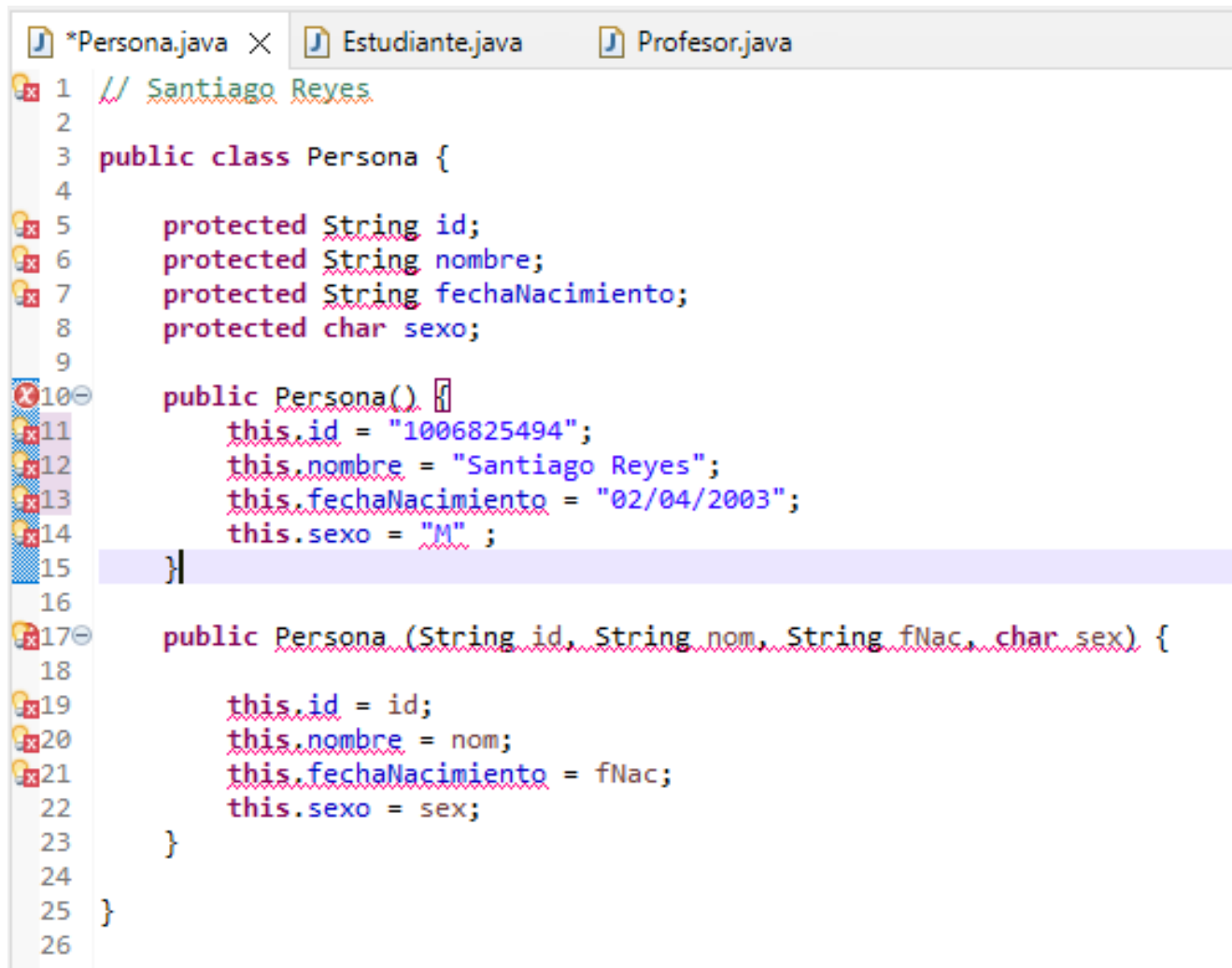
Creamos un programa llamado Herencia2, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Herencia.

π Diagrama de Clases



π Proyecto





```
1 // Santiago Reyes
2
3 public class Persona {
4
5     protected String id;
6     protected String nombre;
7     protected String fechaNacimiento;
8     protected char sexo;
9
10    public Persona() {
11        this.id = "1006825494";
12        this.nombre = "Santiago Reyes";
13        this.fechaNacimiento = "02/04/2003";
14        this.sexo = "M" ;
15    }
16
17    public Persona (String id, String nom, String fNac, char sex) {
18
19        this.id = id;
20        this.nombre = nom;
21        this.fechaNacimiento = fNac;
22        this.sexo = sex;
23    }
24
25 }
26
```

```

1 // Santiago Reyes
2
3 public class Profesor extends Persona {
4
5     private String categoria;
6     private String estudio;
7
8     public Profesor() {
9
10        this.categoria = "P00";
11        this.estudio = "MaestriaSistemas";
12    }
13
14    public Profesor(String categoria, String estudio) {
15
16        this.categoria = categoria;
17        this.estudio = estudio;
18    }
19
20    public Profesor(String id, String nom, String fNac, char sex, String categoria, String estudio) {
21        super(id, nom, fNac, sex);
22        this.categoria = categoria;
23        this.estudio = estudio;
24    }
25
26
27
28 }
29

```

```

1 // Santiago Reyes
2 public class Estudiante extends Persona {
3
4     private String carrera;
5     private double ValorSemestre;
6
7     public Estudiante() {
8
9         this.carrera = "Ing Sistemas";
10        this.ValorSemestre = 12345678;
11    }
12
13    public Estudiante(String carrera, double valorSemestre) {
14
15        this.carrera = carrera;
16        ValorSemestre = valorSemestre;
17    }
18
19    public Estudiante(String id, String nom, String fNac, char sex, String carrera, double valorSemestre) {
20
21        super(id, nom, fNac, sex);
22        this.carrera = carrera;
23        ValorSemestre = valorSemestre;
24    }
25
26 }
27

```

π

Análisis de resultados

Se creó un proyecto java llamado **Herencia2**, en la cual se crearon tres clases una llamada **Persona**, una clase llamada **Estudiante** y otra llamada **Profesor**.

En la clase **Persona** se declararon cuatro atributos protegidos(**protected**), tres atributos de tipo **String** llamado **nombre**, **id**, **fechaNacimiento**, el otro atributo de tipo **char** llamado **sexo**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **id**, **nombre** **fechaNacimiento** y **sexo**. Usamos el método constructor para crear nuevos atributos haciendo uso de **this**.

En la clase **Estudiante** se declararon dos atributos privados, uno de los atributos es de tipo **String** llamado **carrera**, el otro atributo de tipo **double** llamado **valorSemestre**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **carrera** y **valorSemestre**.

Usamos el método constructor para crear nuevos atributos haciendo uso de **this**. Usamos el método constructor **super** para usar los atributos de la clase **Persona**, y generar los atributos **carrera y valorSemestre** haciendo uso de **this**.

En la clase **Profesor** se declararon dos atributos privados, dos atributos de tipo **String** llamados **categoría y estudio**.

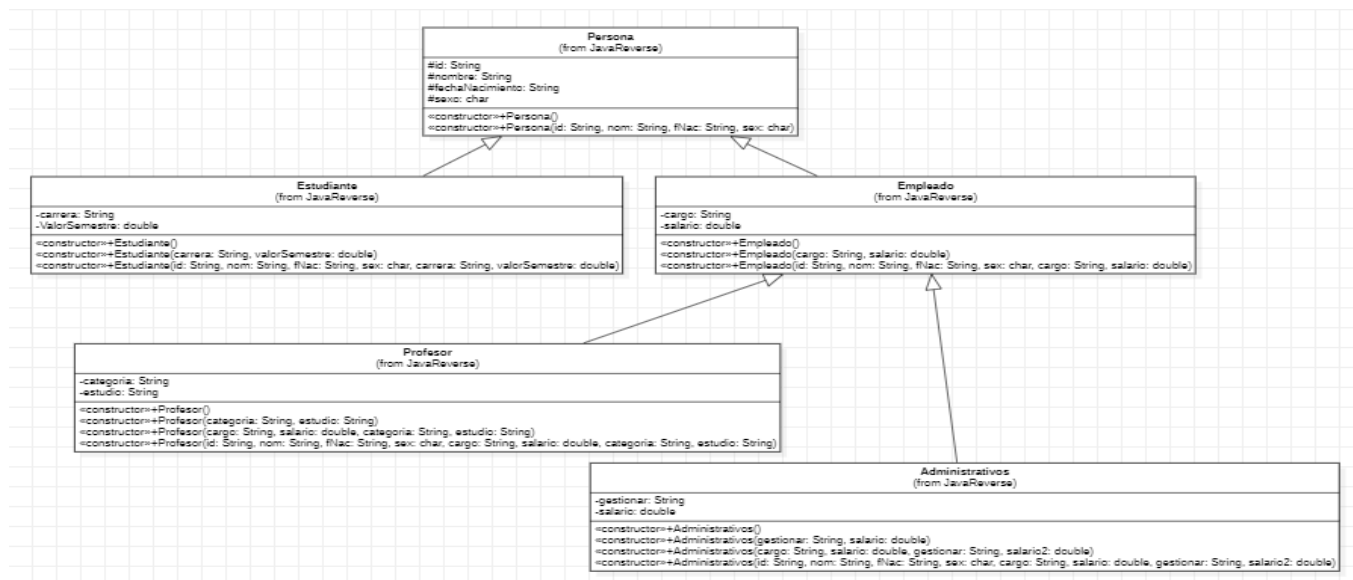
Haciendo uso de **this** declaramos nuevos parámetros para los atributos **categoría y estudio**. Usamos el método constructor para crear nuevos atributos haciendo uso de **this**. Usamos el método constructor **super** para usar los atributos de la clase **Persona**, y generar los atributos **categoría y estudio** haciendo uso de la variable **this**.

En las clases **Estudiante y Profesor** hacemos uso de **extends** lo cual cumple con la función de llamar a la clase llamada **Persona** para poder hacer uso de la relación entre clase (**herencia**), que permite definir nuevas clases a partir de otras ya definidas para crear clases basadas en clases ya existentes, de forma que heredan las propiedades de la clase base.

π Enunciado #3

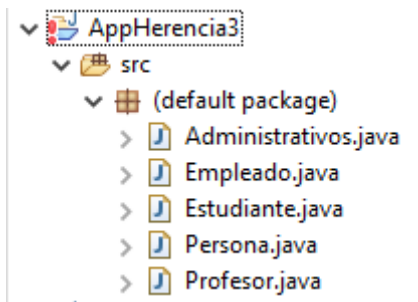
Creamos un programa llamado **Herencia3**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de herencia.

π Diagrama de Clases



π

Proyecto



π

Código Fuente

```
Persona.java × Estudiante.java
1 //Santiago Reyes
2
3 public class Persona {
4
5     protected String id;
6     protected String nombre;
7     protected String fechaNacimiento;
8     protected char sexo;
9
10    public Persona() {
11        this.id = "1006825494";
12        this.nombre = "Santiago Reyes";
13        this.fechaNacimiento = "02/04/03";
14        this.sexo = "M";
15    }
16
17    public Persona(String id, String nom, String fNac, char sex) {
18
19        this.id = id;
20        this.nombre = nom;
21        this.fechaNacimiento = fNac;
22        this.sexo = sex;
23    }
24 }
25
```

```
Persona.java Estudiante.java ×
1 //Santiago Reyes
2
3 public class Estudiante extends Persona {
4
5     private String carrera;
6     private double ValorSemestre;
7
8     public Estudiante() {
9
10        this.carrera = "Ingeniería electrónica";
11        this.ValorSemestre = 5004389;
12    }
13
14    public Estudiante(String carrera, double valorSemestre) {
15
16        this.carrera = carrera;
17        ValorSemestre = valorSemestre;
18    }
19
20    public Estudiante(String id, String nom, String fNac, char sex, String carrera, double valorSemestre) {
21
22        super(id, nom, fNac, sex);
23        this.carrera = carrera;
24        ValorSemestre = valorSemestre;
25
26    }
27
28 }
29
```

```
Persona.java Estudiante.java Empleado.java X Administrativos.java Profesor.java
1 //Santiago Reyes
2
3 public class Empleado extends Persona {
4
5     private String cargo;
6     private double salario;
7
8     public Empleado() {
9
10        this.cargo = "Profesor Informatica";
11        this.salario = 120000;
12
13    }
14
15    public Empleado(String cargo, double salario) {
16
17        this.cargo = cargo;
18        this.salario = salario;
19    }
20
21    public Empleado(String id, String nom, String fNac, char sex, String cargo, double salario) {
22
23        super(id, nom, fNac, sex);
24        this.cargo = cargo;
25        this.salario = salario;
26    }
27
28 }
29
```

```
Persona.java Estudiante.java Empleado.java Administrativos.java X Profesor.java
1 //Santiago Reyes
2
3 public class Administrativos extends Empleado {
4
5     private String gestionar;
6     private double salario;
7
8     public Administrativos() {
9
10        this.gestionar = "presupuestos";
11        this.salario = 130000;
12
13    }
14
15    public Administrativos(String gestionar, double salario) {
16
17        this.gestionar = gestionar;
18        this.salario = salario;
19    }
20
21    public Administrativos(String cargo, double salario, String gestionar, double salario2) {
22        super(cargo, salario);
23        this.gestionar = gestionar;
24        salario = salario2;
25    }
26
27    public Administrativos(String id, String nom, String fNac, char sex, String cargo, double salario, String gestionar,
28        double salario2) {
29        super(id, nom, fNac, sex, cargo, salario);
30        this.gestionar = gestionar;
31        salario = salario2;
32    }
33
34 }
```

```

1 //Santiago Reyes
2
3 public class Profesor extends Empleado {
4
5     private String categoria;
6     private String estudio;
7
8     public Profesor() {
9
10        this.categoria = "P00";
11        this.estudio = "MaestriaSistemas";
12    }
13
14
15    public Profesor(String categoria, String estudio) {
16
17        this.categoria = categoria;
18        this.estudio = estudio;
19    }
20
21    public Profesor(String cargo, double salario, String categoria, String estudio) {
22        super(cargo, salario);
23        this.categoria = categoria;
24        this.estudio = estudio;
25    }
26
27    public Profesor(String id, String nom, String fNac, char sex, String cargo, double salario, String categoria,
28        String estudio) {
29        super(id, nom, fNac, sex, cargo, salario);
30        this.categoria = categoria;
31        this.estudio = estudio;
32    }
33
34 }
35

```

π Análisis de resultados

Se creó un proyecto java llamado **Herencia3**, en la cual se crearon cinco clases una llamada **Persona**, una clase llamada **Estudiante**, una clasellamada **Empleado**, una clase llamada **Profesor** y otra clase llamada **Administrativos**.

En la clase **Persona** se declararon cuatro atributos protegidos(**protected**), tres atributos de tipo **String** llamado **nombre**, **id**, **fechaNacimiento**, el otro atributo de tipo **char** llamado **sexo**.

Haciendo uso de la variable **this** declaramos nuevos parámetros para los atributos **id**, **nombre** **fechaNacimiento** y **sexo**. Usamos el método constructor para crear nuevos atributos haciendouso de **this**.

En la clase **Estudiante** se declararon dos atributos privados, uno de los atributos es de tipo **String** llamado **carrera**, el otro atributo de tipo **double** llamado **valorSemestre**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **carrera** y **valorSemestre**. Usamos el método constructor para crear nuevos atributos haciendo uso de **this**. Usamos el método constructor **super** para usar lo atributos de la clase **Persona**, y generar los atributos **carrera** y **valorSemestre** haciendo uso de **this**.

En la clase **Profesor** se declararon dos atributos privados, dos atributos de tipo **String** llamados **categoría** y **estudio**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **categoría** y **estudio**. Usamos el método constructor para crear nuevos atributos haciendo uso de **this**. Usamos el método constructor **super** para usar lo atributos de la clase **Persona**, y generar los atributos **categoría** y **estudio** haciendouso de la variable **this**.

En la clase **Empleado** se declararon dos atributos privados de tipo String llamados **categoría y estudio**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **categoría y estudio**. Usamos el método constructor para crear nuevos atributos haciendo uso de **this**. Usamos el método constructor **super** para usar lo atributos de la clase **Empleado**, y generar los atributos **categoría y estudio** haciendo uso de **this**.

En la clase **Administrativos** se declararon dos atributos privados, uno de los atributos es de tipo **String** llamado **gestionar**, el otro atributo de tipo **double** llamado **salario**.

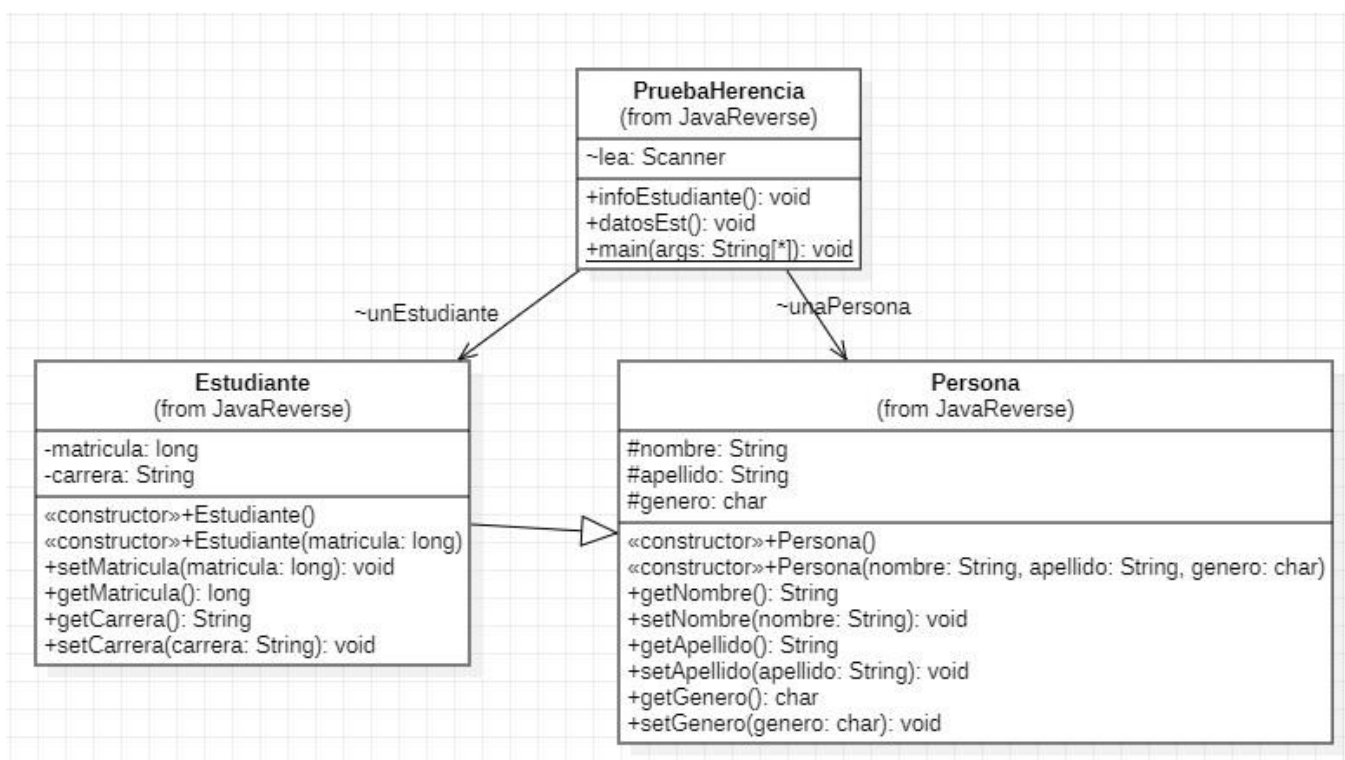
Haciendo uso de **this** declaramos nuevos parámetros para los atributos **gestionar y salario**. Usamos el método constructor para crear nuevos atributos haciendo uso de **this**. Usamos el método constructor **super** para usar lo atributos de la clase **Empleado**, y generar los atributos **gestionar y salario** haciendo uso de **this**.

En las clases **Estudiante y Empleado** hacemos uso de **extends** la cual cumple con la función de llamar a la clase llamada **Persona**; y en las clases **Administrativos y Profesor** hacemos uso de **extends** la cual cumple con la función de llamar a la clase llamada **Empleado** para poder hacer uso de la relación entre clase (**herencia**), que permite definir nuevas clases a partir de otras ya definidas para crear clases basadas en clases ya existentes, de forma que heredan las propiedades de la clase base.

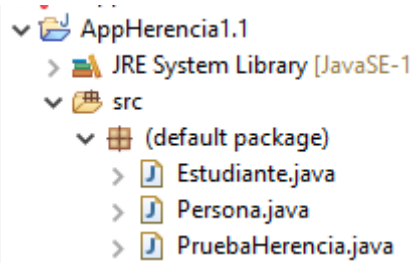
π Enunciado #4

Creamos un programa llamado **Herencia1.1**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de herencia.

π Diagrama de Clases



π Proyecto



π

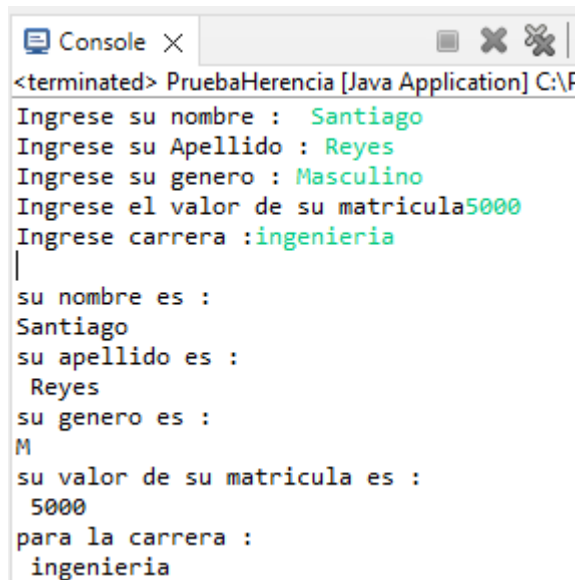
Código Fuente

```
Persona.java × Estudiante.java PruebaHerencia.java
1 //Santiago Reyes
2
3 public class Persona {
4
5     protected String nombre;
6     protected String apellido;
7     protected char genero;
8
9
10    public Persona() {
11        this.nombre = "";
12        this.apellido = "";
13        this.genero = ' ';
14    }
15
16    public Persona (String nombre,String apellido, char genero) {
17        this.nombre = nombre;
18        this.apellido = apellido;
19        this.genero = genero;
20    }
21
22    public String getNombre() {
23        return nombre;
24    }
25
26    public void setNombre(String nombre) {
27        this.nombre = nombre;
28    }
29
30    public String getApellido() {
31        return apellido;
32    }
33
34    public void setApellido(String apellido) {
35        this.apellido = apellido;
36    }
37
38    public char getGenero() {
39        return genero;
40    }
41
42    public void setGenero(char genero) {
43        this.genero = genero;
44    }
45 }
46
47
```

```
Persona.java | Estudiante.java X | PruebaHerencia.java
1 //Santiago Reyes
2
3 public class Estudiante extends Persona{
4     private long matricula;
5     private String carrera;
6
7     public Estudiante() {
8         this.matricula=0;
9     }
10    public Estudiante(long matricula) {
11        this.matricula = matricula;
12    }
13    public void setMatricula(long matricula) {
14        this.matricula = matricula;
15    }
16    public long getMatricula() {
17        return matricula;
18    }
19    public String getCarrera() {
20        return carrera;
21    }
22    public void setCarrera(String carrera) {
23        this.carrera = carrera;
24    }
25
26 }
27
```

```
*PruebaHerencia.java X
2 //Santiago Reyes
3 import java.util.Scanner;
4
5 public class PruebaHerencia {
6     Scanner lea = new Scanner(System.in);
7     Estudiante unEstudiante = new Estudiante();
8     public void infoEstudiante() {
9         System.out.print("Ingrese su nombre : ");
10        String nombre = lea.next();
11        unEstudiante.setNombre(nombre);
12        System.out.print("Ingrese su Apellido : ");
13        String apellido = lea.next();
14        unEstudiante.setApellido(apellido);
15        System.out.print("Ingrese su genero : ");
16        String genero = lea.next();
17        unEstudiante.setGenero(genero.charAt(0));
18        System.out.print("Ingrese el valor de su matricula");
19        Long matricula = lea.nextLong();
20        unEstudiante.setMatricula(matricula);
21        System.out.print("Ingrese carrera :");
22        String carrera = lea.next();
23        unEstudiante.setCarrera(carrera);
24    }
25
26    public void datosEst() {
27        System.out.print("\nsu nombre es :\n" + unEstudiante.getNombre());
28        System.out.print("\nsu apellido es :\n " + unEstudiante.getApellido());
29        System.out.print("\nsu genero es :\n" + unEstudiante.getGenero());
30        System.out.print("\nsu valor de su matricula es :\n " + unEstudiante.getMatricula());
31        System.out.print("\npara la carrera :\n " + unEstudiante.getCarrera());
32    }
33
34    public static void main(String[] args) {
35        PruebaHerencia prueba = new PruebaHerencia();
36        prueba.infoEstudiante();
37        prueba.datosEst();
38    }
39 }

```

```
<terminated> PruebaHerencia [Java Application] C:\F
Ingrese su nombre : Santiago
Ingrese su Apellido : Reyes
Ingrese su genero : Masculino
Ingrese el valor de su matricula5000
Ingrese carrera :ingenieria
|
su nombre es :
Santiago
su apellido es :
Reyes
su genero es :
M
su valor de su matricula es :
5000
para la carrera :
ingenieria
```

Análisis de resultados

Haciendo uso de las aplicaciones Eclipse y StarUML, se creó un proyecto java llamado **Herencia1**, en la cual se crearon tres clases una llamada **Persona**, otra clase llamada **Estudiante** y otra clase llamada **PruebaHerencia**.

En la clase **Persona** se declararon cuatro atributos protegidos(**protected**), tres atributos de tipo **String** llamado **nombre**, **id**, **fechaNacimiento**, el otro atributo de tipo **char** llamado **sexo**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **id**, **nombre** **fechaNacimiento** y **sexo**. Usamos el método constructor para crear nuevos atributos.

En la clase **Estudiante** se declararon dos atributos privados, uno de los atributos es de tipo **String** llamado **carrera**, el otro atributo de tipo **double** llamado **valorSemestre**.

Haciendo uso de **this** declaramos nuevos parámetros para los atributos **carrera** y **valorSemestre**. Usamos el método constructor para crear nuevos atributos. Usamos el método constructor **super** para usar lo atributos de la clase **Persona**, y generar los atributos **carrera** y **valorSemestre** haciendo uso de **this**.

En la clase **Estudiante** hacemos uso de **extends** la cual cumple con la función de llamar a la clase llamada **Persona** para poder hacer uso de la relación entre clases (**herencia**), que permite definir nuevas clases a partir de otras ya definidas para crear clases basadas en clases ya existentes, de forma que heredan las propiedades de la clase padre.

En la clase **PruebaHerencia** se generó un objeto de tipo **Estudiante** llamado **unEstudiante** y se generaron los métodos de **infoEstudiante** y **datosEst**.

En el método llamado **infoEstudiante** se hace uso de la librería **Scanner** para hacer uso del método **set** para así ingresar los valores a los atributos del objeto **unEstudiante**, asignando así valores a los atributos de **nombre**, **id**, **fechaNacimiento** (que se heredan de la clase persona) y los atributos de **carrera** y **valorSemestre**.

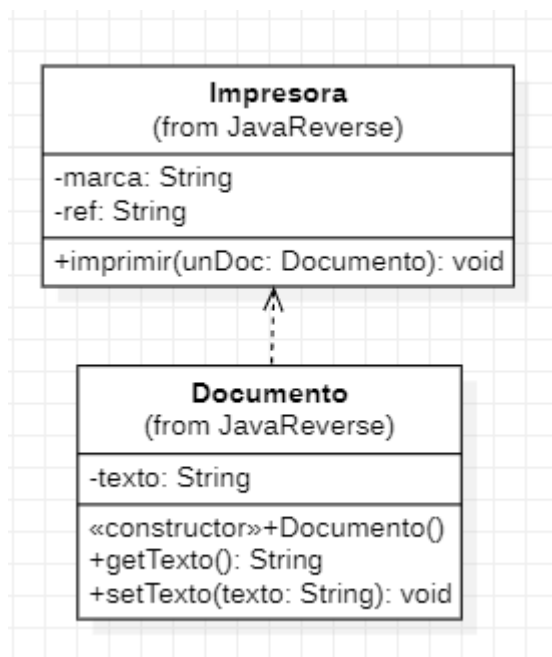
En el método llamado **datosEst** se hace uso del método **get** para así poder llamar los valores de los atributos asignados al objeto **unEstudiante** y poder imprimir el estudiante con sus respectivos atributos a través de la consola.

Tema – Relaciones entre clase: Dependencia

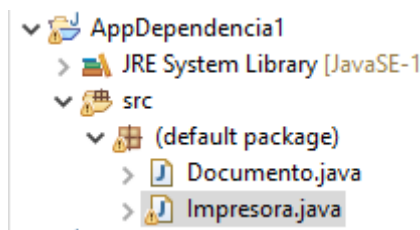
π **Enunciado #1**

Creamos un programa llamado **Dependencia1**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Dependencia.

π **Diagrama de Clases**



π **Proyecto**



π

Código Fuente

```
Impresora.java × Documento.java ×
1 //Santiago Reyes
2
3 public class Impresora {
4
5     private String marca;
6     private String ref;
7
8     public void imprimir (Documento unDoc) {
9         System.out.println("imprimiendo... ");
10    }
11
12 }
13

1 //Santiago Reyes
2
3 public class Documento {
4
5     private String texto;
6
7     public Documento() {
8         this.texto = "Tercer corte";
9     }
10
11    public String getTexto() {
12        return texto;
13    }
14
15    public void setTexto(String texto) {
16        this.texto = texto;
17    }
18
19 }
20
21
22
23
```

π

Análisis de resultados

Se creó un proyecto java llamado **Dependencia1**, en la cual se crearon dos clases una llamada **Impresora**, y otra clase llamada **Documento**.

En la clase **Documento** se declaró un atributo privado (**private**), el atributo es de tipo **String** llamado **texto**.

Haciendo uso de **this** asignamos un nuevo valor para el atributo **texto**. Luego haciendo uso del método **get** el cual retorna (**return**) el atributo de **texto**.

Haciendo uso del método **set** en el cual le daremos un valor al atributo llamado **texto**. En dónde usaremos **this** el cual tomara el valor principal y lo asigna al atributo llamado **texto**.

En la clase **Impresora** se declararon dos atributos privados, de tipo **String** llamados **marca y ref**. y usaremos el método **imprimir** para poder imprimir el objeto **unDoc** de tipo **Documento**.

Estamos utilizando objetos de otra clase, entonces estamos generando una relación entre clases de una relación de dependencia entre las respectivas clases.

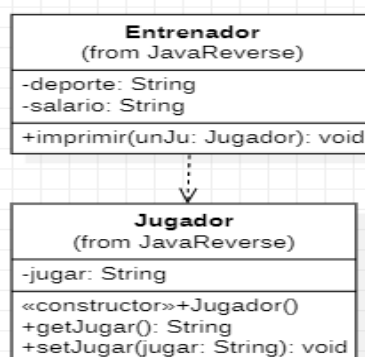
π

Enunciado #2

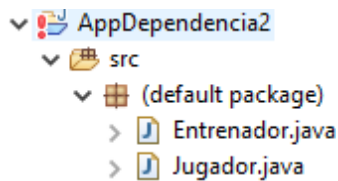
Creamos un programa llamado **Dependencia2**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Dependencia.

π

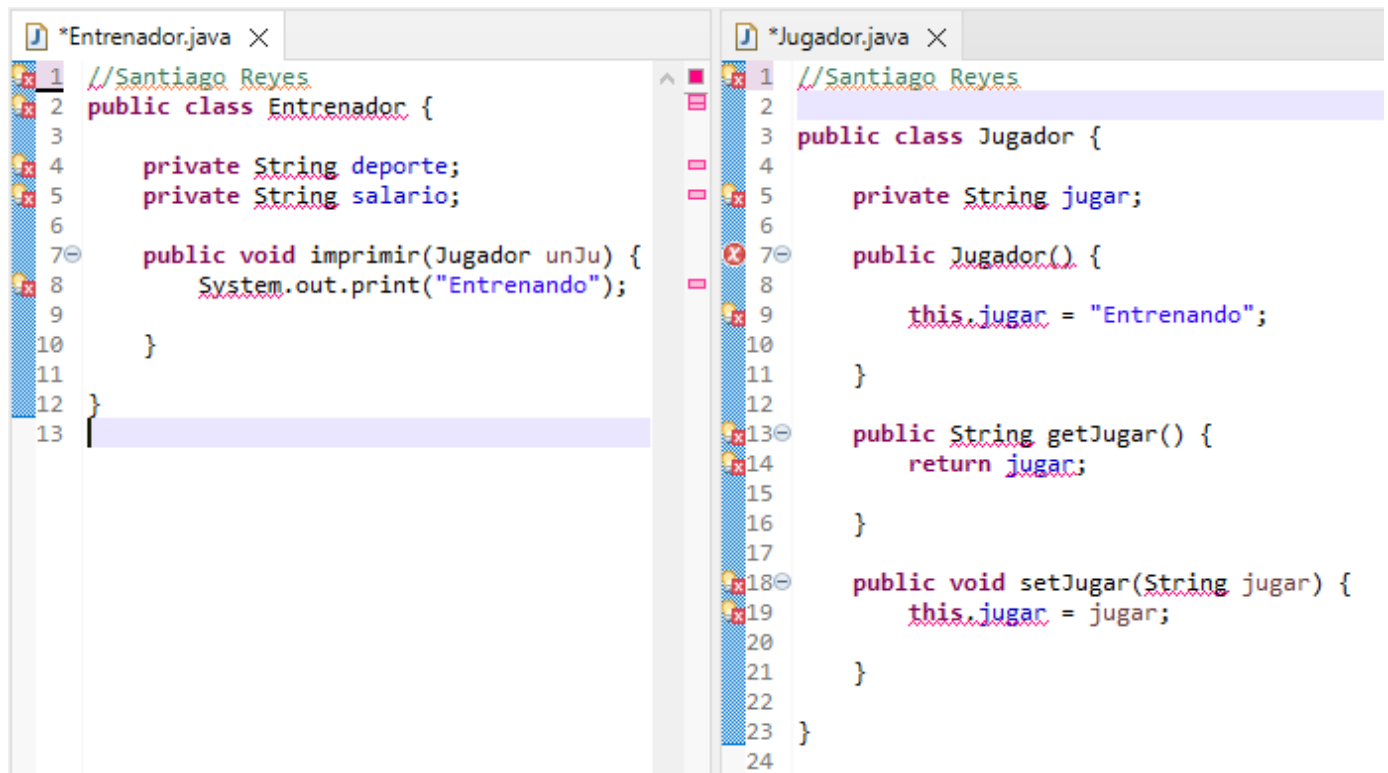
Diagrama de Clases



π Proyecto



π Código Fuente



π Análisis de resultados

Se creó un proyecto java llamado **Dependencia2**, en la cual se crearon dos clases una llamada **Entrenador**, y otra clase llamada **Jugador**.

En la clase **Jugador** se declaró un atributo privado (**private**), el atributo es de tipo **String** llamado **jugar**.

Haciendo uso de **this** asignamos un nuevo valor para el atributo **jugar**. Luego haciendo uso del método **get** el cual retorna (**return**) el atributo de **jugar**.

Haciendo uso del método **set** en el cual le daremos un valor al atributo llamado **jugar**. En dónde usaremos **this** el cual tomara el valor principal y lo asigna al atributo llamado **jugar**.

En la clase **Entrenador** se declararon dos atributos privados, de tipo **String** llamados **deporte** y **salario**. y usaremos el método **imprimir** para poder imprimir el objeto **unJu** de tipo **Jugador**.

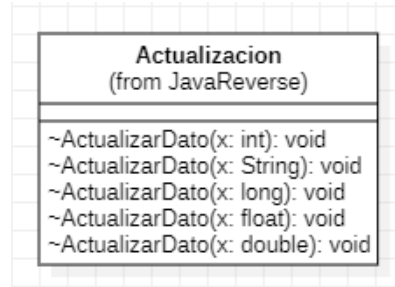
Estamos utilizando objetos de otra clase, entonces estamos generando una relación entre clases de una relación de dependencia entre las respectivas clases.

Tema – Polimorfismo

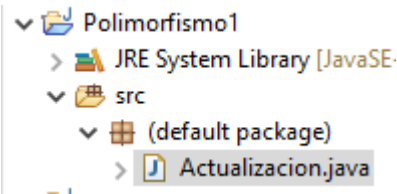
π Enunciado #1

Creamos un programa llamado **Polimorfismo**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Polimorfismo.

π Diagrama de Clases



π Proyecto



π Código Fuente

```
Actualizacion.java X
1 //Santiago Steven Reyes Naranjo
2
3 //sobrecarga de métodos
4 public class Actualizacion {
5
6     void ActualizarDato(int x) {}
7     void ActualizarDato(String x) {}
8     void ActualizarDato(long x) {}
9     void ActualizarDato(float x) {}
10    void ActualizarDato(double x) {}
11
12 }
13
```

π Análisis de resultados

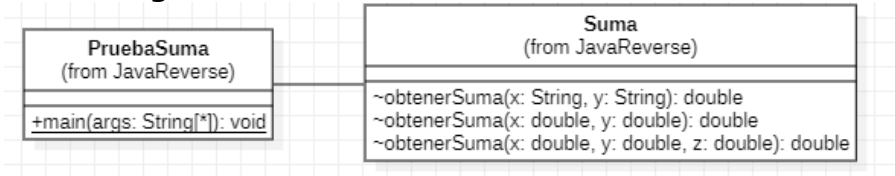
Se creó un proyecto java llamado **Polimorfismo1**, en la cual se creó una clase llamada **Actualización**.

En esta clase se declararon diferentes métodos llamados **ActualizarDato** de tipos **int**, **String**, **long**, **float**, **double**; todos con el mismo nombre, pero con diferente firma (cambiando su tipo de dato) generando así una sobrecarga de métodos que se refiere al uso del mismo identificador u operador en distintos contextos y con distintos significados para así poder hacer uso del mismo método de diferentes formas y en diferentes contextos.

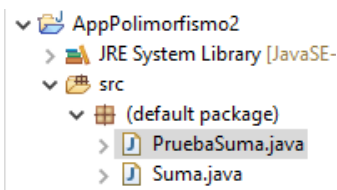
π Enunciado #2

Creamos un programa llamado **Polimorfismo2**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Polimorfismo.

π Diagrama de Clases



π Proyecto



π Código Fuente

```
1 //Santiago Reyes
2
3 public class Suma {
4
5     double obtenerSuma(String x, String y) {
6
7         // se hace la conversion porque en el método que nos dan nos recibe en String y
8         // para poder realizar la suma debe estar en un tipo de dato que permita hacer
9         // operaciones.
10        // return Double.parseDouble(x) + Double.parseDouble(y);
11        double x1 = Double.parseDouble(x);
12        double y1 = Double.parseDouble(y);
13        return x1 + y1;
14    }
15
16    double obtenerSuma(double x, double y) {
17        return x + y;
18    }
19
20    double obtenerSuma(double x, double y, double z) {
21        return x + y + z;
22    }
23
24 }
25
```

The screenshot shows the Eclipse IDE with two tabs open: **PruebaSuma.java** and **Suma.java**. The **Suma.java** tab is active, displaying the source code. The code defines a **public class Suma** with three methods: `obtenerSuma(String x, String y)`, `obtenerSuma(double x, double y)`, and `obtenerSuma(double x, double y, double z)`. The first method includes comments about string conversion and a commented-out return statement. The second and third methods are simple arithmetic operations. The code is enclosed in a closing brace at the end.

```

1 //Santiago Reyes
2
3 public class PruebaSuma {
4     public static void main(String[] args) {
5
6         double x = 2;
7         double y = 3;
8
9         Suma unaSuma = new Suma();
10        System.out.println("la suma de String es " + unaSuma.obtenerSuma(x, y));
11        System.out.println("la suma de 2 Double es " + unaSuma.obtenerSuma(x, y));
12        System.out.println("la suma de 3 Double es " + unaSuma.obtenerSuma(2.2, 3.5, 4.3));
13    }
14 }
15
16 }
17

```

Pruebas

```

Console X
<terminated> PruebaSuma [Java Application]
la suma de String es 5.0
la suma de 2 Double es 5.0
la suma de 3 Double es 10.0

```

Análisis de resultados

Se creó un proyecto java llamado **AppPolimorfismo2**, en la cual se crearon las clases **Suma** y otra llamada **PruebaSuma**.

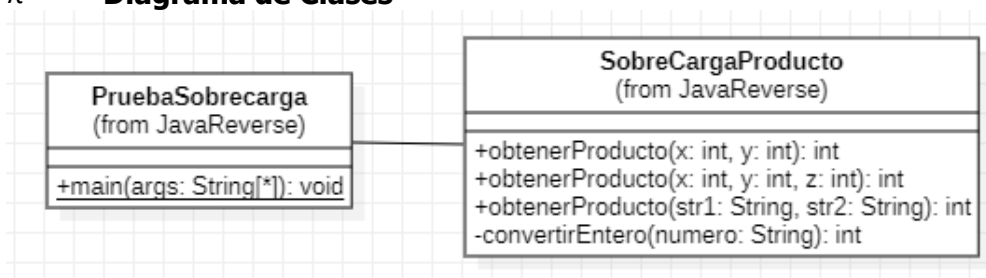
En la clase **Suma** se declararon 3 métodos llamados **obtenerSuma**, con el mismo nombre, pero con diferente firma (cambiando su tipo de dato), el primero con un tipo de dato **String**, el segundo con un tipo de dato **Double**, y el tercero con un dato más que los anteriores métodos y de tipo **Double** generando así una sobrecarga de métodos.

En la clase **PruebaSuma** se llamaron los 3 diferentes métodos haciendo uso de cada uno de ellos para poder operar diferentes tipos de datos haciendo el llamamiento de un mismo método.

Enunciado #3

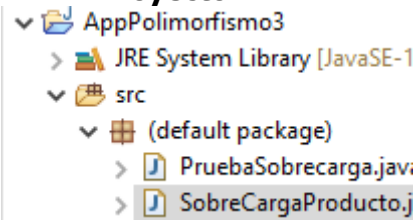
Creamos un programa llamado **Polimorfismo3**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Polimorfismo.

Diagrama de Clases



π

Proyecto



π

Código Fuente

```

1 // Santiago Reyes
2 public class PruebaSobrecarga {
3     public static void main(String[] args) {
4         int a = 2;
5         int b = 4;
6         int c = 5;
7         String str1 = "8";
8         String str2 = "7";
9         SobreCargaProducto sp = new SobreCargaProducto();
10        System.out.println("Método con dos parámetros: " + a + "x" + b + "=" + sp.obtenerProducto(a, b));
11        System.out.println("Método con tres parámetros: " + a + "x" + b + "x" + c + "=" + sp.obtenerProducto(a, b, c));
12        System.out.println(
13            "Método con parámetros tipo String: " + str1 + "x" + str2 + "=" + sp.obtenerProducto(str1, str2));
14    }
15 }
16 }
17

```

```

1 // Santiago Reyes
2
3 public class SobreCargaProducto {
4
5     public int obtenerProducto(int x, int y) {
6         return x * y;
7     }
8
9     public int obtenerProducto(int x, int y, int z) {
10        return x * y * z;
11    }
12
13    public int obtenerProducto(String str1, String str2) {
14        int x = convertirEntero(str1);
15        int y = convertirEntero(str2);
16        return x * y;
17    }
18
19    private int convertirEntero(String numero) {
20        return Integer.parseInt(numero);
21    }
22 }
23
24

```

π

Pruebas

```

Console X
<terminated> PruebaSobrecarga (1) [Java Application] C:
Método con dos parámetros: 2x4=8
Método con tres parámetros: 2x4x5=40
Método con parámetros tipo String: 8x7=56

```

π **Análisis de resultados**

Se creó un proyecto java llamado **AppPolimorfismo3**, en la cual se crearon las clases **SobrecargaProducto** y otra llamada **PruebaSobrecarga**.

En la clase **SobrecargaProducto** se crearon 3 métodos llamados **obtenerProducto**, con el mismo nombre, pero con diferente firma (cambiando su tipo de dato), el primero con un tipo de dato **int**, el segundo con un tipo de dato **int**, y el tercero con un dato más que los anteriores métodos y de tipo **String** pero retornando un tipo de dato **int** generando así una sobrecarga de métodos.

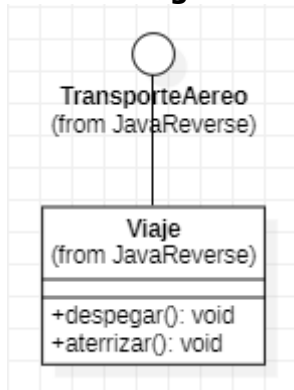
En la clase **PruebaSuma** declararon 3 atributos tipo **int** con 3 números diferentes más 2 atributos de tipo **String** con números también, luego, se generó el objeto de tipo **SobrecargaProducto**, después de esto, se llamaron los 3 **diferentes métodos** haciendo uso de cada uno de ellos para poder operar diferentes tipos de datos haciendo el llamamiento de un mismo método.

Tema – Interface y Clases Abstractas

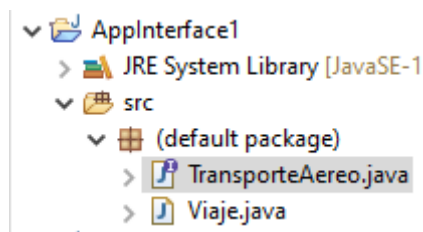
π **Enunciado #1**

Creamos un programa llamado **Interface**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Interface.

π **Diagrama de Clases**

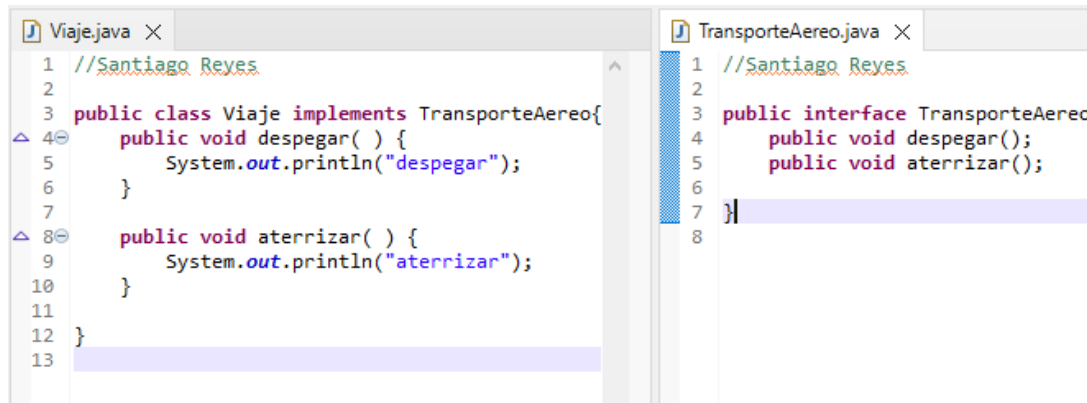


π **Proyecto**



π

Código Fuente



```
1 //Santiago Reyes
2
3 public class Viaje implements TransporteAereo{
4     public void despegar( ) {
5         System.out.println("despegar");
6     }
7
8     public void aterrizar( ) {
9         System.out.println("aterrizar");
10    }
11
12 }
13
```

```
1 //Santiago Reyes
2
3 public interface TransporteAereo
4     public void despegar();
5     public void aterrizar();
6
7 }
8
```

π

Análisis de resultados

Se creó un proyecto java llamado **AppInterface1**, en la cual se creó una clase llamada **Viaje**, y una interface llamada **TransporteAereo**.

En la clase **Viaje** se declaró un método llamado **despegar**. Y otro atributo método llamado **aterrizar**.

En esta clase hacemos uso de del método **implements** el cual llama a la interface **TranporteAereo**.

En la interface **TransporteAereo** se llaman los métodos **despegar** y **aterrizar** provenientes de la clase **Viaje**.

Las interfaces actúan, por tanto, como tipos de clases. Se pueden declarar variables que pertenezcan al tipo de la interfaz esta es una lista de métodos donde solo están las firmas de los métodos, sin implementación, que define un comportamiento específico para un conjunto de objetos entonces estamos generando una relación entre clases de una relación de interfaz entre las respectivas clases.

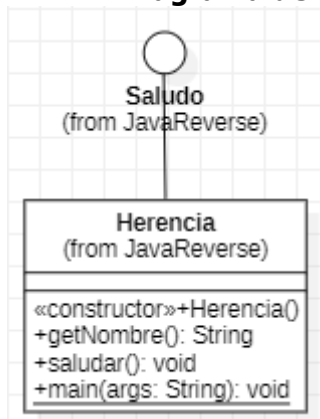
π

Enunciado #2

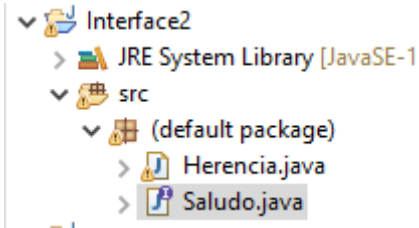
Creamos un programa llamado **Interface2**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Interface.

π

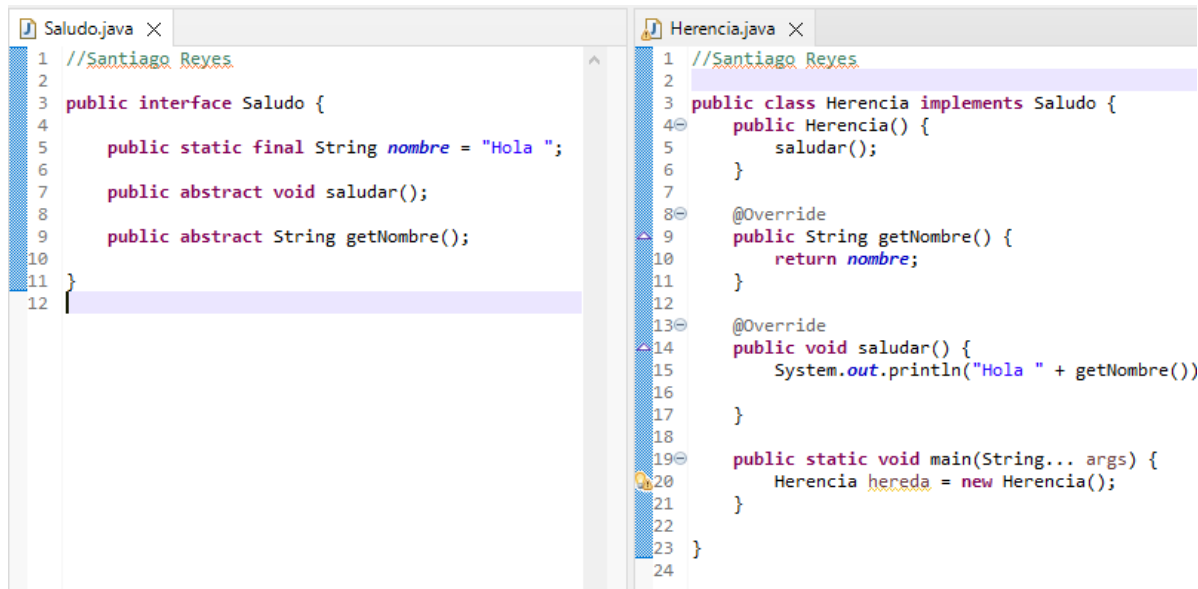
Diagrama de Clases



π Proyecto



π Código Fuente



π Análisis de resultados

Se creó un proyecto java llamado **Interface2**, en la cual se creó una clase llamada **Herencia**, y una interface llamada **Saludo**.

En la clase **Herencia** hacemos uso de del método **implements** el cual llama a la interface **Saludo**, luego se llama el método saludo que se hereda de la interface saludo para así poder ser definido y usado.

En la interface **Saludo** se declara el método **saludar**.

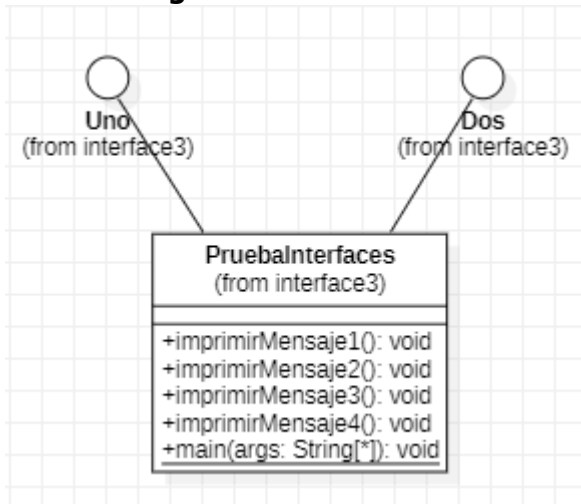
Las interfaces actúan, por tanto, como tipos de clases. Se pueden declarar variables que pertenezcan al tipo de la interfaz esta es una lista de métodos donde solo están las firmas de los métodos, sin implementación, que define un comportamiento específico para un conjunto de objetos entonces estamos generando una relación entre clases de una relación de interfaz entre las respectivas clases.

π Enunciado #3

Creamos un programa llamado **Interface3**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases deInterface.

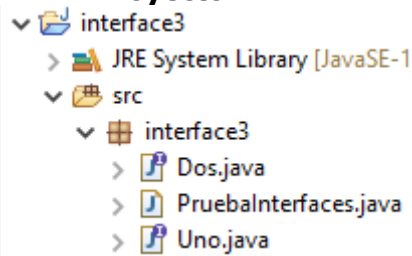
π

Diagrama de Clases



π

Proyecto



π

Código Fuente

Uno.java X	Dos.java X	PruebaInterfaces.java
1 //Santiago Reyes	1 //Santiago Reyes	
2	2	
3 package interface3;	3 package interface3;	
4	4	
5 public interface Uno {	5 public interface Dos {	
6	6	
7 public abstract void imprimirMensaje1();	7 public abstract void imprimirMensaje3();	
8	8	
9 public abstract void imprimirMensaje2();	9 public abstract void imprimirMensaje4();	
10 }	10	
11	11 }	
	12	

```
PruebaInterfaces.java X
1 //Santiago Reyes
2
3 package interface3;
4
5 public class PruebaInterfaces implements Uno, Dos {
6
7     @Override
8     public void imprimirMensaje1() {
9         System.out.println("vas");
10    }
11
12     @Override
13     public void imprimirMensaje2() {
14         System.out.println("a");
15    }
16
17     @Override
18     public void imprimirMensaje3() {
19         System.out.println("pasar");
20    }
21
22     @Override
23     public void imprimirMensaje4() {
24         System.out.println("el semestre :3");
25    }
26
27     public static void main(String[] args) {
28         PruebaInterfaces prueba = new PruebaInterfaces();
29         prueba.imprimirMensaje1();
30         prueba.imprimirMensaje2();
31         prueba.imprimirMensaje3();
32         prueba.imprimirMensaje4();
33    }
34 }
35
36
```

π Pruebas

```
Console X
<terminated> PruebaInterfaces
vas
a
pasar
el semestre :3
|
```

π Análisis de resultados

Se creó un proyecto java llamado **Interface2**, en la cual se creó una clase llamada **Herencia**, y una interface llamada **Saludo**.

En la clase **Herencia** hacemos uso de del método **implements** el cual llama a la interface **Saludo**, luego se llama el método saludo que se hereda de la interface saludo para así poder ser definido y usado.

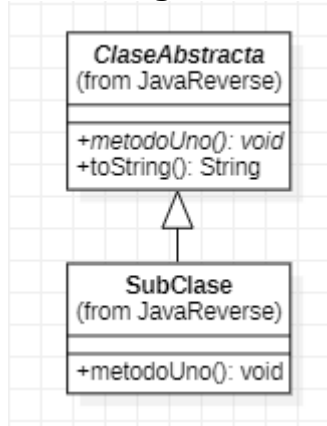
En la interface **Saludo** se declara el método **saludar**.

Las interfaces actúan, por tanto, como tipos de clases. Se pueden declarar variables que pertenezcan al tipo de la interfaz esta es una lista de métodos donde solo están las firmas de los métodos, sin implementación, que define un comportamiento específico para un conjunto de objetos entonces estamos generando una relación entre clases de una relación de interfaz entre las respectivas clases.

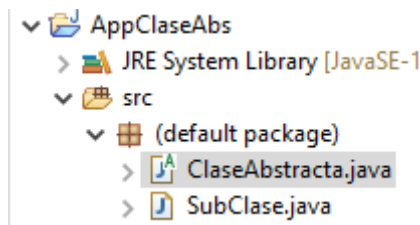
π Enunciado #1

Creamos un programa llamado **ClaseAbstracta**, haciendo uso de las aplicaciones Eclipse y StarUML teniendo en cuenta el uso de las buenas prácticas de la POO. Para aplicar el concepto de relaciones entre clases de Clase Abstracta.

π Diagrama de Clases



π Proyecto



π Código Fuente

```
SubClase.java
1 //Santiago Reyes
2
3 public class SubClase extends ClaseAbstracta{
4
5     public void metodoUno() {
6         System.out.println("método uno");
7     }
8
9 }
10

ClaseAbstracta.java
1 //Santiago Reyes
2
3 public abstract class ClaseAbstracta {
4
5     public abstract void metodoUno();
6     public String toString() {
7         return "Clase Abstracta";
8     }
9
10 }
11
```

π Análisis de resultados

Se creó un proyecto java llamado **AppClaseAbs**, en la cual se creó una clase abstracta llamada **ClaseAbstracta**, y una clase llamada **SubClase**.

En la clase **SubClase** hacemos uso de **implements** el cual llama a la clase abstracta **ClaseAbstracta**, luego se llama el método **metodoUno** que se hereda de la clase abstracta **ClaseAbstracta** para así poder ser definido y usado.

En la clase abstracta **ClaseAbstracta** se declara el método **metodoUno**, con la función de declarar la existencia de dicho método, pero no su implementación.

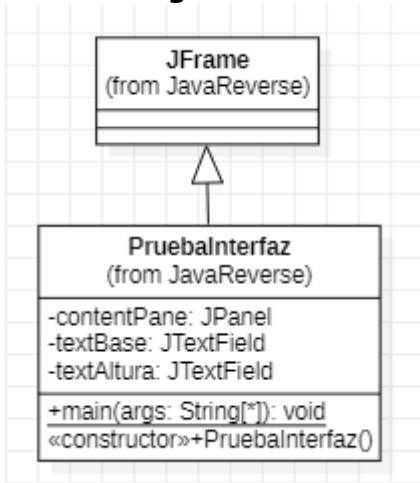
Las clases abstractas no representan algo específico y las podemos usar para crear otras clases. No pueden ser instanciadas, por lo tanto, no se pueden declarar nuevos objetos en ellas.

Tema – Interfaz gráfica.

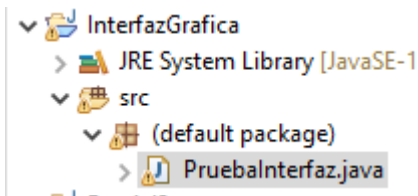
π Enunciado #1

Generar un programa que haga uso de la interfaz gráfica de Java haciendo uso de la aplicación Eclipse.

π Diagrama de Clases



π Proyecto



π Código Fuente

```
PruebaInterfaz.java X
1 //Santiago Steven Reyes Naranjo
2
3 import java.awt.EventQueue;
4
5 import javax.swing.JFrame;
6 import javax.swing.JPanel;
7 import javax.swing.border.EmptyBorder;
8 import javax.swing.JLabel;
9 import javax.swing.ImageIcon;
10 import javax.swing.SwingConstants;
11 import java.awt.Toolkit;
12 import java.awt.Font;
13 import javax.swing.JTextField;
14 import java.awt.Color;
15 import javax.swing.JButton;
16 import javax.swing.JMenuBar;
17 import javax.swing.JMenu;
18 import javax.swing.JMenuItem;
19 import javax.swing.JOptionPane;
20
21 import java.awt.event.ActionListener;
22 import java.awt.event.ActionEvent;
23
24 public class PruebaInterfaz extends JFrame {
25
26     private JPanel contentPane;
27     private JTextField textBase;
28     private JTextField textAltura;
```

```
PruebaInterfaz.java X
28 private JTextField textAltura;
29
30 /**
31  * Launch the application.
32  */
33 public static void main(String[] args) {
34     EventQueue.invokeLater(new Runnable() {
35         public void run() {
36             try {
37                 PruebaInterfaz frame = new PruebaInterfaz();
38                 frame.setVisible(true);
39                 frame.setLocationRelativeTo(null);
40             } catch (Exception e) {
41                 e.printStackTrace();
42             }
43         }
44     });
45 }
46
47 /**
48  * Create the frame.
49  */
50 public PruebaInterfaz() {
51     setIconImage(Toolkit.getDefaultToolkit().getImage("C:\\Users\\santi\\Downloads\\triangulo-5_xl.jpeg"));
52     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
53     setBounds(100, 100, 580, 461);
54     contentPane = new JPanel();
55     contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
56
57     setContentPane(contentPane);
58     contentPane.setLayout(null);
59
60     JMenuBar menuBar = new JMenuBar();
61     menuBar.setBounds(0, 0, 101, 22);
62     contentPane.add(menuBar);
63 }
```

```
PruebaInterfaz.java ×
61 menuBar.setBounds(0, 0, 101, 22);
62 contentPane.add(menuBar);
63
64 JMenu mnNewMenu = new JMenu("New menu");
65 menuBar.add(mnNewMenu);
66
67 JMenuItem mntmNewMenuItem = new JMenuItem("Salir");
68 mntmNewMenuItem.addActionListener(new ActionListener() {
69     public void actionPerformed(ActionEvent e) {
70
71         System.exit(0);
72     }
73 });
74 mnNewMenu.add(mntmNewMenuItem);
75
76 JButton btnCalcular = new JButton("Calcular");
77 btnCalcular.addActionListener(new ActionListener() {
78     public void actionPerformed(ActionEvent e) {
79
80         double base = Double.parseDouble(textBase.getText());
81         double altura = Double.parseDouble(textAltura.getText());
82
83         double area = 0.0;
84
85         area = base*altura/2;
86
87         JOptionPane.showMessageDialog(null, "El área del triángulo es: " + area);
88     }
89 });
90 btnCalcular.setForeground(Color.BLACK);
91 btnCalcular.setBackground(Color.GRAY);
92 btnCalcular.setBounds(396, 199, 89, 23);
93 contentPane.add(btnCalcular);
94
95
96
```



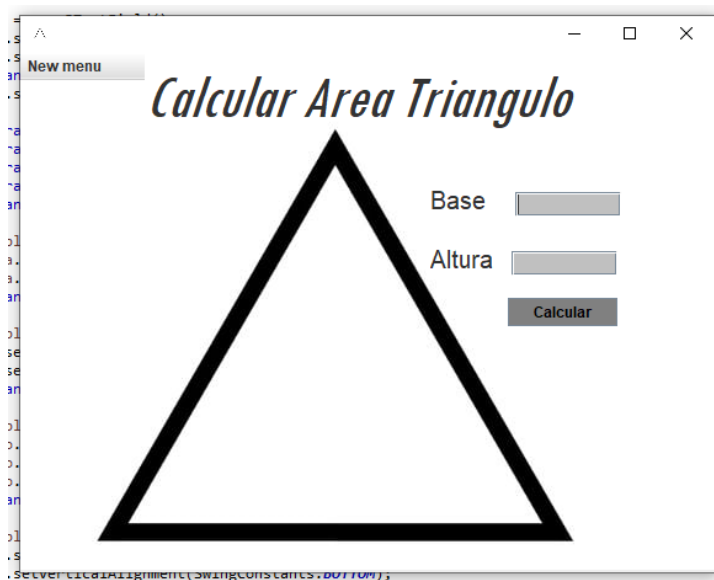
```

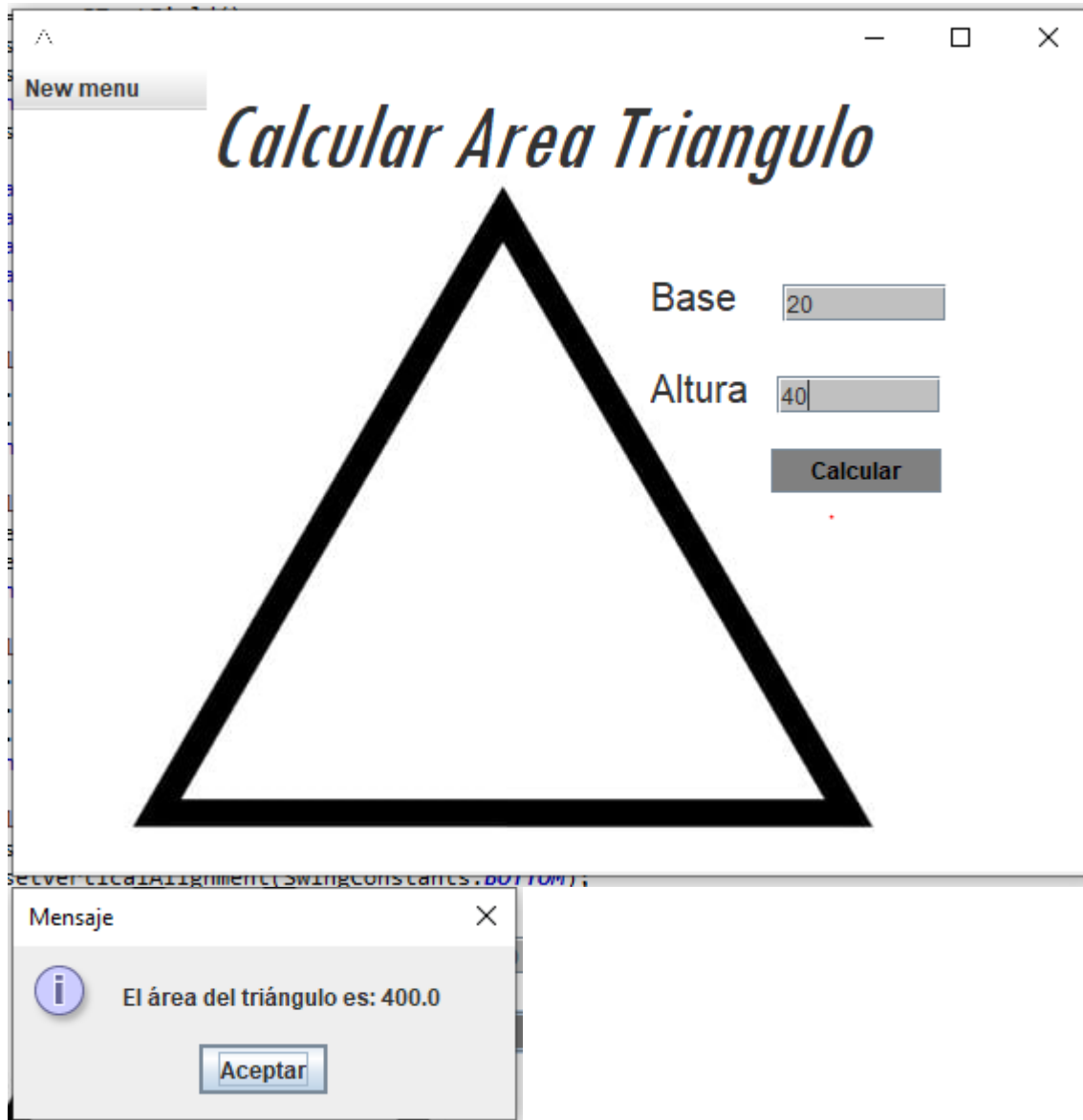
PruebaInterfaz.java x
95     contentPane.add(btnCalcular);
96
97     textBase = new JTextField();
98     textBase.setBackground(Color.LIGHT_GRAY);
99     textBase.setBounds(402, 113, 86, 20);
100    contentPane.add(textBase);
101    textBase.setColumns(10);
102
103    textAltura = new JTextField();
104    textAltura.setColumns(10);
105    textAltura.setBackground(Color.LIGHT_GRAY);
106    textAltura.setBounds(399, 161, 86, 20);
107    contentPane.add(textAltura);
108
109    JLabel lblAltura = new JLabel("Altura");
110    lblAltura.setFont(new Font("Arial", Font.PLAIN, 20));
111    lblAltura.setBounds(333, 154, 59, 29);
112    contentPane.add(lblAltura);
113
114    JLabel lblBase = new JLabel("Base");
115    lblBase.setFont(new Font("Arial", Font.PLAIN, 20));
116    lblBase.setBounds(333, 97, 59, 46);
117    contentPane.add(lblBase);
118
119    JLabel lblTitulo = new JLabel("Calcular Area Triangulo");
120    lblTitulo.setHorizontalAlignment(SwingConstants.CENTER);
121    lblTitulo.setBounds(62, 0, 423, 75);
122    lblTitulo.setFont(new Font("Tw Cen MT Condensed", Font.ITALIC, 50));
123    contentPane.add(lblTitulo);
124
125    JLabel lblFondo = new JLabel("New label");
126    lblFondo.setBounds(-54, 0, 670, 420);
127    lblFondo.setVerticalAlignment(SwingConstants.BOTTOM);
128    lblFondo.setIcon(new ImageIcon("C:\\Users\\santi\\Downloads\\triangulo-5_x1.jpeg"));
129    contentPane.add(lblFondo);
130 }
131 }

```

π

Pruebas





π Referencias bibliográficas

- Buenas prácticas de programación orientada a objetos en JAVA, Capítulo 8 al Capítulo 13.