UNIVERSIDAD
Popular del cesar

Ingeniería de Sistemas

Programación de Computadores II

UNIDAD 2:

PROGRAMACION BASADA EN OBJETOS

2.1 Clases y Objetos (Estado, comportamiento e identidad)
2.2 Abstracción de objetos – Creación de Clases
2.3 Visibilidad de Clases y Objetos
2.3.1 Vista pública de las Clases
2.3.2 Vista privada de las Clase
2.4 Atributos y Métodos
2.4.1 Miembros de Instancia y de Clase
2.4.2 Método constructor
2.4.3 Métodos Getter y Setters (encapsulamiento)
3.4.4 Sobrecarga de métodos
2.5 Relaciones entre clases
2.5.1 Relación de asociación
2.5.2 Relación de agregación
2.5.3 Relación de composición

UNIDAD 2:

OBJETIVOS DE APRENDIZAJE

- Ofrecer a los estudiantes una primera aproximación a la tecnología de objetos en Java
- Presentar a los estudiantes las características básicas de una clase, su estructura básica y su implementación en Java
- Presentar a los estudiantes los conceptos de constructor, métodos getter y setter, y su función dentro de una clase.
- Presentar a los estudiantes los conceptos de variables de instancia y variables de clases
- Presentar a los estudiantes el concepto de sobrecarga de métodos y su implementación en Java
- Que el estudiante reconozca el significado de asociación, agregación y composición entre clases, su implementación en UML y en código Java.

Clases y Objetos: (Estado, comportamiento e identidad)

UNIVERSIDAD Popular del cesar

Bicicleta
marca : **String** Talla: **char** materialBase : **String** noSerial: **int**
cambiarPlato(): **void** cambiarPacha(): **void** frenar() : **void**

Una **clase** es una plantilla que define las variables y los métodos que son comunes para todos los objetos de un cierto tipo.

Objeto1: Bicicleta
marca = “GW” talla = ‘L’ materialBase=“Carbono” noSerial=12345
cambiarPlato(): void cambiarPacha() : void frenar(): void

Objeto2: Bicicleta
marca = “Venzo” talla = ‘XS’ materialBase=“Acero” noSerial=31122
Cambiar Plato(): void Cambiar Pacha(): void Frenar(): void

Objeto 3:Bicicleta
marca = “Optimus” talla = ‘M’ materialBase=“Acero” noSerial=2020
cambiarPlato(): void cambiarPacha(): void Frenar(): void

Los **objetos** son ejemplares de una clase
Instancias de una clase

Clases y Objetos: (Estado, comportamiento e identidad)

UNIVERSIDAD Popular del cesar

Un **objeto** es cada ejemplar construido de una clase, es decir, una instancia de clase

Propiedades

Identidad

Permite diferenciar los objetos de modo no ambiguo independientemente de su estado. ***Cada objeto posee su propia identidad de manera implícita, puesto que ocupa su propia posición en la memoria de la computadora.***

Comportamiento

Conjunto de operaciones que se pueden realizar sobre un objeto.

Estado

El estado de un objeto viene determinado por los valores que toman sus datos o atributos en un instante de tiempo.

Clases y Objetos: (Estado, comportamiento e identidad)

UNIVERSIDAD Popular del cesar

Rectangulo
private double base private double altura
public rectangulo( ) public double getBase() public double getAltura() public void setBase() public void setAltura() public double calcularArea() public double calcularPerimetro()

Rectangulo a = **new** Rectangulo (5,8);

Rectangulo b = **new** Rectangulo (5, 8);

Rectangulo c = **new** Rectangulo (8, 6);

Rectangulo d = **new** Rectangulo (10,10);

Rectangulo e = **new** Rectangulo (10,8);

MEMORIA

e: Rectángulo
base = 10
altura = 8

a: Rectángulo
base = 5
altura = 8

b: Rectángulo
base = 5
altura = 8

c: Rectángulo
base = 8
altura = 6

d: Rectángulo
base = 10
altura = 10

!= identidad
== estado
== comportamiento

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Estructura de una clase)

Se desea calcular el área de un rectángulo ?

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

Nombre de la clase

Atributos
Variables de instancia o de clase

Métodos de la clase

Representación grafica de una clase en UML

Estructura de una clase

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Implementación de una clase)

Se desea calcular el área de un rectángulo ?

Implementación en Java

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

```
7   public class Rectangulo {
8       private double base;
9       private double altura;
10
11      public Rectangulo() {
12          this.base=0;
13          this.altura=0;
14      }
15
16      public double getBase(){ return base; }
17      public void setBase(double base){ this.base = base; }
18      public double getAltura(){ return altura; }
19      public void setAltura(double altura){this.altura = altura;}
20
21
22      public void calcularArea(){
23          System.out.printf("Area=%.2f%n",this.base * this.altura);
24      }
25      public void calcularPerimetro(){
26         System.out.printf("Perimetro=%.2f%n",2 * (this.base +this.altura));
27      }
28
29  }
```

Definición de la clase

Atributos

Método Constructor

Getter y Setter (encapsulamiento)

Métodos miembros

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Clases y objetos)

Se desea calcular el área de un rectángulo ?

r: Rectángulo

base = ~~0~~ 10
altura = ~~0~~ 5

Creación de objetos en Java

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

```
4   public class PruebaRectangulo {
5
6       public static void main(String[] arg){
7
8           Rectangulo r = new Rectangulo();
9           r.setBase(10);
10          r.setAltura(5);
11          System.out.println("Area: " + r.calcularArea());
12          System.out.println("Perimetro: " + r.calcularPerimetro());
13
14      }
15
16  }
```

Creación del Objeto

Asignación de nuevos valores a atributos

Impresión de área y perímetro

Calcula el valor del área y la retorna

Calcula el valor del perímetro y lo retorna

https://github.com/jfseoanesl/jfseoanesl-Unicesar-Programacion-II-Unidad-1.-Simples-classes-and-objects.git

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Estructura de una clase)

Estructura e implementación de una clase en Java

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

Modificador de acceso

Nombre de la clase

```
public class Rectangulo  { //cabecera
 // implementación
        Atributos
        Constructores
        Métodos
}
```

public y **class** son palabras reservadas de Java

Rectangulo es el nombre dado a la clase, debe ser en singular e iniciar en mayúscula.

Por norma el archivo que contiene la clase se debe llamar igual que la clase.

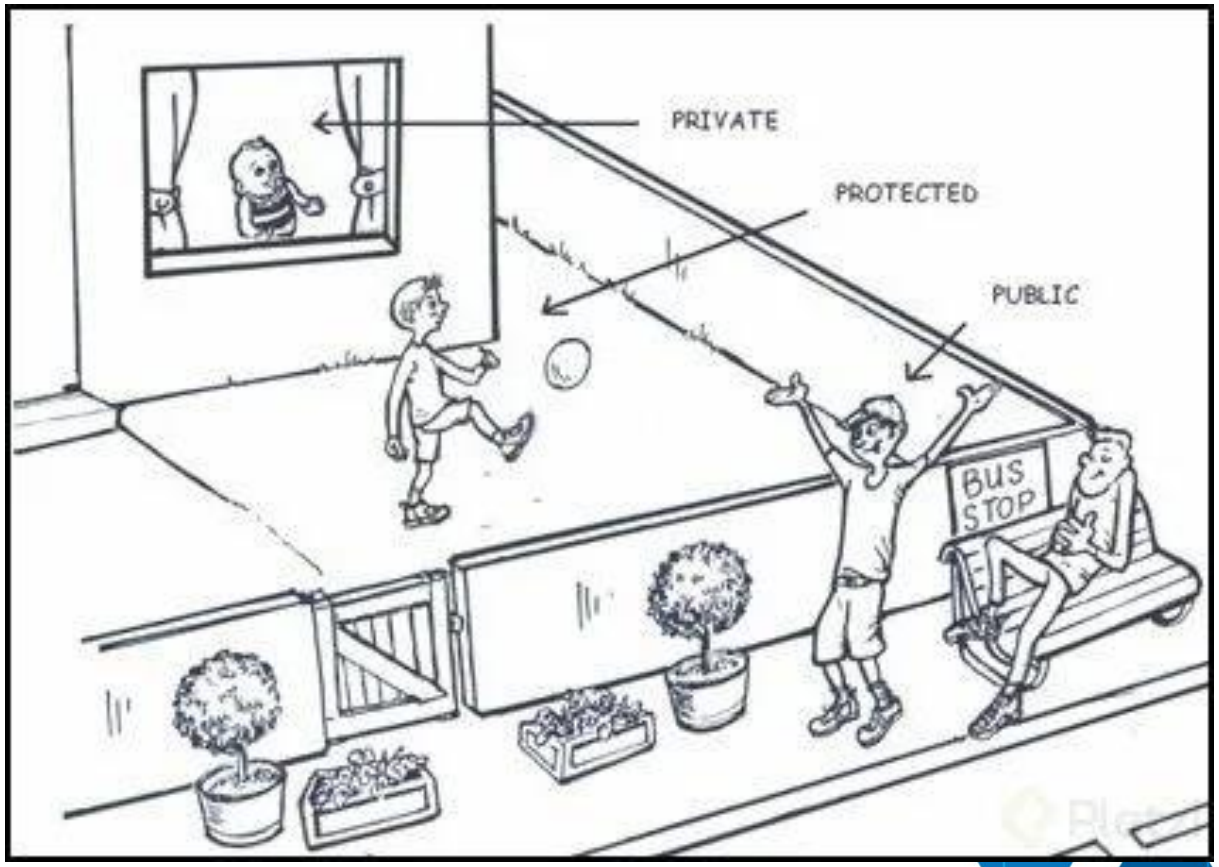
Nombre clase : Rectangulo
Nombre archivo: Rectángulo.java

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Visibilidad de clases y objetos)

Modificadores de acceso en Java

Modificador	Clase	Package	Subclase	Otros
public	✔	✔	✔	✔
protected	✔	✔	✔	•
default	✔	✔	•	•
private	✔	•	•	•



Cuando no se define modificador de acceso en la definición de clase, método o atributo, se asume como **Default**

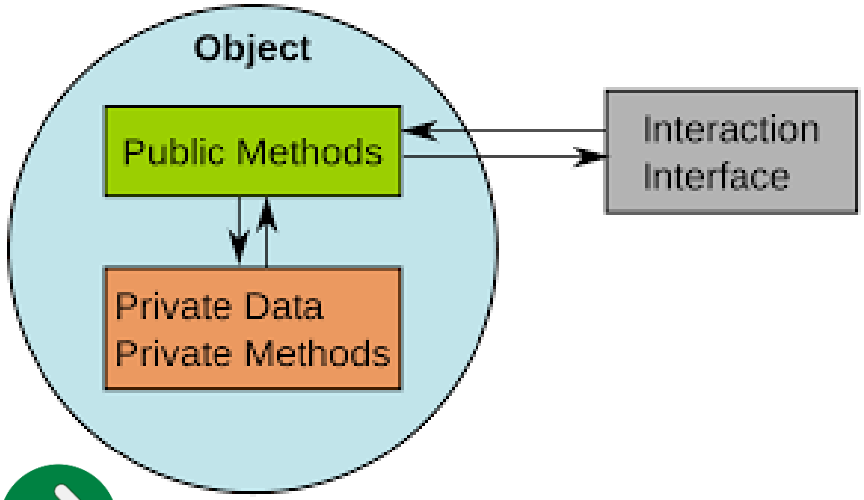
UNIVERSIDAD Popular del cesar

Clases y Objetos: (Visibilidad de clases y objetos)

Modificadores de acceso en Java

Por lo general las variables o atributos son del tipo **private**, mientras que los métodos son tipo **public**

Encapsulamiento



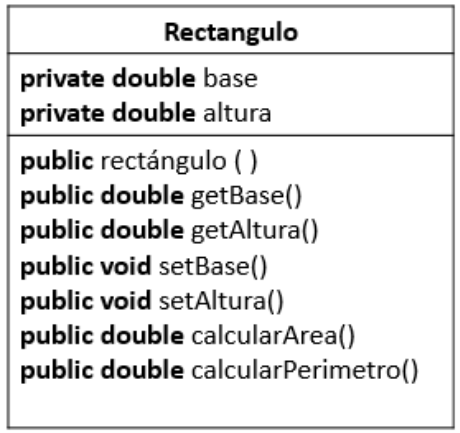

```
7   public class Rectangulo {
8       private double base;
9       private double altura;
10
11      public Rectangulo() {
12          this.base=0;
13          this.altura=0;
14      }
15
16      public double getBase(){ return base; }
17      public void setBase(double base){ this.base = base; }
18      public double getAltura(){ return altura; }
19      public void setAltura(double altura){this.altura = altur
20
21
22      public void calcularArea(){
23          System.out.printf("Area=%.2f%n",this.base * this.alt
24      }
25      public void calcularPerimetro(){
26         System.out.printf("Perimetro=%.2f%n",2 * (this.base +
27      }
28
29  }
```

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y Métodos)

Variables de instancia y de clase

Diseño UML




```
public class Rectangulo  {
 // implementación
     //Variables de instancia
    private double base;
    private double altura;
    //Variables de clases
    public static int color = 1;


        //Métodos omitidos
}
```

Atributos de instancia

- Variables o métodos que se almacenan en los objetos.
- Se requiere la creación de un objeto para acceder a este tipo de atributo o métodos.

Atributos de clase

- Son campos que se almacenan en la clase y no en los objetos.
- Común a todas la instancias de la clase.
- Solo se inicializan una vez.
- Incluyen en su definición la palabra **static**
- Se acceden a través del nombre de la clase y no de los objetos

UNIVERSIDAD Popular del cesar

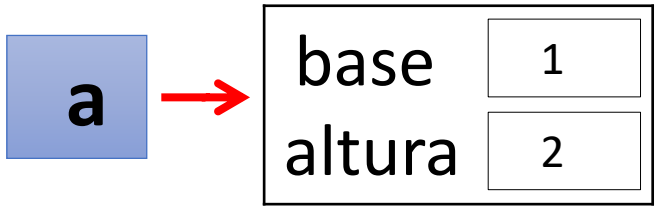
Clases y Objetos: (Atributos y Métodos)

Variables de instancia y de clase

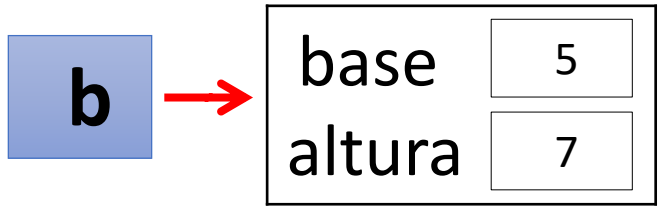
Cada vez que se crea un objeto, **se reserva espacio en memoria para cada una de sus variables de instancia**.

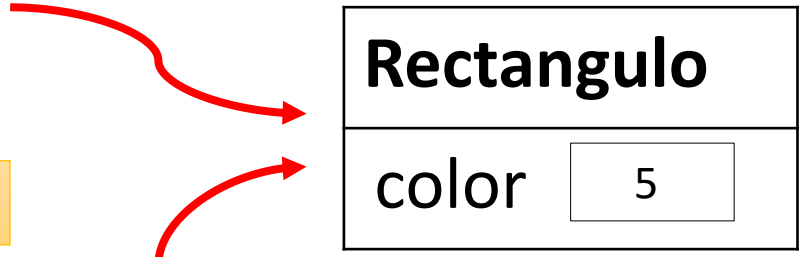
Las variables de clase **implican una sola zona de memoria reservada para todas las instancias de la clase**, y no una copia por objeto, como sucede con las variables de instancia.

Rectangulo a =**new** Rectangulo ( );



Rectangulo b =**new** Rectangulo ( );





Objeto.nombreVariable;
Objeto.nombreMetodo();

a.calcularArea( );

Variables/métodos de instancia

NombreClase.nombreVariable;
NombreClase.nombreMetodo();

Rectangulo.color;

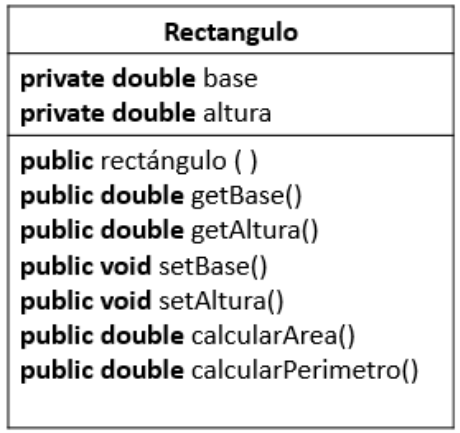
Variables/métodos de clase

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y Métodos)

Método Constructor

Diseño UML




```
public class Rectangulo  {
 // implementación
        //Atributos omitidos
        //constructor por defecto
        public Rectangulo ( ) {
            // inicializa atributos
        }
        //Métodos omitidos
}
```

El **constructor** es el responsable de inicializar un objeto cuando éste es creado

Tiene el mismo nombre de la clase que lo define.

Debe ser publico y no define valor de retorno

Algunos lenguajes implementan metodos **destructores** para destruir objetos y liberar memoria.

En Java no hay destructores, la liberación de memoria es llevada acabo por el **Garbage Collector**.

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y métodos)

Método Constructor

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

```
public Rectangulo() {
     this.base=2;
     this.altura=5;
}
```

Rectangulo a = **new** Rectangulo ( );

a : Rectángulo
base = 2 altura = 5

```
public Rectangulo() {
     this.base=10;
     this.altura=3;
}
```

Rectangulo a = **new** Rectangulo ( );

a : Rectángulo
base = 10 altura = 3

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y métodos)

Creación de objetos – Instanciar clases

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

Para instancia una clase (crear un objeto de un tipo de clase) se utiliza la palabra reservada **new** acompañada de un método **constructor.**

Clase identificadoObjeto = **new ConstructorClase** ( );

```
public class Rectangulo {
 // implementación
}
```

```
6   public class ClasePrincipal {
7
8       public static void main(String [] arg){
9
10          Rectangulo r1;
11          r1 = new Rectangulo();
12
13          Rectangulo r = new Rectangulo();
14
15      }
16
17  }
```

Creación del objeto

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y métodos)

Métodos Getter y Setter

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

```
public class Rectangulo
{
    private double base;
    private double altura;
    //constructores omitidos

}
```

Los métodos **Getter** devuelven información sobre el estado de un objeto.

```
// getter de la base
public double getBase(){
    return this.base;
}

// getter de la altura
public double getAltura(){
    return this.altura;
}
```

Los métodos **Setter** cambian o modifican el estado de un objeto

```
// setter de la base
public void setBase(double b){
    this.base=b;
}

// setter de la altura
public void setAltura(double a){
    this.altura = a;
}
```

this es una palabra reservada que hace referencia al objeto actual

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y métodos)

Métodos Getter y Setter

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

Obtener el valor de los atributos

Establecer valores a los atributos

```
7   public class Rectangulo {
8       private double base;
9       private double altura;
10
11      public Rectangulo() {
12          this.base=0;
13          this.altura=0;
14      }
15
16      public double getBase(){ return base; }
17      public void setBase(double base){ this.base = base; }
18      public double getAltura(){ return altura; }
19      public void setAltura(double altura){this.altura = altur
20
21
22      public void calcularArea(){
23          System.out.printf("Area=%.2f%n",this.base * this.alt
24      }
25      public void calcularPerimetro(){
26         System.out.printf("Perimetro=%.2f%n",2 * (this.base +
27      }
28
29  }
```

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y métodos)

Métodos Getter y Setter

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

```
4   public class PruebaRectangulo {
5
6       public static void main(String[] arg){
7
8           Rectangulo r = new Rectangulo();
9           r.setBase(10);
10          r.setAltura(5);
11          System.out.printf("Para Base: %.2f%n", r.getBase());
12          System.out.printf("Para Altura: %.2f%n", r.getAltura());
13          System.out.println("Area: " + r.calcularArea());
14          System.out.println("Perimetro: " + r.calcularPerimetro());
15
16      }
17
18  }
```

Setter

Getter

```
Output - ClasesYObjetos (run)
run:
Para Base: 10,00
Para Altura: 5,00
Area: 50.0
Perimetro: 30.0
BUILD SUCCESSFUL (total time: 2 seconds)
```

UNIVERSIDAD Popular del cesar

Clases y Objetos: (Atributos y métodos)

Métodos Miembros

Diseño UML

Rectangulo
private double base **private double** altura
public rectángulo ( ) **public double** getBase() **public double** getAltura() **public void** setBase() **public void** setAltura() **public double** calcularArea() **public double** calcularPerimetro()

Todo método tiene dos partes: **una cabecera y un cuerpo**

```
7   public class Rectangulo {
8       private double base;
9       private double altura;
10
11      public Rectangulo() {
12          this.base=0;
13          this.altura=0;
14      }
15
16      public double getBase(){ return base; }
17      public void setBase(double base){ this.base = base; }
18      public double getAltura(){ return altura; }
19      public void setAltura(double altura){this.altura = altur
20
21
22      public void calcularArea(){
23          System.out.printf("Area=%.2f%n",this.base * this.alt
24      }
25      public void calcularPerimetro(){
26         System.out.printf("Perimetro=%.2f%n",2 * (this.base +
27      }
28
29  }
```

Los métodos miembros definen el comportamiento asociado y característicos de los objetos de un tipo de clase.

Métodos miembros

Clases y Objetos: (Atributos y métodos)

Ejercicios Prácticos

Implementar en Java las Clases diseñadas para resolver los siguientes problemas propuestos:

Crear una clase que permita calcular la nota final de un estudiante de la asignatura POO de la UPC. Cree varios objetos estudiantes y calcule su definitiva.

Estudiante
- double nota1, nota2, nota3 + static final double P1=0.3 + static final double P2=0.3 + static final double P3=0.4
+ Estudiante() + double getNota1() + double getNota2() + double getNota3() + void setNota1(double n1) + void setNota2(double n2) + void setNota3(double n3) + double calcularDefinitiva()

Crear una clase que permita calcular el sueldo de un trabajador según sus horas trabajadas, se conoce que todo trabajador maneja una misma tarifa por hora de trabajo, equivalente a $250.00. Cree dos trabajadores y calcule su sueldo, posteriormente modifique la tarifa, y recalcule el sueldo de los dos trabajadores.

Trabajador
+ static final double TARIFA_HORA=250 - double horasTrabajadas
+ Trabajador() + double getHorasTrabajadas() + void setHorasTrabajadas(double h) + double calcularSueldo()

Clases y Objetos: (Atributos y métodos)

Ejercicios Prácticos

Diseñar e implementar en Java las Clases necesarias para resolver los siguientes problemas propuestos:

Se solicita crear un programa en java que permita calcular la superficie de una habitación a partir de las medidas de su longitud y anchura.

Se solicita implementar la clase Habitación, así mismo, se solicita implementar una clase principal en la que deberás instanciar varios objetos de tipo habitación y calcular e imprimir su superficie.

Clases y Objetos: (Sobrecarga de Métodos)

UNIVERSIDAD Popular del cesar

Pueden declararse métodos con el mismo nombre en la misma clase, siempre y cuando tengan distintos conjuntos de parámetros (que se determinan con base en el número, tipos y orden de los parámetros). A esto se le conoce como sobrecarga de métodos.

```
91      public void mostrar(){
92
93          System.out.println("Nombre: " + this.getpNombre());
94          System.out.println("Apellido: " + this.getpApellido());
95          System.out.println("Estado Civil: " + this.getEstadoCivil());
96          System.out.println("Edad: "+ this.getEdad());
97          System.out.println("Ocupacion: " + this.getOcupacion());
98
99      }
```

Firma 1

```
101     public void mostrar(int n){
102
103         if(n==1)
104             System.out.println("Nombre: " + this.getpNombre());
105     }
```

Firma 2

```
107     public void mostrar(int n,int a){
108
109         if(n==1)
110             System.out.println("Nombre: " + this.getpNombre());
111         if(a==1)
112             System.out.println("Apellido: " + this.getpApellido());
113     }
114 }
```

Firma 3

Cuando se hace el llamado a un método sobrecargado, **el compilador de Java selecciona el método adecuado con base en el número y tipo de argumentos que tiene el método** y no por el tipo que devuelve.

mostrar (1, 1**);**

mostrar ( );

mostrar (1**);**

Clases y Objetos: (Sobrecarga de Métodos)

UNIVERSIDAD Popular del cesar

También existe la **sobrecarga de constructores**: Cuando en una clase existen constructores múltiples, se dice que hay sobrecarga de constructores.

```
19      public Rectangulo(){
20          this(0,0);
21      }
22
23      /**
24       * Constructores sobrecargados, base variable y altura fija
25       * @param b de tipo double
26       */
27      public Rectangulo(double b){
28         this(b, 10);
29      }
30
31      /**
32       * Constructores sobrecargados, base variable y altura fija
33       * @param b de tipo double para base
34       * @param a de tipo double para altura
35       */
36      public Rectangulo(double b, double a){
37          this.altura=a;
38          this.base=b;
39      }
```

Firma 1

Firma 2

Firma 3

Los constructores sobrecargados ofrecen diferentes alternativas para crear los objetos de una clase.

Permiten inicializar los objetos con valores deseados.

Rectangulo r = new Rectangulo( );

Rectangulo r = new Rectangulo(5);

Rectangulo r = new Rectangulo(5,5);

Clases y Objetos: (Sobrecarga de Métodos)

UNIVERSIDAD Popular del cesar

También existe la **sobrecarga de constructores**: Cuando en una clase existen constructores múltiples, se dice que hay sobrecarga de constructores.

```
19      public Rectangulo(){
20          this(0,0);
21      }
22
23      /**
24       * Constructores sobrecargados, base variable y altura fija
25       * @param b de tipo double
26       */
27      public Rectangulo(double b){
28         this(b, 10);
29      }
30
31      /**
32       * Constructores sobrecargados, base variable y altura fija
33       * @param b de tipo double para base
34       * @param a de tipo double para altura
35       */
36      public Rectangulo(double b, double a){
37          this.altura=a;
38          this.base=b;
39      }
```

Firma 1

Firma 2

Firma 3

```
4   public class PruebaRectangulo {
5
6       public static void main(String[] arg){
7
8           Rectangulo r = new Rectangulo();
9           r.setBase(10);
10          r.setAltura(5);
11
12          Rectangulo r2 = new Rectangulo(5);
13
14          Rectangulo r3 = new Rectangulo(5,5);
15
16          System.out.printf("Para Base: %.2f%n", r.getBase());
17          System.out.printf("Para Altura: %.2f%n", r.getAltura());
18          System.out.println("Area: " + r.calcularArea());
19          System.out.println("Perimetro: " + r.calcularPerimetro());
20
21      }
22
23  }
```

Clases y Objetos: (Sobrecarga de Métodos)

UNIVERSIDAD Popular del cesar

Ejercicio

Distancia entre puntos

Crea la clase **Punto**. De un punto se tienen que saber sus coordenadas x e y. Define la clase y los métodos habituales.

Así mismo, se solicita implementar un método que permita calcular la distancia hacia otro punto, este método podrá recibir como parámetro un objeto punto o sus coordenadas.

En la clase principal, crea dos puntos y calcula la distancia entre ellos.

Venta de pizzas (*)

Crea la clase Pizza con los atributos y métodos necesarios. Sobre cada pizza se necesita saber el tamaño – “mediana” o “familiar” - el tipo – “margarita”, “cuatro quesos” o “funghi” - y su estado – “pedida” o “servida”. La clase debe almacenar información sobre el número total de pizzas que se han pedido y que se han servido.

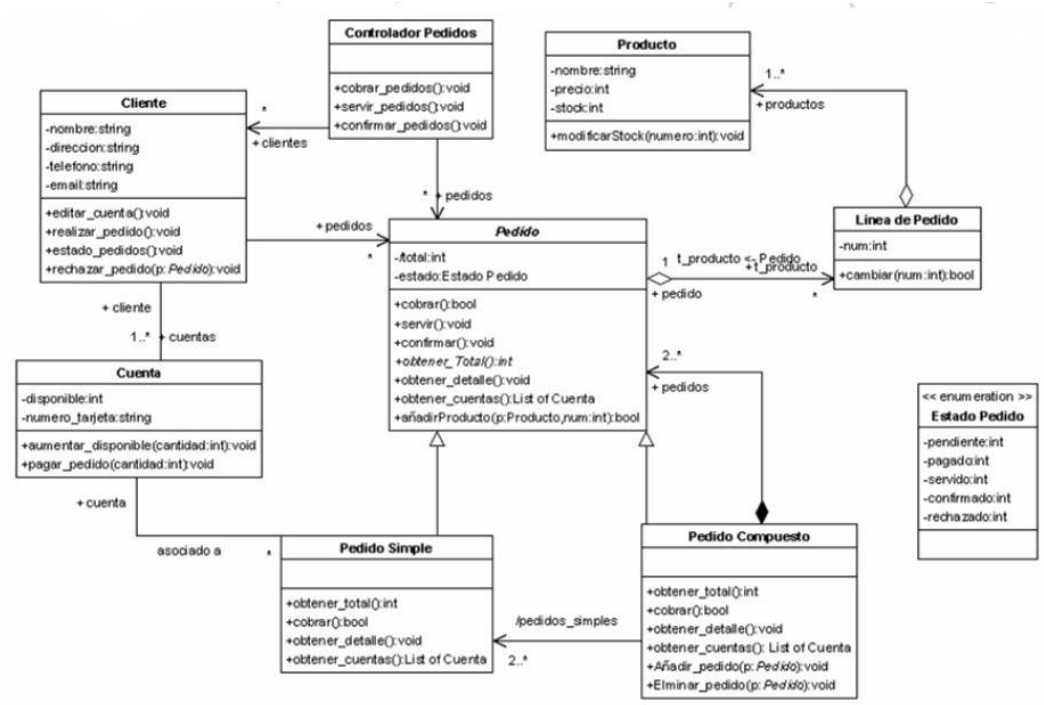
Siempre que se crea una pizza nueva, su estado es “pedida”. Se debe implementar un método servir(), que cambiara su estado a servida, cuando esta es servida.

En una clase principal se solicita crear varias pizzas, y servir algunas de ellas. Imprimir un informe con el total de pizzas pedidas y pizzas servidas.

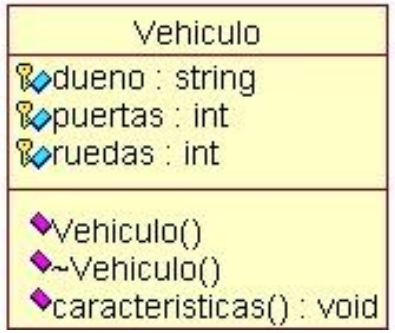
UNIVERSIDAD Popular del cesar

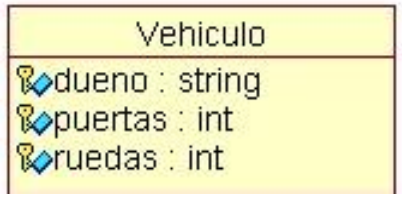
Diagramas de Clases

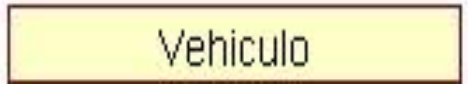
un **diagrama de clases** en UML es un tipo de diagrama de estructura estática que describe la estructura de un sistema mostrando las clases del sistema, sus atributos, métodos, sus relaciones, y las interfaces.



Elementos







Clases

Relaciones entre clases

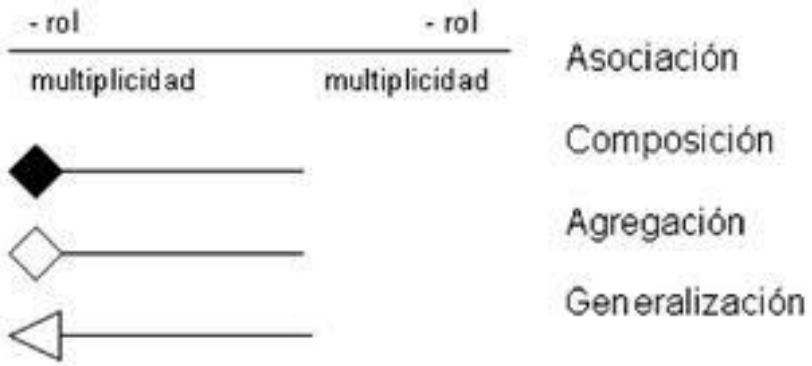


Figura 5. Tipos de relaciones. Notación UML

UNIFIED MODELING LANGUAGE™

Relaciones entre Clases – Asociación

La **asociación** es una relación entre Clases que indica que una instancia de una Clase conoce de la instancia de otra Clase, los objetos de una clase contienen referencias a los objetos de otra clase para acceder a sus atributos o para utilizar las operaciones que ellos ofrecen.

La clases Estudiante y Universidad están relacionadas con la asociación ***estudia en*** que describe que "***un estudiante estudia en la universidad***"

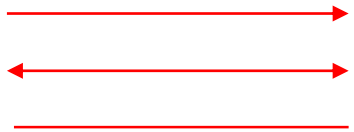
La notación que describe la asociación entre dos clases es **una línea.**

Nombre de la asociación, ofrece una descripción de la relación entre las clases

Cardinalidad de una asociación especifica cuantas instancias de una clases se pueden asociar a una sola instancia de otra clase

Uno a Uno (1 … 1)
Uno a Muchos (1 .. *)
Muchos a Muchos (* .. *)

Navegabilidad, indica que instancias de una clase tienen acceso a instancias de otra clase.



Navegabilidad en un solo sentido

Navegabilidad en ambos sentidos

Alias, especifican con mas detalles la relación

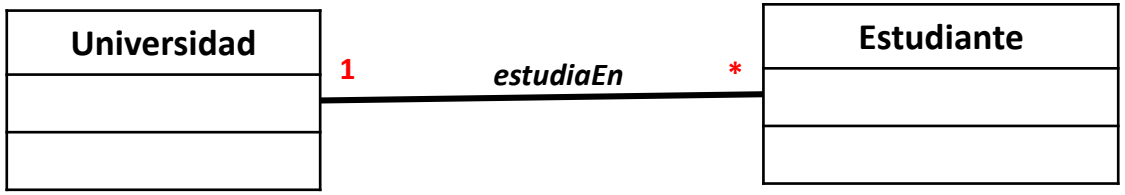
UNIVERSIDAD Popular del cesar

Asociaciones entre Clases – Cardinalidad

Uno a Uno. Dos objetos se relacionan de forma exclusiva, una instancia de un objeto se asocia con solo una instancia de otro objeto.

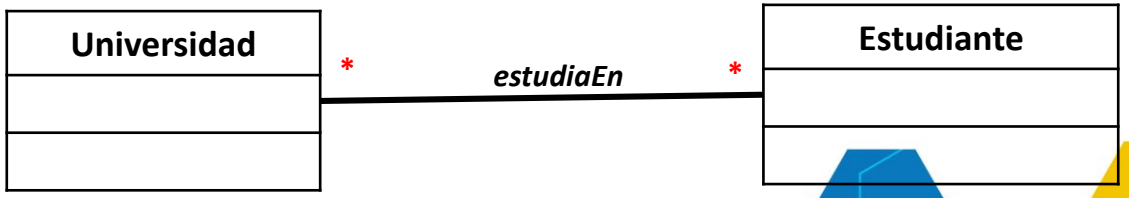
"Un Rector dirige una Universidad y una Universidad es dirigida por un solo Rector"

Uno a Muchos. Un objeto de una clase puede estar asociado a muchos objetos de otra clase.



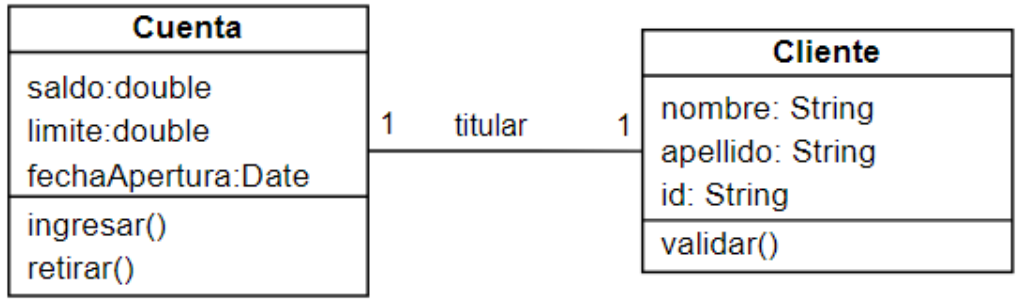
"En una universidad pueden estudiar muchos estudiantes, y cada estudiantes estudia en una sola Universidad"

Muchos a Muchos. Donde cada objeto de cada clase puede estar asociado a muchos objetos de la otra clase.

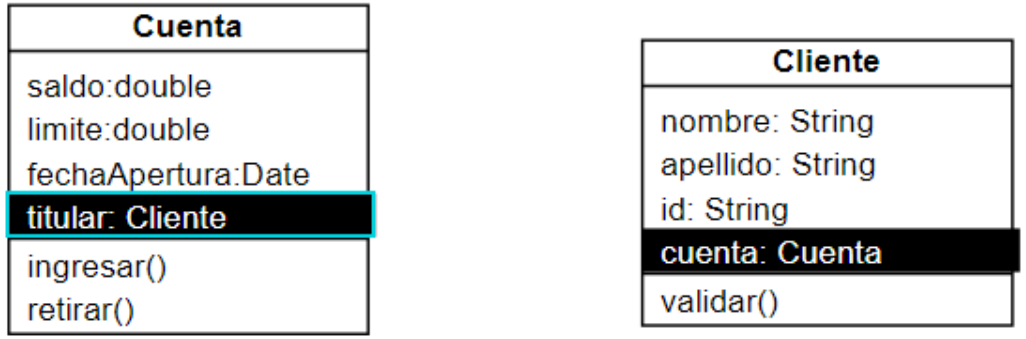


"En una Universidad pueden estudiar muchos estudiantes y un estudiante puede estudiar en Muchas Universidades"

Asociaciones entre Clases – Cardinalidad



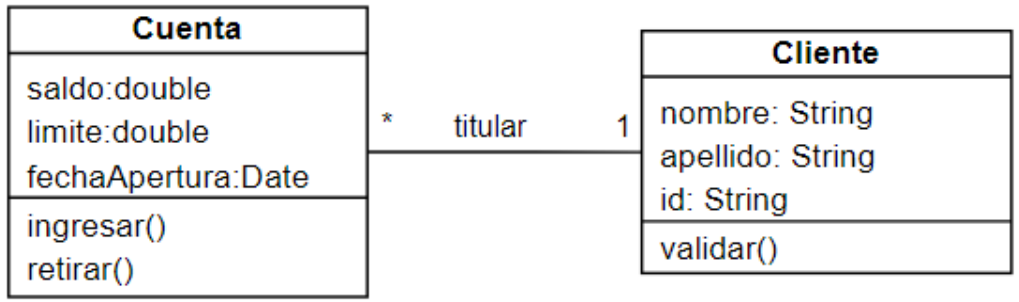
Implementación



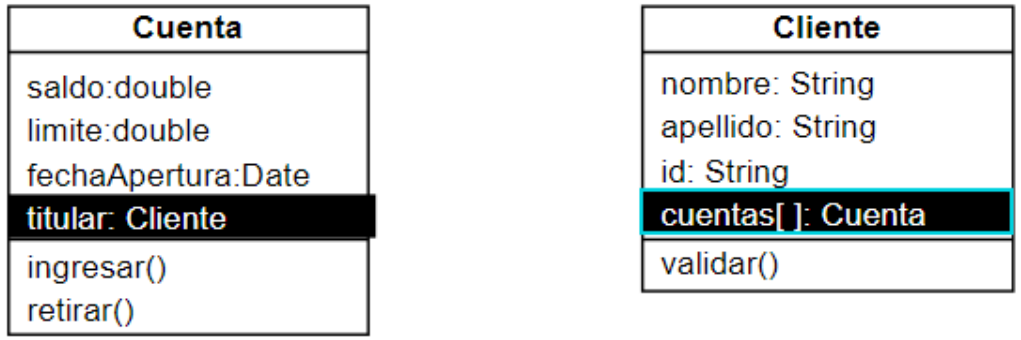

```java
public class Cuenta {

    private double saldo;
    private double limite;
    private Date fechaApertura;
    private Cliente titular;

    public Cuenta(){ }

    // metodos adicionales
}
```

```java
public class Cliente {

    private String nombre;
    private String apellido;
    private String id;
    private Cuenta cuenta;

    public Cliente(){ }

    // metodos adicionales
}
```

Asociaciones entre Clases – Cardinalidad



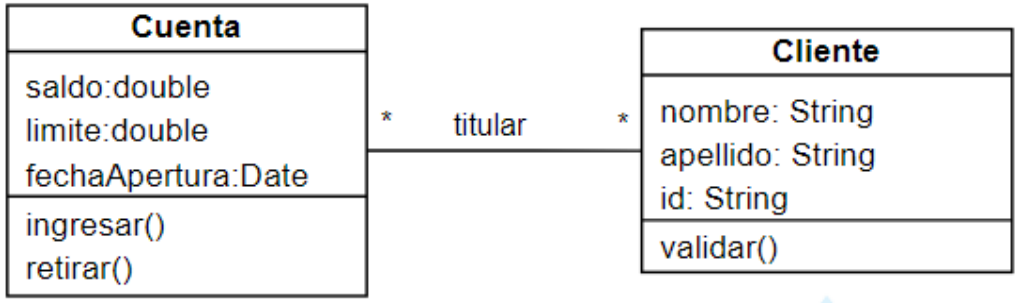
Implementación



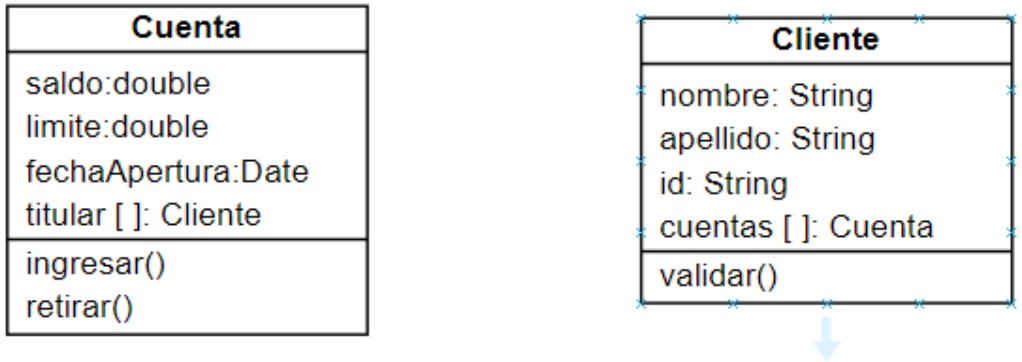

```java
public class Cuenta {

    private double saldo;
    private double limite;
    private Date fechaApertura;
    private Cliente titular;

    public Cuenta(){ }

    // metodos adicionales
}
```

```java
public class Cliente {

    private String nombre;
    private String apellido;
    private String id;
    private Cuenta cuentas[ ];

    public Cliente(){ }

    // metodos adicionales
}
```

Asociaciones entre Clases – Cardinalidad



Implementación




```java
public class Cuenta {

    private double saldo;
    private double limite;
    private Date fechaApertura;
    private Cliente titular[];

    public Cuenta(){ }

    // metodos adicionales
}

public class Cliente {

    private String nombre;
    private String apellido;
    private String id;
    private Cuenta cuentas[ ];

    public Cliente(){ }

    // metodos adicionales
}
```

Relaciones entre Clases – Ejercicios

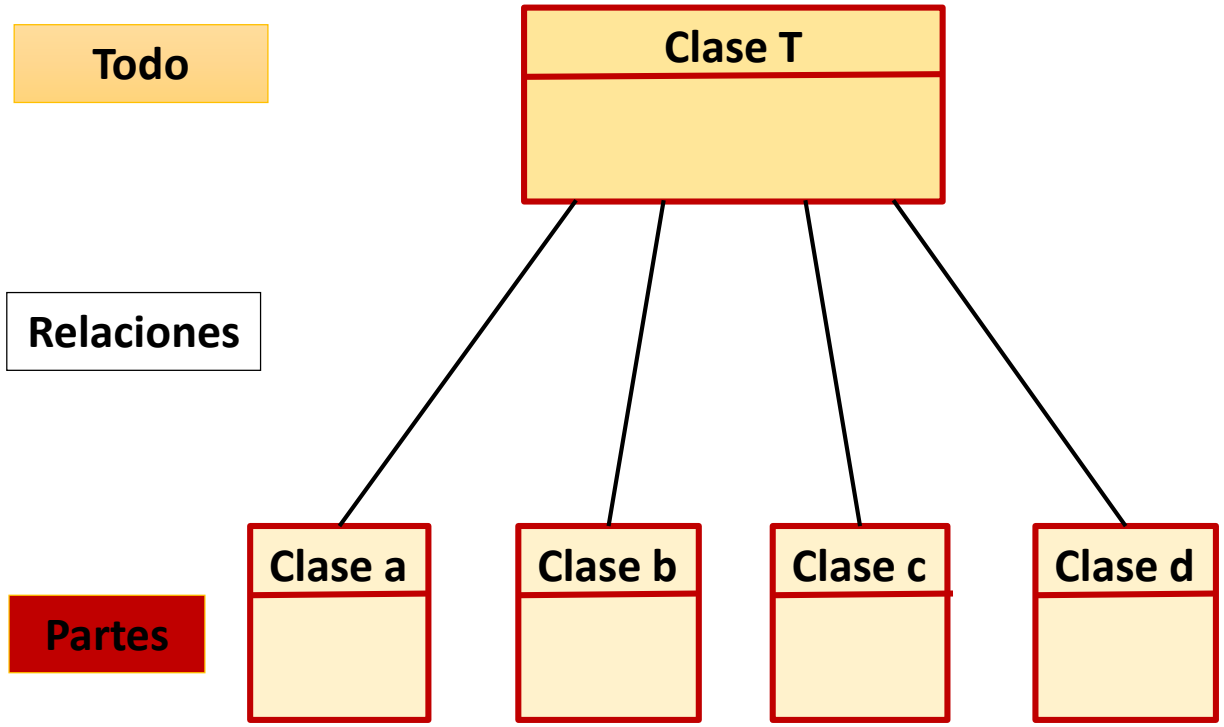
Desarrolle una clase Persona, con atributos para sus datos personales, la dirección de su domicilio, la empresa para la que trabaja (de la cual se puede conocer su nit, nombre y dirección) y la fecha de vinculación. Toda dirección debe indicar la ciudad y el país, además hay que tener en cuenta que una empresa puede tener varias direcciones y varios trabajadores.

Cree los diagramas de clases correspondientes y en una clase principal instancie al menos dos personas, muestre sus datos personales, la empresa para la que trabaja y el tiempo en meses de estar vinculado.

Tipos de asociaciones – Agregación y Composición

UNIVERSIDAD Popular del cesar

Las relaciones de **Agregación** y **Composición** son formas especificas de asociaciones entre un todo y sus partes (**ensamblado**). Donde el todo **(ensamblado)** está compuesto por sus partes.



Cada relación parte- todo se considera una relación por separado

UNIFIED MODELING LANGUAGE™

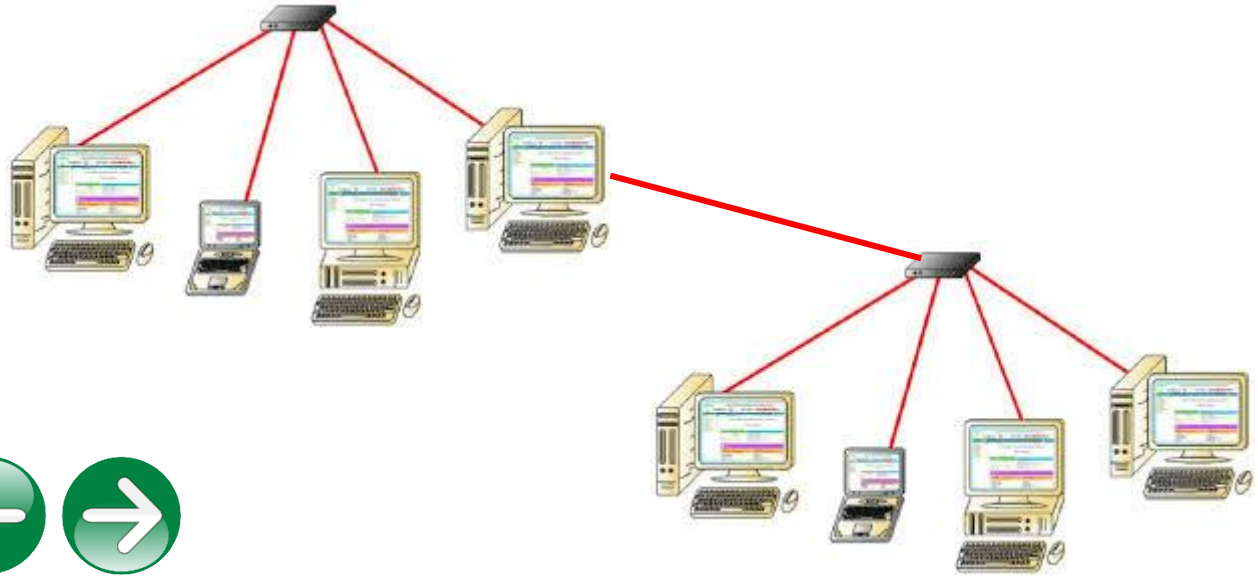
Relaciones entre Clases – Agregación

La **agregación** es un tipo de asociación que expresa la relación **"parte de",** esto es, una relación entre el todo y sus partes

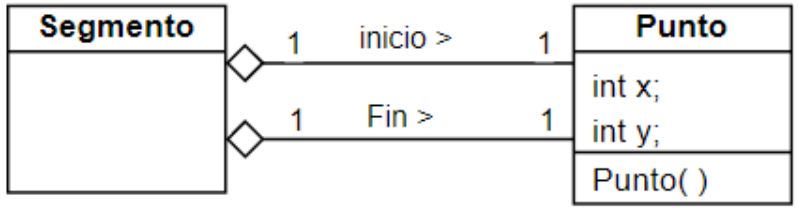
Se representa agregando un diamante vacío en el extremo que corresponde al todo.

En el caso de la agregación, las partes del ensamblado pueden aparecer en varios emsablados.

La existencias de las partes, no esta sujeta a la existencia del ensamblado.



Relaciones entre Clases – Agregación

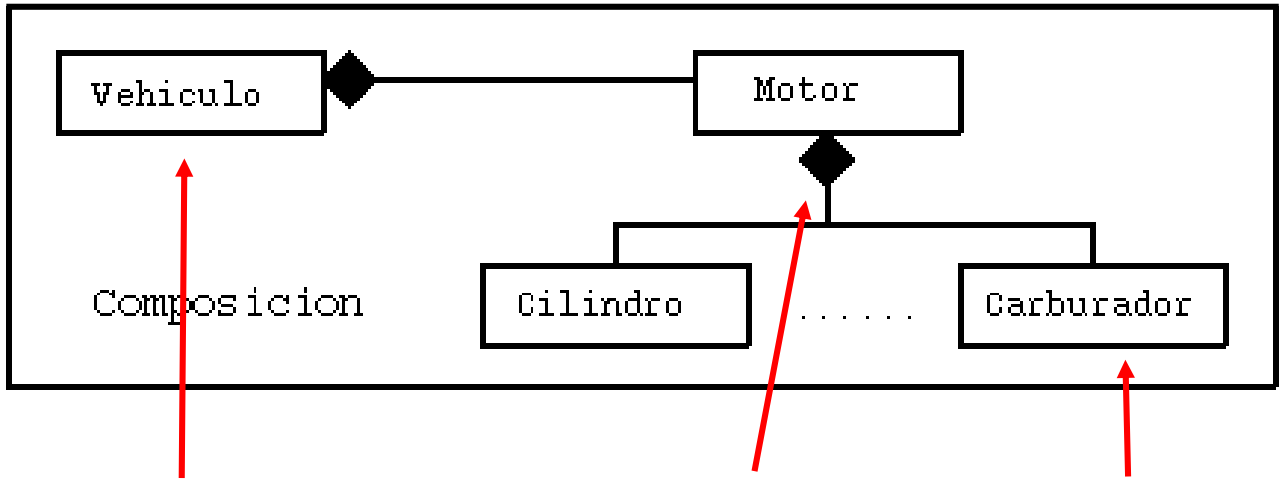



```
Punto a = new Punto(5,3);
Punto b = new Punto(0,0);
Punto c = new Punto(8,4);

Segmento ab = new Segmento(a,b);
Segmento bc = new Segmento(b,c);
```

```
public class Punto {

   private int x;
   private int y;

    public Punto(int x, int y) {
        this.x=x;
        this.y=y;
    }
    // metodos adicionales

}
```

```
public class Segmento {
    private Punto inicio;
    private Punto fin;

    public Segmento(Punto inicio, Punto fin) {
        this.inicio = inicio;
        this.fin = fin;
    }

    public Punto getInicio() {
        return inicio;
    }
    public void setInicio(Punto inicio) {
        this.inicio = inicio;
    }
    public Punto getFin() {
        return fin;
    }
    public void setFin(Punto fin) {
        this.fin = fin;
    }
}
```

Relaciones entre Clases – Composición

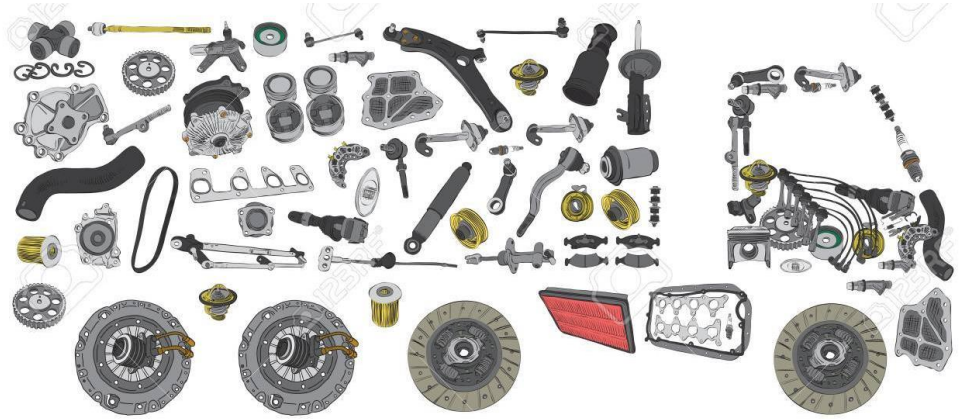


La **composición,** representa una relación más fuerte entre el todo y sus partes, en la cual las partes sólo tienen sentido como parte del todo.

La notación utilizada es un diamante lleno en el extremo que corresponde al todo.

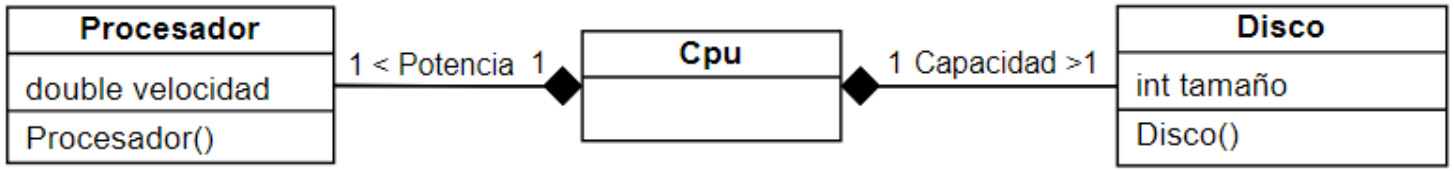
En el caso de la composición, las partes del ensamblado no pueden aparecer en otros emsablados.

Las partes son construidas y destruidas junto con el todo



UNIVERSIDAD Popular del cesar

Relaciones entre Clases – Composición




```
Cpu x32 = new Cpu();
Cpu x64 = new Cpu();
```

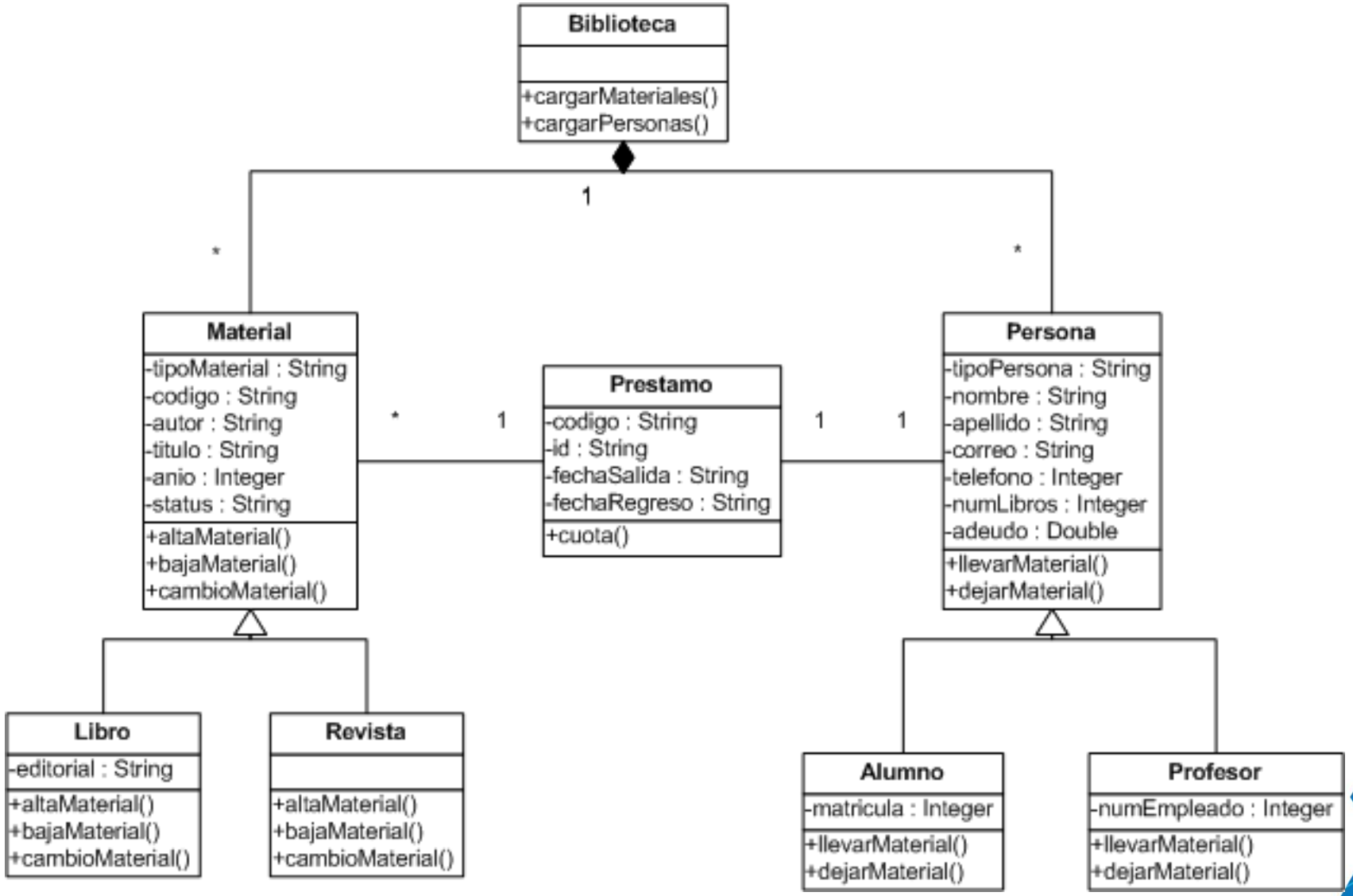
```
public class Procesador {
    private double velocidad;

    public Procesador() {
    }

    public double getVelocidad() {
        return velocidad;
    }

    public void setVelocidad(double velocidad) {
        this.velocidad = velocidad;
    }

}
```

```
public class Disco {
    private int tamaño;

    public Disco() {
    }

    public int getTamaño() {
        return tamaño;
    }

    public void setTamaño(int tam) {
        this.tamaño = tam;
    }

}
```

```
public class Cpu {
    private Disco capacidad;
    private Procesador potencia;

    public Cpu() {
        this.capacidad = new Disco();
        this.potencia = new Procesador();
    }
    public Disco getCapacidad() {
        return capacidad;
    }
    public void setCapacidad(Disco capacidad) {
        this.capacidad = capacidad;
    }
    public Procesador getPotencia() {
        return potencia;
    }
    public void setPotencia(Procesador potencia) {
        this.potencia = potencia;
    }
}
```

UNIVERSIDAD Popular del cesar

Relaciones entre Clases – Diagrama de Clases



UNIVERSIDAD Popular del cesar

Mejorando el diseño de clases

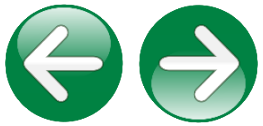
Principios de asignación de responsabilidades:

Solid

Single Responsibility Principle (SRP)
Open/Closed Principle (OCP)
Liskov Substitution Principle (LSP)
Interface Segregation Principle (ISP)
Dependency Inversion Principle (DIP)

Grasp:

Experto en Información
Creador
Alta Cohesión
Bajo Acoplamiento
Fabricación Pura
Indirección
Variaciones Protegidas



Relaciones entre Clases – Ejercicios

Se desea desarrollar un programa para la gestión de una biblioteca, por lo cual, se le pide que diseñe una clase que represente un Libro. En el registro de todo libro de la biblioteca es de suma importancia conocer su isbn, su titulo, el numero de edición, el autor y la editorial. Del autor se debe conocer su nombre, apellido y su nickname, mientras que de la editorial su nombre, y el país de origen. En una clase principal se desea instanciar tres libros e imprimir sus datos.

Se requiere una aplicación que permita el registro y consulta de las cuentas activas de un banco. Del Banco se conoce su nombre, sede, país. Por su parte, de cada Cuenta se conoce su numero, el saldo, el tipo (ahorro/corriente) y el titular de la cuenta. Se debe tener en cuenta que un banco puede tener activas varias cuentas y que una cuenta solo puede pertenecer a un único banco.

De cada empleado de una compañía se conoce su id, nombre, apellido, año de vinculación y puede tener asignado uno o varios cargos, mientras que, un cargo puede tener asignado varios empleados. Se requiere una aplicación que permita el registro de los empleados de la compañía y la impresión de los empleados que cada cargo tiene asignado.

Evaluaciones Grupo 01

Teórico:
Practico: Jueves,
Proyecto – Entrega 1:

Evaluaciones Grupo 02

Teórico:
Practico:
Proyecto – Entrega 1:

UNIVERSIDAD
Popular del cesar