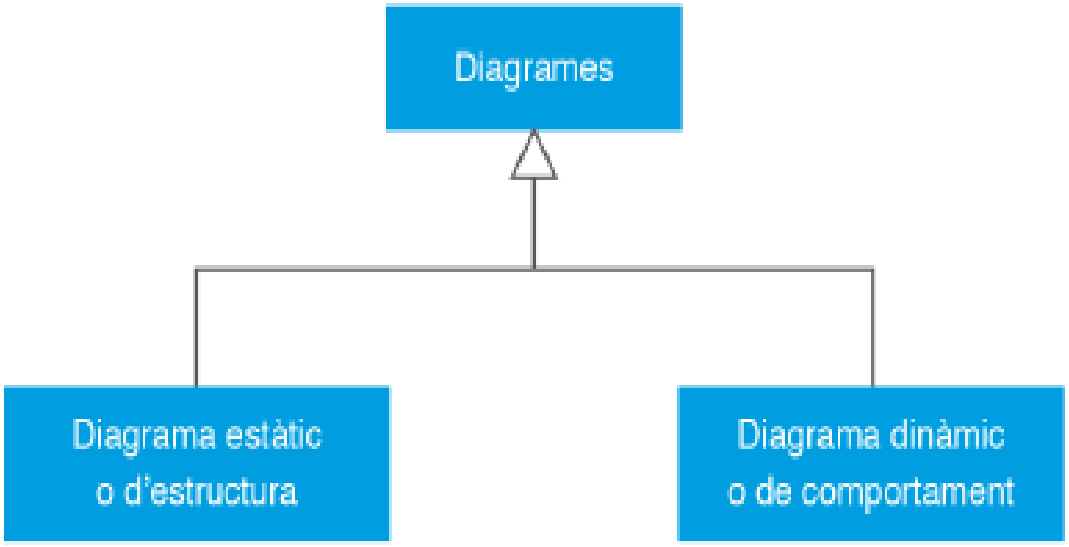
4 Diagramas estáticos

Índice

Clasificación diagramas

FIGURA 2.1. Classificació dels diagrames UML



Diagramas estáticos

Visión estática (o estructural): para ofrecer este tipo de visión se utilizan objetos, atributos, operaciones y relaciones. La visión estática de un sistema da más valor a los elementos que se encontrarán en el modelo del sistema desde un punto de vista de la estructura del sistema. Describen aspectos del sistema que son estructurales y, por tanto, permanentes (lo que el sistema tiene).

Diagramas estáticos

Dentro de los diagramas estáticos o diagramas de estructura se pueden encontrar 7 tipos de diagramas, que son:

Diagrama de paquetes, que representa esencialmente las relaciones de diferentes tipos entre los contenidos de diferentes paquetes de un modelo.

Diagramas estáticos

Diagrama de clases, que probablemente muchos consideran el diagrama principal, dado que describe clasificadores de todo tipo y diferentes tipos de relaciones entre ellos antes de utilizarlos en otros diagramas.

Diagrama de objetos, que representa instancias de clasificadores definidos en un diagrama de clases previo y relaciones entre sí.

Diagramas estáticos

Diagrama de estructuras compuestas, que describe casos en los que, o bien las instancias de un clasificador tienen como partes instancias de otros, o bien en el comportamiento ejecutante de un clasificador participan a instancias de otros.

Diagrama de componentes, que es un diagrama de clases y al mismo tiempo de estructuras compuestas simplificado y más adecuado para determinadas tecnologías de programación.

Diagramas estáticos

Diagrama de despliegue, que describe la configuración en tiempo de ejecución de un software especificado, normalmente, por un diagrama de componentes.

Diagrama de perfil, permite adaptar o personalizar el modelo con construcciones que son específicas de un dominio en particular, de una determinada plataforma, o de un método de desarrollo de software…

Diagramas estáticos

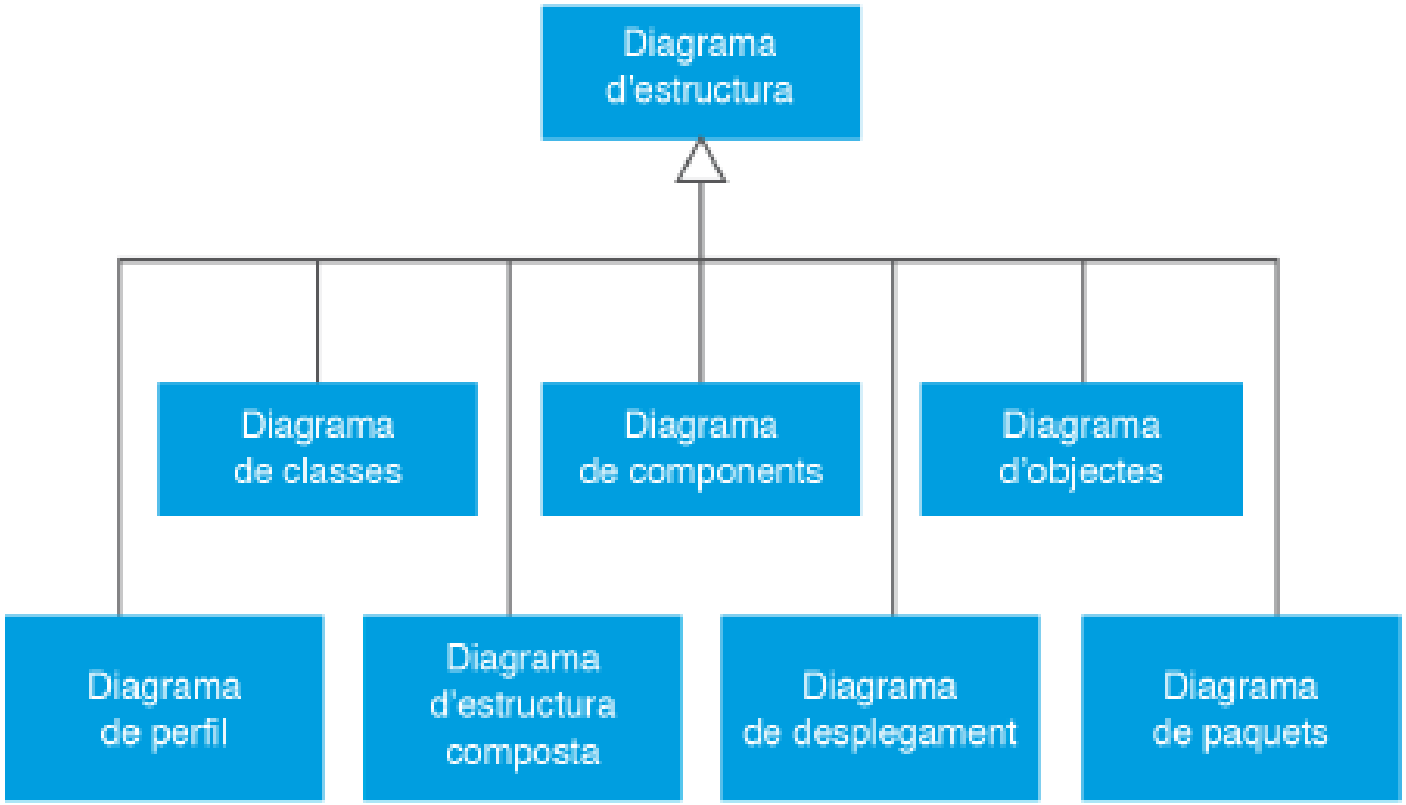


Diagrama de clases

Uno de los diagramas referentes de UML, clasificado en los diagramas de tipo estático, es el **diagrama de clases**. Es uno de los diagramas más utilizados en las metodologías de análisis y diseño que se basan en UML.

Un diagrama de clases representa las clases que serán utilizadas en el sistema y las relaciones que existen entre ellas.

Diagrama de clases

Este tipo de diagramas son utilizados durante las **fases de análisis y diseño** de los proyectos de desarrollo de software. Es en ese momento en que se comienza a crear el modelo conceptual de los datos que utilizará el sistema. Por eso se identifican los componentes (con sus **atributos y funcionalidades**) que tomarán parte en los procesos y se definen las relaciones que habrá entre ellos.

Diagrama de clases

Un diagrama de clases lleva vinculados algunos conceptos que ayudarán a entenderlos la creación y el funcionamiento en su totalidad. Estos conceptos son:

- Clase, atributo y método (operaciones).
- Visibilidad.
- Objeto. Instanciación.
- Relaciones. Herencia, composición y agregación.
- Clase asociativa.

Clases. Atributos y operaciones

Una **clase** describe un conjunto de objetos que comparten los mismos atributos, que representan características estables de las clases, y las operaciones, que representan las acciones de las clases.

Clases. Atributos y operaciones

Los **atributos** (también llamados propiedades o características) son los datos detallados que contienen los objetos. Estos valores corresponden al objeto que instancia la clase y hace que todos los objetos sean distintos entre sí.

Clases. Atributos y operaciones

Las operaciones (también llamadas **métodos** o funcionalidades) implementan las acciones que podrán llevarse a cabo sobre los atributos.

Clases. Atributos y operaciones

Las operaciones (también llamadas **métodos** o funcionalidades) implementan las acciones que podrán llevarse a cabo sobre los atributos.

Clases. Atributos y operaciones

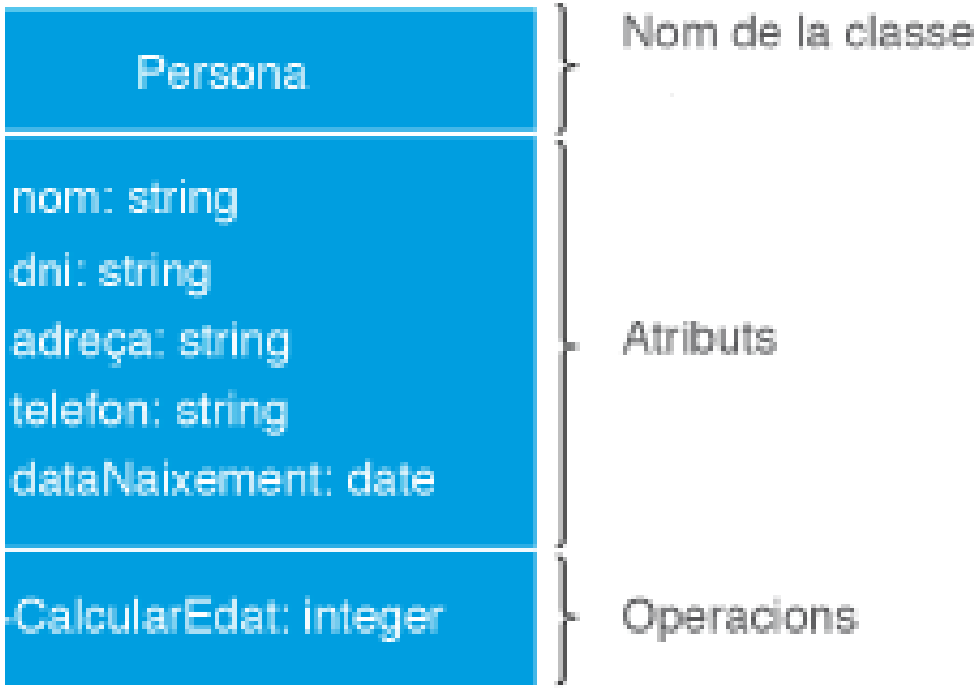


Diagrama de clases

```
Class Persona{
{
     // atributs
     private String nom;
     private String dni;
     private String adreca;
     private String telefon;
     private date dataNaixement;
     // constructor
     public Persona(String nom, String dni, string adreca, string telefon, date
          dataNaixement{
        this.nom = nom;
        this.dni = dni;
        this.adreca= adreca;
        this.telefon = telefon;
        this.dataNaixement = dataNaixement
      }
     // mètodes o operacions
     public integer CalcularEdat() {
          Date dataActual = new Date();
          SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
          String _dataActual = formato.format(dataActual);
          String _dataNaixement = formato.format(dataNaixement);

          String[] dataInici = _dataNaixement.split("/");
          String[] dataFi = _dataActual.split("/");

          int anys = Integer.parseInt(dataFi[2]) - Integer.parseInt(dataInici
               [2]);
          int mes = Integer.parseInt(dataFi[1]) - Integer.parseInt(dataInici
               [1]);
          if (mes < 0) {
               anys = anys - 1;
          } else if (mes == 0) {
               int dia = Integer.parseInt(dataFi[0]) - Integer.parseInt(
                    dataInici[0]);
               if (dia > 0) anys = anys - 1;
          }
          return anys;
     }
}
```

Diagrama de clases

```
1  Class Persona{
2  {
3        // atributs
4        private String nom;
5        private String dni;
6        private String adreca;
7        private String telefon;
8        private date dataNaixement;
9        // constructor
10       public Persona(String nom, String dni, string adreca, string telefon, date
               dataNaixement{

11         this.nom = nom;
12         this.dni = dni;
13         this.adreca= adreca;
14         this.telefon = telefon;
15         this.dataNaixement = dataNaixement;
16       }
```

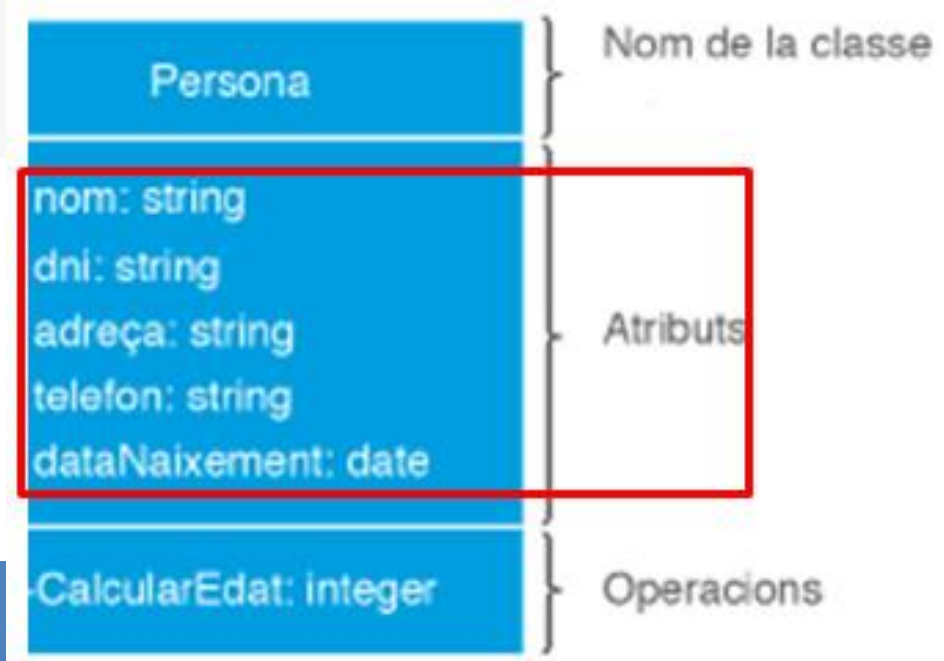
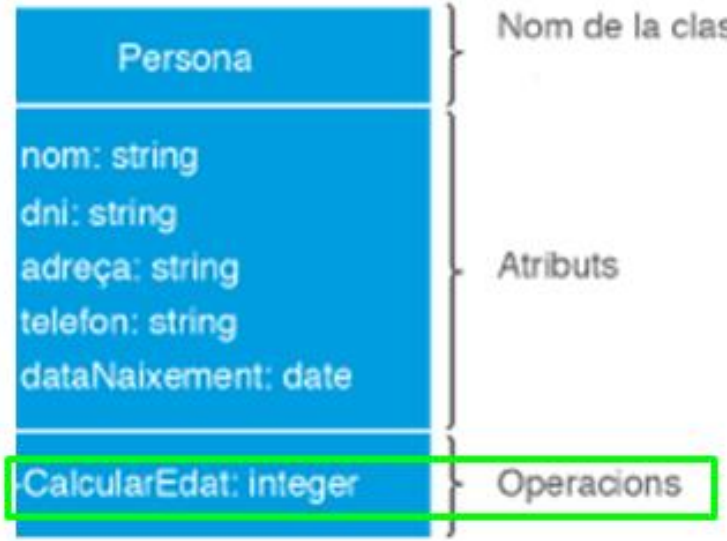


Diagrama de clases

```
17      // mètodes o operacions
18      public integer CalcularEdat() {
19            Date dataActual = new Date();
20            SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
21            String _dataActual = formato.format(dataActual);
22            String _dataNaixement = formato.format(dataNaixement);
23
24            String[] dataInici = _dataNaixement.split("/");
25            String[] dataFi = _dataActual.split("/");
26
27            int anys = Integer.parseInt(dataFi[2]) - Integer.parseInt(dataInici
                  [2]);
28            int mes = Integer.parseInt(dataFi[1]) - Integer.parseInt(dataInici
                  [1]);
29            if (mes < 0) {
30                  anys = anys - 1;
31            } else if (mes == 0) {
32                  int dia = Integer.parseInt(dataFi[0]) - Integer.parseInt(
                        dataInici[0]);
33                  if (dia > 0) anys = anys - 1;
34            }
35            return anys;
36      }
37  }
```



Visibilidad

La visibilidad de un atributo o de una operación definirá el ámbito desde el que podrán ser utilizados estos elementos. Esta característica está directamente relacionada con el concepto de orientación a objetos llamado encapsulación, mediante el cual se permite a los objetos decidir cuál de la su información será más o menos pública para el resto de objetos.

Visibilidad

Las posibilidades para la visibilidad, tanto de atributos como de métodos, son:

+ en UML, o **public**
- en UML, o **private**
en UML, o **protected**
~ en UML, o **package**

Visibilidad

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=665:public-private-y-protected-javatipos-de-modificadores-de-acceso-visibilidad-en-clases-subclases-cu00693b&catid=68&Itemid=188

Diagrama de clases

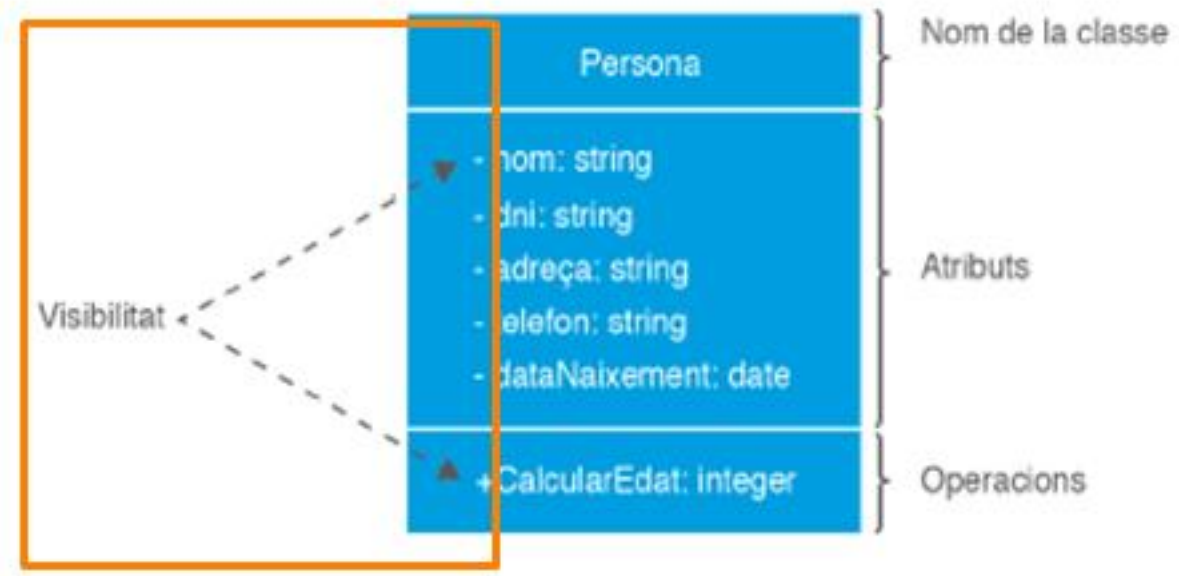


Diagrama de clases

```
1  Class Persona{
2  {
3       // atributs
4       private String nom;
5       private String dni;
6       private String adreca;
7       private String telefon;
8       private date dataNaixement;
9       // constructor
10      public Persona(String nom, String dni, string adreca, string telefon, date
            dataNaixement{
11        this.nom = nom;
12        this.dni = dni;
13        this.adreca= adreca
14        this.telefon = tele
15        this.dataNaixement
16      }
```

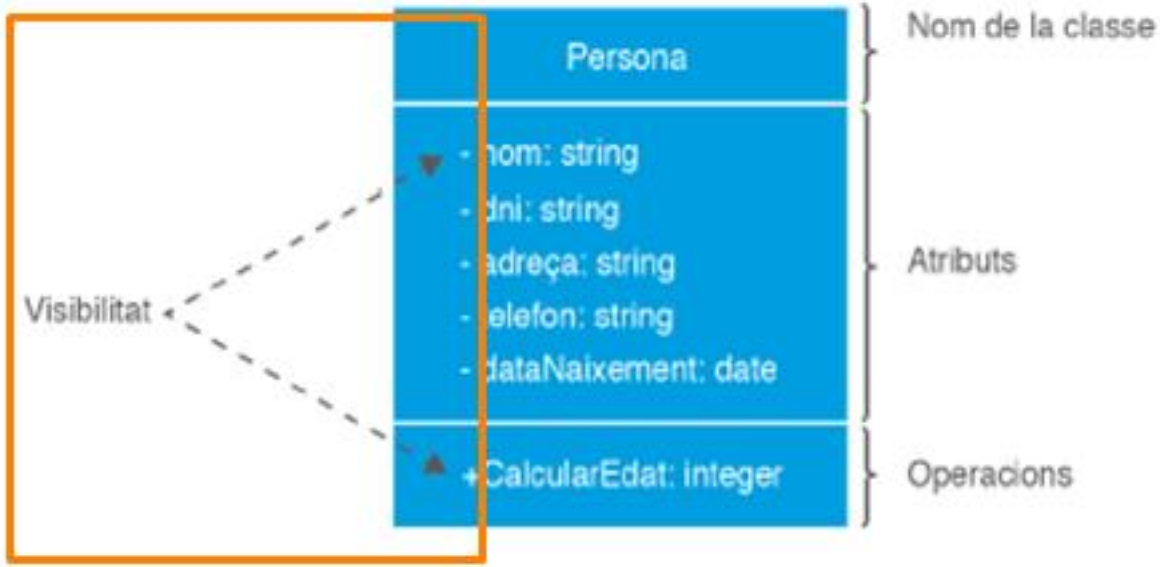
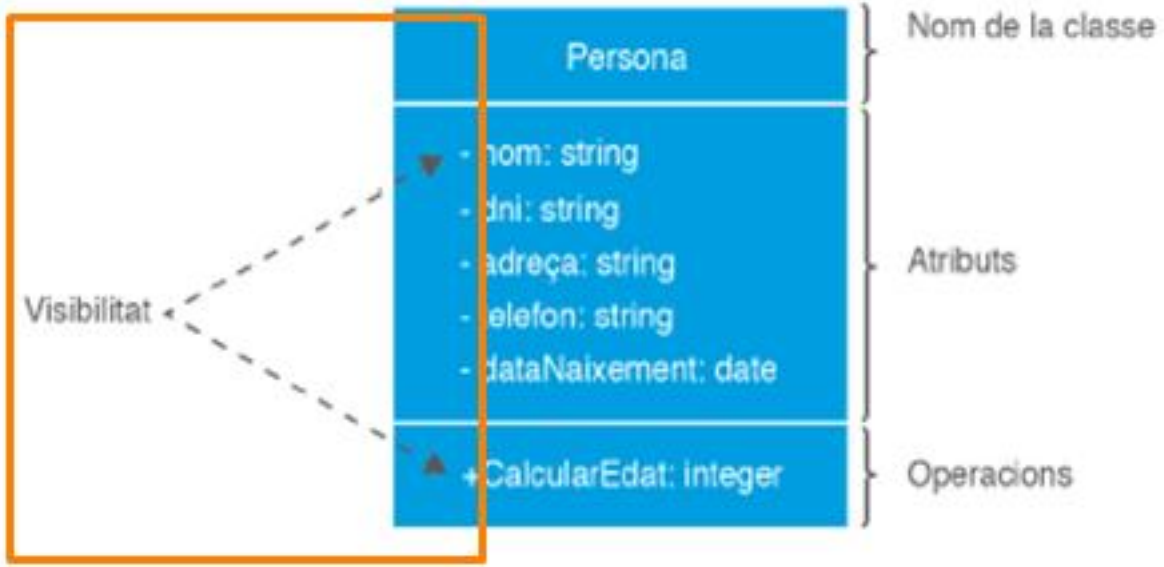


Diagrama de clases

```
17      // mètodes o operacions
18      public integer CalcularEdat() {
19            Date dataActual = new Date();
20            SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
21            String _dataActual = formato.format(dataActual);
22            String _dataNaixement = formato.format(dataNaixement);
23
24            String[] dataInici = _dataNaixement.split("/");
25            String[] dataFi = _dataActual.split("/");
26
27            int anys = Integer.parseInt(dataFi[2]) - Integer.parseInt(dataInici
                  [2]);
28            int mes = Integer.parseInt(dataFi[1]) - Integer.parseInt(dataInici
                  [1]);
29            if (mes < 0) {
30                  anys = anys - 1;
31            } else if (mes ==
32                  int dia = Int
                        dataInici
33                  if (dia > 0)
34            }
35            return anys;
36      }
37  }
```

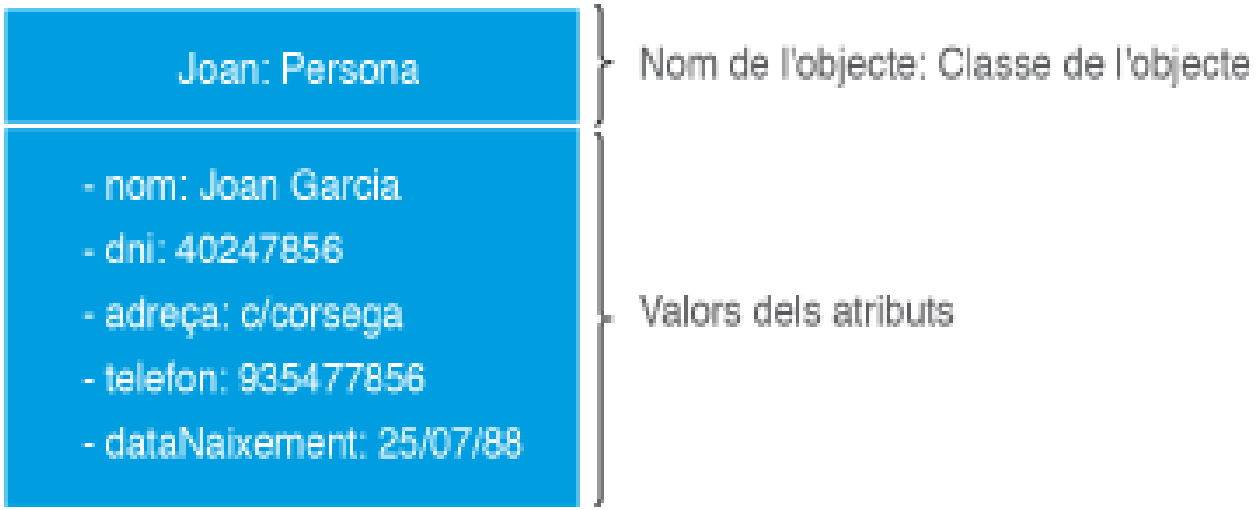


Objetos. Instanciación

Un objeto es una instanciación de una clase. El concepto instanciación indica la acción de crear una instancia de una clase.

La creación de una instancia de una clase se refiere a llamar al método constructor de una clase en tiempo de ejecución de un software

Diagrama de clases



El código para crear un objeto o instancia:

NombreClase nombreObjeto = new Constructor(…);

```
1  public static void main(String[] args) {
2        Persona Joan = new Persona("Joan Garcia","40782949","C/Casanoves 130",
             "93 2983924", 25/03/1977);
3  }
```

Diagrama de clases

Relaciones. Herencia, composición y agregación

Las **relaciones** son elementos imprescindibles en un diagrama de clases. Por relación se entiende que un objeto obj1 pida a otro obj2, mediante un mensaje, que ejecute una operación de las definidas en la clase del obj2.

Diagrama de clases

Relaciones. Herencia, composición y agregación

En el ejemplo de un cliente que reserva una habitación de hotel se puede observar que intervienen dos clases, Persona y Habitación, que están relacionadas, donde la clase Habitación consulta el nombre del cliente correspondiente a la clase Persona.

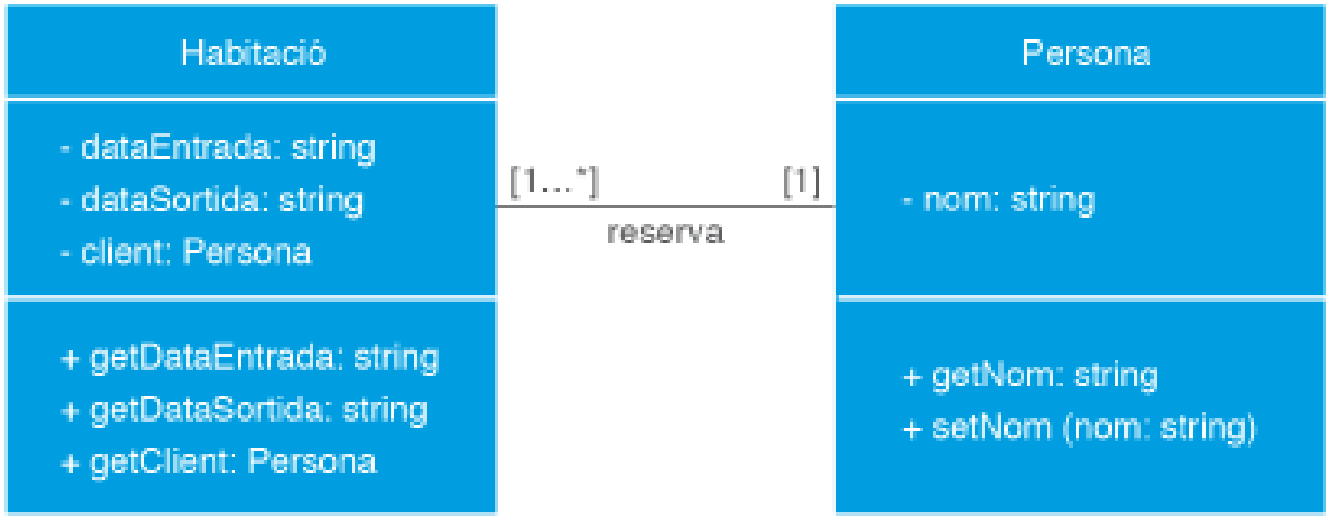


Diagrama de clases

Relaciones. Herencia, composición y agregación

Las relaciones que existen entre las diferentes clases se llaman de forma genérica **asociaciones**.

Una **asociación** es un clasificador que define una relación entre varios clasificadores, que establece conexiones con un cierto significado entre las instancias respectivas.

Diagrama de clases

Relaciones. Herencia, composición y agregación

La **multiplicidad**, representada por unos valores <min....max>, indica el número máximo de enlaces posibles que podrán darse en una relación. Esta multiplicidad nunca podrá ser negativa, siendo, además, el valor máximo siempre mayor o igual al valor mínimo.

Diagrama de clases

Relaciones. Herencia, composición y agregación

Concretamente, indica que, en la asociación Préstamo, para cada objeto de la clase Lector podrá haber, al menos 0 objetos de la clase Libro y, como máximo, 3 objetos. En cambio para cada objeto de la clase Libro podrá existir, al menos 0 y como máximo 1 objeto de la clase Lector.

Diagrama de clases

Relaciones. Herencia, composición y agregación

Se pueden encontrar distintos tipos de relaciones entre clases. Éstas se pueden clasificar de muchas formas. A continuación, se muestra una clasificación de las relaciones:

- Relación de asociación
- Relación de asociación de agregación
- Relación de asociación de composición
- Relaciones de dependencia
- Relación de generalización

Diagrama de clases

Relaciones. Herencia, composición y agregación

1) La **relación de asociación** se representa mediante una línea continua sin flechas ni ningún otro símbolo en los extremos. Es un tipo de relación estructural que define las conexiones entre dos o más objetos, lo que permite asociar objetos que instancien clases que colaboren entre sí.

Diagrama de clases

Relaciones. Herencia, composición y agregación

2) Una **relación de asociación de agregación** es un caso especial de asociación entre dos o más objetos. Se trata de una relación del tipo todo-parte. Este tipo de relación implica dos tipos de objetos, el objeto llamado base y el objeto que estará incluido en el objeto base. **La relación indica que el objeto base necesita del objeto incluido para poder existir y realizar sus funcionalidades. Si desaparece el objeto base, el o los objetos que se encuentren incluidos en el objeto base no desaparecerán y podrán seguir existiendo con sus propias funcionalidades**. La relación de asociación de agregación se representa mediante una línea continua que finaliza en uno de los extremos por un rombo vacío, sin llenar. El rombo vacío se ubicará en la parte de el objeto base.

Diagrama de clases

Relaciones. Herencia, composición y agregación

El objeto base es el objeto llamado Frutería. Los objetos incluidos en la frutería son: Manzanas, Naranjas, Peras y Fresas. Se establece una relación entre estas clases del tipo todo-parte, donde las frutas son parte de la frutería. Eso sí**, si la frutería deja de existir, las frutas siguen existiendo.**

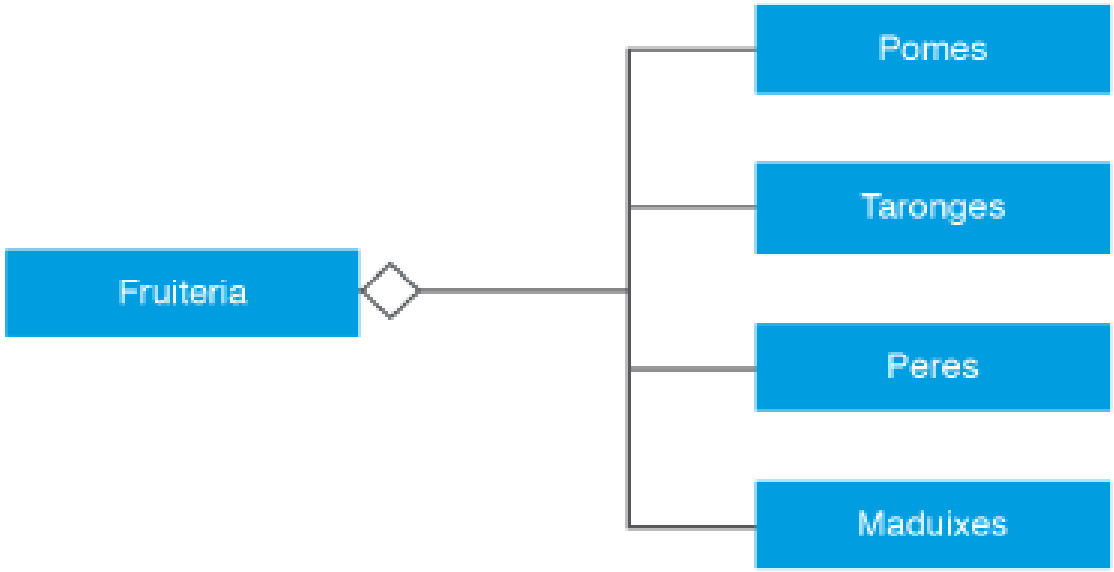


Diagrama de clases

Relaciones. Herencia, composición y agregación

3) Una **relación de asociación de composición** es también un caso especial de asociación entre dos o más objetos. Es una relación del tipo todo-parte. Es una relación muy similar a la relación de asociación de agregación, con la diferencia de que **existe una dependencia de existencia entre el objeto base y el objeto** (o los objetos) que existe está incluido. Es decir**, si deja de existir el objeto base, dejará de existir también el o los objetos incluidos**. El tiempo de vida del objeto incluido depende del tiempo de vida del objeto base. La relación de asociación de composición se representa mediante una línea continua finalizada en uno de los extremos por un rombo pintado, llenado. El rombo pintado se ubicará en la parte del objeto base.

Diagrama de clases

Relaciones. Herencia, composición y agregación

El objeto base Mano se compone de los objetos incluidos Meñique, Anular, Corazón, Índice y Pulgar. Sin el objeto Mano el resto de objetos dejarán de existir.

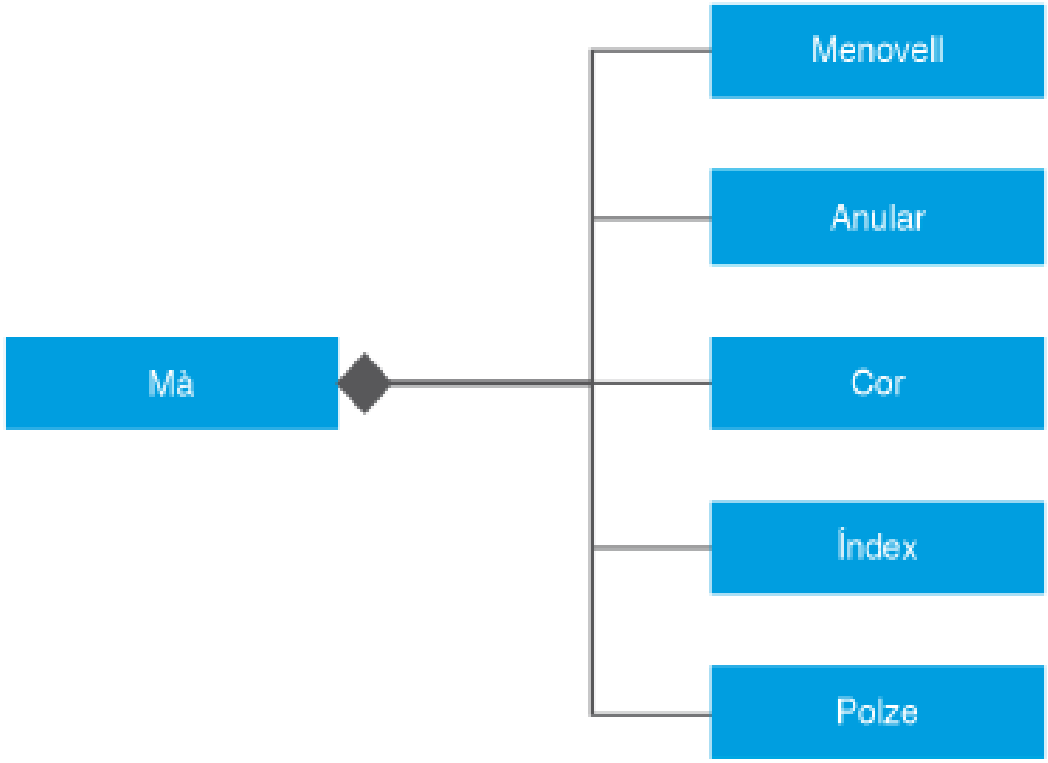


Diagrama de clases

Relaciones. Herencia, composición y agregación

4) Otro tipo de relación entre clases es la **relación de dependencia**. Éste tipo de relación se representa mediante una flecha discontinua entre dos elementos. El objeto del que sale la flecha se considera un objeto dependiente. El objeto en el que llega la flecha se considera un objeto independiente. Se trata de una relación semántica. **Si existe un cambio en el objeto independiente, el objeto dependiente se verá afectado.**

Diagrama de clases

Relaciones. Herencia, composición y agregación

Existe una dependencia normal del elemento película en relación con el elemento actor, ya que actor se utiliza como parámetro en los métodos añadir y eliminar de película. Si hay cambios en actor, el objeto película se verá afectado.

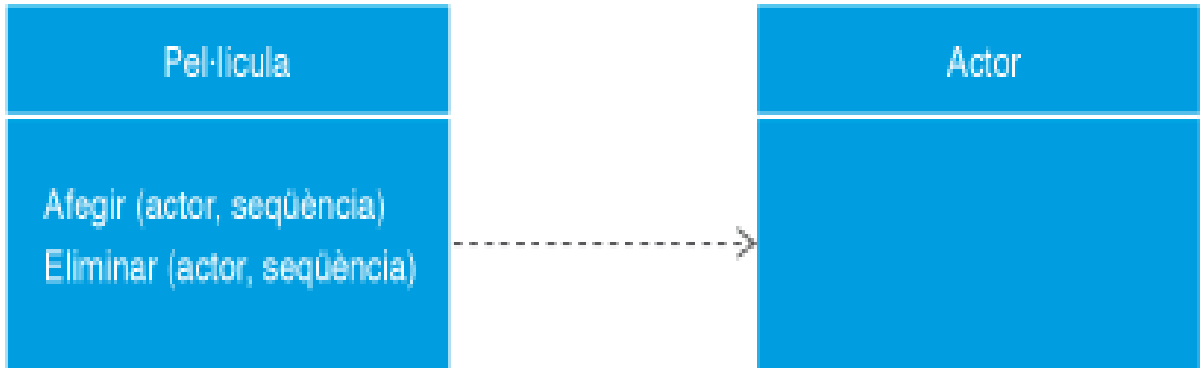


Diagrama de clases

Relaciones. Herencia, composición y agregación

5) La **relación de generalización** se da entre dos clases donde existe un vínculo que se puede considerarse de **herencia**. Una clase es llamada clase madre (o superclase). La otra (u otras) son las llamadas clases hijas o subclases, que heredan los atributos y métodos y comportamiento de la clase madre. Este tipo de relación queda especificado mediante una flecha que sale de la clase hija y que termina en la clase madre.

Diagrama de clases

Relaciones. Herencia, composición y agregación

La herencia se da a partir de estas relaciones de dependencia. Ofrecen como punto fuerte la posibilidad de reutilizar parte del contenido de un objeto (lo considerado superclase), extendiendo sus atributos y métodos al objeto hijo.

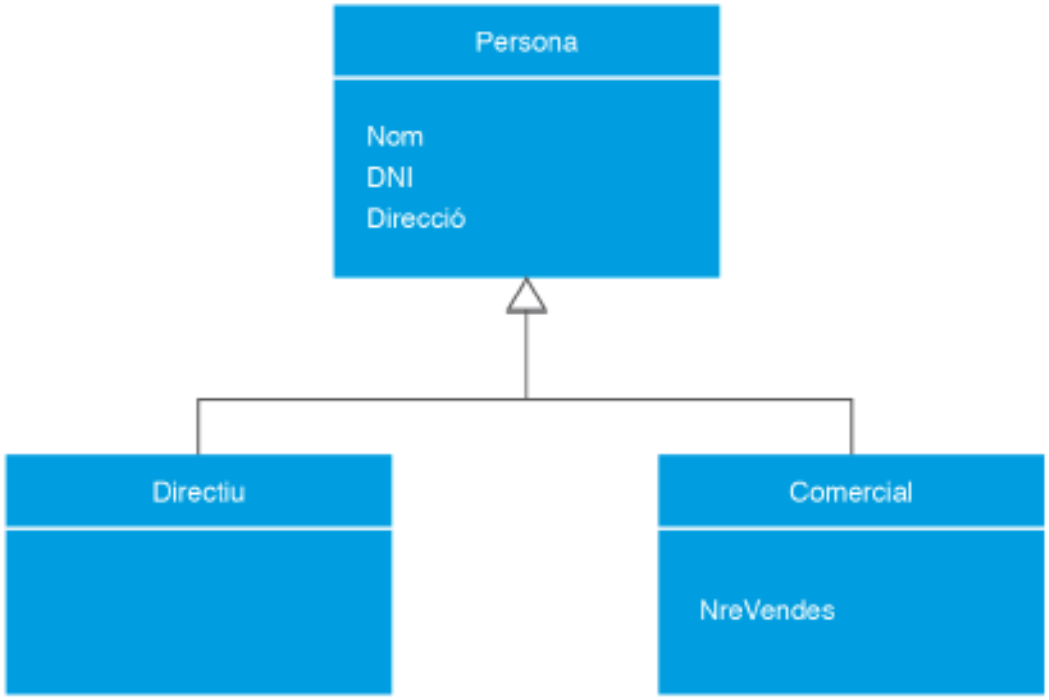


Diagrama de clases

Clase asociativa

Cuando una **asociación tiene propiedades o métodos propios** se representa como una clase unida a la línea de la asociación por medio de una línea discontinua. Tanto la línea como el rectángulo de clase representan el mismo elemento conceptual: la asociación.

Diagrama de clases

Clase asociativa

La nota está directamente relacionada con las clases Estudiant y Asignatura. Cada uno de los alumnos de la asignatura tendrá una determinada nota. La forma de modelizar el UML esta situación es con las clases asociadas.

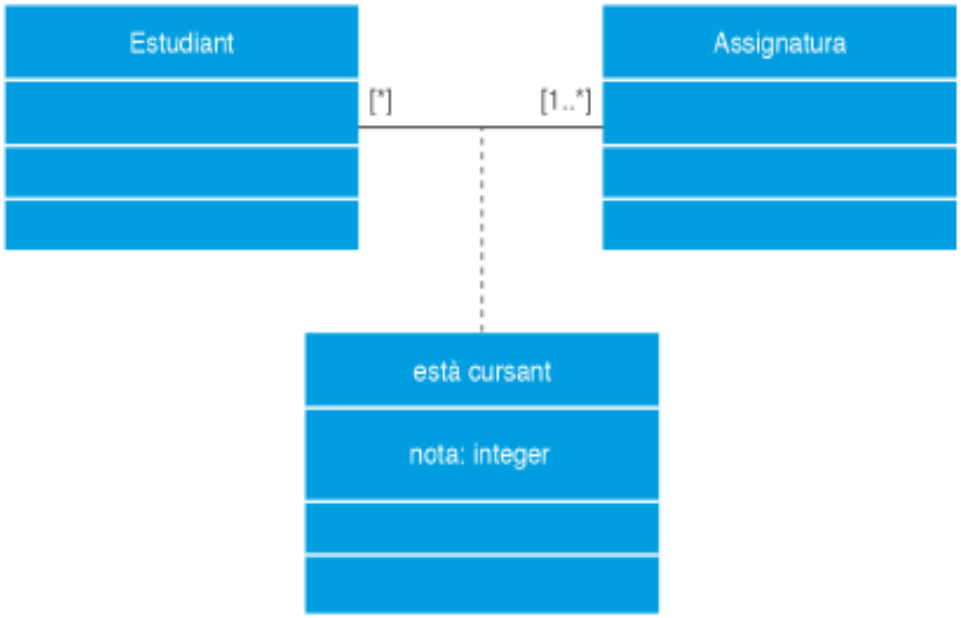


Diagrama de clases

Clase abstracta

Una **clase abstracta** para **Java** es una **clase** de la que nunca se van a crear instancias: simplemente va a servir como superclase a otras **clases**. No se puede usar la palabra clave new aplicada a **clases abstractas**.

```
public abstract class NombreDeLaClase { … }
```

https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=668:clases-y-metodos-abstractos-en-java-abstract-class-clases-del-api-ejemplos-codigo-y-ejercicios-cu00695b&catid=68&Itemid=188

Diagrama de clases

Interfaz

Una interfaz contiene la declaración de las operaciones sin su implementación, que tendrán que ser implementadas por una clase o componente.

Llegados a este punto, podríamos preguntarnos: ¿Qué diferencia hay entre una interfaz y una clase abstracta?

Una interfaz es simplemente una lista de métodos no implementados, así como la declaración de posibles constantes. Una clase abstracta, a diferencia de las interfaces, puede incluir métodos implementados y no implementados.

Diagrama de clases

Interfaz

Esta interfaz se llama ICalculadora, y contiene los métodos suma, resta, multiplicacion y division.

```
1  interface ICalculadora {
2      public abstract int suma (int x, int y);
3      public abstract int resta (int x, int y);
4      public abstract int multiplicacio (int x, int y);
5      public abstract int divisio (int x, int y);
6  }
```

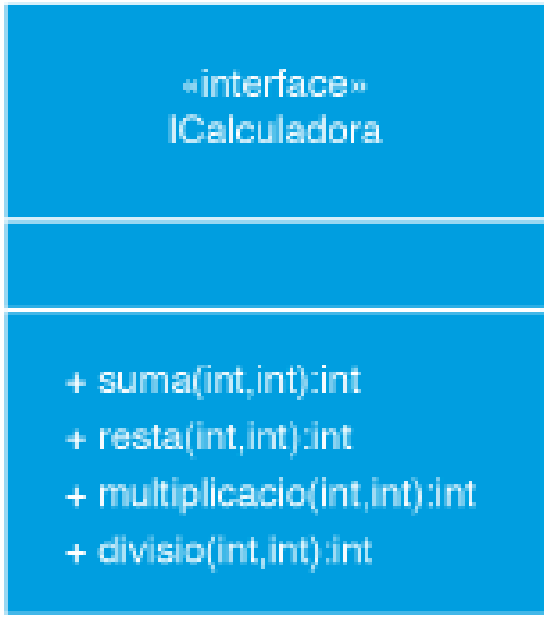


Diagrama de clases

Interfaz

Una vez definida se muestra cómo será la utilización de la interfaz definida en el código siguiente.

```
1  public class Calculadora implements ICalculadora {
2          public int suma (int x, int y){
3              return x + y;
4          }
5          public int resta (int x, int y){
6              return x – y;
7          }
8          public int multiplicacio (int x, int y){
9              return x * y;
10         }
11         public int divisio (int x, int y){
12             return x / y;
13         }
14 }
```