

PARADIGMA ORIENTADO A OBJETOS

Paradigmas de Programación

EN ESTA CLASE

- ▶ Problemas y soluciones
- ▶ El proceso de desarrollo de **software**
- ▶ El concepto de **objeto**
- ▶ El concepto de **clase**
- ▶ **Caso de estudio: Presión arterial**
- ▶ Conceptos pilares

Calendario

Día	Fecha	Actividad
Miercoles	5	Intro POO / Intro Java / Consola / IDE
Jueves	6	Java / Excepciones / Arreglos
Miercoles	12	Relaciones / Herencia
Jueves	13	Polimorfismo / Problema del diamante / Genericidad
Miercoles	19	Excepciones / Colecciones
Jueves	20	Practica
Miercoles	26	SOLID / Practica / Ligaduras / Consulta
Jueves	27	Practica / Opcional: Swing / Opcional: Documentacion en JAVA
Miercoles	2	Ejercitación Integral - En grupo
Jueves	3	Parcial
Viernes	4	Entrega Laboratorio

SISTEMAS DE SOFTWARE

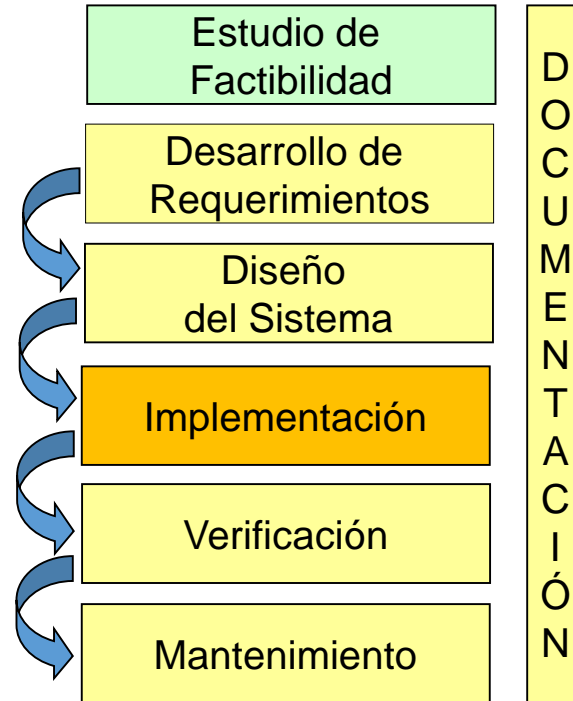
El **desarrollo de software** es un **proceso de** que abarca distintas **etapas** y requiere de la aplicación de una metodología.

Es un proceso **colaborativo** en el que interactúan los miembros del **equipo de desarrollo** con **clientes y usuarios**.

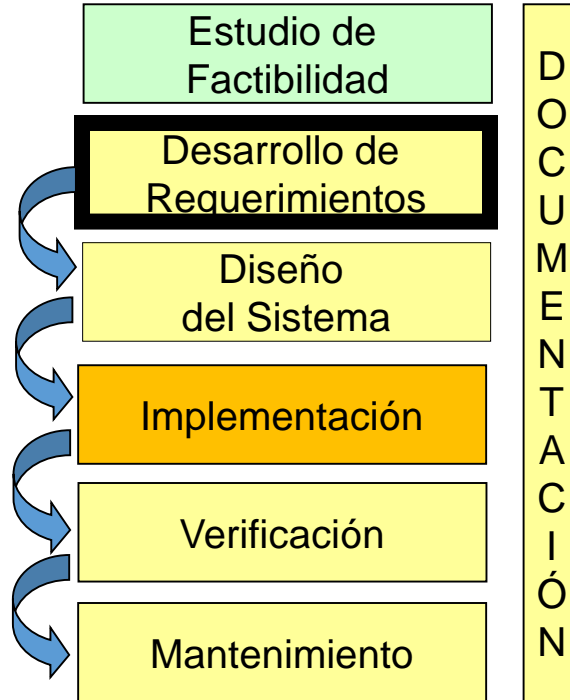
El **ciclo de vida** de un sistema de software comienza cuando se formula la necesidad, oportunidad o idea que le da origen y termina cuando deja de utilizarse.

Las **etapas** del desarrollo puede **organizarse** de diferentes maneras, una alternativa es el **modelo en cascada**.

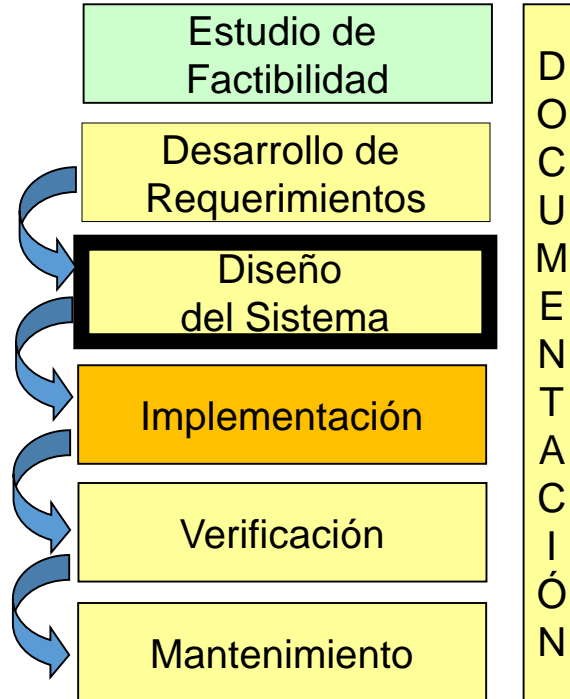
EL PROCESO DE DESARROLLO DE SOFTWARE



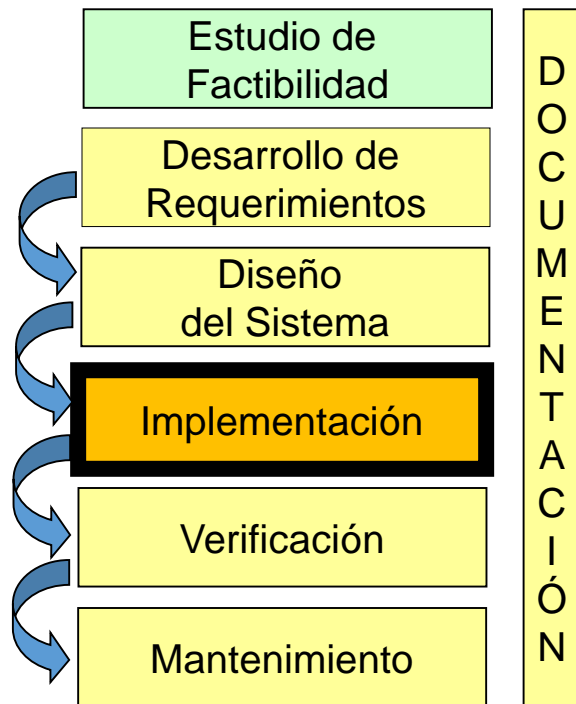
EL PROCESO DE DESARROLLO DE SOFTWARE



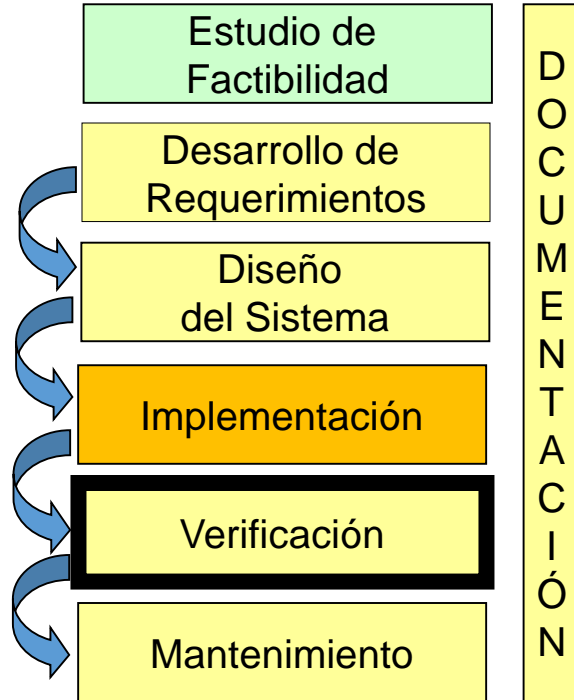
EL PROCESO DE DESARROLLO DE SOFTWARE



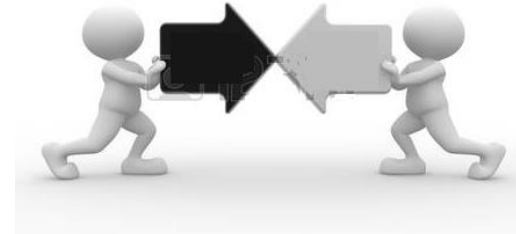
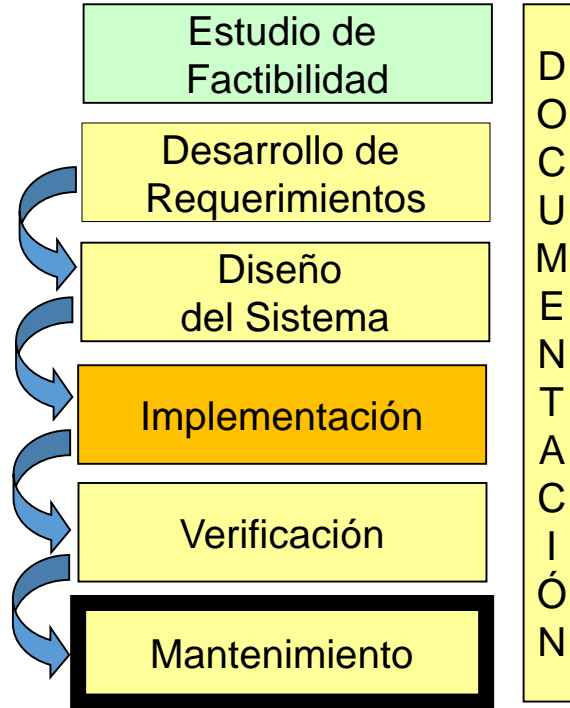
EL PROCESO DE DESARROLLO DE SOFTWARE



EL PROCESO DE DESARROLLO DE SOFTWARE



EL PROCESO DE DESARROLLO DE SOFTWARE



METODOLOGÍAS Y LENGUAJES

El **proceso de desarrollo de software** requiere de la aplicación de una **metodología** y algunas **herramientas** consistentes con esa metodología.

Una **metodología** está formada por un conjunto de **métodos, técnicas y estrategias**.

Actualmente la metodología más difundida está integrada al **paradigma de programación orientada a objetos**.

Las **herramientas** más importantes dentro del proceso de desarrollo de software son el **lenguaje de modelado** y el **lenguaje de programación**.

LENGUAJE DE MODELADO Y LENGUAJE DE PROGRAMACIÓN

Un **lenguaje de modelado** es una notación que permite especificar las partes esenciales de un sistema de software.

El lenguaje de modelado más utilizado tanto en el ámbito académico como comercial es **UML**.

LENGUAJE DE MODELADO Y LENGUAJE DE PROGRAMACIÓN

Un **lenguaje de modelado** es una notación que permite especificar las partes esenciales de un sistema de software.

El lenguaje de modelado más utilizado tanto en el ámbito académico como comercial es **UML**.



Java es uno de los lenguajes de programación más ampliamente difundidos dentro de la POO.

Dos conceptos centrales tanto en la metodología como en las herramientas son **objeto** y **clase**.

CASO DE ESTUDIO: GESTIÓN DE UN HOSPITAL

El Gobierno de la Provincia de Mendoza decide desarrollar un sistema de gestión para los Hospitales bajo su jurisdicción.

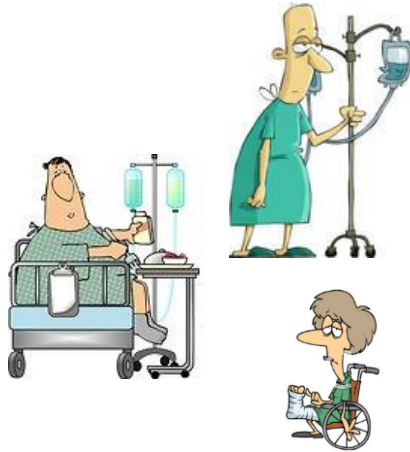
El desarrollo del sistema incluye la revisión de los circuitos administrativos y operativos, para la gestión eficiente y racional de la salud, en el marco de la reglamentación y normas de calidad vigentes.

El sistema va a estar conformado por tres subsistemas relacionados:

- ▶ *Gestión Administrativa*
- ▶ *Gestión Guardia e Internaciones*
- ▶ *Estadísticas*

CASO DE ESTUDIO: GESTIÓN DE UN HOSPITAL

Sistema de
Gestión del
Hospital



CASO DE ESTUDIO: GESTIÓN DE UN HOSPITAL

*Cada **paciente** es un **objeto del problema***



*Cada paciente tiene un nombre, una fecha de nacimiento, una obra social y una hoja de internación, estos son los **atributos** del objeto paciente.*

*Un paciente se interna, se da de alta, se interviene quirúrgicamente, estas acciones definen el **comportamiento** del objeto paciente.*

CASO DE ESTUDIO: GESTIÓN DE UN HOSPITAL

Cada **hoja de internación** también es un **objeto** que puede modelarse a través de un **conjunto de atributos** y un **comportamiento**.

Los **atributos** que modelan a una hoja de internación incluye la los controles de signos vitales y los estudios de diagnóstico.

El **comportamiento** de una hoja de internación incluye operaciones para registrar controles y estudios.



CASO DE ESTUDIO: GESTIÓN DE UN HOSPITAL

Cada **control de signos vitales** es también un **objeto** que puede modelarse a través de un **conjunto de atributos** y un **comportamiento**.

Los **atributos** que modelan a un control de signos vitales son la temperatura corporal y la presión arterial.

La **presión arterial** es también un **objeto** que puede modelarse con dos atributos, máxima y mínima. Ambos valores son números enteros.

En la gestión de un hospital cada **médico** es un objeto, como lo es también cada **enfermera** que realiza un control de signos vitales y cada **esfingomamómetro** con el que se realiza la medición.

EL CONCEPTO DE OBJETO

Un **objeto del problema** es una entidad, física o conceptual, que puede ser modelada a través de sus **atributos** y su **comportamiento**.

Cada **objeto del problema** en ejecución quedará asociado a un **objeto de software**.

Un **objeto de software** es un **modelo**, una representación abstracta de un objeto del problema.

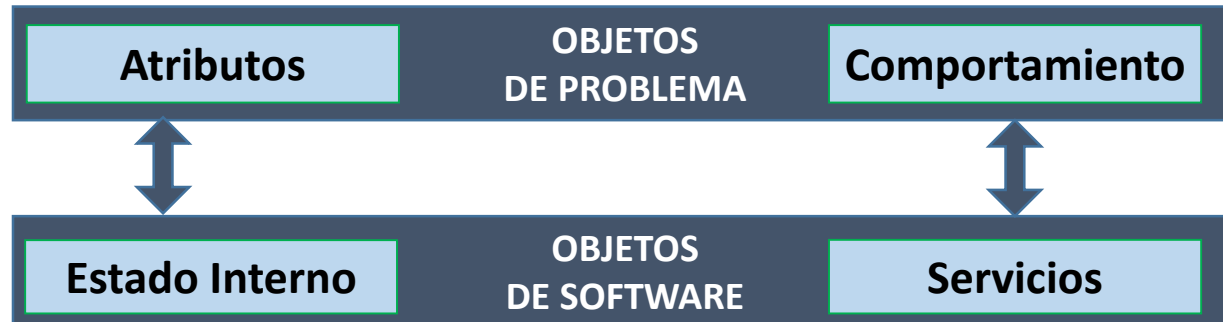
Un **objeto de software** tiene una **identidad** y mantiene los valores de los atributos en su **estado interno**.

El **comportamiento** queda determinado por un conjunto de **servicios** y un conjunto de **responsabilidades**.

EL CONCEPTO DE OBJETO

La palabra **objeto** se utiliza entonces para referirse a:

- Los **objetos del problema**, es decir, las entidades identificadas en las etapas de desarrollo de requerimientos y diseño.
- Los **objetos de software**, esto es, las representaciones que modelan **en ejecución** a las entidades del problema.



EL CONCEPTO DE CLASE

Los objetos del problema pueden agruparse en **clases** de acuerdo a sus **atributos** y **comportamiento**.

Todos los objetos de una clase van a estar modelados por un mismo **conjunto de atributos** y un mismo **comportamiento**.

Desde el punto de vista del diseño una **clase** es un **patrón** que establece los atributos y el comportamiento de un conjunto de objetos.

Un sistema de mediana escala puede modelarse a través de cientos o incluso miles de clases relacionadas entre sí.

EL DIAGRAMA DE UNA CLASE

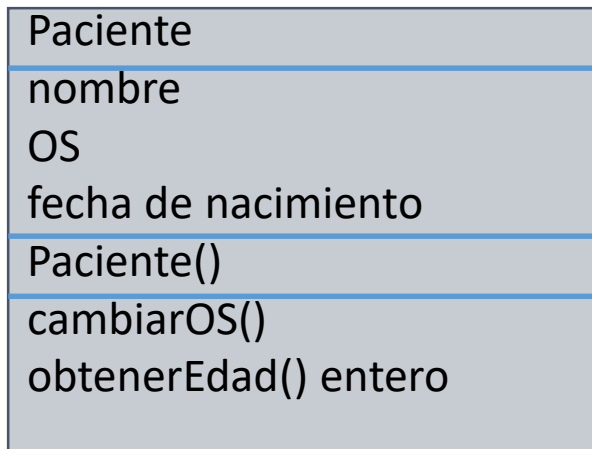
En la etapa de diseño cada clase se modela mediante un **diagrama**:

Nombre
Atributos
Constructores
Comandos
Consultas
Responsabilidades

Los servicios provistos por una clase pueden ser **constructores**, **comandos** o **consultas**.

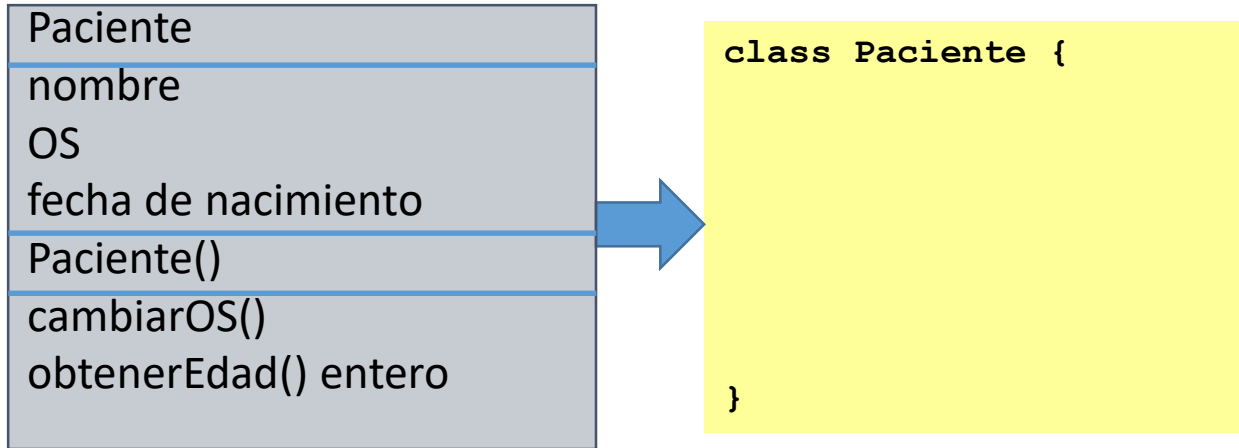
El diagrama puede incluir **notas** o **comentarios** que describen **restricciones** o la **funcionalidad** de los servicios.

EL DIAGRAMA DE UNA CLASE



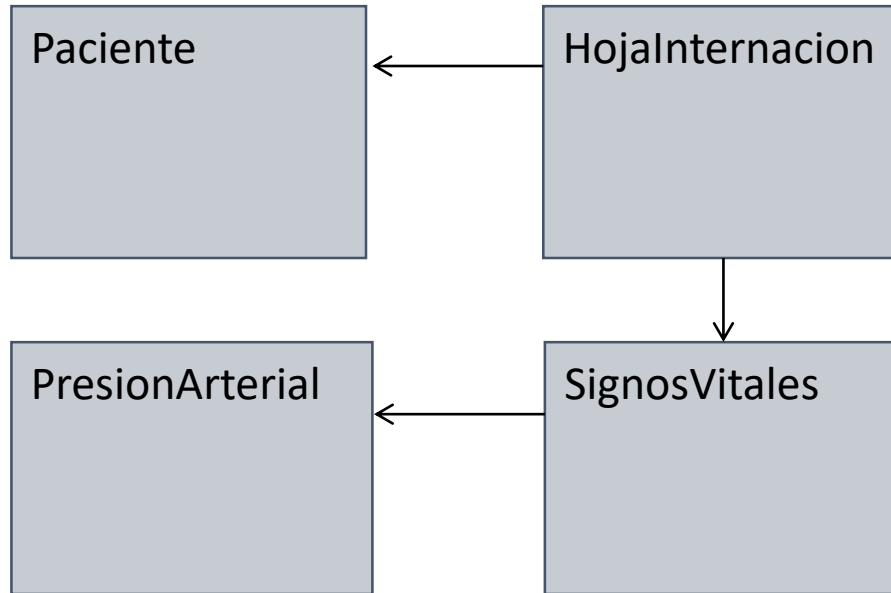
EL CÓDIGO DE UNA CLASE

En la etapa de implementación cada clase es una unidad de código:

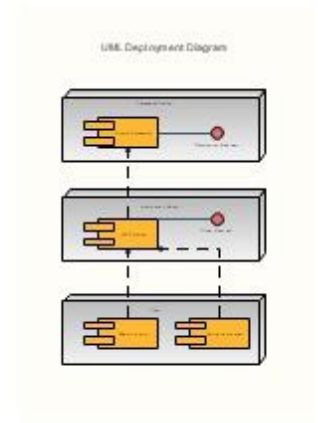
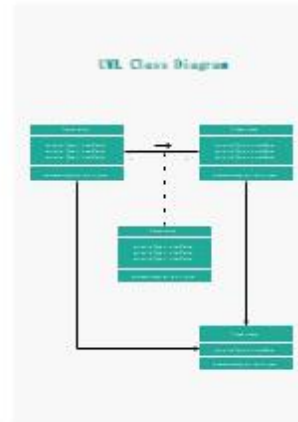
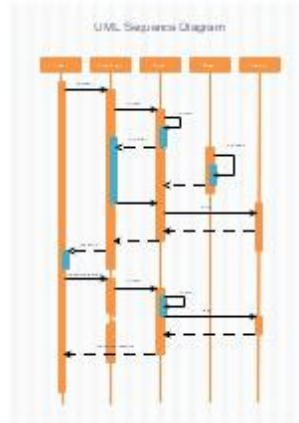
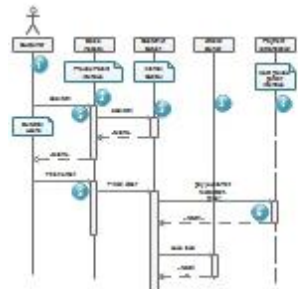
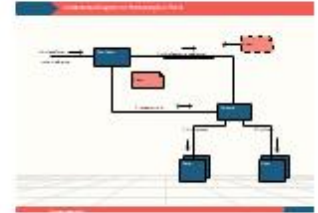
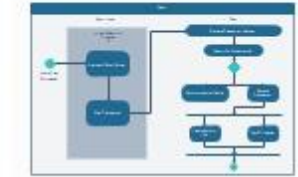
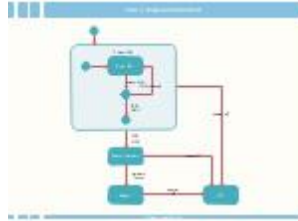


Así, cada clase **modelada** en lenguaje de modelado va a corresponderse a una clase **implementada** en lenguaje de programación.

DIAGRAMA DE CLASES



Diagramas



Paradigma Orientado a Objetos

- ▶ Es un paradigma de programación que usa los objetos en sus interacciones para diseñar aplicaciones y programas informáticos.
- ▶ Como otras metodologías de diseño orientados a la información, crean una representación del dominio del problema en el mundo real y lo transforma en un dominio de soluciones que es el software.
- ▶ Mas cercana a como expresariamos las cosas en la vida real que otros tipos de programación

Paradigma orientado a objetos

- ▶ Existen muchos conceptos en programación orientada a objetos, como clases y objetos, sin embargo, en el desarrollo de software con programación orientada a objetos, existen un conjunto de ideas fundamentales que forman los cimientos del desarrollo de software.
- ▶ 4 conceptos que vamos a ver les llamamos los 4 pilares de la programación orientada a objetos. Esto no quiere decir que fuera de estos 4 pilares no existan otras ideas igual de importantes, sin embargo, estos 4 pilares representan la base de ideas más avanzadas, por lo que es crucial entenderlos.
- ▶ Estos pilares son: **abstracción, encapsulamiento, herencia y polimorfismo**

En POO

- ▶ En cada problema vamos a **implementar** el código en Java de una colección de clases **modeladas** en un diagrama elaborado por un diseñador.
- ▶ Pero antes...

Clases y objetos

- ▶ Modelar 3 tipos de objetos:
 - Grupo musical
 - Botella
 - Auto

Y ahora...

- ▶ Veremos Java...

<https://roadmap.sh/java>

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Especificación de Requerimientos

*La **presión arterial** es la fuerza de presión ejercida por la sangre circulante sobre las arterias y constituye uno de los principales signos vitales de un paciente.*

Se mide por medio de un aparato, que usa la altura de una columna de mercurio para reflejar la presión de circulación.

La presión sistólica se define como el máximo de la curva de presión en las arterias y ocurre cerca del principio del ciclo cardíaco durante la sístole o contracción ventricular; la presión diastólica es el valor mínimo de la curva de presión en la fase de diástole o relajación ventricular del ciclo cardíaco.

La presión del pulso refleja la diferencia entre las presiones máxima y mínima medidas.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Especificación de Requerimientos

Estas medidas de presión no son estáticas, experimentan variaciones naturales entre un latido del corazón a otro y a través del día y tienen grandes variaciones de un individuo a otro.

La hipertensión se refiere a la presión sanguínea que es anormalmente alta, y se puede establecer un umbral para la máxima y otro para la mínima que permitan considerar una situación de alarma.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Diseño de una clase: Atributos

PresionArterial

<<Atributos de instancia>>

maxima, minima: entero

valores representados
en milímetros de
mercurio

Cada control de la presión arterial se modela con dos **atributos**: máxima y la mínima.

El diseñador resolvió no representar el pulso como un atributo.

¿Por qué?

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Diseño de una clase: Constructores

PresionArterial

<<Constructor>>

PresionArterial(ma, mi: entero)

requiere $ma > mi > 0$

El constructor tiene siempre el mismo nombre que la clase.

Es un servicio que se ejecuta cuando se crea un objeto de la clase.

El comentario establece una **restricción**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Diseño de una clase: Consultas

PresionArterial

<<Consultas>>

obtenerMaxima(): entero

obtenerMinima(): entero

Cada una de las dos consultas retorna un **resultado**.

En este caso, cada consulta retorna el valor de un atributo de instancia, se trata de consultas triviales, adoptamos la **convención** de usar la palabra **obtener** seguida del nombre del atributo.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Diseño de una clase: Consultas

PresionArterial

<<Consultas>>

obtenerPulso(): entero

alarmaHipertension(): boolean

obtenerPulso():
máxima-mínima

alarmaHipertensión:
maxima>140 o minima>80

Cada una de las dos consultas retorna un resultado computado a partir de los valores de los atributos.

Cada comentario especifica la **funcionalidad** de la consulta, esto es, como se computa el resultado.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Diseño de una clase: Responsabilidades

La clase PresionArterial forma parte de un sistema que incluye a otras clases.

En este ejemplo no incluye comandos ni una sección para las responsabilidades, pero uno de los comentarios establece una responsabilidad para las clases que **usan** a PresionArterial.

La clase PresionArterial **requiere** que cuando se crea un objeto, en el constructor el parámetro que representa el valor máxima es mayor que mínima y ambos mayores a 0.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

PresionArterial

<<Atributos de instancia>>

maxima, minima: entero

<<Constructores>>

PresionArterial(ma, mi: entero)

<<Consultas>>

obtenerMaxima(): entero

obtenerMinima(): entero

obtenerPulso(): entero

alarmaHipertension(): boolean

valores representados en
milímetros de mercurio

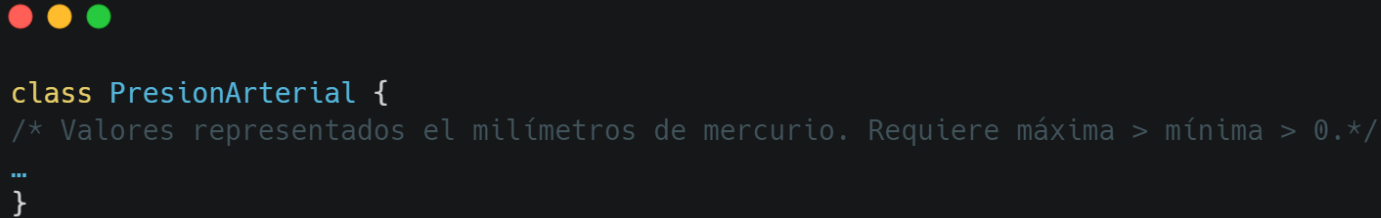
requiere $ma > mi > 0$

obtenerPulso():
máxima-mínima

alarmaHipertensión:
 $maxima > 140$ o $minima > 80$

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java



```
class PresionArterial {  
    /* Valores representados el milímetros de mercurio. Requiere máxima > mínima > 0.*/  
    ...  
}
```

La palabra **reservada** `class` está seguida por el nombre de la clase.

Las `{ }` son los **delimitadores** una unidad de código, existen otros delimitadores como los corchetes y los paréntesis.

Java es libre de la línea y sensible a las minúsculas y mayúsculas.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

```
class PresionArterial {  
    /* Valores representados el milímetros de mercurio. Requiere máxima > mínima > 0.*/  
    //Atributos de instancia  
    ...  
}
```

El símbolo `//` precede a un **comentario** de una línea.

Los símbolos `/* */` delimitan a un comentario de varias líneas.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial

<<atributos de instancia>>

maxima, minima: entero



```
//Atributos de instancia  
private int maxima;  
private int minima;
```

Cada **atributo de instancia** se declara como una **variable** de **tipo elemental int**.

El símbolo **;** **termina** cada instrucción.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial

<<Constructor>>

PresionAterial(ma, mi: entero)

requiere $ma > mi > 0$

```
//Constructor
public PresionArterial(int ma,int mi){
//Requiere ma > mi > 0
    maxima = ma;
    minima = mi;
}
```

El **constructor** asigna a cada atributo de instancia el valor de un **parámetro formal**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial

<<Consultas>>

obtenerMaxima():entero

obtenerMinima():entero



```
public int obtenerMaxima(){  
    return maxima; }
```

```
public int obtenerMinima(){  
    return minima; }
```

La instrucción **return** retorna el valor de un **atributo**, el tipo corresponde a la declaración del método.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial
<<Consultas>>
obtenerPulso():entero

```
public int obtenerPulso(){  
    //pulso= máxima-mínima  
    return maxima-minima;  
}
```

La instrucción **return** retorna el valor de una **expresión aritmética**, el tipo corresponde a la declaración del método.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial

<<Consultas>>

alarmaHipertension():boolean

```
public boolean alarmaHipertension(){  
    //maxima>140 o minima>80  
    return maxima > 140 || minima > 80;  
}
```

La consulta retorna como resultado un valor booleano que resulta de evaluar la expresión lógica.

El operador `||` denota disyunción con **cortocircuito**, si la primera subexpresión computa true, no se computa la segunda.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial

<<Consultas>>

alarmaHipertension():boolean

```
public boolean alarmaHipertension(){  
    //maxima>140 o minima>80  
    boolean b=maxima>140|| minima>80;  
    return b;  
}
```

Esta versión, equivalente a la anterior, utiliza una variable local booleana para retener y retornar el valor de la expresión computada.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

PresionArterial

<<Consultas>>

alarmaHipertension():boolean

```
public boolean alarmaHipertension(){  
    //maxima>140 o minima>80  
    boolean b = false;  
    if (maxima>140 || minima>80)  
        b = true;  
    return b;  
}
```


CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Implementación en Java

- ▶ Los atributos **no son visibles** desde el exterior de la clase porque los hemos declarado **privados**.
- ▶ Los métodos que no modifican los valores de los atributos se llaman **consultas**.
- ▶ El nombre de una consulta está precedido por el **tipo del resultado**.
- ▶ La consulta incluye una instrucción **return** seguida de una **expresión de tipo compatible** con el **tipo del resultado**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Convenciones

- ▶ Declaramos los atributos como privados para que solo sean accesibles dentro de la clase.
- ▶ Para **consultar** el valor de cada atributo definimos un **método** que retorna el valor del atributo.
- ▶ Cada identificador de clase comienza con una mayúscula, a las variables por lo general le asignamos nombres que comienzan en minúscula.
- ▶ Retenemos el orden y los comentarios del diagrama en el código para reflejar su estructura.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

```
class PresionArterial {  
  
    //Atributos de instancia  
    private int maxima;  
    private int minima;  
  
    //Constructor  
    public PresionArterial(int ma,int mi){  
        //Requiere ma > mi > 0  
        maxima = ma;  
        minima = mi;  
    }  
  
    public int obtenerMaxima()  
    {    return maxima;}  
  
    public int obtenerMinima()  
    {    return minima;}  
  
    public int obtenerPulso() {  
        // pulso=maxima-minima  
        return maxima-minima;}  
  
    public boolean alarmaHipertension(){  
        return maxima > 140 || minima > 80;}  
}
```

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

La clase **PresionArterial** forma parte de un sistema que modela la solución de un problema.

El sistema incluye a muchas otras clases, relacionadas entre sí, probablemente escritas por distintos programadores.

Antes de integrar la clase **PresionArterial** al sistema, el programador debe **verificar** que es correcta para un conjunto de **casos de prueba**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

La verificación no garantiza la correctitud, pero permite detectar algunos errores y depurarlos.


Definimos una clase **TestPresion** que **usa** a la clase **PresionArterial** y verifica sus servicios para algunos casos de prueba.

```
class PresionArterial {  
    ...  
}
```

```
class TestPresion {  
    public static void main (String a[]){  
        ...  
    }  
}
```

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación



```
class TestPresion {  
  
    public static void main (String a[]){  
        PresionArterial med1;  
        PresionArterial med2;  
  
        med1 = new PresionArterial (115,60);  
        med2 = new PresionArterial (110,62);  
  
        int p1 = med1.obtenerPulso();  
        int p2 = med2.obtenerPulso();  
  
        System.out.println ("Primera medición pulso "+p1);  
        System.out.println ("Segunda medición pulso "+p2);}  
}
```

La ejecución del programa va a comenzar con la activación del método **main**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

```
class TestPresion {  
    public static void main (String a[]){  
        PresionArterial med1;  
        PresionArterial med2;  
  
        med1 = new PresionArterial (115,60);  
        med2 = new PresionArterial (110,62);  
  
        int p1 = med1.obtenerPulso();  
        int p2 = med2.obtenerPulso();  
  
        System.out.println ("Primera medición pulso "+p1);  
        System.out.println ("Segunda medición pulso "+p2);}  
}
```

Se **declaran** dos **variables** de clase **PresionArterial**

La clase **TestPresion** **usa** a la clase **PresionArterial**

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

```
class TestPresion {  
  
    public static void main (String a[]){  
        PresionArterial med1;  
        PresionArterial med2;  
  
        med1 = new PresionArterial (115,60);  
        med2 = new PresionArterial (110,62);  
  
        int p1 = med1.obtenerPulso();  
        int p2 = med2.obtenerPulso();  
  
        System.out.println ("Primera medición pulso "+p1);  
        System.out.println ("Segunda medición pulso "+p2);}  
}
```

Se **crean dos objetos** de clase **PresionArterial**, cada uno de los cuales queda ligado a una variable.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

```
class TestPresion {  
  
    public static void main (String a[]){  
        PresionArterial med1;  
        PresionArterial med2;  
  
        med1 = new PresionArterial (115,60);  
        med2 = new PresionArterial (110,62);  
  
        int p1 = med1.obtenerPulso();  
        int p2 = med2.obtenerPulso();  
  
        System.out.println ("Primera medición pulso "+p1);  
        System.out.println ("Segunda medición pulso "+p2);}  
}
```

Se envía el **mensaje obtenerPulso ()** al objeto ligado a la variable **med1**, retorna un valor de tipo **int** que se asigna a la variable **p1**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

```
class TestPresion {  
  
    public static void main (String a[]){  
        PresionArterial med1;  
        PresionArterial med2;  
  
        med1 = new PresionArterial (115,60);  
        med2 = new PresionArterial (110,62);  
  
        int p1 = med1.obtenerPulso();  
        int p2 = med2.obtenerPulso();  
  
        System.out.println ("Primera medición pulso "+p1);  
        System.out.println ("Segunda medición pulso "+p2);}  
}
```

Se envía el **mensaje obtenerPulso ()** al objeto ligado a la variable **med2**, retorna un valor de tipo **int** que se asigna a la variable **p2**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

Verificación

```
class TestPresion {  
  
    public static void main (String a[]){  
        PresionArterial med1;  
        PresionArterial med2;  
  
        med1 = new PresionArterial (115,60);  
        med2 = new PresionArterial (110,62);  
  
        int p1 = med1.obtenerPulso();  
        int p2 = med2.obtenerPulso();  
  
        System.out.println ("Primera medición pulso "+p1);  
        System.out.println ("Segunda medición pulso "+p2);}  
}
```

Se produce una salida por **consola**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

JAVA

```
PresionArterial med1;  
PresionArterial med2;
```

Es equivalente a :

```
PresionArterial med1,med2;
```

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

JAVA

```
med1 = new PresionArterial (115,60);
```

```
public PresionArterial(int ma,int mi){  
    //Requiere ma > mi  
    maxima = ma;  
    minima = mi; }
```

Crea un **objeto** de clase **PresionArterial**

Se asigna espacio en memoria para almacenar los valores de los atributos y se invoca el constructor de la clase

PresionArterial

Los parámetros formales se inicializan con los valores de los parámetros reales.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

JAVA

```
int p1 = med1.obtenerPulso();
```

```
public int obtenerPulso() {  
    //pulso= máxima-mínima  
    return maxima-minima;  
}
```

Declara una variable **p1** de tipo **int** y le asigna el valor que resulta de computar la expresión a la derecha del símbolo **=**

El objeto ligado a la variable **med1** recibe el mensaje **obtenerPulso()**

En respuesta al mensaje el objeto ejecuta el método **obtenerPulso** y retorna un valor de tipo **int**.

CASO DE ESTUDIO: CONTROL DE LA PRESIÓN ARTERIAL

JAVA

```
System.out.println ("Primera medición pulso "+p1)
```

System es un objeto que recibe el **mensaje**
out.println.

El parámetro de este mensaje es una **cadena de caracteres**, esto es un objeto de la clase **String**, provista por Java.

La cadena se genera **concatenando cadenas de caracteres**.

Referencias y Constructores

- ▶ Se usa el constructor de la clase MiClase `a = new MiClase();` Todos los objetos son creados en el heap (memoria asignada dinámicamente durante la ejecución).
- ▶ Lo que se retorna es una referencia al nuevo objeto (puede ser pensada como un puntero). Java tiene un proceso de recolección de basura (Garbage Collection) que automáticamente recupera zonas no referenciadas.
- ▶ Si deseamos hacer algún tipo de limpieza antes de liberar el espacio de un objeto, la clase debería incluir un método con nombre `finalize()`. Éste se invoca justo antes de recolectar su memoria.

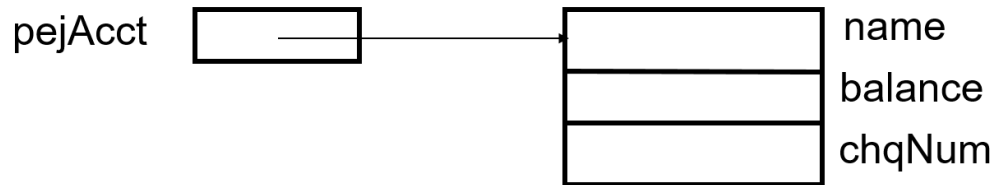
Identificador de objetos vs objetos

Cheque pejAcct;

pejAcct  // Referencia nula

pejAcct.deposit(1000000); // error

pejAcct = new Cheque("Peter", 1000, 40);



Este ejemplo asume que la clase Cheque ya existe y posee miembros datos (= atributos):
name, balance y chqNum

Manejo de referencias

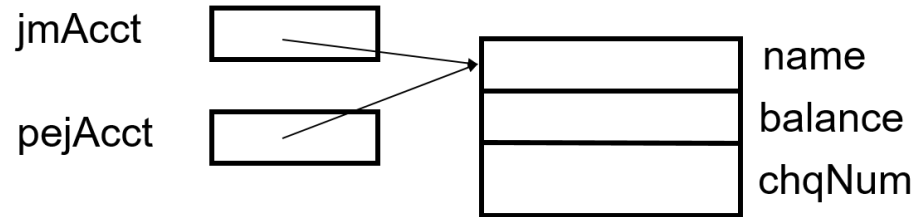
- Los identificadores de objetos son referencias.
- Referencia significa puntero (i.e. no el contenido)
 - ▶ = es copiar la referencia
- Usar método clone para crear copia del objeto completo (más adelante).
 - ▶ == es comparación de referencias
- Usar equals para comparar contenidos
 - objeto.unMetodo(pejAcct) pasa un referencia
 - objeto.unMetodo(tipo_básico) pasa el valor
 - return pejAcct retorna una referencia
- Si queremos retornar una copia, usar **clone()** para crearla y luego retornarla

Identificador de objetos vs objetos

Cheque jmAcct;

jmAcct 

jmAcct = pejAcct;

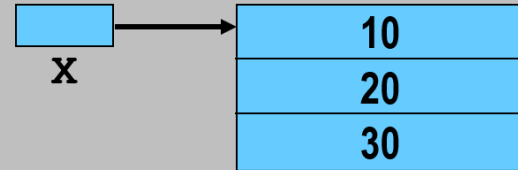


Referencias

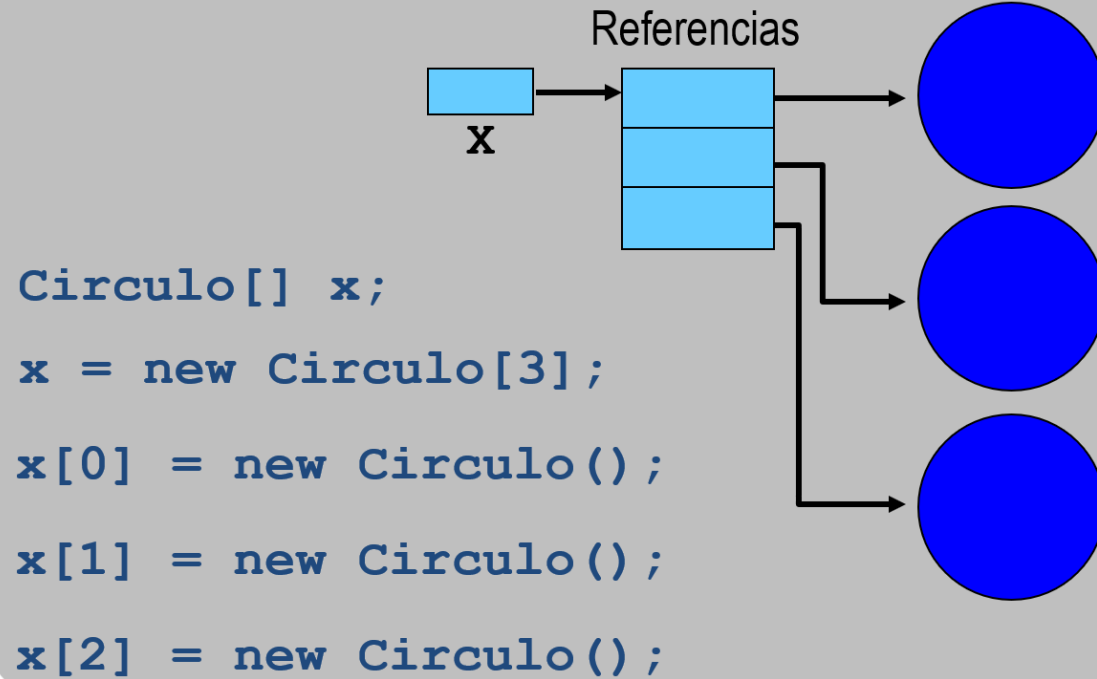
- ▶ Los objetos son referenciados
- ▶ Esta es una forma “controlada” de usar: Direcciones y punteros
- ▶ Al declarar una instancia de una clase obtenemos una referencia a esa instancia.
- ▶ Mientras no se asigne un objeto con new, su valor es null.
- ▶ En caso de tipos primitivos (8) se tiene la variable y acceso directo (no es referencia)

Referencias

```
int[] x; // equivalente a int x[]  
x = new int[3];  
  
x[0] = 10;  
x[1] = 20;  
x[2] = 30;
```



Referencias



Referencias

```
boolean[] respuestas = {true, false, true};
String[]  nombres    = {"Ana María", "Carlos"};
Circulo[]  circulos   = {
    new Circulo(),
    new Circulo(20),
    new Circulo(5.5)
};
String[][] humor = {
    { "Coco Legrand", "Alvaro Salas" },
    { "Les Luthiers" },
    { "Groucho Marx", "Buster Keaton",
      "Jerry Lewis", "Woody Allen" }
};
```

Pilares

The background of the slide is a light gray with a subtle pattern of overlapping hexagons and circles. Some hexagons are solid white, while others are white with a dark gray outline. Thin gray lines connect some of the hexagons, creating a network-like structure. The overall aesthetic is clean and modern.

Abstracción

- ▶ Desde el punto de vista del desarrollo de software, podemos ver que con una clase podemos realizar una abstracción de una entidad del mundo real.
- ▶ Tomemos por ejemplo la clase `PresionArterial` que hicimos ... ¿Por qué solamente la información que dijimos? Una medida del mundo real tiene más propiedades, como ... Sin embargo, debemos preguntarnos, ¿Son estas informaciones relevantes para nuestro software?

Encapsulación y ocultamiento

- ▶ La **encapsulación** se refiere al **ocultamiento de los datos miembros de un objeto**, es decir, **encapsular los atributos y métodos del objeto**, de manera que sólo se pueda cambiar mediante las operaciones definidas para ese objeto.
- ▶ Entonces la encapsulación es un **mecanismo de protección o aislamiento de atributos y métodos**, es decir, el aislamiento **protege a los datos asociados** de un objeto **contra su modificación** por quien no tenga derecho a acceder a ellos, **eliminando efectos secundarios e interacciones** en cuanto al ocultamiento de los datos miembros de un objeto.
- ▶ En otros términos, es la **capacidad de visibilidad de atributos y métodos de un objeto**, esta visibilidad va de acuerdo al **nivel de encapsulamiento**, tenemos tres niveles principales:

Encapsulación

- ▶ La encapsulación permite ocultar los datos y la funcionalidad de un objeto. Además, facilita la reutilización de objetos
- ▶ La interfaz de la clase de un objeto permite que otros objetos accedan a los atributos y a los métodos públicos de un objeto

Acoplamiento

- ▶ **¿Qué es el acoplamiento?**
- ▶ El acoplamiento es el grado en que los módulos de un programa **dependen** unos de otros.
- ▶ Si para hacer cambios en un módulo del programa es necesario hacer cambios en otro módulo distinto, existe acoplamiento entre ambos módulos.
- ▶ En Programación Orientada a Objetos, si una clase X usa una clase Y, se dice que X depende de Y. Esto es, X no puede realizar su trabajo sin Y, por lo tanto, existe acoplamiento entre las clases X e Y.
- ▶ Como se observa, el acoplamiento es direccional, puede haber acoplamiento de la clase X con la clase Y, pero esto no implica que exista en sentido inverso.

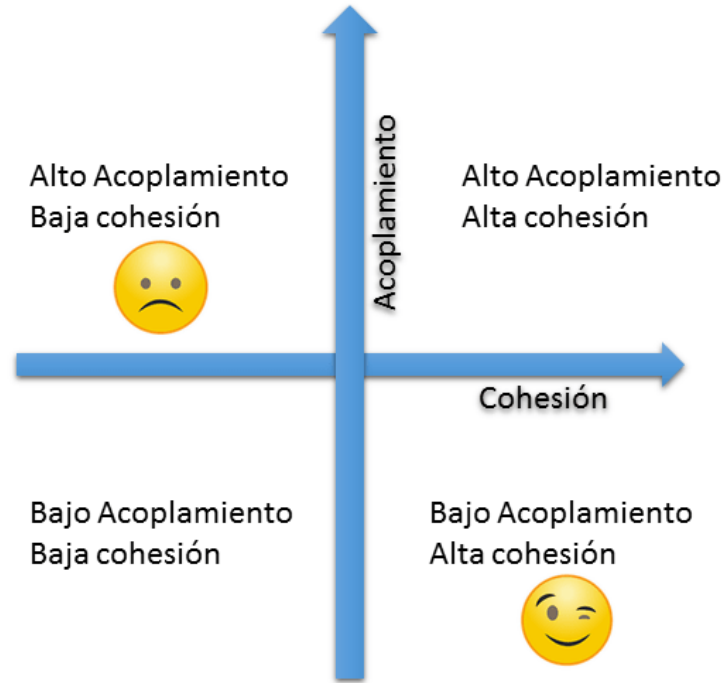
Cohesión

- ▶ La cohesión es la medida en la que un componente o clase realiza únicamente la tarea para la cual fue diseñada (Una clase debe de hacer lo que respecta a su entidad, y no hacer acciones que involucren a otra clase o entidad).
- ▶ En ingeniería del software, algo tiene alta cohesión si tiene un alcance definido, unos límites claros y un contenido delimitado y perfectamente ubicado.
- ▶ Si hablásemos de POO, una clase tendrá alta cohesión si sus métodos están relacionados entre sí, tienen un contenido claro y temática común, trabajan con tipos similares, etc. Todo bien encerrado dentro de la clase, y perfectamente delimitado.
- ▶ En resumen, **un código altamente cohesionado tiende a ser mas autocontenido y con menos dependencias.**

Acomplamiento vs Cohesión

- ▶ Acomplamiento y cohesión son dos conceptos que están relacionados entre sí. De forma natural, si en nuestro desarrollo buscamos una alta cohesión, obtendremos un bajo acomplamiento.
- ▶ El riesgo de centrarse sólo en la cohesión será llegar a construir un macrocomponente que termine siendo poco cohesionado y haga demasiadas cosas no relacionadas con el objeto (**responsabilidad**).
- ▶ Centrarse sólo en el acomplamiento también puede tener el riesgo de atomizar la estructura. Componentes muy simples y pequeños, con la responsabilidad bien delimitada, pero que acabaran reutilizándose en distintos contextos, por lo que afectarán a la cohesión del programa y empezarán a acoplarse entre módulos.
- ▶ Además, **una estructura altamente desacoplada será un código menos entendible**, menos legible, ya que para comprenderlo necesitamos explorar muchas referencias de distintos módulos.

Entonces....



Resumen

OBSERVACIONES

En esta clase

- ▶ Hemos mencionado algunos **términos conocidos**:
 - *programa, declaración, variable, expresión, tipo de dato elemental...*
- ▶ Hemos definido algunos **términos nuevos**:
 - *objeto, clase, atributo, servicio, constructor, método, consulta...*
- ▶ En las próximas clases nos concentraremos en comprender y vincular los conceptos asociados a estos nuevos términos e integrarlos con algunos de los conceptos aprendidos previamente.

RECOMENDACIONES

Entre la teoría y la práctica

- ▶ El aprendizaje de Java va a requerir **autonomía**.
- ▶ El objetivo es que **aprendan a aprender** un lenguaje de programación.
- ▶ Los prácticos 7 y 8 se proponen en la **primera semana** y están orientados a presentar los aspectos básicos del lenguaje y POO.

RECOMENDACIONES

- ▶ Accedan al ambiente IDE de su elección antes de la próxima clase e implementen las clases **PresionArterial** y **TestArterial**
- ▶ Destinen unos 20 minutos a leer esta presentación antes de la próxima clase teórica.
- ▶ Pueden avanzar en la resolución de los prácticos 7 y 8 en paralelo. Ambos deberían completarse en una semana.