Orientació a objectes amb PHP

PID_00243574

Manel Díaz Llobet

Temps mínim de dedicació recomanat: 5 hores

Universitat Oberta de Catalunya

Manel Díaz Llobet

Índex

Introducció

La programació ha experimentat una gran evolució des dels seus orígens fins al dia d'avui.

Els primers programadors s'havien d'enfrontar no sols amb la complexitat lògica dels programes que havien de desenvolupar, sinó també amb el maquinari específic de l'ordinador per al qual programaven. Encara més, un programa que funcionés en un ordinador, no ho havia de fer en un altre, ja que alguns llenguatges eren dependents del maquinari. Aquests llenguatges es van anomenar "**llenguatges assembladors**", perquè s'assemblaven, o sigui, traduïen instruccions del llenguatge assemblador a instruccions en el codi màquina, o sigui, el codi binari emprat pels processadors per a executar les instruccions.

Aviat es va superar aquest inconvenient, amb la invenció dels programes compiladors i intèrprets. La complexitat llavors "només" requeia en la lògica del programa.

Amb els primers compiladors i intèrprets, la programació es feia línia a línia, amb una instrucció anomenada GOTO, per a saltar a línies de codi allunyades de l'actual. Aquest sistema de programació feia molt complex el desenvolupament de programes mitjanament complexos, i va donar lloc al que es va anomenar "codi espagueti", per l'analogia d'un plat d'espagueti i l'execució d'un programa.

L'any 1968, Edsger Dijkstra va publicar una carta oberta anomenada "La sentència GOTO considerada perjudicial", que va establir les bases de la programació estructurada, o sigui, amb les estructures de programació bàsiques: seqüències, condicionals i bucles, a més de procediments i funcions.

La sentència GOTO

L'original en anglès de la sentència GOTO d'Edsger Dijkstra es pot trobar a: http://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf

La programació estructurada, junt amb la programació modular (dividir un programa en parts i ubicar cadascuna de les parts en fitxers diferents que es poden reutilitzar) i les bases de dades van fer que la programació de gestió arribés al gran públic. Això, unit al fet que el maquinari era cada vegada més potent i més barat, va donar lloc a una explosió exponencial de desenvolupaments de sistemes, cada vegada més grans i més complexos.

Es feia necessari un nou paradigma de desenvolupament d'aplicacions que fes més fàcil i ràpid tant el desenvolupament inicial dels sistemes com el posterior manteniment. A començaments de la dècada dels 90, es va començar a fer servir un estil de desenvolupament que oferia un alt grau de reutilització, de separació de tasques i que facilitava en gran mesura el treball en equip. Van ser els inicis de l'orientació a objectes.

1. El paradigma de l'orientació a objectes

1.1. Problemes de la programació estructurada

La programació estructurada va suposar un clar avenç envers la programació en codi assemblador, i va proporcionar moltes eines altament útils per a desenvolupar programes cada vegada més complexos.

Però va arribar un punt que la complexitat dels programes i sistemes que eren necessaris desenvolupar va superar el paradigma de la programació estructurada. El procés de desenvolupament i, sobretot, el procés de manteniment de les aplicacions es feia molt difícil, a causa sobretot de les fortes interaccions entre diferents parts del codi programat. A aquesta interacció entre diferents parts de codi se l'anomena "acoblament" del codi. Un alt nivell d'acoblament indica que, si canviem o substituïm una part del codi, és necessari canviar-ne moltes altres parts. Per a un bon manteniment de les aplicacions, haurien de tenir un baix nivell d'acoblament, que és una de les característiques que l'orientació a objectes intenta aconseguir.

El motiu principal d'aquest problema era que la programació estructurada fa una diferenciació molt forta entre el que són les dades i els processos necessaris per a gestionar-les. Encara més, a la programació estructurada les dades poden estar molt separades dels processos necessaris per a gestionar-les. Aquest és un enfocament potencialment conflictiu per a sistemes d'alta complexitat i/o abast.

1.2. Solucions de l'orientació a objectes

El paradigma de l'orientació a objectes es basa a gestionar les dades del programa amb "procediments" i "funcions" al més a prop possible de la definició de les dades, creant unes "càpsules" de funcionalitat que es poden fer servir per mòduls externs. Aquests mòduls externs, però, no poden accedir a la gestió de les dades ni als seus valors, excepte si ho fan de forma ordenada, segons s'ha definit en la creació d'aquesta "càpsula" de codi.

Mètodes

Els "procediments" i les "funcions" a l'orientació a objectes s'anomenen mètodes.

El principal (o un dels principals) avantatges de l'orientació a objectes és la **reutilització** de codi.

A la programació estructurada, quan es comença un nou projecte, cal programar-ho tot gairebé des de zero (tornar a programar el menú d'opcions, l'accés a la base de dades, etc.), menys el que es pugui reutilitzar fent servir la tècnica

del "copiar i enganxar", amb tots els problemes que això comporta. Una altra manera de reutilitzar en la programació estructurada és la inclusió de mòduls de codi (`#include`). A l'orientació a objectes, el codi és **reutilitzable**, perquè s'intenta aconseguir "peces" de codi, com les peces del motor d'un cotxe: si se n'espatlla una, se substitueix per una altra de compatible (que fins i tot pot ser més moderna i tenir més rendiment que la substituïda) i el motor funciona igual o millor que abans. De la mateixa manera, un **objecte** de codi pot substituir-ne un altre que contingui errors o que no funcioni (sempre que el nou objecte sigui compatible amb l'anterior, com les peces d'un cotxe). Fins i tot pot fer que el programa funcioni millor que amb l'objecte antic.

Els objectes són **càpsules** de codi tancades, és a dir, que es poden fer servir, però no es pot accedir al seu contingut (de la mateixa manera que un usuari pot fer servir un GPS, però no pot accedir a la seva electrònica interna).

Això comporta que aquestes càpsules de codi, programades per qualsevol programador, es puguin fer servir, o sigui, que puguin ser **reutilitzades** per tots aquells programadors que ho necessitin, de la mateixa manera que una peça del motor d'un cotxe es pot fer servir en motors d'altres cotxes.

A l'orientació a objectes, el desenvolupador no s'ha de preocupar de programar a baix nivell el sistema de menús. El motiu és que els botons, finestres i menús són objectes ja programats, provats i encapsulats. El desenvolupador només els ha de reutilitzar, escrivint el codi que gestioni el comportament d'aquell objecte en l'aplicació que s'està programant. Així, el desenvolupador es pot concentrar a programar la interfície de la seva aplicació, fent servir els components que el llenguatge proporciona i la lògica de la seva aplicació. Per exemple, si es vol programar una calculadora, el programador s'hauria de centrar a programar les operacions matemàtiques que ofereix aquesta calculadora, no a programar que un botó s'enfonsi en fer-hi clic. Aquest comportament del botó ja està programat internament en l'objecte "botó" i és comú a tots els botons. Aquest fet comporta uns altres grans avantatges de l'orientació a objectes.

> L'augment de la velocitat de desenvolupament d'aplicacions, una claredat més gran del codi font, la reducció del nivell d'acoblament, l'alta reutilització del codi i la facilitat de manteniment de les aplicacions són altres grans avantatges de l'orientació a objectes.

A més, la quantitat d'errors disminueix, ja que molt del codi que fa servir l'aplicació està prèviament programat i provat dintre dels objectes que es fan servir. Com a possibles desavantatges es podria dir que és una manera de programar diferent de la programació estructurada, i s'ha de fer un cert esforç per tal d'entendre-la, per a poder desenvolupar els objectes de forma que siguin al màxim de reutilitzables.

A més, aquest tipus de desenvolupament requereix un esforç més gran de disseny del codi i de plantejament de l'estructura del programa a realitzar. Això, lluny de ser un desavantatge, dóna peu a sistemes més robusts i eficients, en què la tasca de fer el manteniment del sistema no es converteix en un malson de mòduls plens de procediments i funcions entrellaçats entre si.

2. Conceptes bàsics de l'orientació a objectes

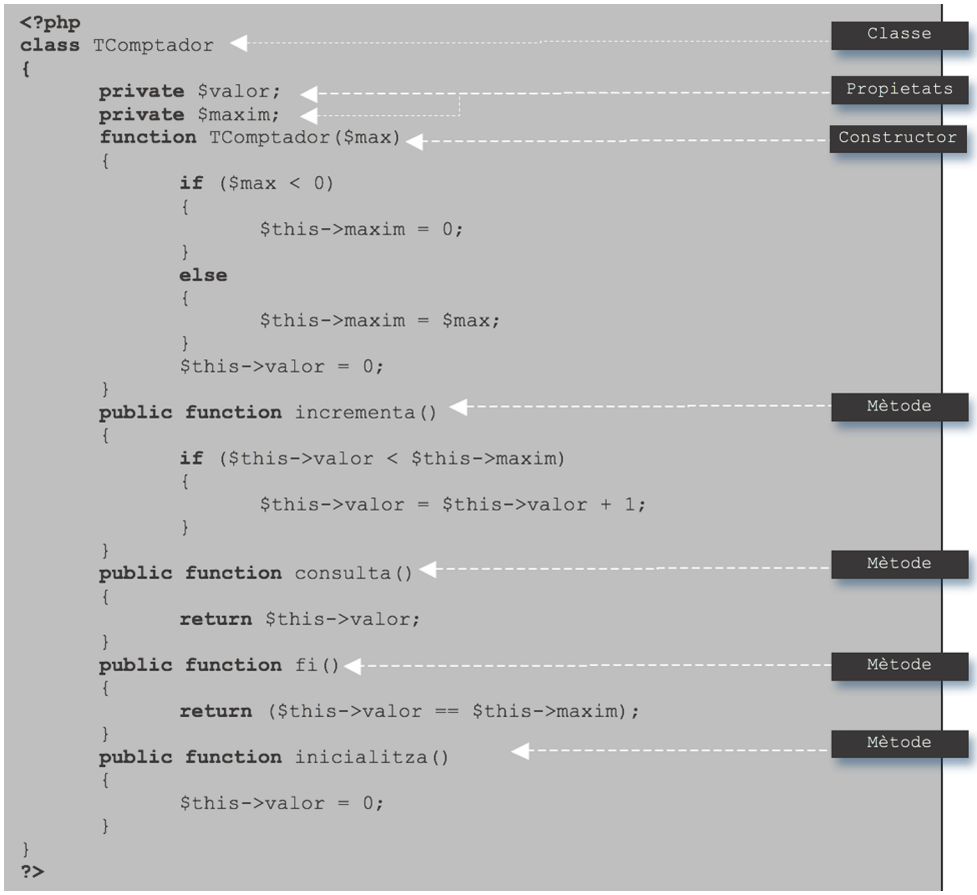
2.1. Classes i objectes

Un objecte és una entitat del món real, relacionada amb el problema que es vol resoldre, que es gestiona de forma autònoma i es relaciona amb altres entitats.

Aquestes entitats podran ser físiques (objectes que existeixen físicament, com per exemple una persona) o conceptuals (objectes que no existeixen físicament, com ara un llibre electrònic o el matrimoni entre dues persones). Un objecte contindrà tant les dades com els processos necessaris per a gestionar completament l'entitat.

Per exemple, gestionar un comptador a programació requereix certa informació, com ara el valor inicial, el valor final i el valor actual. Però no només això, sinó que un comptador cal que compti. Així, cal tenir un procés per a incrementar el comptador, un altre per a inicialitzar-lo i un altre que informi si s'ha arribat al final.

Així, l'objecte comptador es podria definir de la següent manera:




```php
<?php
class TComptador
{
	private $valor;
	private $maxim;
	function TComptador($max)
	{
		if ($max < 0)
		{
			$this->maxim = 0;
		}
		else
		{
			$this->maxim = $max;
		}
		$this->valor = 0;
	}
	public function incrementa()
	{
		if ($this->valor < $this->maxim)
		{
			$this->valor = $this->valor + 1;
		}
	}
	public function consulta()
	{
		return $this->valor;
	}
	public function fi()
	{
		return ($this->valor == $this->maxim);
	}
	public function inicialitza()
	{
		$this->valor = 0;
	}
}
?>
```

Nom de les classes

Tradicionalment el nom de les classes comença per la lletra T majúscula, perquè provenen dels antics Tipus abstractes de dades (*Type*). Es pot pensar en una classe com un tipus de dades, amb procediments associats, que han estat definits per l'usuari.

En realitat, el que s'acaba de definir és, per dir-ho d'alguna manera, una mostra del que hauria de ser un comptador. O sigui, que tots els comptadors que fem servir al nostre programa haurien de ser com aquest: amb dues variables per a guardar el valor actual i el màxim i cinc funcions per a poder funcionar.

El que s'acaba de descriure **no és un objecte**, és la definició de com haurien de ser tots els objectes que fessin la tasca de comptador. A aquesta definició se l'anomena "**classe**".

Definició

Classe: definició de les dades (propietats) que caracteritzen un conjunt d'objectes del mateix tipus i de les funcionalitats (mètodes) que calen per manipular-les.

En aquest cas, el nom de la classe és `TComptador`, tal com està indicat a la segona línia del codi anterior.

Llavors, que és un objecte? La resposta és molt simple: un objecte és la instanciació d'una classe. En instanciar una classe indiquem que l'objecte és una possible representació concreta de la classe i que, per tant, tindrà les funcionalitats de la classe i en concretarà els valors. Per exemple, cadascun dels comptadors que fem servir en un programa serà un objecte diferent de la classe `TComptador`.

Definició

Objecte: és la instanciació d'una classe. També se l'anomena "instància d'una classe".

Per exemple, es pot tenir aquest programa:

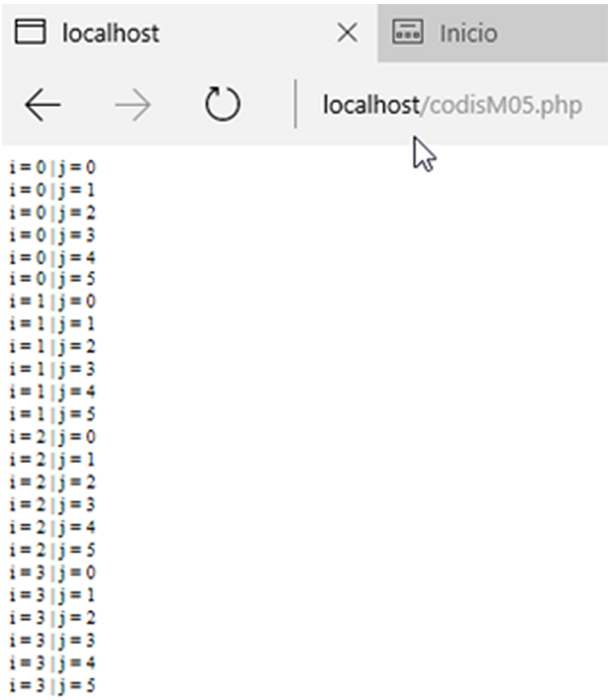
```
<?php
include_once ("TComptador.php");

$i = new TComptador(4);
$j = new TComptador(6);
while (!($i->fi()))
{
      while (!($j->fi()))
      {
         echo "i = " . $i->consulta() . " | j = "
                              . $j->consulta() . "<br>";
         $j->incrementa();
      }
      $j->inicialitza();
      $i->incrementa();
}
?>
```

Creació d'objectes

Utilització d'objectes

Com es pot veure, aquí hi ha definits dos objectes de la classe `TComptador` (`$i` i *`$j`*) que es fan servir a les instruccions del codi. Cadascun d'aquests objectes funciona independentment de l'altre, però tots dos tenen totes les variables i funcions que la classe `TComptador` ha definit. En aquest cas, `$i` és un comptador que comptarà de 0 fins a 3 i `$j` comptarà de 0 fins a 5.

Aquest seria el resultat de l'execució del codi anterior, amb la utilització de dos objectes de la classe `TComptador`.



2.2. Propietats / mètodes / esdeveniments

Durant tot l'exemple anterior s'ha dit que una classe té una sèrie de variables per a emmagatzemar la informació que li cal per a funcionar. A aquestes "variables" de la classe se les anomena "**propietats**" de la classe.

Analogia OO - Relacional

Es pot establir una analogia amb les entitats i atributs del model relacional. La classe seria com l'entitat del model relacional i les propietats de la classe serien com els atributs de l'entitat del model relacional. Un objecte d'una classe seria com una fila de dades en una taula, amb els valors de totes les propietats equivalents als valors de tots els atributs de la taula.

Definició

Propietat: una propietat és una característica pròpia de la classe. El conjunt de totes les propietats defineixen el que és una classe. El conjunt de tots els valors de les propietats d'un objecte d'una classe en un moment donat defineixen l'estat d'un objecte.

Així, una classe `TCotxe` tindrà les propietats "matrícula", "cilindrada", "marca" o "model", però no tindrà la propietat "cognoms". Aquesta propietat la tindrà una classe `TEstudiant`, per exemple.

Les propietats d'una classe poden prendre valors quan es declaren objectes d'aquella classe. Com s'ha dit abans, un objecte és la utilització d'una classe. Per a poder donar valor a una propietat, o sigui, per a utilitzar-la, cal definir un objecte de la classe.

A l'exemple anterior, la propietat *$valor* de qualsevol objecte s'incrementa en una unitat quan es crida el procediment `incrementa()`.

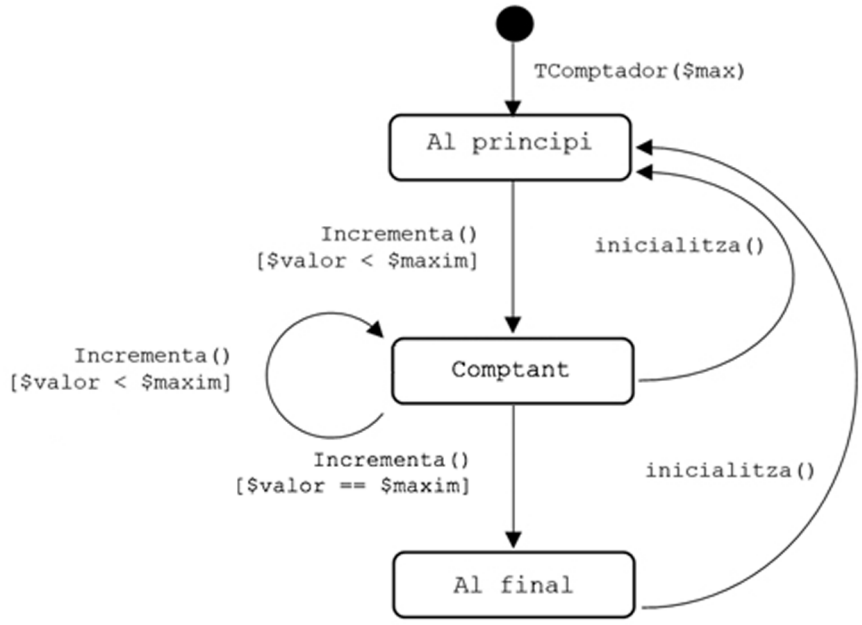
De la mateixa manera que una classe necessita variables per a emmagatzemar la informació (propietats), també necessita realitzar certes accions amb aquesta informació. Per a això, es defineixen procediments i funcions associats a la classe, que actuen sobre les propietats de la classe i sobre els paràmetres externs que es puguin definir. Cadascun d'aquests procediments i funcions s'anomenen "mètodes de la classe".

Definició

Mètode: un mètode és una funcionalitat d'una classe. És un procediment o una funció que gestiona algun aspecte de la classe. Els mètodes de la classe poden accedir a les propietats de la mateixa classe sense cap mena de restricció. El conjunt de tots els mètodes de la classe defineix les funcionalitats de la classe. Els mètodes d'una classe poden fer que la classe canviï el seu estat.

A la definició de propietat i a la de mètode s'hi ha introduït la idea d'estat d'una classe.

Què és l'estat d'una classe? L'estat d'una classe ve definit pels valors de les propietats de la classe en un moment donat, i varia amb l'execució de mètodes. En l'exemple del comptador, els estats de la classe serien els següents:



Com es pot veure, els diferents mètodes poden fer que l'estat de l'objecte canviï si es donen unes certes condicions. Els mètodes `consulta()` i `fi()` es poden cridar des de qualsevol dels tres estats ("`Al principi`", "`Comptant`" i "`Al final`") sense que canviï l'estat de l'objecte.

D'altra banda, el mètode incrementa(), quan s'executa a l'estat "`comptant`", es queda en el mateix estat si es compleix la condició `$valor < $maxim` i canvia a l'estat "`Al final`" si es compleix la condició `$valor == $maxim`.

També es pot veure com no des de tots els estats es poden cridar tots els mètodes. Des de l'estat "`Al final`" no es pot cridar el mètode `incrementa()`.

A més de propietats i mètodes, alguns objectes poden reaccionar a estímuls externs. Quan es fa clic en un botó, una finestra es tanca. Quan es passa el punter del ratolí per sobre d'una imatge, aquesta canvia, etc.

A aquests fets externs que fan que l'objecte canviï se'ls anomena "**esdeveniments**".

Definició

Esdeveniment: un esdeveniment és un fet que passa fora de l'objecte, que l'objecte pot detectar i al qual l'objecte pot reaccionar, executant algun mètode per a donar servei a l'esdeveniment.

Els esdeveniments s'apliquen sobretot a objectes visuals, o sigui, que es mostren per pantalla.

2.3. Interfície de la classe: privada, pública i protegida

Podeu connectar una memòria USB a l'entrada VGA de l'ordinador? Sabeu pilotar un Boeing 747?

La resposta a la primera pregunta és "no", i la resposta a la segona pregunta és "probablement, no". Per què?

Doncs perquè no es pot connectar una memòria USB al port VGA de l'ordinador perquè tenen **interfícies** incompatibles, o sigui, la connexió de la memòria USB i l'entrada del cable VGA són físicament diferents.

I no saps pilotar un Boeing 747 perquè no coneixes el funcionament del quadre de comandament de la cabina, o sigui, no coneixes la **interfície** de comunicació amb l'avió.

Definicions

Interfície: una interfície és el sistema de comunicació que s'estableix entre dues entitats. És la manera que té una entitat de dir-li a l'altra *què* ha de fer, però no *com* ho ha de fer.

Interfície de la classe: la interfície d'una classe a la programació orientada a objecte és el sistema de comunicació que s'estableix entre els objectes de la classe i el codi extern que els vol utilitzar. A cada part de l'objecte (propietats i mètodes) s'hi pot accedir per l'exterior o no, tot depèn de com es configuri. Al conjunt de propietats i mètodes de la classe que es pot accedir des de codi extern se l'anomena "interfície pública de la classe".

Fent una analogia amb el món real, el volant d'un cotxe, l'accelerador, el fre, el canvi de marxes i els interruptors serien part de la interfície pública del cotxe. Aquests són els artefactes que utilitzen les entitats externes (els conductors) per a comunicar-li al cotxe que és el que ha de fer (accelerar, frenar, girar...).

Però l'entitat externa al cotxe no li diu en cap moment *com* ho ha de fer per a accelerar, frenar o girar. Afortunadament és així, perquè si per a conduir un cotxe s'hagués de conèixer a fons la mecànica del motor o el sistema hidràulic de frenada, només conduirien els enginyers de la marca. Del fet de poder fer servir un objecte (un cotxe o un `TComptador`) sense haver de conèixer el seu funcionament intern se'n diu "**abstracció**".

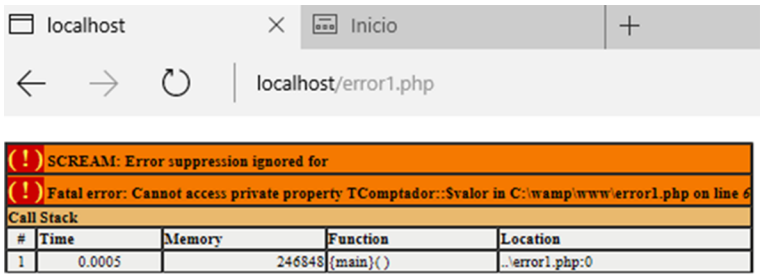
El concepte d'abstracció ha fet que el desenvolupament de sistemes informàtics avancés molt, ja que no és necessari conèixer com funciona internament un botó, una llista desplegable o un accés a una base de dades. Només cal conèixer com fer-los servir. Igual que conduir un cotxe.

A l'orientació a objectes, cada classe defineix la seva interfície, o sigui, com permet que elements externs interactuïn amb ella. Una classe pot mostrar o amagar el seu contingut. A l'exemple del comptador, la classe `TComptador` deixa que des de l'exterior es pugui cridar tots els seus mètodes, però no deixa que es pugui modificar ni tan sols consultar el valor de les seves dues propietats.

Si s'intenta executar el següent codi:

```php
<?php
      include_once ("TComptador.php");
   $i = new TComptador(4);
   echo "$i->valor = " . $i->valor;
?>
```

es genera un error com aquest:



Aquest error s'ha generat perquè des de fora de l'objecte s'ha intentat accedir a una propietat interna de l'objecte que no es troba a la seva interfície pública, o sigui que és privada.

Per a definir la interfície d'una classe, hi ha tres paraules reservades:

- **Públic**: (***PHP:*** *public*) totes les propietats i mètodes declarats com a públics pertanyen a la interfície pública de la classe i poden ser consultats i modificats des de codi extern a la classe.

- **Privat**: (***PHP:*** *private*) totes les propietats i mètodes declarats com a privats no pertanyen a la interfície pública de la classe i no poden ser consultats ni modificats pel codi extern a la classe (però si pel codi intern de la classe, o sigui, programat a dintre de la mateixa classe).

- **Protegit**: (***PHP:*** *protected*) totes les propietats i mètodes declarats com a protegits no pertanyen a la interfície pública de la classe, però poden ser consultats i/o modificats per objectes de classes heretades de la classe actual. (El concepte d'herència es veurà més endavant.)

Així, al que sigui declarat com a públic hi pot accedir tothom. El que sigui declarat com a privat, només s'hi pot accedir pel codi de la mateixa classe. I el que sigui declarat com a protegit, només s'hi pot accedir pel codi de la mateixa classe o dels seus descendents, o sigui, les classes filles de la classe actual.

Al codi de l'exemple, es pot veure que les dues propietats de la classe `TComptador` estan declarades com a privades, i els cinc mètodes de la classe estan declarats com a públics. És per això que des del programa principal es pot cridar els mètodes de cadascun dels objectes, però no es pot accedir a les propietats, tal com mostra l'error de l'exemple anterior.

2.4. Constructors i destructors

Quan es crea un objecte, és possible que s'hagin de reservar recursos per al seu funcionament, com per exemple definir un vector o obrir un fitxer. Aquests recursos es demanen al sistema operatiu per tal que l'objecte els faci servir durant la seva vida útil.

Quan un objecte ja no s'ha de fer servir més, cal que alliberi aquells recursos que ha demanat al començament, o sigui, que li retorni els recursos demanats al sistema operatiu perquè els faci servir un altre programa.

Hi ha dos mètodes especials en què es poden fer aquestes tasques: el constructor i el destructor.

Definició

Constructor: un constructor és un mètode especial d'una classe que s'executa automàticament en el moment de crear (instanciar) un objecte de la classe. Aquest mètode es pot fer servir, entre d'altres, per a inicialitzar propietats i/o reservar recursos perquè l'objecte els faci servir durant la seva vida útil.

Destructor: un destructor és un mètode especial d'una classe que s'executa (automàticament en alguns llenguatges) en el moment que un objecte ja no s'ha de fer servir més. Serveix principalment per a alliberar els recursos demanats en el constructor.

Constructors parametritzats

Els constructors poden tenir o no paràmetres a la capçalera. En el cas de l'exemple, el constructor té un paràmetre (`$max`) que indica el número màxim del comptador.

Els constructors que tenen paràmetres a la capçalera s'anomenen "constructors parametritzats".

És una bona pràctica de programació orientada a objectes fer servir, almenys, constructors per a inicialitzar les propietats de l'objecte. Si cal reservar algun recurs, gairebé és obligatori fer servir constructors i destructors.

A les versions de PHP anteriors a la 7, el constructor de la classe es podia definir com un mètode amb el mateix nom de la classe, sense indicar-ne la visibilitat (públic, protegit o privat) ja que un constructor *sempre* ha de ser públic (igual que un destructor) per a poder cridar-lo des de fora de la classe. Un exemple de constructor d'aquestes característiques es pot veure a l'exemple de codi del subapartat 2.1.

A partir de la versió PHP5, els constructors d'una classe tenen un nom unificat, o sigui, que no depèn de com es digui la classe. Per a ser compatible amb versions anteriors, es continua utilitzant la nomenclatura vella, o sigui, el nom de la classe com a nom del constructor, però aquesta pràctica es considera obsoleta a PHP7 i segurament es deixarà de mantenir en versions futures. Així que és millor fer servir la nomenclatura nova. En l'exemple del nostre `TComptador`, el constructor de la classe seria:

```
function __construct($max)
{
       echo "<br> Creant TComptador <br>";
       if ($max < 0)
       {
              $this->maxim = 0;
       }
       else
       {
              $this->maxim = $max;
       }
       $this->valor = 0;
}
```

Com es pot veure, la paraula clau per a definir el constructor és `__construct()`. També es pot observar la utilitat del constructor per a inicialitzar les propietats de l'objecte. En aquest cas, si el número màxim passat per paràmetre és negatiu, es posa a 0, i si és positiu, es guarda aquell valor. També s'assigna el valor de 0 com a valor actual del comptador.

En cas dels destructors és molt similar. El nom unificat dels destructors d'una classe és `__destruct()`. Per exemple, en el cas del comptador no tindria gaire sentit programar un destructor de la classe, ja que el constructor no demana recursos i el destructor no té res a alliberar, però per a veure que funciona es pot fer servir aquest exemple:

```
function __destruct()
{
       echo "<br> destruint TComptador <br>";
}
```

Quan aquest destructor s'executa, l'únic que fa és mostrar per pantalla un missatge. Sabem que un constructor s'executa quan es crea un nou objecte, amb la instrucció `$i = new TComptador(4)`, en el cas del nostre comptador. Però, quan s'executa un destructor?

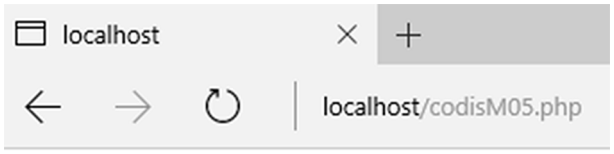
A PHP un destructor s'executa quan en el codi no hi ha cap més referència a un objecte. O sigui, després de la darrera instrucció on es faci servir `$i` (o `$j`, o qualsevol altre objecte), s'executarà el destructor de l'objecte.

Si modifiquem el constructor de la classe amb el codi del nou constructor i afegim el codi del destructor a la definició de la classe `TComptador` i executem el codi del subapartat 2.1., el resultat és aquest:

Crida als destructors

El comportament dels destructors a PHP no és el mateix que a molts altres llenguatges de programació.

En un gran nombre de llenguatges de programació cal cridar de forma explícita el destructor quan el programador considera que ja no es farà servir més l'objecte creat.




```
Creant TComptador

Creant TComptador
i = 0 | j = 0
i = 0 | j = 1
i = 0 | j = 2
i = 0 | j = 3
i = 0 | j = 4
i = 0 | j = 5
i = 1 | j = 0
i = 1 | j = 1
i = 1 | j = 2
i = 1 | j = 3
i = 1 | j = 4
i = 1 | j = 5
i = 2 | j = 0
i = 2 | j = 1
i = 2 | j = 2
i = 2 | j = 3
i = 2 | j = 4
i = 2 | j = 5
i = 3 | j = 0
i = 3 | j = 1
i = 3 | j = 2
i = 3 | j = 3
i = 3 | j = 4
i = 3 | j = 5

destruint TComptador

destruint TComptador
```

2.5. Herència i polimorfisme

Quan una marca de cotxes treu un nou model al mercat, és fruit d'un disseny i unes proves, s'han de fer unes línies de producció per a fabricar-lo en massa, etc.

Si mesos després la marca de cotxe treu al mercat el mateix model, però descapotable. Cal tornar a fer tot el disseny, les proves i refer totes les línies de producció des de zero? Doncs la resposta és, evidentment, que no.

El que es fa és aprofitar la plataforma desenvolupada pel primer model i modificar-la per a crear el segon model.

Aquest concepte també s'aplica a la programació orientada a objectes, i se'n diu "**herència**".

Definició

Herència: l'herència és la tècnica per la qual es permet reutilitzar les dades i mètodes d'una classe ja existent (A) en la creació d'una altra classe (B). És útil quan la classe B té moltes característiques comunes amb la classe A. La classe B només haurà de definir les noves funcionalitats i propietats que no es troben a la classe A. A la classe A se l'anomena "classe mare" i a la classe B, "classe filla" o "classe heretada". La classe heretada és una especialització de la classe mare, o sigui, hereta les propietats i el comportament de la classe mare, i les refina per a adaptar-se a les seves necessitats o especificitats. D'aquesta manera, es reutilitza una gran part del codi i les dades de la classe mare.

Les marques de cotxes també fabriquen cotxes amb canvi manual i amb canvi automàtic. Quan un cotxe accelera i cal canviar de marxa, el mecanisme per a fer-ho del manual és diferent del mecanisme de l'automàtic, però l'acció és la mateixa: canviar de marxa. A la programació orientada a objectes, a aquest fet se l'anomena "**polimorfisme**".

Definició

Polimorfisme: és una tècnica per la qual un mètode que s'executa ho fa d'una manera o una altra depenent del context, de forma transparent a l'usuari.

Hi ha moltes operacions polimòrfiques. Potser la més famosa de totes és la suma "+". Aquesta operació es comporta de forma diferent segons si es vol sumar nombres enters, caràcters o matrius:

$$2 + 3 = 5$$

'A' + 'B' = "AB"

$$\begin{vmatrix} 5 & 6 \\ 2 & 5 \end{vmatrix} + \begin{vmatrix} 4 & 2 \\ 1 & 1 \end{vmatrix} = \begin{vmatrix} 9 & 8 \\ 3 & 6 \end{vmatrix}$$

Com a exemple, farem una segona classe que s'anomenarà `TComptaSalt`, que serà com `TComptador`, però, en lloc d'incrementar en una unitat, incrementarà segons un valor que l'usuari li indiqui. Per a desenvolupar aquesta nova classe, reaprofitarem la classe `TComptador`.

En aquest cas, `TComptador` serà la classe mare i `TComptaSalt` serà la classe filla o heretada.

A més, aquesta nova classe tindrà un mètode nou, `posar_salt($increment)`, que servirà per a indicar a la classe l'increment de cada volta. El comportament del mètode incrementa de `TComptaSalt` és diferent del de `TComptador`, ja que a `TComptaSalt` no s'incrementarà d'un en un. Però el nom del mètode serà el mateix tant a `TComptador` com a `TComptaSalt`. El mètode `incrementa` és **polimòrfic**.

```
class TComptaSalt extends TComptador
{
      private $salt;
      public function posar_salt ($increment)
      {
            if ($increment <= 0)
            {
                  $this->salt = 1;
            }
            else
            {
                  $this->salt = $increment;
            }
      }
      public function incrementa ()
      {
            if ($this->valor + $this->salt < $this->maxim)
            {
                  $this->valor = $this->valor +
                                 $this->salt;
            }
            else
            {
                  $this->valor = $this->maxim;
            }
      }
}
```

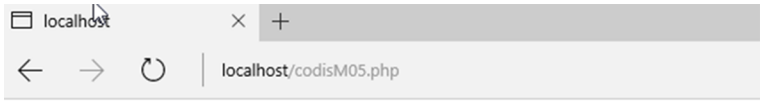
Com es pot veure a l'exemple, per a fer una classe heretada cal posar després del nom de la nova classe la paraula reservada `extends` i el nom de la classe mare.

Dels dos mètodes de la classe, el primer (`posar_salt`) és nou i es programa com qualsevol altre mètode. El segon (`incrementa`) és una redefinició del mètode `incrementa` de la classe mare. Aquest mètode incrementa de la nova classe té un comportament diferent del de la classe mare; per tant, és un mètode polimòrfic. Per a definir mètodes polimòrfics, la capçalera del mètode de la classe heretada ha de coincidir amb la capçalera del mètode de la classe mare. O sigui, que si a la classe mare el mètode no té paràmetres, a la classe filla tampoc.

Si s'executa el següent codi:

```php
<?php
include_once ("TComptador.php");
$i = new TComptador(4);
while (!($i->fi()))
{
	echo "i = " . $i->consulta() . "<br>";
	$i->incrementa();
}
$s = new TComptaSalt(10);
$s->posar_salt(3);
while (!($s->fi()))
{
	echo "s = " . $s->consulta() . "<br>";
	$s->incrementa();
}
?>
```

El resultat és:



Creant TComptador
i = 0
i = 1
i = 2
i = 3

Creant TComptador
s = 0

Notice: Undefined property: TComptaSalt::$valor in C:\wamp\www\TComptador.php on line 68

Call Stack

#	Time	Memory	Function	Location
1	0.0006	250912	{main}( )	..\codisM05.php:0
2	0.0034	269272	TComptaSalt->incrementa( )	..\codisM05.php:14

Notice: Undefined property: TComptaSalt::$maxim in C:\wamp\www\TComptador.php on line 68

Call Stack

#	Time	Memory	Function	Location
1	0.0006	250912	{main}( )	..\codisM05.php:0
2	0.0034	269272	TComptaSalt->incrementa( )	..\codisM05.php:14

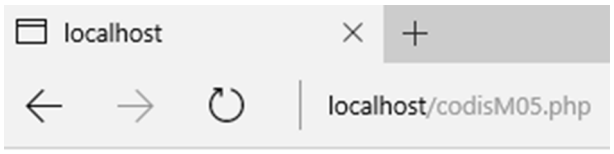
Que ha passat? El primer comptador, de la classe `TComptador`, s'ha executat correctament.

Però el segon comptador, de la classe `TComptaSalta`, ha generat aquests errors. Si els analitzem amb una mica més de profunditat, es pot veure que les propietats `$valor` i `$maxim` no estan definides.

El motiu és que aquestes propietats estan definides com a **privades** a la classe mare. Per tal que la classe filla les hereti, per a poder-les fer servir cal modificar la visibilitat de les propietats de la classe mare i posar-les com a `protected`:

```php
<?php
class TComptador
{
	protected $valor;
	protected $maxim;
.
.
```

Si després de fer aquest canvi s'executa l'anterior codi, el resultat és:




```
Creant TComptador
i = 0
i = 1
i = 2
i = 3

Creant TComptador
s = 0
s = 3
s = 6
s = 9

destruint TComptador

destruint TComptador
```

Com es pot veure, es crea el primer comptador que fa la seva feina –amb la crida al mètode `$i->incrementa()`–. En aquest cas, el mètode incrementa que s'executa és el de la classe mare, o sigui, de `TComptador` (incrementa d'1 en 1).

Després es crea el segon comptador, que també fa la seva feina –també amb la crida a `$s->incrementa()`–, però en aquest cas el mètode que realment s'executa és el de la classe filla, `TComtaSalta` (incrementa de 3 en 3, com s'ha indicat amb `$s->posar_salt(3)`.

3. Utilització de l'orientació a objectes a la gestió de dades

A la programació orientada a objectes, el fet de decidir quines classes crear i de què s'ha d'encarregar cadascuna de les classes és una feina difícil. S'ha de valorar que les classes siguin reutilitzables, que tinguin un baix nivell d'acoblament amb la resta, que no siguin gaire grans (per a augmentar la modularitat) ni gaire petites per a no tenir una gran quantitat de classes per a gestionar.

Nivell o grau d'acoblament

El grau o nivell d'acoblament d'una classe es pot definir com la dependència que té una classe respecte a la resta de classes amb les quals es relaciona.

Si aquesta dependència es fa a través de la interfície de la classe, el nivell d'acoblament és baix. Si la dependència es fa a força de modificar propietats públiques de la classe, el nivell d'acoblament és alt.

Si es fon una bombeta, es canvia per una altra amb la mateixa interfície (rosca, potència, voltatge, etc.). Nivell d'acoblament baix.

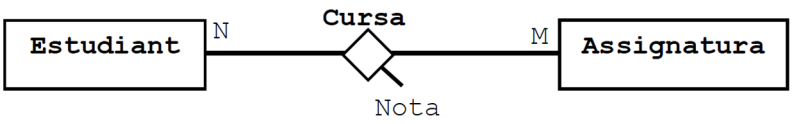
Si ens canviem el número de telèfon, cal informar a tothom del nou número. Nivell d'acoblament alt.

Hi ha diverses maneres de decidir les classes que cal crear per a una aplicació de gestió de dades. L'explicació d'aquestes diferents tècniques queda fora de l'abast d'aquest mòdul didàctic. Però el que sí que s'explicarà és *una* possible manera d'organitzar les classes que es necessitin, de forma que el sistema informàtic resultant sigui fàcil de mantenir, amb un mínim nivell d'acoblament i amb una mida de classes raonable.

3.1. Quines classes s'han de crear?

Per a decidir quines classes s'han de crear cal fixar-se en el model físic de dades, en concret en la creació de taules. Es parteix de la base que cada classe gestionarà una taula de la base de dades.

Així, si tenim aquesta base de dades d'exemple:




```
Estudiant (DNI, nom, edat)
Assignatura (codi, nom, crèdits)
Cursa (DNI, codi_assig, nota)
Foregin key(DNI) references Estudiant(DNI)
Foreign key(codi_assig) references Assignatura(codi)
```

```
create table estudiant
(
   DNI char(9),
   nom varchar (100),
   edat integer,
primary key (DNI)
);

create table assignatura
(
   codi char(5),
   nom varchar(50),
   credits integer,
primary key (codi)
);

create table cursa
(
   DNI char(9),
   codi_assig char(5),
   nota integer,
   primary key (DNI,codi_assig),
   foreign key (DNI) references estudiant(DNI),
   foreign key (codi_assig) references assignatura (codi)
);
```

Suposem que la gestió que es vol fer d'aquesta base de dades té les següents funcionalitats:

1) Cal poder donar d'alta estudiants a la base de dades.

2) Cal poder assignar i desassignar assignatures als estudiants de la base de dades. Quan s'assigna una assignatura a un estudiant, encara no té nota (nota a `NULL`). No es pot desassignar una assignatura d'un alumne si aquest té una nota posada.

3) Cal poder posar nota a una assignatura d'un estudiant donat. La nota és un nombre natural de 0 a 10.

4) Cal poder donar de baixa un estudiant de la base de dades, sempre que aquest estudiant no estigui cursant cap assignatura.

5) Tenint en compte el codi d'una assignatura, cal mostrar el seu nom i nombre de crèdits.

6) Cal que es pugui mostrar un llistat de tots els estudiants que cursen una determinada assignatura mirant el codi de l'assignatura, amb el DNI i el nom de l'estudiant, el nombre de crèdits i el nom de l'assignatura i la nota que han obtingut.

Aquestes serien les especificacions que es donen per a desenvolupar una aplicació web. A partir d'aquestes especificacions, el primer pas serà crear les classes corresponents a les taules de la base de dades. Com que hi ha tres taules, es crearan tres classes:

1) Classe `TEstudiant`:

```
<?php
class TEstudiant
{
      private $DNI;
      private $nom;
      private $edat;

      function __construct($v_dni, $v_nom, $v_edat)
      {
            $this->DNI = $v_dni;
            $this->nom = $v_nom;
            $this->edat = $v_edat;
      }
?>
```

2) Classe `TAssignatura`:

```
<?php
class TAssignatura
{
 private $codi;
 private $nom;
 private $credits;

 function __construct($v_codi, $v_nom, $v_credits)
 {
      $this->codi = $v_codi;
      $this->nom = $v_nom;
      $this->credits = $v_credits;
 }
?>
```

3) Classe `TCursa`:

```php
<?php
class TCursa
{
	private $DNI;
	private $codi_assig;
	private $nota;

	function __construct($v_dni, $v_codi_assig)
	{
		$this->DNI = $v_dni;
		$this->codi_assig = $v_codi_assig;
		$this->nota = NULL; //al començament no té nota
	}
?>
```

Com es pot veure, el primer pas per a crear les classes és posar-los el nom que els correspon (cada classe amb el nom que indica de quina taula es fa responsable). També es defineixen, mitjançant propietats privades, els atributs de la taula. El constructor de la classe haurà de permetre assignar els valors passats com a paràmetre a les propietats de la classe.

3.2. Quin contingut té cada classe?

Una vegada les classes estan creades, amb les seves propietats i el constructor definit, cal preocupar-se pels mètodes que tindrà cada classe. Per a assignar-los, és necessari mirar les especificacions de l'aplicació que s'ha de desenvolupar i decidir, per cada funcionalitat, quina classe serà la responsable d'implementar-la.

Un dels mètodes existents per a prendre aquesta decisió és fixar-se en la darrera instrucció SQL que caldrà executar per a implementar la funcionalitat. Per exemple:

1) Cal poder donar d'alta estudiants a la base de dades. Quina classe serà l'encarregada de fer tot el que sigui necessari per a "*poder donar d'alta estudiants a la base de dades*"? En altres paraules, quina instrucció SQL caldrà executar després de fer les comprovacions pertinents per a poder donar d'alta un estudiant a la base de dades? Doncs un `INSERT INTO ESTUDIANT VALUES` (...) La taula afectada serà la d'Estudiants. Així, la classe encarregada de donar d'alta nous estudiants serà `TEstudiant`.

Si seguim el mateix mètode amb la resta de funcionalitats, tenim:

2) Cal poder assignar i desassignar assignatures als estudiants de la base de dades.

SQL: `INSERT INTO CURSA VALUES (...)` i `DELETE FROM CURSA WHERE...`

Classe encarregada: `TCursa`

3) Cal poder posar nota a una assignatura d'un estudiant donat. La nota és un nombre natural de 0 a 10.

SQL: `UPDATE CURSA SET NOTA = ... WHERE ...`

Classe encarregada: `TCursa`

4) Cal poder donar de baixa un estudiant de la base de dades, sempre que aquest estudiant *no* estigui cursant cap assignatura.

SQL: `DELETE FROM ESTUDIANT WHERE ...`

Classe encarregada: `TEstudiant`

5) Tenint en compte el codi d'una assignatura, cal mostrar el seu nom i nombre de crèdits.

SQL: `SELECT NOM, CREDITS FROM ASSIGNATURA WHERE CODI = ...`

Classe encarregada: `TAssignatura`

6) Cal que es pugui mostrar un llistat de tots els estudiants que cursen una determinada assignatura mirant el codi de l'assignatura, amb el DNI i el nom de l'estudiant, el nombre de crèdits i el nom de l'assignatura i la nota que han obtingut.

SQL: `SELECT E.DNI, E.NOM, A.NOM, A.CREDITS, C.NOTA FROM (`**`ESTUDIANT`** `E INNER JOIN` **`CURSA`** `C ON E.DNI = C.DNI) INNER JOIN` **`ASSIGNATURA`** `A ON C.CODI_ASSIG = A.CODI WHERE A.CODI = ...`

Classe encarregada: com es pot veure a la sentència SQL, és una consulta amb dos *inner joins*. Aquí podríem triar qualsevol de les tres taules, però com que el WHERE restringeix les dades en funció del codi de l'assignatura, hem decidit assignar la funcionalitat a la classe associada a la taula assignatura. Al cap i a la fi, el llistat d'estudiants que es mostra és per a una assignatura.

D'altra banda, fent una anàlisi més profunda del problema (que s'hauria de fer en un disseny previ), es pot arribar a la conclusió que seria necessari crear un mètode que informés de si un estudiant (o una assignatura) ja són a la base de dades. Aquests mètodes seran *existeix_estudiant* a *TEstudiant*, *existeix_assignatura* a *TAssignatura* i *existeix_assignacio* a *TCursa*.

Una vegada feta aquesta anàlisi de les funcionalitats, ja podem escriure el codi de cadascuna de les classes i les capçaleres de tots els mètodes. La implementació seria la següent:

1) Classe `TEstudiant` amb capçaleres de mètodes.

```php
<?php
class TEstudiant
{
	private $DNI;
	private $nom;
	private $edat;

	function __construct($v_dni, $v_nom, $v_edat)
	{
		$this->DNI = $v_dni;
		$this->nom = $v_nom;
		$this->edat = $v_edat;
	}
	public function existeix_estudiant()
	{
		$res = false;
		return $res;
	}
	public function alta_estudiant()
	{
		$res = false;
		return $res;
	}
	public function baixa_estudiant()
	{
		$res = false;
		return $res;
	}
}?>
```

2) Classe **`TAssignatura`** amb capçaleres de mètodes.

```php
<?php
class TAssignatura
{
	private $codi;
	private $nom;
	private $credits;

	function __construct($v_codi, $v_nom, $v_credits)
	{
		$this->codi = $v_codi;
		$this->nom = $v_nom;
		$this->credits = $v_credits;
	}
	public function existeix_assignatura()
	{
		$res = false;
		return $res;
	}
	public function consultar_assignatura (&$v_nom,
	&$v_credits)
	{
		$res = false;
		return $res;
	}
	public function llistat_estudiants()
	{
		$res = false;
		return $res;
	}
}?>
```

3) Classe **`TCursa`** amb capçaleres de mètodes.

```php
<?php
class TCursa
{
	private $DNI;
	private $codi_assig;
	private $nota;
	function __construct($v_dni, $v_codi_assig)
	{
		$this->DNI = $v_dni;
		$this->codi_assig = $v_codi_assig;
		$this->nota = NULL; //al començament no té nota
	}
	public function existeix_assignacio()
	{
		$res = false;
		return $res;
	}
	public function assignar ()
	{
		$res = false;
		return $res;
	}
	public function desassignar()
	{
		$res = false;
		return $res;
	}
	public function posar_nota ($v_nota)
	{
		$res = false;
		return $res;
	}
}?>
```

Com es pot veure, a tots els mètodes hi ha definida una variable `$res` (de resultat) que sempre es retorna. L'objectiu d'aquesta variable és comprovar si s'ha produït un error en realitzar l'operació demanada. És un control d'error molt simple, però cal fer algun control, per tal d'informar l'usuari si s'arriba a produir algun error.

Hi ha mètodes sense paràmetres i d'altres amb paràmetres. En els casos on no hi ha paràmetres, els valors necessaris s'obtenen a partir de l'estat de l'objecte. És a dir, els paràmetres ja s'han passat al constructor de la classe. Un exemple d'això són els mètodes `alta_estudiant` o `baixa_estudiant` de `TEstudiant`.

Un altre mètode amb paràmetres és el de `consultar_assignatura`, de `TAssignatura`. A aquest mètode s'hi passen dos paràmetres que ja es troben en les propietats de la classe. Això és així perquè aquest mètode ha de retornar dues informacions: el nom de l'assignatura i el nombre de crèdits. Com que les propietats són privades, hi ha d'haver algun mecanisme per a extreure aquesta informació de la classe. En aquest cas, el mecanisme triat és el pas de paràmetres **per referència**.

I el darrer mètode a comentar és el de `posar_nota` de `TCursa`. A aquest mètode no se li passa la nota quan es crida el constructor, perquè un estudiant al començament no té nota. Quan es cridi `assignar()`, només caldrà el DNI de l'estudiant i el codi de l'assignatura. Quan es cridi `desassignar()`, cal-

drà comprovar que la nota sigui NULL (que no tingui nota) i quan es cridi `posar_nota()` és quan realment l'estudiant tindrà una nota en aquella assignatura.

Una vegada s'ha creat l'estructura de classes, es pot començar a implementar la funcionalitat de cadascun dels mètodes. És una bona pràctica començar per aquells mètodes de classes que gestionin taules provinents d'entitats (`TEstudiant` i `TAssignatura`, en aquest cas) ja que seran les que necessitaran estar implementades abans de poder implementar les classes que gestionen taules provinents de relacions (`TCursa` en aquest cas). Si no tenim ni alumnes ni assignatures a la base de dades, com ho farem per a assignar-los?

Començarem per implementar el mètode d'alta d'estudiants de `TEstudiant`. En l'anàlisi que s'ha fet dels mètodes necessaris per a realitzar les funcionalitats, s'ha arribat a la conclusió que per a cada mètode dels que haurem d'implementar s'haurà d'executar una sentència SQL. Per a poder executar aquesta sentència, caldrà fer un accés a la base de dades, establint la connexió, realitzant la consulta, obtenint les dades retornades o el resultat de la instrucció, etc.

4. Lectura complementària: el model MVC

4.1. Introducció

En aplicacions petites, fent la creació de les classes necessàries per a gestionar les taules de la base de dades, n'hi haurà prou per a fer tot el seu desenvolupament. Això passarà difícilment en aplicacions reals, on fàcilment ens podem trobar amb centenars de taules.

Quan cal desenvolupar una aplicació de gestió on la base de dades té un nombre elevat de taules i per a cada taula s'ha de fer la gestió d'un gran nombre de mètodes (a banda d'altes, baixes, modificacions i consultes, pot haver-hi diferents llistats, modificacions, consultes parcials, etc.), és complex buscar i trobar dintre del conjunt de *tots* els mètodes de *totes* les classes, el mètode adient que faci el que el programador necessita.

Per a evitar o minimitzar al màxim aquest problema es pot aplicar un model de disseny anomenat Model-Vista-Controlador (Model-View-Controller), on hi ha tres estereotips de classes, o sigui, tres conjunts de classes ben diferenciats.

- **Classes de MODEL**: totes les classes necessàries per a accedir a la base de dades. Inclou també una classe especial per a centralitzar els accessos a la base de dades que han de fer les classes encarregades de gestionar les taules.

- **Classes de VISTA**: totes les classes necessàries per a comunicar-se amb la interfície d'usuari. Són les classes que implementen els filtres de dades introduïts per l'usuari i apliquen els formats necessaris a les sortides de dades.

- **Classes de CONTROLADOR**: totes les classes necessàries per a coordinar les peticions realitzades per les classes de VISTA, que reben peticions directament dels usuaris, i els mètodes de les classes de MODEL, que implementen la solució a aquestes peticions.

4.2. Classes especials: TAccesBD i TControlador

En aquest subapartat s'expliquen en detall les classes especials TAccesBD i TControlador.

4.2.1. TAccesBD

La classe `TAccesBD` està pensada per ser l'únic punt de comunicació entre el programa i la base de dades. Això té alguns avantatges:

- És una classe reutilitzable, o sigui, que es pot incloure l'arxiu on està implementada en un altre projecte, i es pot fer servir de la mateixa manera.

- Si la manera d'accedir a la base de dades canvia en un futur (per exemple, una migració de MySQL a SQL-Server), els únics canvis que cal fer són a la classe `TAccesBD` i no al codi del programa

- Com que està unificat l'accés a la base de dades a través d'aquesta classe, si es detecta i es corregeix un error en `TAccesBD`, automàticament queda corregit per a tots els accessos a la base de dades que es facin des del programa.

- Té un baix acoblament, de manera que seria fàcil canviar aquesta classe per una altra (amb la mateixa interfície) que millorés el rendiment o donés accés a una base de dades implementada amb una altra tecnologia, com ara Objecte-Relacional, NoSQL o orientada a objecte pura.

El codi d'aquesta classe especial es troba a l'annex d'aquest document.

Amb els comentaris que hi ha al codi i el contingut del mòdul "Accés a bases de dades amb MySQL", el funcionament de la classe és autoexplicatiu.

Una vegada disposem d'un fitxer (`TAccesBD.php`) amb la implementació de la classe, cal incloure'l a la classe (o classes) que necessitin accedir a la base de dades. En aquest cas, les tres classes que s'estan implementant.

Amb l'ús de `TAccesBD`, cal modificar la implementació de les tres classes actuals perquè facin servir un objecte de la classe `TAccesBD` per a accedir a la base de dades. Les modificacions que cal fer són:

- Afegir una propietat nova, privada, que serà l'objecte que es farà servir per a accedir a la base de dades.

- Al constructor de cadascuna de les classes s'ha de crear l'objecte de la classe `TAccesBD` i assignar-lo a la propietat. Així estarà disponible per a tots els mètodes de la classe per quan el necessitin.

- Crear un destructor de cadascuna de les classes per a destruir l'objecte de la classe `TAccesBD` que es va crear en el constructor.

- Fer servir l'objecte de la classe `TAccesBD` per a gestionar la base de dades.

A la classe `TEstudiant`, se li passa al constructor les dades de la connexió amb la base de dades. El valor d'aquestes dades es defineix en un objecte superior que crida un objecte de la classe `TEstudiant`, per a unificar en un sol punt totes les dades de la connexió.

Així, el codi de `TEstudiant` amb les modificacions fetes i els mètodes `existeix_estudiant` i `alta_estudiant` quedaria de la següent manera:

```
<?php
include_once ("TAccesBD.php");

class TEstudiant
{
      private $DNI;
      private $nom;
      private $edat;
      private $abd;       //Objecte de TAccesBD

      function __construct($v_dni, $v_nom, $v_edat, $servidor, $usuari, $paraula_pas, $nom_bd))
      {
            $this->DNI = $v_dni;
            $this->nom = $v_nom;
            $this->edat = $v_edat;
            $var_abd = new TAccesbd($servidor,$usuari,$paraula_pas,$nom_bd);
            $this->abd = $var_abd;
            $this->abd->connectar_BD();
      }
      function __destruct()
      {
            if (isset($this->abd))    //si existeix l'objecte...
            {
                  unset($this->abd);        //el destruim
            }
      }
      public function existeix_estudiant ()
      {
            $res = false;
            if ($this->abd->consulta_SQL("select count(*) as quants
          from estudiant where DNI = '" .
          $this->abd->escapar_dada($this->DNI)."'"))
            {
                  if ($this->abd->consulta_fila())
                  {
                        $res = ($this->abd->consulta_dada('quants') > 0);
                  }
            }
            return $res;
      }
      public function alta_estudiant()
      {
            $res = false;
            //es comprova que l'estudiant no està ja a la base de dades
            if (!($this->existeix_estudiant()))
            {     //si efectivament no hi és, s'insereix
                  if ($this->abd->consulta_SQL("insert into estudiant values ('" .
                        $this->abd->escapar_dada($this->DNI) . "','" .
                        $this->abd->escapar_dada($this->nom) . "'," .
                        $this->abd->escapar_dada($this->edat). ")"))
                  {                 {
                        $res = true;
                  }
            }
            return $res;
      }
}
?>
```

Al codi anterior, al mètode `existeix_estudiant()` es pot veure el funcionament de la classe `TAccesBD`:

Primer es munta el text de la consulta SQL concatenant els paràmetres necessaris. Aquesta cadena de caràcters se li passa al mètode `consulta_SQL()` de l'objecte `abd` definit com a propietat privada de la classe `TEstudiant`. És molt important notar que *totes* les variables necessàries per a muntar qualsevol instrucció SQL (tant si són consultes com insercions, modificacions o eliminacions de dades) han de passar primer per la funció `escapar_dada($dada)`

de TAccesBD, per tal d'eliminar caràcters "perillosos" per a la consulta, i evitar d'aquesta manera possibles atacs d'injecció de SQL. La classe TAccesBD fa servir la instrucció mysqli_real_scape_string per a fer aquesta funció de seguretat.

Si la consulta és correcta sintàcticament, i hi ha dades, retornarà una o més files de dades. Per a accedir a la primera fila de dades retornades per la consulta cal cridar el mètode consulta_fila(). Si tot va bé, després de fer-ho es podrà accedir a les dades de la primera fila amb el mètode consulta_dada(), passant-hi com a paràmetre el nom del camp retornat a consultar.

Tot això s'ha de fer si es tracta d'un SELECT. Si la sentència SQL executada no és un SELECT, la instrucció consulta_SQL() retornarà CERT o FALS per a indicar si l'execució de la sentència ha acabat amb èxit o no, com es mostra al mètode d'alta d'un estudiant. En el cas d'alta d'un estudiant, primer es comprova que l'estudiant ja sigui a la base de dades, seguint les consideracions generals sobre la programació de gestió.

Per a inserir un estudiant a la base de dades caldria fer un codi com aquest:

```
<?php
//programa que fa servir les classes
include_once "TEstudiant.php";

$e = new TEstudiant('52111111C','Joan Serra Dot',23);
if ($e->alta_estudiant())
{
      echo ("<br>Alta de l'estudiant realitzada correctament");
}
else
{
      echo ("<br>Error en realitzar l'alta de l'estudiant");
}
?>
```

Un altre mètode que cal comentar és el llistat d'estudiants d'una certa assignatura. Com ja s'ha comentat, aquest mètode està codificat a la classe TAssignatura. Aquest mètode realitza les següents accions:

1) Comprova que hi ha estudiants que cursen aquella assignatura.

2) Mostra la capçalera del llistat i fa la consulta SQL, que retornarà les dades del llistat.

3) Accedeix a la primera fila de dades i mentre hi hagi files de dades.

a) Accedeix a la informació de la fila actual i la mostra per pantalla.

b) Avança la següent fila de dades, obtenint la següent fila, o bé arribant al final.

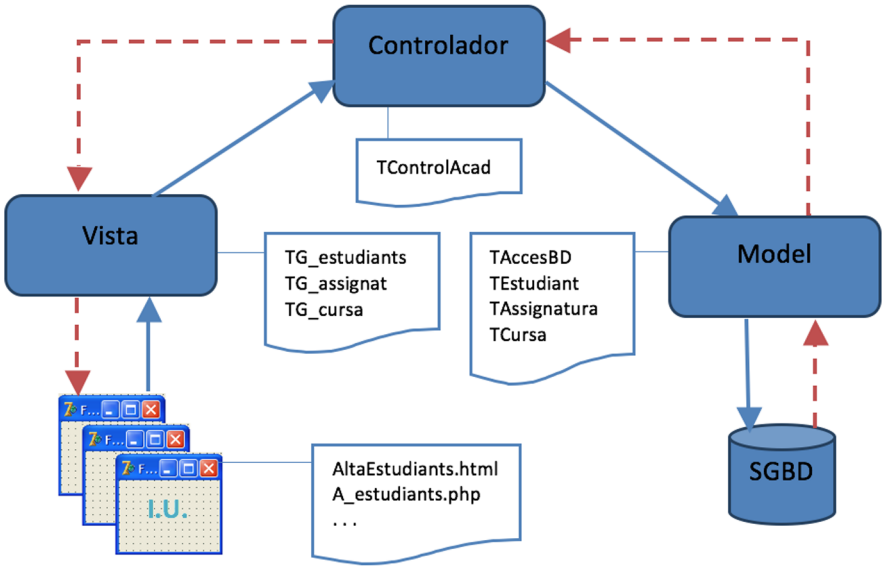
4) En sortir del bucle es consulta el nombre de files retornades, per a mostrar-lo també per pantalla.

5) Finalment es tanca la consulta.

```
public function llistat_estudiants()
      {
            $res = false;
            //Es comprova si l'assignatura te alumnes assignats
            if ($this->abd->consulta_SQL("select count(*) as quants
       from cursa where codi_assig = '" .
       $this->abd->escapar_dada($this->codi)."'"))
            {
                  if ($this->abd->consulta_fila())
                  {
                        if ($this->abd->consulta_dada('quants') != 0)
                        {
                              //Existeix l'assignatura i te alumnes-> fem llistat
                              $res = true;
                              echo "<br><br> LLISTAT D'ALUMNES DE L'ASSIGNATURA
                              AMB CODI ". $this->codi;
                              echo "<br> ----------------------------";

                              if ($this->abd->consulta_SQL("
               SELECT E.DNI, E.NOM as nom_est, A.NOM as nom_as,
                  A.CREDITS, C.NOTA " . "FROM (ESTUDIANT E INNER
                  JOIN CURSA C ON E.DNI = C.DNI) " ."INNER JOIN
                  ASSIGNATURA A ON C.CODI_ASSIG = A.CODI WHERE
                  A.CODI = '" .
                  $this->abd->escapar_dada($this->codi)."'")
                   )
                              {
                                    $fila = $this->abd->consulta_fila();
                                    while ($fila != null)
                                    {
echo "<br>------------------------";
echo "<br>DNI:         " . $this->abd->consulta_dada('DNI');
echo "<br>NOM:         " . $this->abd->consulta_dada('nom_est');
echo "<br>Assignatura: " . $this->abd->consulta_dada('nom_as');
echo "<br>Credits:     " . $this->abd->consulta_dada('CREDITS');
echo "<br>Nota:        " . $this->abd->consulta_dada('NOTA');
echo "<br>";
                                          $fila = $this->abd->consulta_fila();
                                    }
                                    $quants = $this->abd->files_afectades();
                                    echo "<br> -------------------------";
                                    echo "<br> TOTAL ESTUDIANTS=" . $quants . "<br>";
                              }
                              $this->abd->tancar_consulta();
                        }
                  }
            }
            return $res;
      }
```

4.2.2. TControlador

La classe `TControlador` està pensada per a sistemes que estan formats per moltes classes encarregades de gestionar les taules de la base de dades. Aquesta classe prové d'un patró de disseny anomenat "Model-Vista-Controlador" (MVC o Model-View-Controller, en anglès), on la lògica de l'aplicació informàtica es divideix en tres grans estereotips de classes:



El patró de disseny MVC

El patró de disseny MVC és una de les moltes maneres diferents d'estructurar les classes necessàries per a desenvolupar un sistema informàtic.

Hi ha llenguatges de programació on fins i tot s'obliga a programar fent servir aquest patró de disseny.

Els patrons de disseny i programació

Els patrons de disseny i programació són tècniques provades i documentades per a fer certes tasques tant de disseny com de programació. Aquestes tècniques funcionen i solen ser les millors que s'han trobat fins al moment.

D'altra banda, existeixen els "antipatrons de disseny i programació", que són tècniques que sovint es fan servir, però que donen lloc a més problemes que no pas en solucionen. Un exemple d'això és fer servir variables globals o programar sense un disseny previ.

1) Classes de VISTA. Són les classes que implementen la interfície d'usuari, com per exemple el codi associat a fer clic en un botó. Si parlem d'aplicacions web, aquestes classes proveirien el codi en HTML, Javascript, PHP, etc. (però no CSS, que és estrictament de disseny d'interfície d'usuari). En aquest cas, n'hi haurà tres classes (`TG_estudiants`, `TG_assignat`, `TG_cursa`), que correspondran a tres pàgines web diferents dintre de la mateixa interfície d'usuari i que permetran gestionar les dades de les taules de la base de dades. També hi haurà el codi necessari per cridar, des de la interfície d'usuari, mètodes d'aquestes tres classes.

2) Classes de MODEL. En aquest estereotip s'hi agrupen totes les classes que fan les operacions (que impliquen consultes i actualitzacions de dades) sobre la base de dades. Són les classes que accedeixen a la base de dades (`TAccesBD`) i també les que demanen les instruccions SQL necessàries per a realitzar una funcionalitat (`TEstudiant`, `TAssignatura`, `TCursa`). Aquestes classes reben una petició des de CONTROLADOR, en forma d'execució del mètode adient de la classe corresponent, accedeixen a la base de dades (potser cridant altres mètodes d'altres classes, si cal), obtenen el resultat (o generen un error) i el retornen a CONTROLADOR.

3) Classes de CONTROLADOR. Fan de mitjancer entre les classes d'interfície (VISTA) i d'accés a la base de dades (MODEL). Reben peticions de VISTA, seleccionen quina o quines classes de MODEL són les encarregades de servir la petició (és a dir, quina classe de model té implementada la funcionalitat que

es demana des de VISTA) i criden el mètode corresponent, passant-hi els paràmetres que li arriben des de VISTA. Un cop feta la crida, obtenen la resposta i la retornen a VISTA. Aquestes classes són molt útils quan el nombre de mètodes de les classes de MODEL és molt elevat. Cal tenir en compte que dintre de l'estereotip CONTROLADOR pot haver-hi diverses classes, depenent de la part del sistema que controlin (per exemple, `TControlAcadèmic`, `TControlNómines`, `TControlContinguts`).

A continuació es mostrarà tot el procés, des de la introducció de dades a la interfície d'usuari fins a la inserció de les dades a la base de dades:

1) `AltaEstudiants.html`. Pàgina HTML d'introducció de dades (***MVC: Interfície d'usuari***).

```
<html>
<head>
<title>ALTA D'ESTUDIANTS</title>
</head>
<body>
<p>Introdueix les dades per donar d'alta un estudiant a la base de dades</p>
<form action="A_estudiants.php" method="post">
 <p>DNI: <input type="text" name="DNI" /></p>
 <p>Nom: <input type="text" name="nom" /></p>
 <p>Edat: <input type="text" name="edat" /></p>
 <p><input type="submit" value="Donar d'alta"/></p>
</form>
</body>
</html>
```

2) `A_estudiants.php`. Codi PHP associat a la pàgina d'introducció de dades (***MVC: Interfície d'usuari***).

```
<?php
//codi associat a la pàgina HTML d'entrada de dades de l'alta d'estudiant
include_once("TG_estudiants.php");
$g_e = new TGestioEst($_POST['DNI'], $_POST['nom'], $_POST['edat']);
if ($g_e->alta_estudiant())
{
      echo "Alta realitzada correctament";
}
else
{
      echo "Error en realitzar l'alta";
}
?>
```

3) `TG_Estudiants.php`. Codi PHP associat a la gestió d'estudiants (***MVC: Vista***).

```php
<?php
//classe de VISTA de gestió d'estudiants
//Pot fer alta i baixa d'estudiants
include_once ("TControlAcad.php");
class TGestioEst
{
	private $DNI;
	private $nom;
	private $edat;
	function __construct($v_dni, $v_nom, $v_edat)
	{
		$this->DNI = $v_dni;
		$this->nom = $v_nom;
		$this->edat = $v_edat;
	}
	public function alta_estudiant()
	{
		$c = new TControlAcad();
		$res = $c->alta_estudiant($this->DNI, $this->nom, $this->edat);
		return $res;
	}
	public function baixa_estudiant()
	{
		$c = new TControlAcad();
		$res = $c->baixa_estudiant($this->DNI);
		return $res;
	}
}
?>
```

3) `TControlAcad.php`. Codi PHP de control acadèmic: índex de mètodes de classes associats al control acadèmic. Crida totes les funcionalitats demanades a les especificacions (***MVC:*** *Controlador*).

```
<?php
include_once ("TEstudiant.php");
include_once ("TAssignatura.php");
include_once ("TCursa.php");
class TControlAcad
{
      public function alta_estudiant ($DNI, $nom, $edat)
      {
            $e = new TEstudiant($DNI, $nom, $edat);
            $res = $e->alta_estudiant();
            return $res;
      }
      public function baixa_estudiant ($DNI)
      {
            $e = new TEstudiant($DNI, '',0);
            $res = $e->baixa_estudiant();
            return $res;
      }
      public function consulta_assignatura ($codi_assig,
      &$nom, &$credits)
      {
            $a = new TAssignatura ($codi_assig, '', 0);
            $res = $a->consulta_assignatura ($nom, $credits);
            return $res;
      }
      public function llistat_estudiant ($codi_assig)
      {
            $a = new TAssignatura ($codi_assig, '', 0);
            $res = $a->llistat_estudiants();
            return $res;
      }
      public function assignar ($DNI, $codi_assig)
      {
            $c = new TCursa ($DNI, $codi_assig, null);
            $res = $c->assignar();
            return $res;
      }
      public function desassignar($DNI, $codi_assig)
      {
            $c = new TCursa ($DNI, $codi_assig, null);
            $res = $c->desassignar();
            return $res;
      }
      public function posar_nota ($DNI, $codi_assig, $nota)
      {
            $c = new TCursa ($DNI, $codi_assig);
            $res = $c->posar_nota($nota);
            return $res;
      }
}
?>
```

4) `TEstudiant.php`. Codi PHP de la classe encarregada de la gestió de la taula ESTUDIANT *(**MVC**:Model)*. (Només es mostren els mètodes necessaris per a fer la inserció de dades.)

```
<?php
include_once ("TAccesBD.php");
class TEstudiant
{
   private $DNI;
       private $nom;
       private $edat;
       private $abd;
       function __construct($v_dni, $v_nom, $v_edat)
       {
              $this->DNI = $v_dni;
              $this->nom = $v_nom;
              $this->edat = $v_edat;
              $var_abd = new TAccesbd("localhost","root","","bdprova");
              $this->abd = $var_abd;
              $this->abd->connectar_BD();
       }
       function __destruct()
       {
              if (isset($this->abd))
              {
                     unset($this->abd);
              }
       }
   public function existeix_estudiant ()
       {
              $res = false;
              if ($this->abd->consulta_SQL("select count(*) as quants
      from estudiant where DNI = '" .
      $this->abd->escapar_dada($this->DNI)."'"))
              {
                     if ($this->abd->consulta_fila())
                     {
                            $res = ($this->abd->consulta_dada('quants') > 0);
                     }
              }
              return $res;
       }
       public function alta_estudiant()
       {
              $res = false;
              //es comprova que l'estudiant no està ja a la base de dades
              if (!($this->existeix_estudiant()))
              {      //si efectivament no hi és, s'insereix
                     if ($this->abd->consulta_SQL("insert into estudiant values ('" .
                            $this->abd->escapar_dada($this->DNI) . "','" .
                            $this->abd->escapar_dada($this->nom) . "'," .
                            $this->abd->escapar_dada($this->edat). ")"))        {
                            $res = true;
                     }
              }
              return $res;
       }

       . . .

}
?>
```

5) `TAccesBD.php`. Codi PHP de la classe especialitzada d'accés a la base de dades (***MVC:Model***). Només es mostren els mètodes necessaris per a fer la inserció de dades. El codi de la classe completa és a l'annex.

```
<?php
class TAccesbd
{
       private $bd;
       private $host;
       private $user;
       private $pass;
       private $connexio;
       private $dades;
   private $fila;
       function __construct($host , $user , $pass, $bd)
       {
              $this->bd = $bd;
              $this->host = $host;
              $this->user = $user;
              $this->pass = $pass;
       }

public function escapar_dada ($dada)
       {
              return mysqli_real_escape_string($this-
>connexio,$dada);
       }

public function consulta_SQL ($consulta)
       {
              $this->dades = mysqli_query ($this->connexio,
$consulta);
              if (mysqli_errno($this->connexio) != 0)
              {
                     $res = false;
              }
              else
              {
                     $res = true;
              }
              return $res;
       }
. . .
}
?>
```

Es pot observar que aquest mètode de desenvolupament crea molts fitxers diferents. Això té avantatges i inconvenients. Com s'ha esmentat anteriorment, si el sistema que cal desenvolupar és petit, amb poques taules per a gestionar i amb un equip reduït de programadors, no cal muntar tot aquest sistema d'estereotips (MVC). Es pot programar també amb classes (*sempre* és recomanable) però amb crides més directes des de la interfície d'usuari.

Exemple de crides directes des de la interfície d'usuari

Per a construir una caseta per al gos no calen plànols d'arquitecte. Amb un dibuix del que es vol fer n'hi ha prou.

Ara bé, per a sistemes mitjans o grans, on l'equip desenvolupador està format per diversos programadors o fins i tot per diversos equips de programadors, amb aquest sistema (MVC) el repartiment de la feina és molt més eficient si està "compartimentada" en estereotips de classes. Encara més, hi pot haver equips especialitzats en la interfície d'usuari, en la programació de la lògica d'algun dels mòduls, etc.

Exemple d'equips especialitzats en la interfície d'usuari

Per a construir un gratacel és absolutament necessari i obligatori fer plànols i repartir la feina de construcció entre diferents grups d'especialistes.

5. Annex: codi emprat als exemples

5.1. TComptador i TComtpaSalt

```
<?php
class TComptador
{
   protected $valor;
   protected $maxim;

   function __construct($max)
   {
      echo "<br> Creant TComptador <br>";
      if ($max < 0)
      {
         $this->maxim = 0;
      }
      else
      {
         $this->maxim = $max;
      }
      $this->valor = 0;
   }

   public function incrementa()
   {
      if ($this->valor < $this->maxim)
      {
         $this->valor = $this->valor + 1;
      }
   }

   public function consulta()
   {
      return $this->valor;
   }

   public function fi()
   {
      return ($this->valor == $this->maxim);
   }

   public function inicialitza()
```

```
    {
       $this->valor = 0;
    }

    function __destruct()
    {
       echo "<br> destruint TComptador <br>";
    }
}

class TComptaSalt extends TComptador
{
    private $salt;

    public function posar_salt ($increment)
    {
       if ($increment <= 0)
       {
          $this->salt = 1;
       }
       else
       {
          $this->salt = $increment;
       }
    }
    public function incrementa ()
    {
       if ($this->valor + $this->salt < $this->maxim)
       {
          $this->valor = $this->valor + $this->salt;
       }
       else
       {
          $this->valor = $this->maxim;
       }
    }
}
?>
```

5.2. Exemple d'ús de les classes TComptador i TComptaSalt

```
<?php
include_once ("TComptador.php");
$i = new TComptador(4);
while (!($i->fi()))
{
    echo "i = " . $i->consulta() . "<br>";
```

```
   $i->incrementa();
}
$s = new TComptaSalt(10);
$s->posar_salt(3);
while (!($s->fi()))
{
   echo "s = " . $s->consulta() . "<br>";
   $s->incrementa();
}
?>
```

5.3. AltaEstudiants.HTML

```
<html>
<head>
<title>ALTA D'ESTUDIANTS</title>
</head>
<body>
<p>Introdueix les dades per donar d'alta un estudiant a la base de dades</p>
<form action="A_estudiants.php" method="post">
   <p>DNI: <input type="text" name="DNI" /></p>
   <p>Nom: <input type="text" name="nom" /></p>
   <p>Edat: <input type="text" name="edat" /></p>
   <p><input type="submit" value="Donar d'alta"/></p>
</form>
</body>
</html>
```

5.4. A_estudiants.php

```
<?php
//codi associat a la pàgina HTML d'entrada de dades de l'alta d'estudiant
include_once("TG_estudiants.php");
$g_e = new TGestioEst($_POST['DNI'], $_POST['nom'], $_POST['edat']);
if ($g_e->alta_estudiant())
{
   echo "Alta realitzada correctament";
}
else
{
   echo "Error en realitzar l'alta";
}
?>
```

5.5. TG_estudiants.php

```
<?php
```

```
//classe de VISTA de gestió d'estudiants
//Pot fer alta i baixa d'estudiants
include_once ("TControlAcad.php");
class TGestioEst
{
   private $DNI;
   private $nom;
   private $edat;
   function __construct($v_dni, $v_nom, $v_edat)
   {
      $this->DNI = $v_dni;
      $this->nom = $v_nom;
      $this->edat = $v_edat;
   }
   public function alta_estudiant()
   {
      $c = new TControlAcad();
      $res = $c->alta_estudiant($this->DNI, $this->nom, $this->edat);
      return $res;
   }
   public function baixa_estudiant()
   {
      $c = new TControlAcad();
      $res = $c->baixa_estudiant($this->DNI);
      return $res;
   }
}
?>
```

5.6. TControlAcad.php

```
<?php
//Classe de CONTROLADORS que s'encarrega de la gestió
//acadèmica dels estudiants
include_once ("TEstudiant.php");
include_once ("TAssignatura.php");
include_once ("TCursa.php");

class TControlAcad
{
   private $servidor;
   private $usuari;
   private $paraula_pas;
   private $nom_bd;

   function __construct()
   {
```

```
        $this->servidor = "localhost";
        $this->usuari = "root";
        $this->paraula_pas = "";
        $this->nom_bd = "bdprova";
    }

    public function alta_estudiant ($DNI, $nom, $edat)
    {
        $e = new TEstudiant($DNI, $nom, $edat, $this->servidor,
                            $this->usuari, $this->paraula_pas,
                            $this->nom_bd);
        $res = $e->alta_estudiant();
        return $res;
    }

    public function baixa_estudiant ($DNI)
    {
        $e = new TEstudiant($DNI, '',0, $this->servidor, $this->usuari,
                            $this->paraula_pas, $this->nom_bd);
        $res = $e->baixa_estudiant();
        return $res;
    }

    public function consulta_assignatura ($codi_assig,
                                          &$nom, &$credits)
    {
        $a = new TAssignatura ($codi_assig, '', 0, $this->servidor,
                               $this->usuari, $this->paraula_pas,
                               $this->nom_bd);
        $res = $a->consulta_assignatura ($nom, $credits);
        return $res;
    }

    public function llistat_estudiant ($codi_assig)
    {
        $a = new TAssignatura ($codi_assig, '', 0, $this->servidor,
                               $this->usuari, $this->paraula_pas,
                               $this->nom_bd);
        $res = $a->llistat_estudiants();
        return $res;
    }

    public function assignar ($DNI, $codi_assig)
    {
        $c = new TCursa ($DNI, $codi_assig, null, $this->servidor,
                         $this->usuari, $this->paraula_pas,
                         $this->nom_bd);
```

```
        $res = $c->assignar();
        return $res;
    }

    public function desassignar($DNI, $codi_assig)
    {
        $c = new TCursa ($DNI, $codi_assig, null, $this->servidor,
                         $this->usuari, $this->paraula_pas,
                         $this->nom_bd);
        $res = $c->desassignar();
        return $res;
    }

    public function posar_nota ($DNI, $codi_assig, $nota)
    {
        $c = new TCursa ($DNI, $codi_assig, $this->servidor,
                         $this->usuari, $this->paraula_pas,
                         $this->nom_bd);
        $res = $c->posar_nota($nota);
        return $res;
    }

}
?>
```

5.7. TEstudiant.php

```
<?php
//Classe de MODEL encarregada de la gestió de la taula ESTUDIANT de la base de dades
include_once ("TAccesBD.php");

class TEstudiant
{
    private $DNI;
    private $nom;
    private $edat;
    private $abd;

    function __construct($v_dni, $v_nom, $v_edat,
    {
        $this->DNI = $v_dni;
        $this->nom = $v_nom;
        $this->edat = $v_edat;
        $var_abd = new TAccesbd($servidor,$usuari,$paraula_pas,$nom_bd);
        $this->abd = $var_abd;
        $this->abd->connectar_BD();
    }
```

```
    function __destruct()
    {
       if (isset($this->abd))
       {
          unset($this->abd);
       }
    }

    public function existeix_estudiant ()
    {
       $res = false;
       if ($this->abd->consulta_SQL("
           select count(*) as quants
           from estudiant
           where DNI = '" .
$this->abd->escapar_dada($this->DNI)."'")
)
       {
          if ($this->abd->consulta_fila())
          {
             $res = ($this->abd->consulta_dada('quants') > 0);
          }
       }
       return $res;
    }

    public function alta_estudiant()
    {
       $res = false;
       //es comprova que l'estudiant no està ja a la base de dades
       if (!($this->existeix_estudiant()))
       {   //si efectivament no hi és, s'insereix
          if ($this->abd->consulta_SQL("insert into estudiant
values ('" .
             $this->abd->escapar_dada($this->DNI) . "','" .
             $this->abd->escapar_dada($this->nom) . "'," .
             $this->abd->escapar_dada($this->edat). ")"))
          {
             $res = true;
          }
       }
       return $res;
    }

    public function baixa_estudiant()
    {
```

```
      $res = false;
      if ($this->existeix_estudiant())
      {
         //Es comprova que l'estudiant no te cap assignatura assignada
         if ($this->abd->consulta_SQL("
             select count(*) as quants
             from cursa
             where DNI = '" .
$this->abd->escapar_dada($this->DNI) ."'"))
         {
            if ($this->abd->consulta_fila())
            {
               if ($this->abd->consulta_dada('quants') == 0)
               {
                  //l'estudiant no te assignatures, l'eliminem
                  if ($this->abd->consulta_SQL("
                      delete from estudiant
                      where DNI = '" .
$this->abd->escapar_dada($this->DNI) . "'"));
                  {
                     $res = true;
                  }
               }
            }
         }
      }
      return $res;
   }
}
?>
```

5.8. TAssignatura.php

```
<?php
include_once ("TAccesBD.php");

class TAssignatura
{
   private $codi;
   private $nom;
   private $credits;
   private $abd;

   function __construct($v_codi, $v_nom, $v_credits, $servidor,
                        $usuari, $paraula_pas, $nom_bd)
   {
      $this->codi = $v_codi;
```

```
        $this->nom = $v_nom;
        $this->credits = $v_credits;
        $var_abd = new TAccesbd($servidor,$usuari,$paraula_pas,$nom_bd);
        $this->abd = $var_abd;
        $this->abd->connectar_BD();
    }

    function __destruct()
    {
        if (isset($this->abd))
        {
            unset($this->abd);
        }
    }



    public function existeix_assignatura ()
    {
        $res = false;
        if ($this->abd->consulta_SQL("
             select count(*) as quants
             from assignatura
             where codi = '" .
$this->abd->escapar_dada($this->codi) . "'"))
        {
            if ($this->abd->consulta_fila())
            {
                $res = ($this->abd->consulta_dada('quants') > 0);
            }
        }
        return $res;
    }

    public function consulta_assignatura (&$v_nom, &$v_credits)
    {
        $res = false;
        if ($this->existeix_assignatura())
        {
            if ($this->abd->consulta_SQL("
                 select nom, credits
                 from assignatura
                 where codi = '" .
$this->abd->escapar_dada($this->codi) . "'"))
            {
                if ($this->abd->consulta_fila())
                {
                    $v_nom = $this->abd->consulta_dada('nom');
```

```
                $v_credits = $this->abd->consulta_dada('credits');
                $res = true;
             }
          }
       }
       return $res;
    }

    public function llistat_estudiants()
    {
       $res = false;
       //Es comprova si existeix l'assignatura
       if ($this->abd->consulta_SQL("
           select count(*) as quants
           from cursa
           where codi_assig = '" .
$this->abd->escapar_dada($this->codi) ."'"))
       {
          if ($this->abd->consulta_fila())
          {
             if ($this->abd->consulta_dada('quants') != 0)
             {
                //Existeix l'assignatura i te alumnes-> fem el llistat
                $res = true;

echo "<br><br>LLISTAT D'ALUMNES DE L'ASSIGNATURA AMB CODI ".
$this->codi;
echo "<br> ------------------------------------------------";

                if ($this->abd->consulta_SQL("
                    SELECT E.DNI, E.NOM as nom_est, A.NOM as nom_as,
                    A.CREDITS, C.NOTA " .
                    "FROM (ESTUDIANT E INNER JOIN CURSA C ON
                    E.DNI = C.DNI) " .
                    "INNER JOIN ASSIGNATURA A ON C.CODI_ASSIG = A.CODI
                    WHERE A.CODI = '" .
$this->abd->escapar_dada($this->codi) ."'"))
                {
                   $fila = $this->abd->consulta_fila();
                   while ($fila != null)
                   {
echo "<br>-----------------------";
echo "<br>DNI:         " . $this->abd->consulta_dada('DNI');
echo "<br>NOM:         " . $this->abd->consulta_dada('nom_est');
echo "<br>Assignatura: " . $this->abd->consulta_dada('nom_as');
echo "<br>Credits:     " . $this->abd->consulta_dada('CREDITS');
echo "<br>Nota:        " . $this->abd->consulta_dada('NOTA');
```

```
echo "<br>";
                    $fila = $this->abd->consulta_fila();
                }
                $quants = $this->abd->files_afectades();
                echo "<br> ----------------------------------------
                ----------------------------------";
                echo "<br> TOTAL ESTUDIANTS DE L'ASSIGNATURA = " .
                $quants . "<br>";
            }
            $this->abd->tancar_consulta();
        }
      }
    }
    return $res;
  }
}
?>
```

5.9. TAccesBD

```
<?php
//Classe de MODEL especialitzada en l'accés a la base de dades.
//Centralitza totes les peticions de sentències SQL de les altres
//classes de MODEL
class TAccesbd
{
   //Aquestes quatre propietats privades guarden les dades de
   //connexió a la base de dades
   private $bd;
   private $host;
   private $user;
   private $pass;

   //La pròpia classe de gestió de la base de dades guarda els
   //objectes necessaris per realitzar aquesta gestió.
   //Així, des del programa extern que la faci servir, no cal
   //preocupar-se ni de com es fa aquesta connexió ni com es guarda

   private $connexio;    //Connexió real amb la base de dades.
   private $dades;       //Dades reals retornades per una consulta amb
                         //èxit (i dades, clar...)
   private $fila;       //Una de les files de les dades de la consulta
                        //realitzada


   //Constructor de la classe. Serveix per poder crear objectes de la
   //classe. Li hem d'indicar les dades de connexió
```

```
    function __construct($host , $user , $pass, $bd)
    {
       $this->bd = $bd;
       $this->host = $host;
       $this->user = $user;
       $this->pass = $pass;
    }


    //Realitza la connexió a la base de dades, amb les dades de
 //connexió indicades al constructor de la classe
    public function connectar_BD()
    {
       $res = true;
       $this->connexio = @mysqli_connect ($this->host, $this->user,
                                            $this->pass, $this->bd);
       mysqli_set_charset($this->connexio,"utf8");
       if (!$this->connexio)
       {
          $res = false;
          die("No s'ha pogut realitzar la connexió. ERROR:" .
                mysqli_connect_error());
       }
       return $res;
    }


    //Desconnecta de la base de dades.
    public function desconnectarBd()
    {
       if (isset($this->connexio))
       {
          mysqli_close($this->connexio);
       }
    }

 // Per tal d'evitar atacs de SQL-Injection, aquesta funció
 // retorna la dada que se li passi filtrada de caràcters
 // "perillosos"
    public function escapar_dada ($dada)
    {
       return mysqli_real_escape_string($this->connexio,$dada);
    }


    /*Executa un SQL, ja sigui Select (el retorn del qual és un
     dataset) o insert, update o delete (que retorna cert o fals
```

```
    depenent de com hi hagi anat)
    Es dona per fet que la connexió s'ha realitzat prèviament
    Si el SQL és un SELECT ,retorna el conjunt global de resultats del
    SELECT.
    Posteriorment s'haurà de fer servir la funció "Consulta_fila" per
    obtenir la primera fila (o posteriors) de resultats
*/
    public function consulta_SQL ($consulta)
    {
        $this->dades = mysqli_query ($this->connexio, $consulta);
        if (mysqli_errno($this->connexio) != 0)
        {
            $res = false;
        }
        else
        {
            $res = true;
        }
        return $res;
    }

    //Funció que obté la fila de dades d'una consulta prèviament
    //realitzada. Retorna totes les dades de la fila per ser
     //consultades posteriorment amb "consulta_dada"
    public function consulta_fila ()
    {
        $this->fila = null;
        if ($this->dades != null)
        {
            $this->fila = mysqli_fetch_assoc($this->dades);
        }
        return $this->fila;
    }

    /*
Funció que després de fer un "consulta_fila", i donat el nom d'un
camp de la consulta, retorna la dada del camp del registre actual.
    Tant si el nom del camp no existeix als resultats, com si ja s'han
    consultat totes les files del resultat, la funció retorna NULL
    */
    public function consulta_dada ($camp)
    {
        $res = null;
        if ($this->fila != null)
        {
            $res = $this->fila[$camp];
        }
```

```
        return $res;
    }

    //Retorna el número de files involucrades a la darrera
    //consulta_SQL
    public function files_afectades ()
    {
        $res = 0;
        if ($this->connexio != null)
        {
            $res = mysqli_affected_rows($this->connexio);
        }
        return ($res);
    }

    //Elimina la memòria associada a la darrera consulta realitzada
    public function tancar_consulta ()
    {
        mysqli_free_result($this->dades);
    }

    //Retorna el darrer missatge d'error produït a una consulta SQL
    public function missatge_error ()
    {
        $res = "";
        if (mysqli_errno($this->connexio) != 0)
        {
            $res = mysqli_error($this->connexio);
        }
        return $res;
    }
}
?>
```