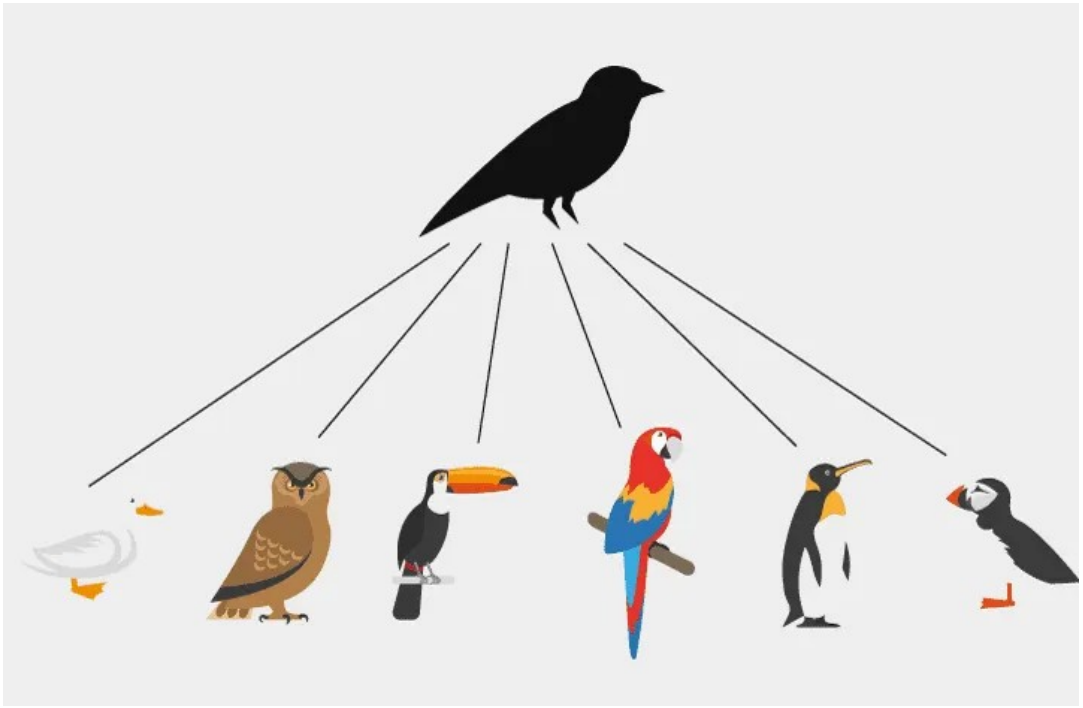


PHP 14

Classes i objectes

18)Classes i objectes

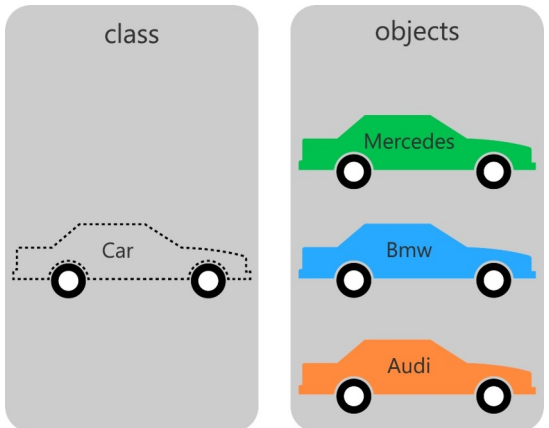
- **Programació orientada a objectes** (POO):
  - Tècnica per dissenyar aplicacions:
    - Aplicació de web.
    - Aplicació d'escriptori en Windows.
    - Aplicació mòbil, etc.
- És un **concepte per dissenyar**:
  - Tot gira al voltant d'**objectes** i **classes**.

- Qualsevol cosa és un **objecte**:
  - Si mirem **al nostre voltant** podem trobar molts objectes.
    - L'ordinador, el cotxe, la casa, les persones...
  - Cada objecte té dues elements que podem abstreure:
    - Les seues **propietats**.
    - Els seus **comportaments**.
  - Per **exemple**, en un **cotxe**:
    - **Propietats**: color, model, marca, tipus, mida...
    - **Comportaments**: va cap avant i cap enrere, frena, emplena el dipòsit, passa la ITV...

- Els objectes **en programació** són molt **pareguts** als de la **vida real**. Poden tindre:
  - Propietats (**atributs**).
  - Comportaments (funcions, que en POO es coneixen com a **mètodes**).

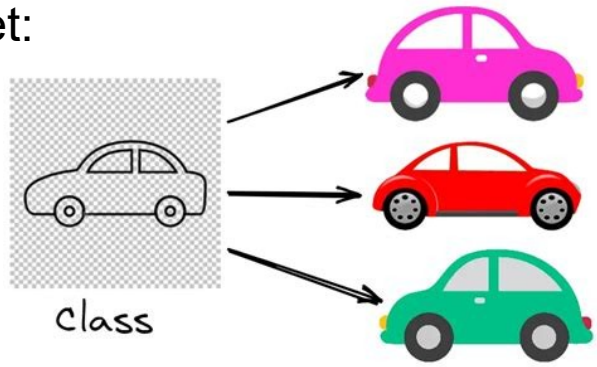
- En el **món real** cada objecte té **diferents comportaments** i **propietats**.
- Per exemple, podem tindre **diversos objectes televisors**:
  - Cada un tindrà una **mida**, **marca** i **funcions diferents**:
    - Al final **tots** es **fiquen en marxa** i **s'apaguen**.
- Els **objectes comparteixen funcions**:
  - Característica que s'anomena **herència** (ho veurem més endavant).

- **Classes**: **definicions dels objectes** (com si foren patrons o plantilles).
- Per exemple
  - Classe: **Cotxe**.
  - Objecte: Un **Ford Kuga** en **particular**.



- Una **classe** representa totes les **propietats** (***atributs***) i **comportaments** (***mètodes***) d'un objecte.

- Per **exemple**:
  - **Classe** Cotxe, té:
    - **Color**.
    - **Marca**.
    - Nombre de **portes**.
  - **Objecte** de Cotxe, en concret:
    - Color: **Rosa**.
    - Marca: **Ford**.
    - Nombre de portes: **2**.

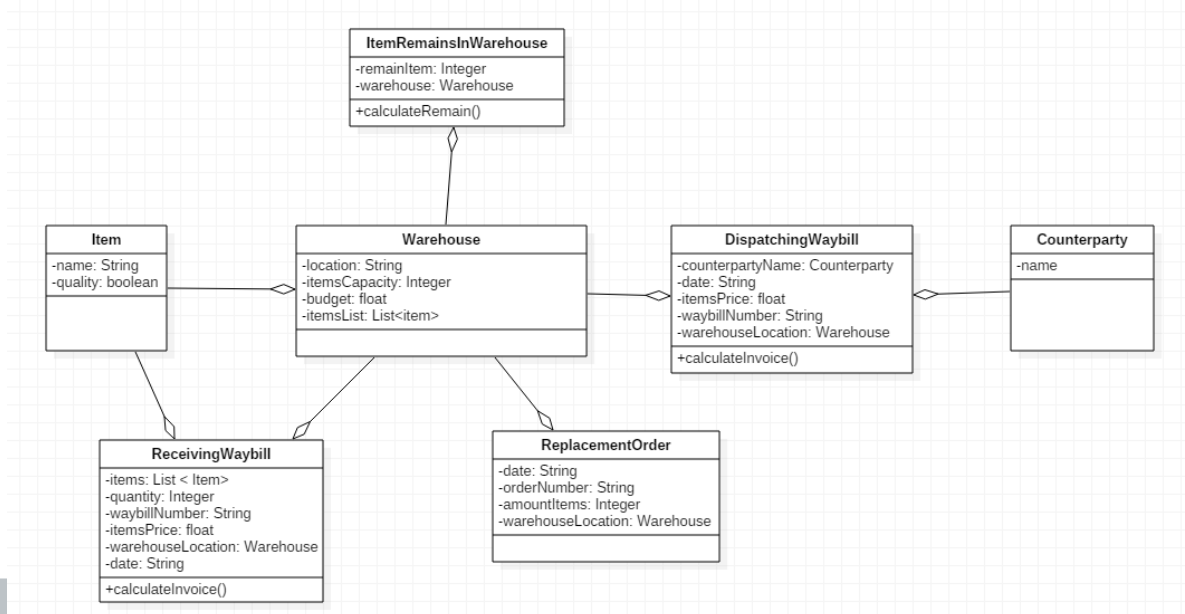


- **Instància**: objecte d'una classe.
  - L'objecte **Ford** és una **instància** de la classe **Cotxe**.

- **Avantatges** d'ús de les tècniques de **POO**:
  - **Reusabilitat del codi**:
    - El **codi** es tindrà més **separat**: les **Classes** podran ser **usades en altres projectes**.
      - Per **exemple**, si hi ha una **classe de calculadora** en un projecte:
        - Es vol **utilitzar en un altre projecte**:
          - Només s'ha de **copiar el codi d'aquesta classe**.
  - **Fàcil de mantenir**: Suposem que hem de **canviar el comportament** d'algun **mètode** d'una aplicació que fa de **calculadora**.
    - Amb POO **no caldrà anar a tots els llocs** on es fa servir la calculadora:
      - **Només al lloc on està definida la classe** de la calculadora.

- **Encapsulació**: **Amagar** les coses: **abstraure** la **lògica** d'un mètode **de** la seua **implementació**.
  - Només ens interessa **que fa el mètode** que farem servir:
    - **No ens cal saber com ho fa**.
  - **Exemple**:
    - Tenim una **classe que crea arxius PDF** i anem a utilitzar-la:
      - Només necessitem saber quins **mètodes** usar:
        - Però **no com funcionen** aquests mètodes **internament**:
          - D'això es va encarregar **el creador** de la classe.

- **Modularitat**:
  - Es creen **diverses classes** per a resoldre **un problema** o **crear** un **sistema**.
    - Permet realitzar **canvis** de manera més fàcil **sense afectar tot** el **sistema**.



- **Exemple de classe en PHP**:

```php
<?php

class Cotxe {

    //atributs / propietats
    public $color;
    public $nombre_portes;
    public $marca;
    public $gasoli;

    //mètodes
    function plenarDiposit($gasoli_nou){
        $this->gasoli = $this->gasoli + $gasoli_nou;
    }

    function accelerar(){
        $this->gasoli = $this->gasoli - 1;
    }

}

?>
```

Nom de la classe:
Per conveni, comença per majúscula

$this: referència a l'objecte invocador

Declaració / instanciació d'objectes : new()

- Els objectes s'han de guardar en variables, per exemple:

```
<?php
    $cotxe = new Cotxe(); // Instanciem un objecte la classe Cotxe
    $cotxe->color = 'Blau'; // Donem valor a algunes de les propietats
    $cotxe->marca = 'Ford'; // Per accedir als atributs utilitzem l'operador ->
    $cotxe->nombre_portes = 4;
    $cotxe->plenarDiposit(10); // utilitzem els mètodes
    $cotxe->accelerar(); // Per accedir als mètodes utilitzem també ->
    $cotxe->accelerar();
    $cotxe→accelerar();
?>
```

Exemple:

Definició de Classe	Instanciació d'objecte

class Persona{
  $nom;
  $edat;
 	
  function menjar($aliment){
  //codi del mètode
  }
 	
  function caminar($destinacio){
  //codi del mètode
  }
 	
  function estudiar($assignatura){
  //codi del mètode
  }
}
	$silvia = new Persona();
	
	$ruben = new Persona();

Visibilitat

- **Atributs**: **variables** definides en les classes que emmagatzemaran dades que poden **variar** d'**objecte a objecte**.
  - Similars a qualsevol variable que hem utilitzat fins ara.
  - Hem de **definir** la **seua visibilitat** en l'entorn de l'aplicació web.
- També hem de definir la **visibilitat** dels **mètodes**.
  - **Tres nivells** especificats pels tres **modificadors** disponibles.
    - `public`.
    - `private`.
    - `protected`.

- **public**: en PHP els **atributs** són declarats **public excepte si s'indica el contrari**:
  - **Es pot interactuar amb ells des de l'exterior d'un objecte**.
  - **No** és **sempre** el comportament més **desitjable**:
    - Podrien ser accedits o modificats des d'un context no desitjat.

- **private**: **només** s'hi podrà **accedir** des de l'**interior** de l'**objecte** que la **conté**.
  - Què passarà amb l'**herència** (concepte que veurem posteriorment) d'aquesta classe?

- **protected**: Només tindrà **visibilitat** per a la **pròpia classe** i les **seues classes descendents**.

- **Exemple**:

```
<?php
class Producte {
   public $preu;
   public $nom;
   public function metode($arg1, $arg2) {
        //codi del mètode
   }
 }
 //Utilitzem la classe definida anteriorment
 $P1 = new Producte();
 $P1->preu = 15;
 echo "Preu del producte".$P1->preu; //accés des de fora de la classe
```

- Per **accedir als atributs i mètodes** des de **dins** d'un altre **mètode** de la classe:
  - Pseudovariable especial: **`$this`** (**punter a l'objecte actual**)

```php
<?php
class Producte {
    private $nom;
    private $preu;
    private $id;
    public function getNom() {
      return $this->nom;
    }
    public function setNom($nom) {
      if ($nom! = $this->getNom())
        $this->nom = $nom;
    }
}
```

El símbol `$` s'utilitza en `$this`, no en en nom de l'atribut

Correcte: `$this->nom`

Incorrecte: `$this->$nom`

- Utilitzant d'**atributs privats i protegits**:
  - Podrem definir **mètodes públics**:
    - **Accessibles des de fora** de l'objecte definit:
      - **Interaccionen** amb els **atributs privats o protegits**.

- Podem **amagar** o **protegir** el que no volem que es puga **fer des de fora** de l'objecte.

- **getters** i **setters**: mètodes encarregats de **retornar o modificar** els **valors dels atributs** de les classes.
  - Els IDEs **automatitzen** la seua **creació**: eventualment **s'hauran de crear**.

```php
<?php

class Producte {
     //atributs
    private $nom;
    private $preu;
    private $id;

//getters i setters
public function getNom() {
      return $this->nom;
}
public function setNom($nom) {
      $this->nom = $nom;
}
public function getPreu() {
      return $this->preu;
}
public function setPreu($preu) {
      $this->preu = $preu;
}
public function getId () {
      return $this->id;
}
public function setId($id) {
      $this->id = $id;
}

}
```

- **Constants**: **atributs no han de variar el seu valor**.
  - Es declaren precedits per la **paraula clau const**.
  - **No porten el caràcter $ al davant**.
  - Per convenció els **noms** dels **atributs constants** es solen definir en **lletres majúscules**.
  - Si s'**intenta canviar el valor** obtindrem un **error**.
- Els atributs constants:
  - **Només** poden **emmagatzemar dades primitives**:
    - integer, float, string, array.
  - **No poden emmagatzemar objectes**.

- **Sintaxi** per **accedir** a les constants:
  - **Diferent** a com accedim als **altres atributs** (`->`).

- Si accedim **des de fora de la classe**:
  - Les **constants** es consideren **atributs públics**:
    - S'utilitza el **nom de la classe** seguit de '**::**' i el **nom de la constant**.

```
NomClasse::NOM_CONSTANT
```

- Si accedim **des de dins de la classe**:
  - S'utilitza la paraula reservada **self** seguida de '**::**' i del **nom de la constant**.

```
self::NOM_CONSTANT
```

- **Exemple**:

```
<?php
class Producte {
   const DISPONIBLE = 0;
   const FORA_DE_STOCK = 1;
   // ...  resta de codi ...
   function prova () {
     echo self::DISPONIBLE; //accés a constant des de dins de la classe
   }
 }
 echo Producte::DISPONIBLE; //accés a constant des de fora de la classe
?>
```

- **Constructor**: **mètode** especial per **inicialitzar els atributs** definits en la **classe**.
- És **cridat automàticament** cada vegada que es **crea** un **objecte**:
  - **No disposa de visibilitat**.
    - No es pot invocar des de dins o fora de l'objecte.
    - Amb l'excepció de la crida al constructor de la classe pare.
  - **Sintaxi**: similar a la dels **mètodes** de la classe:
    - Amb un **nom reservat**: **__construct**
    - **Sense** definició de **visibilitat**.

- **Exemple**:

```
<?php
class Producte{
    private $nom;
    private $preu;
    private $id;
    // ... resta del codi ...
    function __construct ($nom, $preu, $id) { // constructor
      $this->nom = $nom;
      $this->preu = $preu;
      $this->id = $id;
    }
}
?>
```

- Si **definim el constructor**, :
  - Haurem de crear l'objecte indicant la **quantitat exacta d'arguments** que s'han determinat i en l'**ordre correcte**:

```
$P1=new Producte(1,"Producte 1",50);
```

- Si **no definim el constructor**:
  - Només es podrà crear un nou objecte instanciant-lo **sense arguments**:

```
$P1=new Producte();
```

- **Destructor**: **mètode** especial que es **dispara automàticament** abans que s'elimine un objecte.
  - **No cal invocar-lo**.
  - Es definirà amb el **nom** de `__destruct()`.
  - **No accepta arguments**.
  - La **funcionalitat varia** depenent de la classe i de les inicialitzacions del constructor.
  - Normalment es farà servir per alliberar recursos inicialitzats:
    - Connexió amb la bases de dades,
    - Punters a fitxers,
    - Variables, etc ...

- **Exemple**:

```
<?php
class Producte{
    private $nom;
    private $preu;
    private $id;
    // ... resta del codi ...
    function __destruct(){  // destructor
        echo "S'ha destruït l'objecte <br>";
    }
}
?>
```

- Per **eliminar un objecte** en PHP es pot utilitzar la funció **`unset()`**.
- **Exemple**:

```php
<?php
    $p1=new Producte(1,"Producte 1",50); //s'executarà el constructor
    unset($p1); //s'executarà el destructor
?>
```