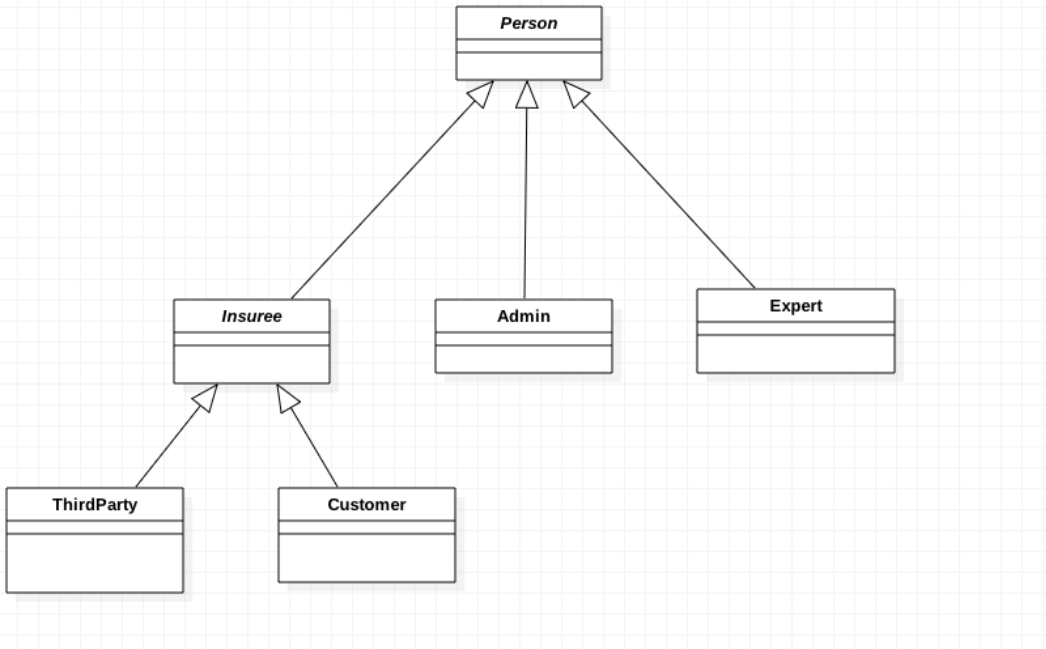
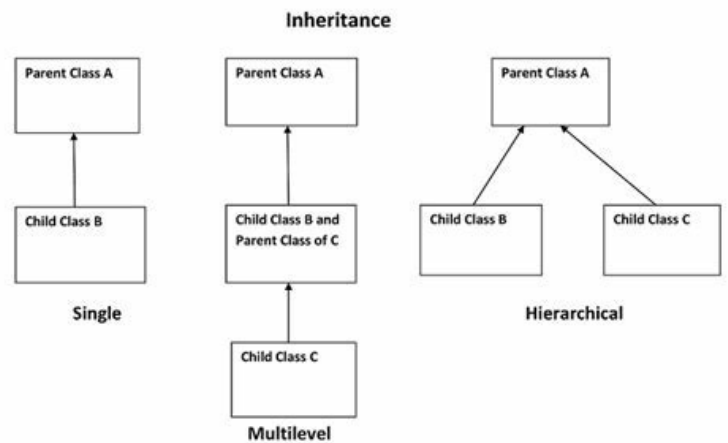


PHP 15

Herència (I)



19)Herència (I)

- **Herència**: una o diverses **classes** poden **derivar** d'una **classe base** o '**pare**'.

- Una classe que hereta d'una altra es diu **subclasse**:
  - **Hereta** tots els **atributs** i **mètodes** de la classe '**pare**'.
  - **Afegeix** les **seues pròpies** característiques (atributs i mètodes).
  - Sol dir que **una subclasse estén a la classe base**.

- El concepte d'**herència** és el mateix que en el **món real**:
  - Els **fills hereten** característiques dels **pares**.

- Una **subclasse disposarà** de tots els **atributs** i **mètodes** de la **classe base**.

- La classe **base no** podrà **utilitzar** cap de les **característiques** que estenguen les **classes hereves**.

- Només s'ha d'**aplicar herència** quan:
  - Hi ha una **relació "és un"** entre les classes.
  - La **subclasse** va a **tindre** en **comú tots** els **atributs** i **mètodes** de la **classe base:**
    - No només alguns.

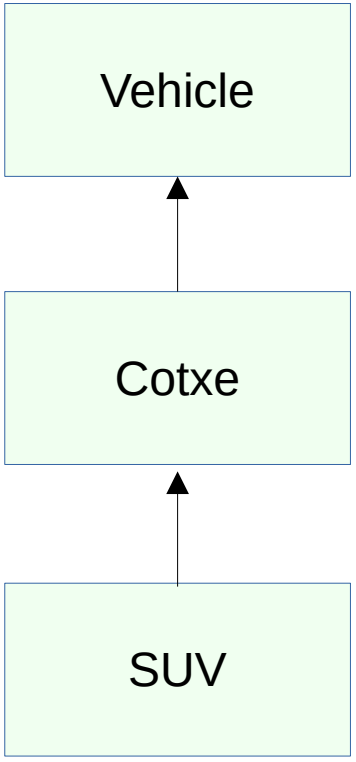
- **Error comú**: aplicar herència quan hi ha classes amb alguns atributs i mètodes en comú.
  - Influeix **negativament** en el **disseny** de classes:
    - S'està **generalitzant classes només** per **reutilitzar** codi:
      - **No** perquè tinguen una **relació** conceptual **pare-fill**.

Principals Avantatges:

- **Reutilització eficient del codi de la classe base**.
  - Mateix codi en dues classes, pare i fill: **només ho haurem escrit una vegada**.
- **Propagació dels canvis realitzats a la classe base**.
  - **Canvi** en les **classes base**: **accessible** a les **classes heretades**:
    - Sempre que la **visibilitat** introduïda ho permeta.
  - **Anar amb compte: Canvis** les **classes base** poden causar **problemes** en les **classes heretades** creades per **altres persones**.
  - Cal assegurar que els canvis són compatibles amb les classes heretades.
    - Per **exemple**: podríem **afegir nous atributs i mètodes** i **no modificar els existents**.

- L'**herència és transitiva**.
  - Una **classe** pot **heretar característiques** de la **classe base** que es troba **nivells més amunt** en la jerarquia d'herència.

- Per **exemple**:
  - Si la classe **SUV** és una subclasse de la classe **Cotxe**.
  - La classe **Cotxe** és una subclasse de la classe **Vehicle**:
  - Aleshores el **SUV** heretarà atributs tant de **Cotxe** com de **Vehicle**.



- PHP únicament suporta **herència simple**.

- És possible **simular l'herència múltiple** (una **classe hereta de diverses classes**):
  - Mitjançant l'ús d'**interfícies**.

- **Sintaxi** herència PHP:

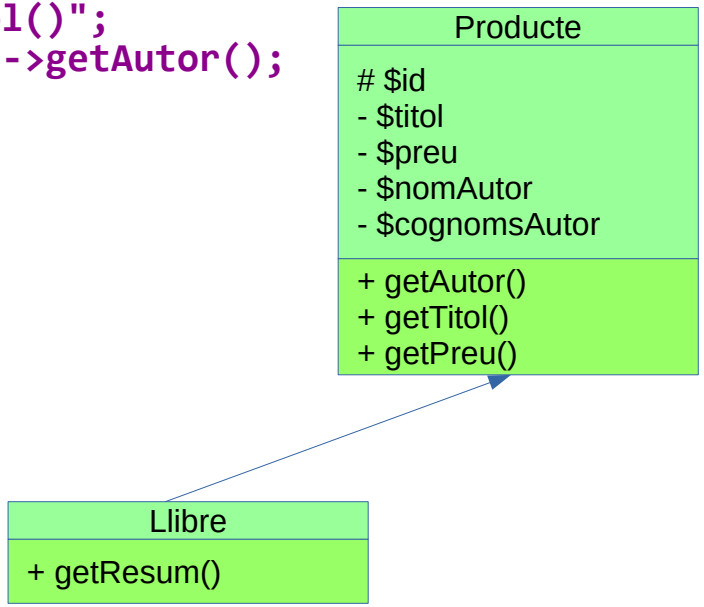
```
Fill extends Pare {
   //codi de la classe Fill
}
```

- **Exemple**: Classe Base (Producte)

```
class Producte {
    //atributs
   protected $id;
   private $titol;
   private $preu;
   private $nomAutor;
   private $cognomsAutor;
    //constructor
   function __construct ($id,
                $titol,
                $preu,
                $nomAutor,
                $cognomsAutor) {
     $this->id = $id;
     $this->titol = $titol;
     $this->nomAutor = $nomAutor;
     $this->cognomsAutor = $cognomsAutor;
     $this->preu = $preu;
   }
   //mètodes (getters)
   public function getId() {
     return $this->id;
   }
   public function getAutor() {
     return $this->nomAutor." ".
          $this->cognomsAutor;
   }
   public function getTitol() {
     return $this->titol;
   }
   public function getPreu() {
     return $this->preu;
   }
} //final de la classe base
```

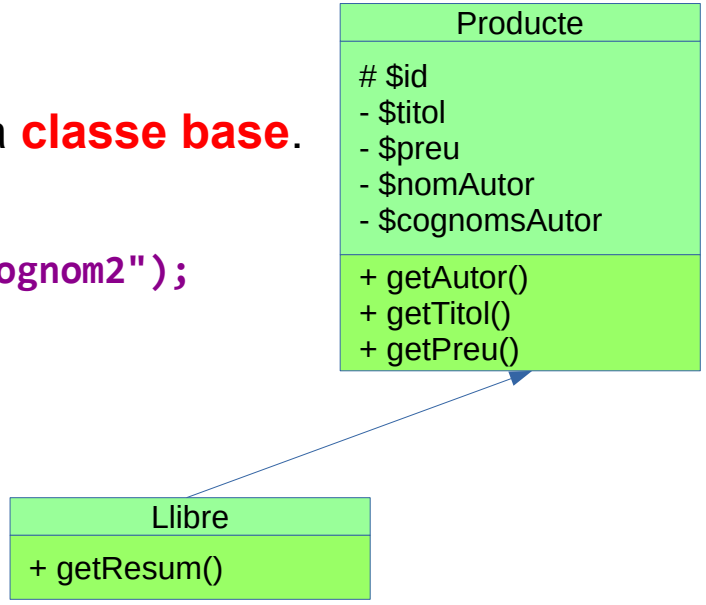
- **Exemple**: Classe que Hereta (Llibre)

```
class Llibre extends Producte {
   public function getResum() {
     $resum = "Id: ". $this->id.", Títol: ". $this->getTitol()";
     $resum.= ", Preu: ".$this->getPreu()", Autor: ". $this->getAutor();
     return $resum;
   }
 }
```



- **Exemple**: Producte → Llibre
- No hem definit **cap constructor específic** en classe Llibre:
  - **Heretarà** el de la **classe base**.
- Crear un objecte instanciat de la **classe Llibre**:
  - Indicar paràmetres que necessitava el **constructor** de la **classe base**.

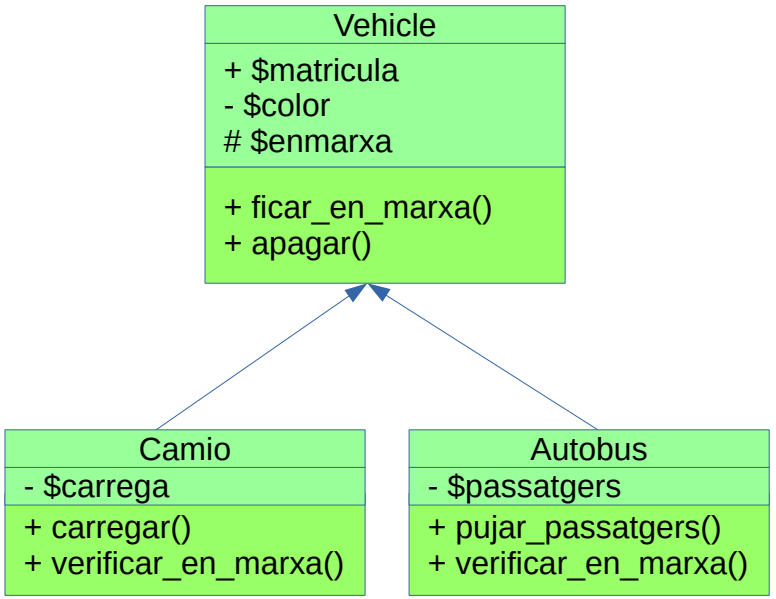
```
$llibre1 = new Llibre (1, "titol", 20, "Autor", "Cognom1 Cognom2");
echo $llibre1->getResum();
```



- **Exemple**: Classe Base (Vehicle)

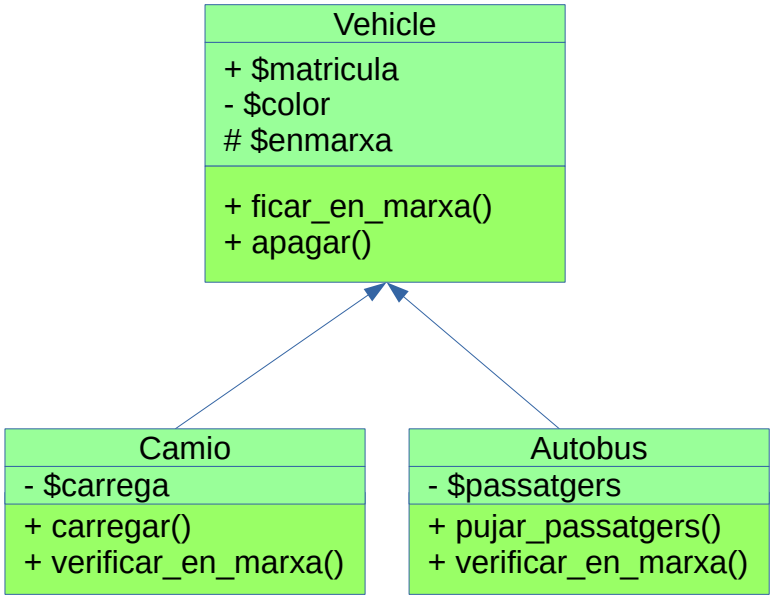
```
class Vehicle{
    //Atributs de la classe Vehicle, que s'heretaran
    public $matricula;
    private $color;
    protected $enmarxa;
    //Mètodes de la classe Vehicle
    public function ficar_en_marxa(){
        $this->enmarxa = true;
        echo 'Vehicle en marxa <br>';
    }

    public function apagar(){
        $this->enmarxa = false;
        echo 'Vehicle apagat <br>';
    }
} //final de la classe base
```



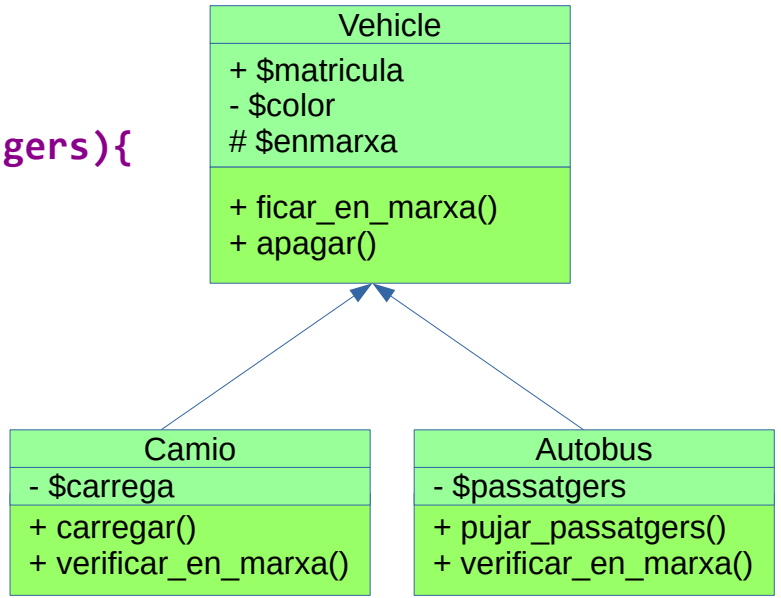
- **Exemple**: Classe Heretada (Camio)

```
class Camio extends Vehicle{
    private $carrega; //nou atribut classe Camio
    //nou mètode classe Camio
    public function carregar($quantitat_a_carregar){
        $this->carrega = $quantitat_a_carregar;
        echo "S'ha carregat quantitat: ".
             $quantitat_a_carregar." <br>";
    }
    //nou mètode classe Camio
    public function verificar_en_marxa(){
        if ($this->enmarxa == true){
            echo 'Camió en marxa <br>';
        }else{
            echo 'Camió apagat <br>';
        }
    }
} //final de la classe Camio
```



- **Exemple**: Classe Heretada (Autobus)

```
class Autobus extends Vehicle{
    private $passatgers; //nou atribut classe Autobus
    //nou mètode classe Autobus
    public function pujar_passatgers($quantitat_passatgers){
        $this->passatgers = $quantitat_passatgers;
        echo 'Han pujat '.
         $quantitat_passatgers.' passatgers <br>';
    }
    //nou mètode classe Autobus
    public function verificar_en_marxa(){
        if ($this->enmarxa == true){
            echo 'Autobús en marxa <br>';
        }else{
            echo 'Autobús apagat <br>';
        }
    }
} //final de la classe Autobus
```



Herència i constructors:

- Quan definim un **constructor propi** a la **classe filla**:
  - Podríem voler **passar** els **arguments** al **constructor** de la **classe base.**

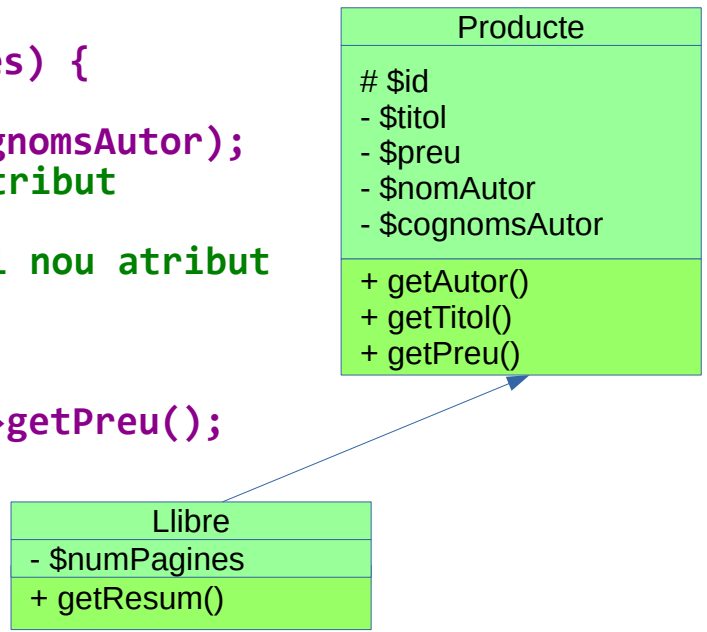
- Per **accedir** al **mètode** la **classe base** s'utilitza la següent sintaxi:

```
parent::__construct();
```

- Així **invocarem al constructor de la classe base** de la qual que hereta la classe actual.

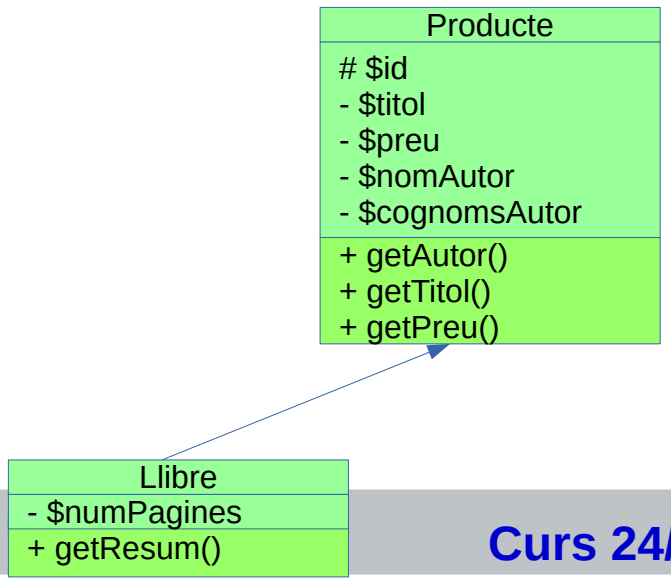
- **Exemple**: Classe Heretada (Producte → Llibre)

```
class Llibre extends Producte {
   private $numPagines; //Nou atribut de la classe Llibre
   function __construct ($id, $titol,$preu,
                         $nomAutor, $cognomsAutor, $numPagines) {
     //crida al constructor de la classe pare
      parent::__construct ($id, $titol, $preu, $nomAutor, $cognomsAutor);
      $this->numPagines = $numPagines; //assignem també nou atribut
   }
   public function getNumPagines() { // nou mètode getter del nou atribut
     return $this->numPagines;
   }
   public function getResum() {
     $resum = "Títol:". $this->getTitol(). ", Preu:". $this->getPreu();
     $resum.= ", Autor:". $this->getAutor().
      ", Núm.:". $this->getNumPagines();
     return $resum;
   }
 } //final de la classe Llibre
```



- **Exemple**: Classe Heretada (Producte → Llibre)
- Per crear un **nou llibre**, ara podrem utilitzar el nou constructor:

```
//hem afegit un paràmetre al constructor original de Producte
$llibre1 = new Llibre(1, "títol", 20, "Autor", "Cognom1 Cognom2", 440);
echo $llibre1->getResum();
```



Herència i constructors:

- Els **destructors de la classe base no seran cridats implícitament per PHP**.
- Si volem executar un **destructor de la classe pare**:
  - S'haurà de **cridar explícitament** a l'interior del destructor definit (sobreescrit):

```
parent::__destruct();
```

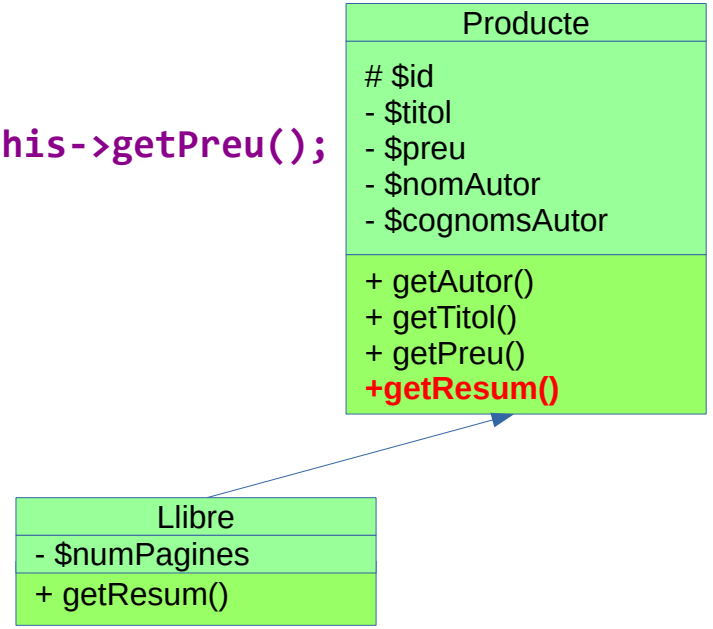
- **Si les classes filles no implementen** un **destructor**:
  - **S'heretarà i utilitzarà el de la classe base**.

Sobreescriptura:

- **Sobreescriure**: **afegir funcionalitat** a un **mètode heretat**:
  - Cridat de la mateixa manera en les dues classes.
  - Estem sobreescrivint el funcionament del mateix mètode.
  - Es pot **utilitzar** amb **constructors** i amb **qualsevol mètode heretat** d'una classe base.
- El més convenient és **sobreescriure mètodes** per **completar l'algorisme** del **mètode** de la **classe base**.
- **No és recomanable** sobreescriure un mètode i **canviar completament** el seu comportament original (en la classe pare):
  - Per això el **millor** és crear un **nou mètode** en la classe filla.

- **Exemple**: Classe Pare (Producte → Llibre)

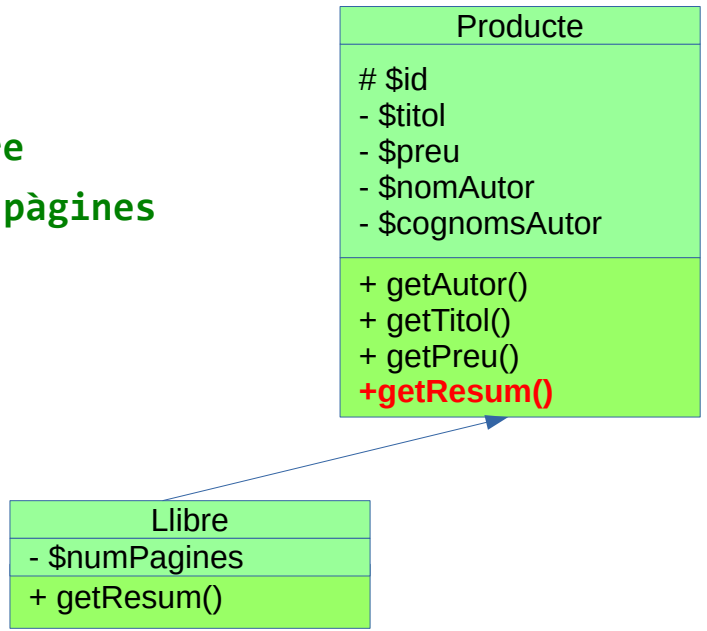
```
class Producte {
    // ...resta del codi
    protected function getResum() {
        $resum = "Títol:". $this->getTitol(). ", Preu:". $this->getPreu();
        $resum.= ", Autor:". $this->getAutor();
        return $resum;
    }
} //final de la classe Producte
```



Sobreescriptura

- **Exemple**: Classe Filla (Producte → Llibre)

```
class Llibre extends Producte {
   // resta del codi
   public function getResum() {
     $resum = parent::getResum(); // mètode de la classe pare
     $resum.= ", Núm.:". $this->getNumPagines(); // afegeix pàgines
     return $resum;
   }
 } //final de la classe Llibre
```



Visibilitat i Herència:

- **Atributs** i **mètodes** declarats com a **públics (`public`)**:
  - Es poden **utilitzar** des de **qualsevol context**:
    - Des de la mateixa **classe**, des d'una **classe heretada** o des de l'**exterior**.

- **Atributs** i **mètodes** declarats com a **privats (`private`):**
  - **Només accessibles des de l'interior** de la **classe on estan declarats**.
  - **No es podran sobreescriure característiques privades en les classes heretades**:
    - Les classes **filles no tindran accés** a aquestes **característiques privades**.

Visibilitat i Herència:

- **Atributs** i **mètodes** declarats com a **protegits (`protected`)** :
  - Només podran ser **accedits**:
    - Des de la **classe que els declara o**
    - Una classe que **hereta**.
  - **Mai** podran ser accedits des de **l'exterior** d'una **classe**.

- Utilitzant les característiques de **visibilitat** es pot:
  - **Exposar** característiques requerides per un client (que podria ser una altra classe).
  - **Evitar** que es puguen **modificar lliurement atributs d'un objecte**.

- Cal procurar que **qualsevol modificació d'un atribut**:
  - **Es faça mitjançant mètodes d'accés**:

- **Exemple**: Classe Pare (**Producte** → Llibre)

```
class Producte {
     // Atributs
   protected $id;
   private $titol;
   private $preu;
   private $nomAutor;
   private $cognomsAutor;

     //Constructor
   function __construct ($id, $titol, $preu,
                  $nomAutor, $cognomsAutor) {
     $this->id = $id;
     $this->titol = $titol;
     $this->preu = $preu;
     $this->nomAutor = $nomAutor;
     $this->cognomsAutor = $cognomsAutor;
   }
```

```
//mètodes getters
public function getId() {
  return $this->id;
}
public function getAutor() {
  return $this->nomAutor. "".
           $this->cognomsAutor;
}
public function getTitol() {
  return $this->titol;
}
public function getPreu() {
  return $this->preu;
}
protected function getResum() {
  $resum = "Títol:". $this->getTitol().
           ", Preu:". $this->getPreu();
  $resum.= ", Autor:". $this->getAutor();
  return $resum;
}
```

- **Exemple**: Classe Pare (**Producte** → Llibre)

```
  // mètodes setters
  public function setId($id) {
    $this->id = $id;
  }
  public function setTitol($titol) {
    $this->titol = $titol;
  }
  public function setPreu($preu) {
    $this->preu = $preu;
  }
  public function setNomAutor ($nomAutor) {
    $this->nomAutor = $nomAutor;
  }
  public function setCognomsAutor ($cognomsAutor) {
    $this->cognomsAutor = $cognomsAutor;
  }
} //final classe Producte
```

- **Exemple**: Classe Filla (Producte → **Llibre**)

```
class Llibre extends Producte {
   private $numPagines; // Atribut nou afegit
     //Constructor que modifica el de la classe pare
   function __construct ($id, $titol, $preu, $nomAutor, $cognomsAutor, $numPagines) {
     parent::__construct ($id, $titol, $preu, $nomAutor, $cognomsAutor); //crida al pare
     $this->numPagines = $numPagines;
   }
     //mètodes getter / setter
   public function getNumPagines() {
     return $this->numPagines;
   }
   public function setNumPagines ($numPagines) {
     return $this->numPagines = $numPagines;
   }
     //reescriptura d'un mètode del pare, aprofitant el seu contingut
   public function getResum() {
     $resum = parent::getResum();  // crida a la mateixa funció del pare
     $resum.= ", Núm.:". $this->getNumPagines();
     return $resum;
   }
 }
```