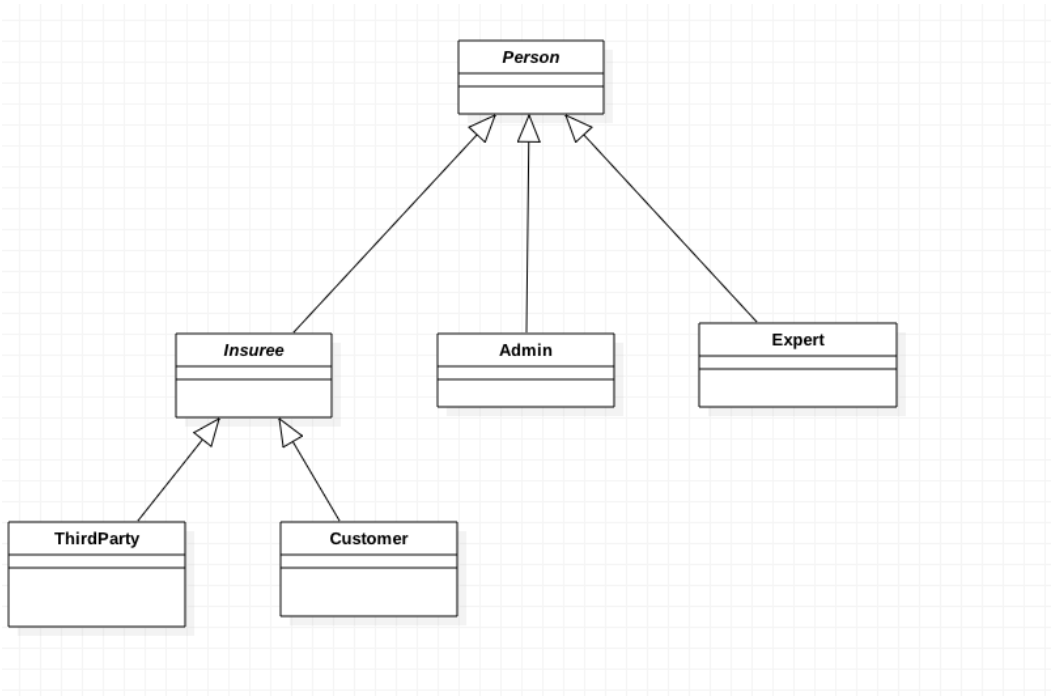

PHP 16

Herència (II)

19)Herència (II)

Sobreescriure mètodes.

- **Sobreescriptura** o **substitució** de **mètodes:** mecanisme pel qual una **classe** que **hereta** pot **redefinir** els **mètodes** que està heretant.
- **Exemple**: cafetera.
- Hi ha **molts tipus de cafeteres**:
  - Totes **fan cafè**.
  - Però el **mecanisme** per fer el cafè és **diferent** depenent del tipus de cafetera:
    - Cafeteres **express**.
    - Cafeteres amb **filtre**.
    - Cafeteres amb **càpsules**, etc.

- La nostra cafetera "**pare**" (de la qual va a **heretar totes** les cafeteres):
  - Pot tenir definit un mètode `ferCafe()`.
  - **No** necessàriament **totes** les **cafeteres** que **hereten fan el cafè** d'igual **manera**.
  - Podem definir un **mètode** per fer cafè **estàndard** en la classe **pare** cafetera.
  - En les classes `CafeteraExpress` i `CafeteraFiltre`:
    - **Sobreescriure** el mètode `ferCafe()` ajustant-lo al procediment propi d'aquestes.

- **Exemple**: Classe pare **Suport**.

```
class Suport {
    public $titol;
    protected $numero;
    private $preu;
     //constructor
    function __construct ($tit, $num, $preu) {
       $this->titol = $tit;
       $this->numero = $num;
       $this->preu = $preu;
    }
     //funció per mostrar els valors per pantalla
    public function imprimirCaracteristiques() {
        echo "<br>". $this->titol;
        echo "<br>". $this->preu. "(IVA no inclòs)";
    }
//getters
    public function getPreuSenseIva() {
       return $this->preu;
    }

    public function getPreuAmbIva() {
       return $this->preu * 1.21;
    }

    public function getNumeroIdentificacio() {
       return $this->numero;
    }
} // final de la classe suport
```

- **Exemple**: Classe filla **Bluray**.

```
class Bluray extends Suport { //Bluray hereta de Suport
   private $durada; //afegim un nou atribut als que ja té la classe pare
     //definim el constructor de la nova classe que hereta de suport
   function __construct ($tit, $num, $preu, $durada) { //constructor amb el nou atribut durada
      //aprofitem el constructor de la classe pare
      parent::__construct ($tit, $num, $preu);
      $this->durada = $durada;
   }
   public function imprimirCaracteristiques () { //sobreescrivim aquest mètode del Pare
      echo "Pel·lícula en BluRay: ";
     //aprofitem el codi declarat en la classe pare amb l'operador d'àmbit parent::
     parent::imprimirCaracteristiques ();
     echo "<br> Durada: ". $this->durada; //mostrem el nou atribut també
   }
}
```

- En aquest **exemple**:
  - La classe `Suport` està inclòs en el mateix **arxiu** de la classe `Bluray`.
    - Podem escriure els dos codis en el mateix fitxer.

- També es poden ficar en **fitxers independents**:
  - Hem d'incloure la classe `Suport` en el fitxer de la classe **Bluray**:
    - Amb la instrucció `include` o `require` de PHP.

- Si volem provar les classes de l'**exemple** podríem fer:

```
$bluray01 = new Bluray("Avatar", 22, 15.5, "240 minuts");
echo "<strong>". $bluray01->titol. "</strong>";
echo "<br> Preu: ". $bluray01->getPreuSenseIva(). " Euros";
echo "<br> IVA inclòs: ". $bluray01->getPreuAmbIva(). " Euros";
echo "<br>". $bluray01->imprimirCaracteristiques();
```

- Mostraria com a **resultat**:

Avatar

Preu:15.5 Euros

IVA inclòs:18.755 Euros

Pel·lícula en BluRay:

Avatar

15.5 (IVA no inclòs)

Durada: 240 minuts

- En PHP 5 es van introduir **classes** i **mètodes abstractes**.
- **Classes abstractes**:
  - **No es poden instanciar**.
  - **Qualsevol classe amb almenys un mètode abstracte:**
    - **S'ha de definir com a abstracta**.
- **Mètodes abstractes**:
  - Simplement **declaren la interfície** (nom i paràmetres que requereix) del mètode.
  - **No poden definir la implementació**.

- Quan **s'hereta d'una classe abstracta**:
- **Tots** els **mètodes** definits com a **abstractes** en la declaració de la classe pare:
  - Han de ser **definits** en la **classe filla**:
    - **Amb la mateixa visibilitat** (o amb una **menys restrictiva**):
    - Si el **mètode abstracte** està definit com a **protegit**:
      - La **implementació** de la **funció** ha de ser definida com a **protegida o pública**.
      - **Mai** com a **privada**.
  - Les **definicions** dels mètodes han de **coincidir**:
    - La **declaració de tipus i el nom d'arguments requerits han de ser els mateixos**.

- Objectiu de l'ús de classes **abstractes**:
  - Definir una **estructura** (**plantilla**):
    - Pot estar **parcialment implementada**:
      - Pot **implementar certes funcionalitats**.
      - **Deixar** que els seus **hereus acaben** d'**implementar-la**.
    - Per a **qualsevol classe** que vulga **estendre-la**.

- **Exemple**:

```
abstract class ClasseAbstracta{
     // Forçar l'extensió de classe
     // per definir aquest mètode
     abstract protected function getValor();
     abstract protected function valorPrefix($prefix);
     // Mètode comú
     public function imprimir() {
         print $this->getValor()."<br>";
     }
}
```

```
class ClasseConcreta1 extends ClasseAbstracta
     //s'ha d'implementar
     protected function getValor() {
         return "ClasseConcreta1";
     }
     //s'ha d'implementar
     public function valorPrefix($prefix) {
         return $prefix."ClasseConcreta1";
     }
}
```

```
class ClasseConcreta2 extends ClasseAbstracta{
     //s'ha d'implementar
     public function getValor() {
         return "ClasseConcreta2";
     }
     //s'ha d'implementar
     public function valorPrefix($prefix) {
         return $prefix."ClasseConcreta2";
     }
}
```

- Si volem provar les classes de l'**exemple** podríem fer:

```
$ClasseAbstracta = new ClasseAbstracta(); //error fatal
$Classe1 = new ClasseConcreta1;
$Classe1->imprimir();
echo $Classe1->valorPrefix('PRE_')."<br>";
$Classe2 = new ClasseConcreta2;
$Classe2->imprimir();
echo $Classe2->valorPrefix('PRE_')."<br>";
```

No es pot instanciar una classe abstracta

- Mostraria com a **resultat**:

ClasseConcreta1
PRE_ClasseConcreta1
ClasseConcreta2
PRE_ClasseConcreta2

- **Característiques principals de les classes abstractes**:
  - Una **classe abstracta no pot instanciar-se**:
    - Sí que **es pot instanciar una classe filla no abstracta**.
  - Una **classe abstracta** ha de tindre com a **mínim un mètode abstracte**.
  - Una classe **classe abstracta A** pot ser **estesa** per una **classe abstracta B**.
    - La classe B **pot implementar o no** els **mètodes abstractes** de la classe A.
    - **Si no fóra abstracta** sí que estaria **obligada** a **implementar** els mètodes.

```
abstract class Animal {
  abstract function so();
}
```

```
abstract class Animalet extends Animal {
   abstract function correr();
   public function dormir() {
     return "Està dormint";
   }
}
```

- Si una classe C estén l'anterior classe abstracta B que estenia la classe abstracta A:
  - Ha d'implementar tots els mètodes abstractes de B.
  - Ha d'implementar els mètodes abstractes d'A que no s'havien implementat en B.

```
class Gat extends Animalet {
   public function so() { //ha d'implementar-la
     return "Miauuuuuuuu !!!";
   }
   public function correr () { //ha d'implementar-la
     return "Està corrent";
   }
//la funció dormir, que no és abstracta,no té perquè implementar-la
}
```

- Si volem provar les classes de l'**exemple** podríem fer:

```
$gat = new Gat();
echo $gat->so(). "<br>";
echo $gat->correr(). "<br>";
echo $gat->dormir(). "<br>";
```

- Mostraria com a **resultat**:

Miauuuuuuuu !!!

Està corrent

Està dormint

- Els **mètodes abstractes** es definiran amb una **visibilitat**:
  - Les seues implementacions en les classes hereves:
    - Han de tenir la mateixa visibilitat o una de menys restrictiva.

```php
abstract class Animal {
  abstract function so();
}
```

```php
abstract class Animalet extends Animal {
   public function dormir() {
     return "Està dormint";
   }
   abstract protected function correr();
 }
```

```php
class Gat extends Animalet {
     //public: mateixa accessibilitat que Animal
   public function so() {
     return "Miauuuuuuuu !!!";
   }
     //private: més restrictiva que Animalet (protected)
   private function correr() { //error
     return "Està corrent";
   }
 }
```

Només pot ser **protected** o **public**

- Si ho executàrem, obtindríem el següent resultat:

```
"Fatal error: Access level to Gat::correr() must be protected (as in class Animalet) or weaker in ..."
```

- La funció **`correr()`** a **`Gat`** ha de ser **`protected`** o **`public`** però no **`private`**:
  - **`private`** és una visibilitat més restrictiva.
- Els altres **mètodes no abstractes** que heretem:
  - Hauran de tenir la **mateixa visibilitat** en cas de **sobreescriure**.

- **Mètodes abstractes heretats**:
  - Poden ser implementats amb **arguments opcionals** no definits en la classe pare.

- Si el **nou argument** no és **opcional** (no té un valor per **defecte**):
  - **Provocarà un error:**

Fatal error: Declaration of ... must be compatible with ...

```
abstract class Animalet extends Animal {
   public function dormir() {
     return "Està dormint";
   }
   abstract public function correr();
 }
```

Classe pare:
Sense paràmetres

- Argument en classe que **hereta**:

```php
class Gat extends Animalet {
   public function so() {
     return "Miauuuuuuuu !!!";
   }
    //amb argument opcional
   public function correr($content = TRUE) {
     if (!$content){
          return "Està corrent cabrejat";
     }
     return "Està corrent";
   }

}
```

Opcional: **Correcte**

```php
class Gat extends Animalet {
   public function so() {
     return "Miauuuuuuuu !!!";
   }
    //amb argument no opcional
   public function correr($content) {
     if (!$content){
          return "Està corrent cabrejat";
     }
     return "Està corrent";
   }

}
```

No és opcional: **Error**

- **Exemple** classe **Abstracta**:

```
abstract class Poligon{
 // declarem mètode abstracte
  abstract function calcul();
}
```

```
class classQuadrat extends Poligon{
  function calcul() {
       echo "Àrea d'un quadrat: a = costat * costat <br>";
  }
}
```

```
class classRectangle extends Poligon{
  function calcul()  {
     echo "Àrea d'un rectangle: a = base * altura <br>";
  }
}
```

```
class classTriangle extends Poligon{
  function calcul () {
     echo "Àrea d'un triangle: a = (base * altura) / 2 <br>";
  }
}
```

- **Exemple** classe **Abstracta**:
- El codi per provar-les:

```
<?php
//Creem els objectes necessaris
$quadrat = new classQuadrat();
$rectangle = new classRectangle();
$triangle = new classTriangle();
//Comprovem la crida a la funció càlcul en cada objecte
$quadrat->calcul();
$rectangle->calcul();
$triangle->calcul();
?>
```

- Mostra a l'eixida:

Àrea d'un quadrat: a = costat * costat

Àrea d'un rectangle: a = base * altura

Àrea d'un triangle: a = (base * altura) / 2

- **Interfície**: **plantilla pura** que únicament **definirà funcionalitats**.
- Es **pareix** molt a una **classe abstracta** pel que fa a la seua estructura:
  - **Diferència**: **cap** dels mètodes que defineix tenen **implementada** la seua **lògica**.
- Ja no **parlem de** classes filles:
  - Parlem de **classes** que **implementen** la **interfície** :
    - Encarregades d'**implementar**, **obligatòriament**, la **funcionalitat** definida pels **mètodes** de la **interfície**.
- Exemple **Sintaxi**:

```
interface Logger {
    public function log($missatge);
}
```

- Perquè una classe **implemente** una **interfície**: operador **implements**.

```
class FileLogger implements Logger {
   private $gestor; // atributs propis de la classe
   private $fitxerLog;
   function __construct($nomFitxer, $modeObertura = 'a') { //constructor
     $this->fitxerLog = $nomFitxer;
     $this->gestor = fopen($nomFitxer, $modeObertura) or die ( 'No es pot obrir el fitxer');
   }
   public function log ($missatge) { //funció de la interfície, obligatòria
     $missatge = date ( "F j, I, g: i : s a"). ':'. $missatge.PHP_EOL ;
     fwrite ($this->gestor, $missatge);
   }
   function __destruct() { //destructor
     if ($this->gestor) { //quan acabem, tanquem el fitxer
       fclose($this->gestor);
     }
   }
 }
```

Característiques principals de les interfícies (I).

1)Una **interfície només indicarà la definició dels mètodes**.

- **Codi intern**: es concretarà les **classes que implementen** la interfície.

- **Les interfícies no es poden instanciar**:
  - **Instanciarem** les **classes que implementen** una **interfície**.

2)Tots els **mètodes** declarats en una **interfície** han de ser **públics**.

3)Els **mètodes** de la **classe** que **implementa** una interfície:

- **Mateixa visibilitat** que en la interfície: sempre **públics**.
  - Diferència classes **abstractes**:
    - Mètodes abstractes classes filles: **mateixa visibilitat** o menys **restrictiva**.

Característiques principals de les interfícies (II).

4)Classe que implemente una interfície:

- Ha de **definir els mètodes definits en la interfície**.
- Pot definir els seus **mètodes propis** i **atributs**.

5)Només es poden definir constants (no es poden definir atributs):

- **Heretades automàticament** per les **classes** que la **implementen**.
- Diferència classes **abstractes**:
  - Es pot definir tot tipus d'**atributs** (amb qualsevol **visibilitat**).

Característiques principals de les interfícies (III).

6) **Compte** quan es **defineix** la **interfície** i els **noms** dels seus **mètodes**:

- Un **canvi** en la **interfície**:
  - Cal **modificar totes** les **classes** que la **implementen**.

7)Les **interfícies també poden ser heretades** per mitjà de la paraula **extends**.

```
interface InterficieA {
   public function primerMetode($nom);
 }
interface InterficieB extends InterficieA {
   public function segonMetode();
 }
```

```
class Exemple implements InterficieB {
   public function primerMetode($nom) {
   }
   public function segonMetode() {
   }
}
```

Característiques principals de les interfícies (i IV).

8) **Una classe només pot estendre d'una classe abstracta**:

- **Una classe pot implementar més d'una interfície**:
  - **Separant** cadascuna **per** una **coma**.
    - Haurà d'implementar els **mètodes de cadascuna de les interfícies**.

```
interface InterficieA {
   public function primerMetode($nom);
 }
interface InterficieB {
   public function segonMetode();
 }
```

```
class Exemple implements InterficieA, InterficieB {
   public function primerMetode($nom) {
   }
   public function segonMetode() {
   }
}
```

Diferències entre classes abstractes i interfícies.

A)Classe abstracta: ha de contenir com a **mínim un mètode abstracte** (abstract):

- Només s'especifica el nom i **no s'implementa**.
- Els altres mètodes de la classe poden estar completament implementats.

- **Interfície: no pot implementar mètodes** :
  - Només pot definir el seu nom.

Diferències entre classes abstractes i interfícies.

B)Classe abstracta: un mètode `abstract` es pot definir `public`, `protected` o `private`.

- Subclasses que **hereten** :
  - Implementen mètodes amb la **mateixa visibilitat o** una amb **menor** restricció.

- En una **interfície** tots els **mètodes** són **públics**.

Diferències entre classes abstractes i interfícies.

C)Classe abstracta: es pot definir:

- **Mètodes**, amb la seua visibilitat.
- **Atributs**, amb la seua visibilitat.
- **Constants**.

- **Interfície**, es pot definir:
  - **Mètodes**, que només poden ser públics.
  - **Constants**.

Diferències entre classes abstractes i interfícies.

D)Classe :

- Pot **heretar** només d'**una classe pare (abstracta o no)**.
- Pot **implementar més d'una interfície**.

Diferències entre classes abstractes i interfícies.

E)Usant **classes abstractes**:

- Una classe **filla** pot **sobreescriure o no** un **mètode** definit en la classe pare.
- Si el **mètode** és **abstracte** sí que està **obligada** a implementar-lo i per tant sobreescriure.

- Una **classe** que **implementa** una **interfície** :
  - **Obligada** a **sobreescriure tots** els **mètodes**.
    - Una interfície només els defineix però no els implementa.

- **Classe abstracta només** té **mètodes abstractes**:
  - S'està usant **com si fora una interfície**.

- Les **classes abstractes** s'utilitzen per **compartir funcions**.

- Les **interfícies** s'utilitzen per:
  - Compartir **com s'ha de fer** alguna cosa.
  - **Què ha de tenir com** a **mínim**.