ENTORNOS DE DESARROLLO

UML - POO - PHP

WALTER ISAMEL SAGASTEGUI LESCANO

1 DE ENERO DE 2021
CAMPUSFP
HUMANES-GETAFE

INDEX

1. POO

CLASE – OBJETO

- Para crear una instacia de una clase se us el operador **new**.
- Para acceder a las propiedades y métodos de un objeto se usa el operador **->**.
- Cada vez que se instancia una clase se crea un objeto nuevo.
- Para asignar valores o referirnos a la propiedad dentro de una clase se utiliza la palabra reservada **$this**.
- Terminos usados en una clase: (1) propiedades, comportamiento (2) atributos, acciones (3) variables de instancia, métodos=funciones.

CONSTRUCTOR

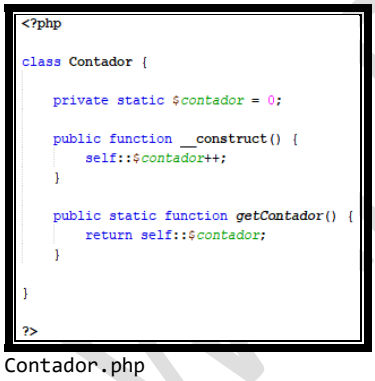
- La función del método constructor es conseguir que el objeto sea creado con valores iniciales para sus variables de instancia es decir inicializar las variables de instancias cuando se crea el objeto.
- El método constructor se ejecuta cada vez que se instancia la clase.
- No siempre es necesario contar con un constructor.
- Dentro del código de un constructor se suelen asignar los valores de algunas o todas las propiedades de dicho objeto.

CONSTANTE

- Una constante es un tipo de atributo que pertenece a la clase cuyo valor nunca cambia.
- Los nombres de las constantes siempre estan en mayuscula que es una convención de nomenclatura.
- Para declarar una constante, debe preceder su nombre con la palabra clave **const**.
- Una constante no lleva un **$** delante de su nombre.
- A diferencia de los atributos, no se puede acceder al valor de una constante mediante el operador -> desde un objeto (ni $this ni $alumno no funcionará) pero con el operador :: porque una constante pertenece a la clase y no a ningún objeto.
- Para acceder a una constante, debe especificar el nombre de la clase, seguido del símbolo de dos puntos dobles, seguido del nombre de la constante.
- El operador -> permite el acceso a un elemento del objeto, mientras que el operador :: permite el acceso a un elemento de la clase.
- Las constantes de clase son útiles para evitar tener un código tonto, es decir, es decir nos dice como funciona nuestro código.

SELF – PARENT

- Cuando queramos accesar a una constante o metodo estatico por ejemplo desde dentro de la clase podemos usar esta palabra reservada self.
- Cuando queramos acceder a una constante o metodo de una clase padre, la palabra reservada parent nos sirve para llamarla desde una clase extendida.
- En la siguiente imagen es imporante entender porque se usa self y no &this y es porque no se puede acceder a un atributo estático con $this pero si con self que significa con uno mismo representa a la clase mientras que $this representa el objeto creado actualmente. Si un atributo estático es modificado, no solo se modifica en el objeto creado sino en la estructura completa de la clase.



```php
<?php

class Contador {

    private static $contador = 0;

    public function __construct() {
        self::$contador++;
    }

    public static function getContador() {
        return self::$contador;
    }

}

?>
```

Contador.php

```php
<?php

require_once "Contador.php";

$test1 = new Contador;
$test2 = new Contador;
$test3 = new Contador;

echo Contador::getContador(); //Output: 3

?>
```

Principal.php

ESTATICO

Los atributos y métodos estáticos así como las constantes de clase son elementos específicos de la clase, es decir que no es útil crear un objeto para usarlo.

ENUNCIADOS

Hacer una clase que permita mostrar el número de veces que se ha creado una instancia de la clase. Para esto necesitaremos un atributo perteneciente a la clase digamos $contador que se incrementa en el constructor.

HERENCIA

- Una clase puede heredar de otra usando la palabra reservada **extends**.
- Una clase Padre solo puede heredar sus variables y métodos con visibilidad publica (**public**) o protegida (**protected**) y no privada (**private**).
- Las clases que heredan de otra clase pueden cambiar el comportamiento de la clase padre sobreescribiendo sus método.
- PHP no permite la herencia multiple, es decir que no puedes utilizar extends de la siguiente forma: **class Chind extends Parent, Other {}**

ABSTRACTA

- Una clase abstracta solo puede heredar sus variables y métodos con visibilidad publica (**public**) o protegida (**protected**) ademas de sus métodos abstractos(**abstract**).
- Para crear una clase Abstracta debes de utilizar la palabra reservada **abstract** antes de la palabra **class**.
- Las clases abstractas son clases que no se instancian y sólo pueden ser heredadas, trasladando así un funcionamiento obligatorio a clases hijas. Mejoran la calidad del código y ayudan a reducir la cantidad de código duplicado.
- Las clases abstractas pueden extenderse unas a otras, así como extender clases normales.

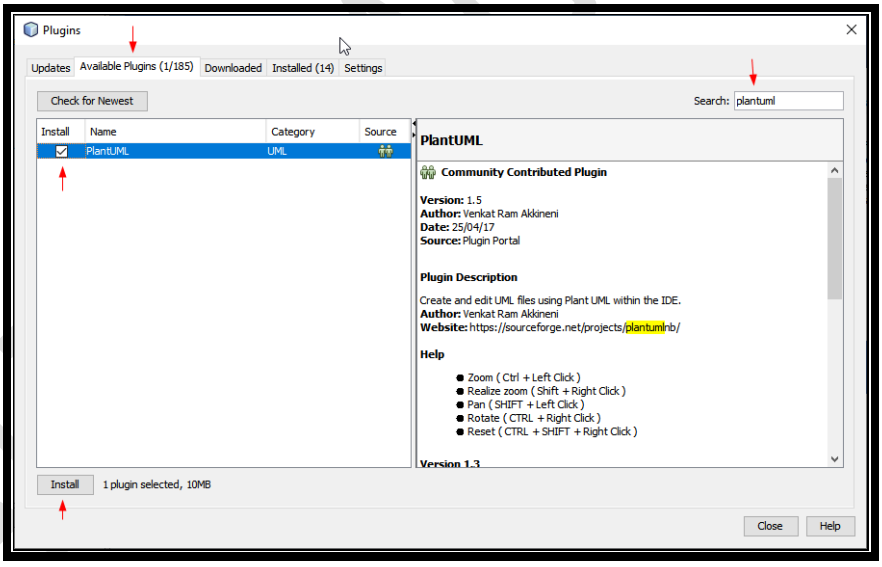
2. PLANTUML

PlantUML es un proyecto Open Source (Código Abierto) que permite escribir rápidamente:

Diagramas de Secuencia
Diagramas de Casos de uso
Diagramas de Clases
Diagramas de Objetos
Diagramas de Actividades (here is the legacy syntax)
Diagramas de Componentes
Diagramas de Despliegue
Diagramas de Estados

3. INSTALAR EL PLUGIN PLANTUML CON NETBEANS

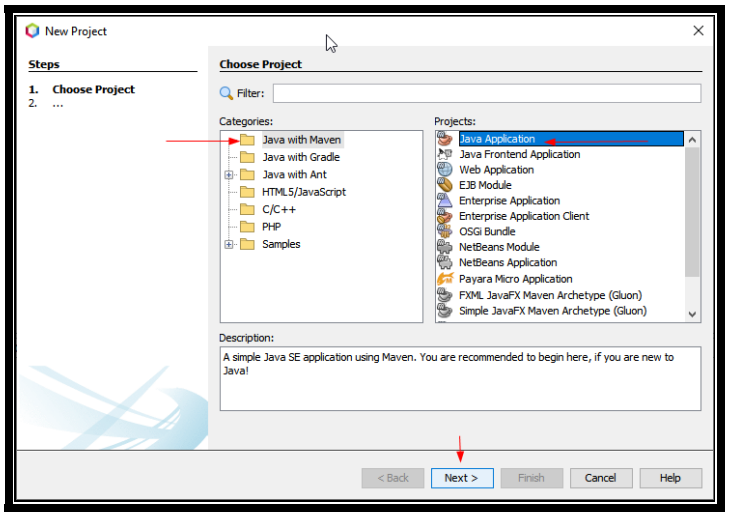
- Ir al menú Tools > Plugins

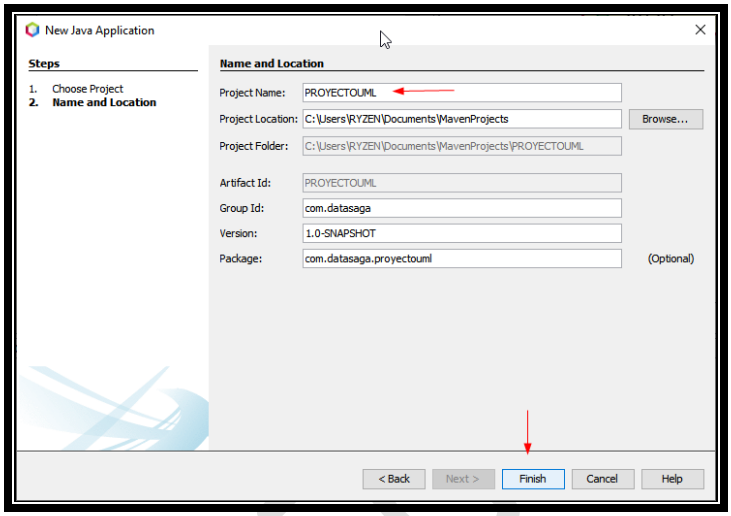


4. CREAR UN PROYECTO JAVA PARA NUESTROS DIAGRAMAS UML

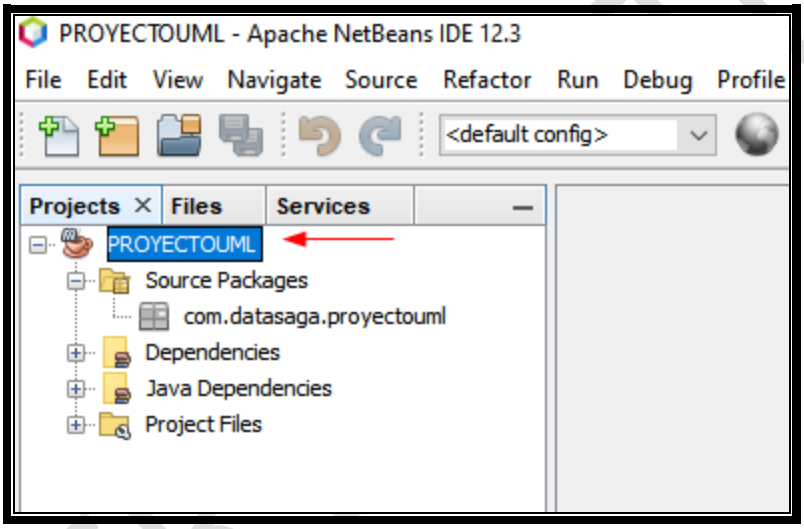
- Ir File > New Project…

- Luego lo que indica la siguiente imagen



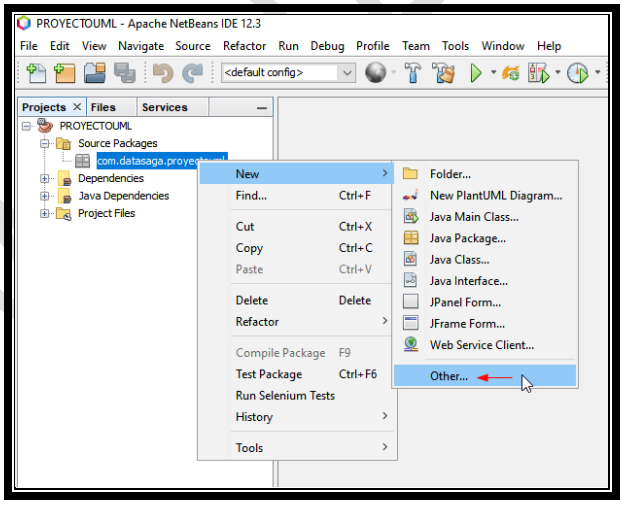


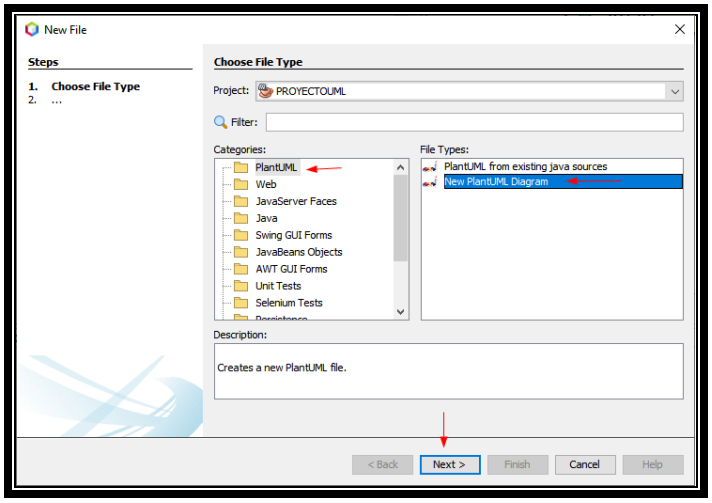
- En el explorador de proyectos de netbean se crea la siguiente estrucutra de carpetas, como indiga la siguiente imagen

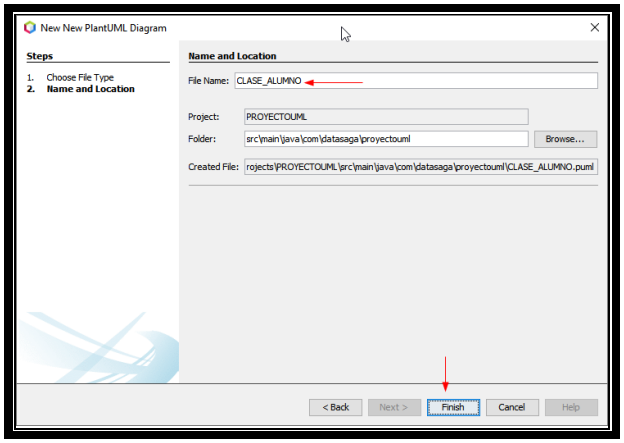


5. CREAR NUESTRO PRIMER DIAGRAMA DE CLASE EN PLANTUML

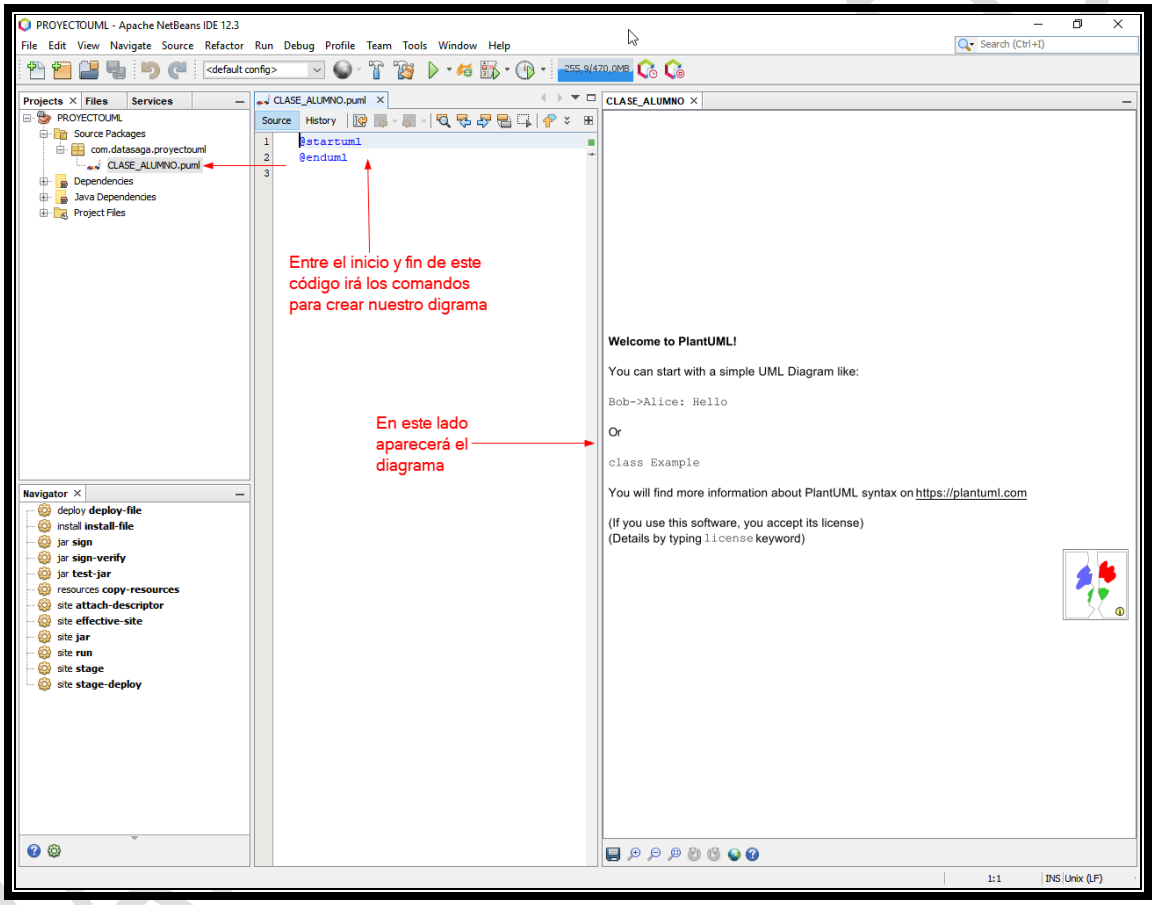
- Crear un diagrama nuevo, a partir del proyecto java que creamos anteriormente



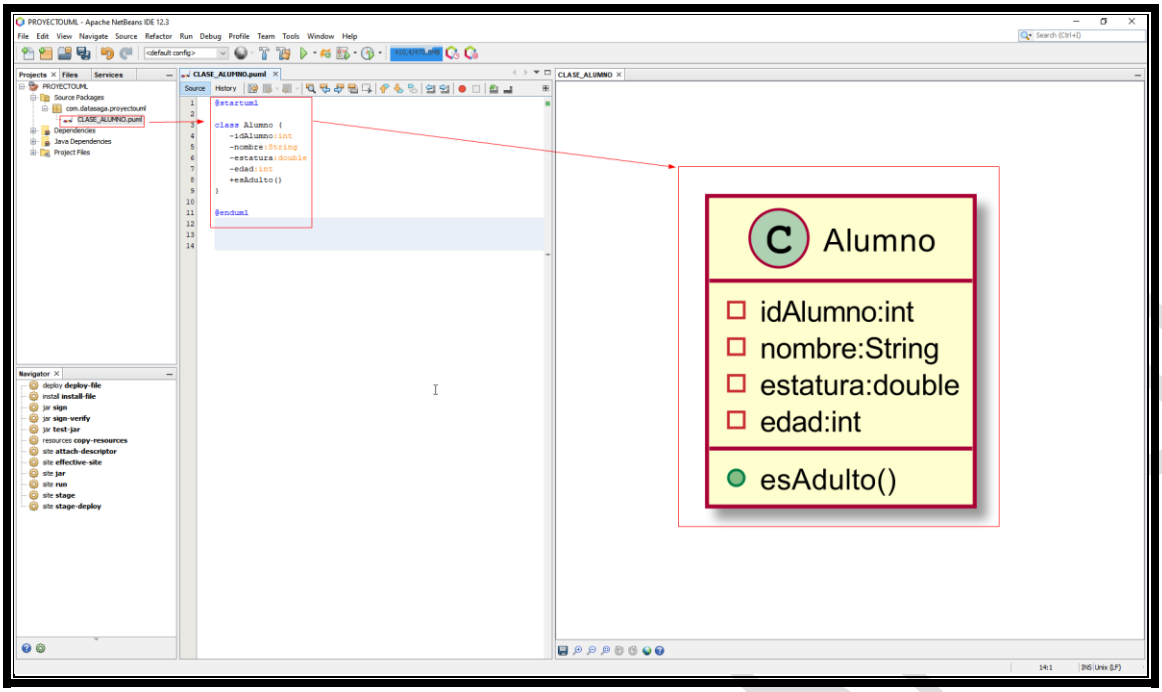




- Del paso anterior, da como resultado la siguiente imagen



- Finalmente nuestro primer diagrama de la clase Alumno



6. CODIGO BASICO DE PLANTUML PARA GENERAR DIAGRAMAS UML

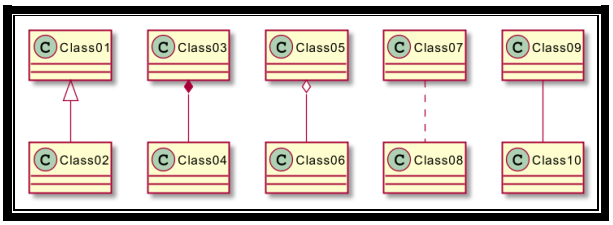
5.1. RELACION ENTRE CLASES

Las relaciones entre clases se definen usando los siguientes símbolos:

Type	Symbol	Drawing
Extension	<|--	
Composition	*--	
Aggregation	o--	

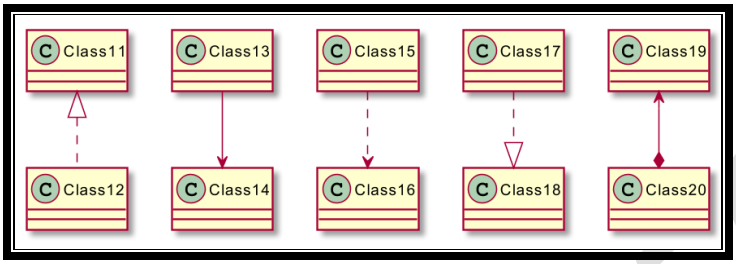
EJEMPLO 1: DISTINTAS FORMAS DE RELACIONAR CLASES

```
@startuml

Class01 <|-- Class02
Class03 *-- Class04
Class05 o-- Class06
Class07 .. Class08
Class09 -- Class10

@enduml
```



EJEMPLO 2: DISTINTAS FORMAS DE RELACIONAR CLASES

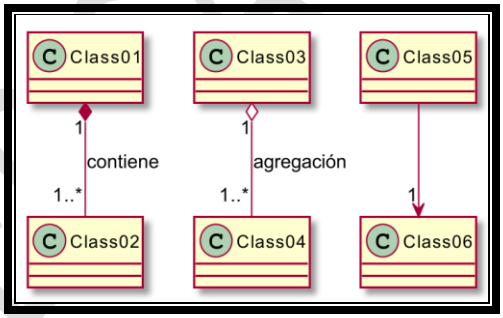
```
@startuml

Class11 <|.. Class12
Class13 --> Class14
Class15 ..> Class16
Class17 ..|> Class18
Class19 <--* Class20

@enduml
```



5.2. ETIQUETAS EN LAS RELACIONES

- Es posible añadir etiquetas en las relaciones, usando :, seguido del texto de la etiqueta.
- Para la cardinalidad, puede usar comillas dobles "" en cada lado de la relación.

```
@startuml

Class01 "1" *-- "1..*" Class02 : contiene

Class03 "1" o-- "1..*" Class04 : agregación

Class05 --> "1" Class06

@enduml
```



5.3. DEFINIENDO LA VISIBILIDAD

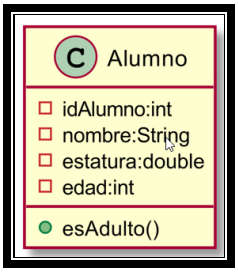
Cuando defines propiedades o métodos, puedes usar caracteres para establecer la visibilidad que les correspondan:

Character	Icon for field	Icon for method	Visibility
-	□	■	private
#	◇	◆	protected
~	△	▲	package private
+	○	●	public

7. EJERCICIOS

7.1 Implementar en PHP el siguiente diagrama de clase Alumno, asi como una clase Data que se encargará de devolver una lista de objetos Alumnos que luego será recorrido por el principal para mostratar todos los alumnos.

```
@startuml

class Alumno {
    -idAlumno:int
    -nombre:String
    -estatura:double
    -edad:int
    +esAdulto()
}

@enduml
```



SOLUCION

Alumno.php

```php
<?php
class Alumno {

    const CENTRO = "CampusFP";

    private $idAlumno;
    private $nombre;
    private $estatura;
    private $edad;

    public function __construct($idAlumno, $nombre, $estatura, $edad) {
        $this->idAlumno = $idAlumno;
        $this->nombre = $nombre;
        $this->estatura = $estatura;
        $this->edad = $edad;
    }

    public function getIdAlumno() {
        return $this->idAlumno;
    }

    public function getNombre() {
        return $this->nombre;
    }

    public function getEstatura() {
        return $this->estatura;
    }

    public function getEdad() {
        return $this->edad;
    }

    public function setIdAlumno($idAlumno) {
        $this->idAlumno = $idAlumno;
    }

    public function setNombre($nombre) {
        $this->nombre = $nombre;
    }

    public function setEstatura($estatura) {
        $this->estatura = $estatura;
    }

    public function setEdad($edad) {
        $this->edad = $edad;
    }

    public function __toString() {
        return $this->idAlumno.",".$this->nombre.",".$this->estatura.",".$this->edad;
    }

    public function esAdulto() {
        if ($this->edad >= 18) {
            return "Adulto";
        } else {
            return "Menor";
        }
    }
}
?>
```

Data.php

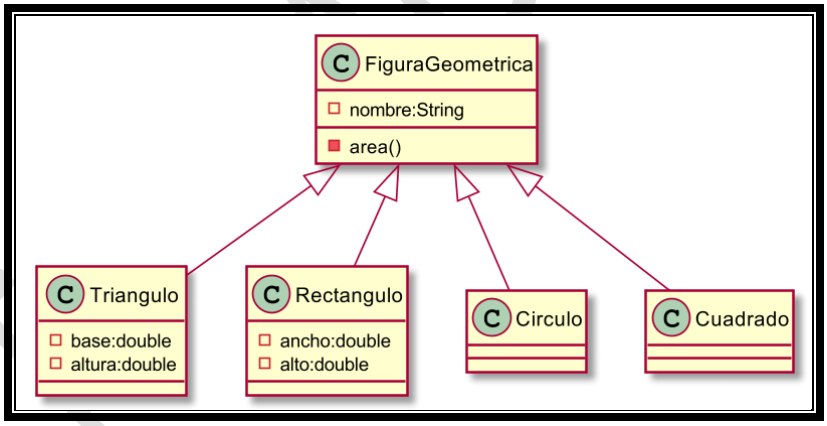
```php
<?php

require_once "Alumno.php";

class Data {

    public static function obtenerAlumnos() {
        $alumnos_al = array();

        $a1 = new Alumno(1, "Luis", 1.72, 17);
        $a2 = new Alumno(2, "Carlos", 1.73, 19);
        $a3 = new Alumno(3, "Miguel", 1.74, 18);
        $a4 = new Alumno(4, "Maria", 1.75, 16);

        array_push($alumnos_al, $a1);
        array_push($alumnos_al, $a2);
        array_push($alumnos_al, $a3);
        array_push($alumnos_al, $a4);

        return $alumnos_al;
    }

}
?>
```

Principal.php

```php
<?php

require_once "Alumno.php";
require_once "Data.php";

$alumnos_al = Data::obtenerAlumnos();

echo "Centro: " . Alumno::CENTRO . "</br></br>";

echo "FORMA 1: USO DE LOS METODOS GET Y SET."."</br></br>";
foreach ($alumnos_al as $a) {
    echo "Id: " . $a->getIdAlumno() . "</br>";
    echo "Nombre: " . $a->getNombre() . "</br>";
    echo "Estatura: " . $a->getEstatura() . "</br>";
    echo "Edad: " . $a->getEdad() . "</br>";
    echo "Soy ".$a->esAdulto()."</br></br>";
}

echo "FORMA 2: USO DEL METODO TOSTRING"."</br></br>";
foreach ($alumnos_al as $a) {
    echo $a.","."Soy ".$a->esAdulto()."</br>";
}
?>
```

6.2 Implementar en php el siguiente diagrama de la clase de herencia.

```
@startuml

class FiguraGeometrica {
   -nombre:String
   -area()
}

class Triangulo {
   -base:double
   -altura:double
}

class Rectangulo {
   -ancho:double
   -alto:double
}

FiguraGeometrica <|-- Triangulo
FiguraGeometrica <|-- Rectangulo
FiguraGeometrica <|-- Circulo
FiguraGeometrica <|-- Cuadrado

@enduml
```



En la solución que se ofrece sólo esta implementado la clase Triangulo y Rectangulo, hay que implementar la Clase Circulo y Cuadrado.

En este ejemplo se muestra el método **area** que esta en el padre, que será implementada por cada uno de los hijos que son los que heredan el método **area**.

En Principal_1.php se simula los datos obtenidos de una clase Data que luego se muestran en pantalla.

En Principal_2.php se muestra el resultado del conteo de objetos que se obtuvieron de la clase Data.

SOLUCION

FiguraGeometrica.php

```php
<?php

abstract class FiguraGeometrica {

    private $nombre;

    public function __construct($nombre) {
        $this->nombre = $nombre;
    }

    public function getNombre() {
        return $this->nombre;
    }

    public function setNombre($nombre) {
        $this->nombre = $nombre;
    }

    public function __toString() {
        return "FiguraGeometrica";
    }

    public function area() {
        echo "Método que será definido por cada uno de sus hijos";
    }
}

?>
```

Rectangulo.php

```php
<?php

require_once "FiguraGeometrica.php";

class Rectangulo extends FiguraGeometrica {

    private $ancho;
    private $alto;

    public function __construct($ancho, $alto, $nombre) {
        parent::__construct($nombre);
        $this->ancho = $ancho;
        $this->alto = $alto;
    }

    public function getAncho() {
        return $this->ancho;
    }

    public function getAlto() {
        return $this->alto;
    }

    public function setAncho($ancho) {
        $this->ancho = $ancho;
    }

    public function setAlto($alto) {
        $this->alto = $alto;
    }

    public function area() {
        return $this->ancho * $this->alto;
    }

    public function __toString(): string {
        return parent::__toString() . " " . parent::getNombre() . " Area : " . $this->area();
        //return $this->__toString() . " Area " . $this->getNombre() . " : " . $this->area();
    }

}
?>
```

Triangulo.php

```php
<?php

require_once "FiguraGeometrica.php";

class Triangulo extends FiguraGeometrica {

    private $base;
    private $altura;

    public function __construct($base, $altura, $nombre) {
        parent::__construct($nombre);
        $this->base = $base;
        $this->altura = $altura;
    }

    public function getBase() {
        return $this->base;
    }

    public function getAltura() {
        return $this->altura;
    }

    public function setBase($base) {
        $this->base = $base;
    }

    public function setAltura($altura) {
        $this->altura = $altura;
    }

    public function area() {
        return $this->base * $this->altura / 2;
    }

    public function __toString() {
        return parent::__toString() . " " . parent::getNombre() . " Area : " . $this->area();
        //return $this->__toString() . " Area " . $this->getNombre() . " : " . $this->area();
    }

}
?>
```

Data.php

```php
<?php

require_once "Rectangulo.php";
require_once "Triangulo.php";

class Data {

    public static function obtenerFigurasGeometricas() { //Simular una base de datos
        $figurasgeometricas_al = array();

        $fg1 = new Rectangulo(2, 3, "Rectangulo");
        $fg2 = new Rectangulo(4, 5, "Rectangulo");
        $fg3 = new Triangulo(6, 7, "Triangulo");
        $fg4 = new Triangulo(8, 9, "Triangulo");

        array_push($figurasgeometricas_al, $fg1);
        array_push($figurasgeometricas_al, $fg2);
        array_push($figurasgeometricas_al, $fg3);
        array_push($figurasgeometricas_al, $fg4);

        return $figurasgeometricas_al;
    }

}
?>
```

Principal_1.php

```php
<?php

require_once "Data.php";

$figurasgeometricas_al = Data::obtenerFigurasGeometricas();

foreach ($figurasgeometricas_al as $fg) {
    echo $fg . "</br>";
}
?>
```

Principal_2.php

```php
<?php

require_once "Data.php";

$figurasgeometricas_al = Data::obtenerFigurasGeometricas();
$ct = 0;
$cr = 0;
foreach ($figurasgeometricas_al as $fg) {
    if ($fg instanceof Triangulo) {
        $ct++;
    }
    if ($fg instanceof Rectangulo) {
        $cr++;
    }
}
echo "Cantidad de objetos Triangulo: " . $ct . "</br>";
echo "Cantidad de objetos Rectangulo: " . $cr . "</br>";
?>
```

6.3

En una empresa se pretende desarrollar un sistema que permita determinar cuánto se le debe pagar a sus trabajadores. Se sabe que en esta empresa existen 4 tipos de trabajadores:

Al 1er Tipo de trabajador se le paga un salario semanal fijo sin tomar encuentra el número de horas que ha trabajado.
El 2do Tipo de trabajador tiene un salario base más un % sobre las ventas que ha realizado.
El 3er Tipo de trabajador se le paga por el número de artículos que produce.
El 4to Tipo de trabajador se le paga por horas y además puede tener un tiempo extra.

Se pide poder imprimir el nombre y apellido de los trabajadores así como el pago que le corresponde.

Realizar el diagrama con PlantUML.