

PROGRAMACIÓN II



SEMANA 11 - 1

INTERFACES Y POLIMORFISMO

Eric Gustavo Coronel Castillo
ecoronel@uch.edu.pe
gcoronelc.blogspot.com

Interfaces y Polimorfismo

Un interfaz es una lista de acciones que puede llevar a cabo un determinado objeto. La diferencia con una clase, es que en una interfaz solo tenemos el prototipo del método, en una clase normalmente se tiene también la implementación.

Las interfaces son un nivel superior de abstracción, que luego deben ser implementadas en clases.

Por su parte el polimorfismo se refiere a la posibilidad de definir clases diferentes que tienen métodos definidos de forma idéntica, pero que se comportan de manera distinta. Así como la herencia está relacionada con las clases y su jerarquía, el polimorfismo se relaciona con los métodos.

Índice

1	¿QUÉ ES UNA INTERFAZ?	4
2	USO DE INTERFACES	8
2.1	DEFINICIÓN DE UNA INTERFAZ	8
2.2	IMPLEMENTACIÓN DE UNA INTERFAZ	9
2.3	CONSTANTES E INTERFACES	11
2.4	LAS INTERFACES SE PUEDEN EXTENDER	11
3	POLIMORFISMO	14
3.1	DEFINICIÓN	14
3.2	IMPLEMENTACIÓN MEDIANTE HERENCIA	14
3.3	IMPLEMENTACIÓN MEDIANTE INTERFACES	14
4	PROYECTO: PERÚ TRAINING	16
4.1	REQUERIMIENTO DE SOFTWARE	16
4.2	ABSTRACCIÓN	17
4.3	ESTRUCTURA DEL PROYECTO EN NETBEANS	19
4.4	CODIFICACIÓN	20
4.5	EJECUCIÓN DEL PROYECTO	28

1 ¿Qué es una Interfaz?

Un interfaz es la definición de un conjunto de operaciones que caracteriza el comportamiento de un objeto, y que deben ser desarrolladas en las clases que implementan la interfaz, esto quiere decir que las interfaces solo contienen los prototipos de los métodos, más no la implementación.

Una clase que implementa una interfaz debe implementar todos los métodos declarados por la interfaz. Muchas clases pueden implementar la misma interfaz, estas clases no necesitan compartir la misma clase padre. También, una clase puede implementar más de una interfaz.

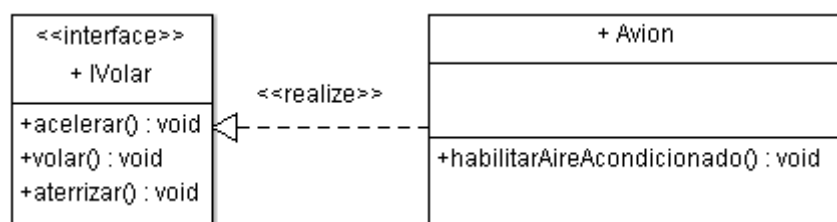


Figura 1 Diagrama de una interfaz y su implementación.

En la Figura 1 tenemos una interfaz **IVolar** que es implementada por la clase **Avion**, note que la clase puede implementar métodos propios, en este caso el método **habilitarAireAcondicionado**; esta interfaz puede ser implementada por muchas clases.

Para declarar una interfaz se utiliza la palabra clave `interface` tal como se ilustra a continuación:

```
<?php
interface IVolar {

    function acelerar();
    function volar();
    function aterrizar();

}
?>
```

Los métodos definidos en una interfaz son métodos abstractos.

Para implementar una interfaz se debe utilizar la palabra clave `implements`, tal como se ilustra a continuación:

```
<?php
require_once '../interfaces/IVolar.php';

class Avion implements IVolar {

    public function acelerar() {

        // Implementación

    }

    public function volar() {

        // Implementación

    }

    public function aterrizar() {

        // Implementación

    }

    public function habilitarAireAcondicionado() {

        // Implementación

    }

}
?>
```

Podemos tener múltiples clases que implementan la interfaz `IVolar`. Un avión puede volar, un ave también puede volar, superman puede volar, etc.

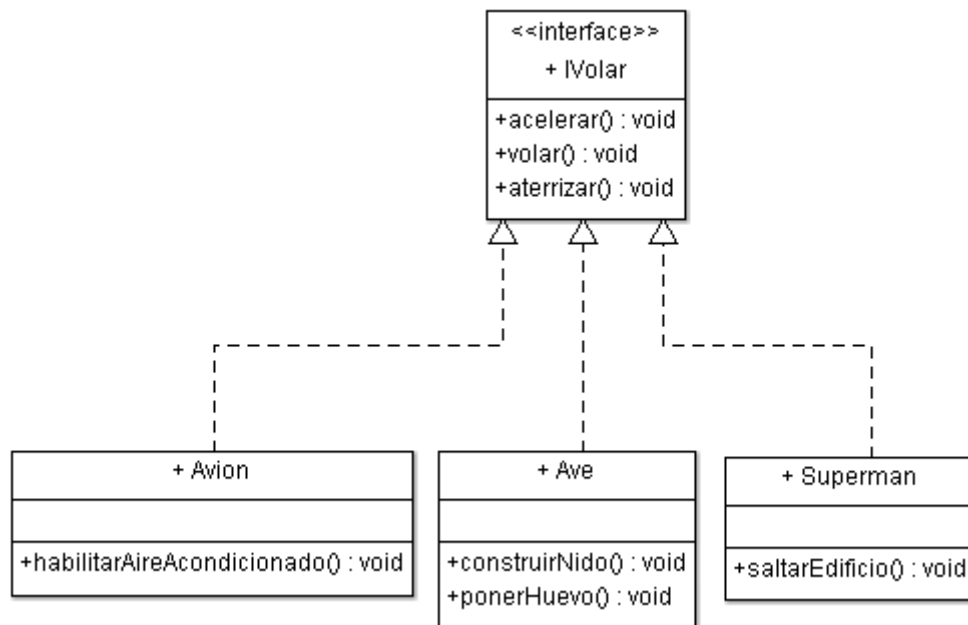


Figura 2 Múltiples implementaciones de la interfaz IVolar.

Un avión es un vehículo, y puede volar. Un ave es un animal, y puede volar. Estos ejemplos muestran que una clase puede heredar de una superclase pero también puede implementar algunas otras interfaces.

Esto parece herencia múltiple, pero no lo es en absoluto. El peligro de la herencia múltiple es que una clase podría heredar dos distintas implementaciones de un mismo método. Esto no es posible con los interfaces, porque una declaración de un método de la interfaz no provee ninguna implementación.

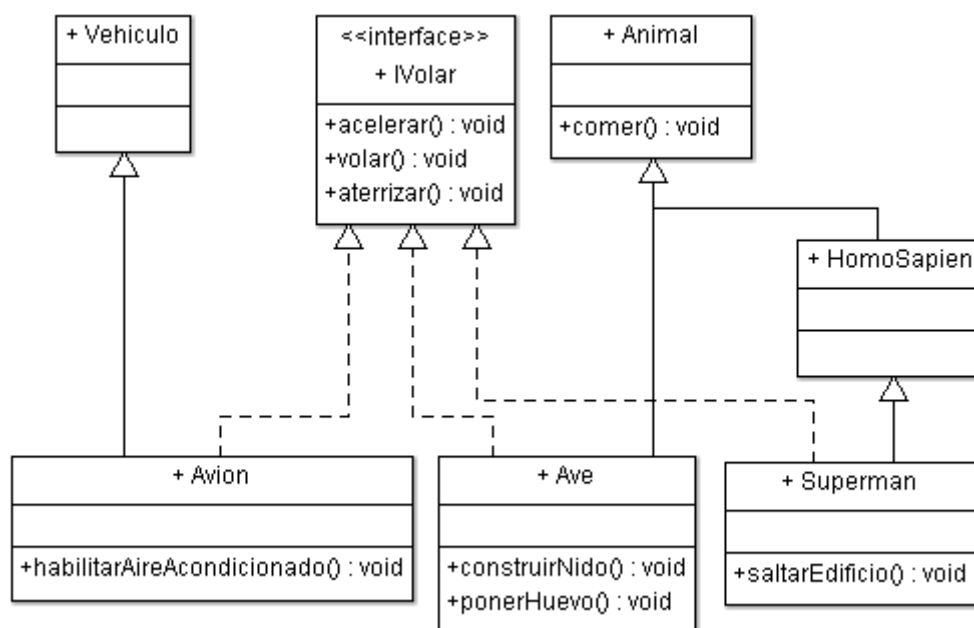


Figura 3 Una mixtura de herencia e implementación de interfaces.

A continuación tenemos la implementación de la clase Ave:

```
<?php

require_once '../interfaces/IVolar.php';

class Ave extends Animal implements IVolar {

    public function acelerar() {
        // Implementación
    }

    public function volar() {
        // Implementación
    }

    public function aterrizar() {
        // Implementación
    }

    public function habilitarAireAcondicionado() {
        // Implementación
    }

    public function constriutNido(){
        // Implementación
    }

    public function ponerHuevo(){
        // Implementación
    }

}
?>
```

La cláusula **extends** debe estar antes de la cláusula **implements**. La clase **Ave** puede implementar sus propios métodos.

2 Uso de Interfaces

2.1 Definición de una Interfaz

Sintaxis:

```
interface nombre_interfaz {  
  
    function nombre_método1( lista_de_parámetros );  
  
    function nombre_método2( lista_de_parámetros );  
  
    . . .  
  
}
```

El acceso a una interfaz es público, lo que quiere decir que puede utilizarse desde cualquier parte.

Los métodos que se declaran no tienen cuerpo y terminan con un punto y coma, son esencialmente métodos abstractos.

A continuación tenemos un ejemplo ilustrativo:

```
<?php  
  
interface IDemo1 {  
    function suma ( $a, $b );  
}  
  
?>
```

A continuación tenemos otro ejemplo:

```
<?php  
  
interface IDemo2 {  
    function producto($a, $b);  
}  
  
?>
```

2.2 Implementación de una Interfaz

Una vez definida un interfaz, una o más clases pueden implementarla. Para implementar una interfaz, se debe incluir la cláusula `implements` en la definición de una clase, y luego crear los métodos definidos por la interfaz.

Sintaxis:

```
class nombre_clase [extends superclase] [implements interfaz1, ...] {  
  
    // Cuerpo de la clase  
  
}
```

Si una clase implementa más de una interfaz, las interfaces se separan por comas. Los métodos que se implementan de una interfaz deben ser declarados como `public`. Además, el formato del método implementado debe coincidir exactamente con el formato especificado en la definición de la interfaz.

A continuación un ejemplo ilustrativo que implementa las interfaces IDemo1 e IDemo2:

```
<?php  
  
require_once 'IDemo1.php';  
require_once 'IDemo2.php';  
  
class CDemo implements IDemo1, IDemo2 {  
  
    public function suma ( $a, $b ) {  
        return ($a+$b);  
    }  
  
    public function producto($a, $b) {  
        return ($a*$b);  
    }  
  
}  
?>
```


Además, las clases que implementan interfaces pueden tener sus propias definiciones. Por ejemplo la siguiente versión de **CDemo** añade el método **potencia()**:

```
<?php

require_once 'IDemo1.php';
require_once 'IDemo2.php';

class CDemo implements IDemo1, IDemo2 {

    public function suma ( $a, $b ) {
        return ($a+$b);
    }

    public function producto($a, $b) {
        return ($a*$b);
    }

    public function potencia($b, $e) {
        $p = 1;
        for ($k = 1; $k <= $e; $k++){
            $p *= $b;
        }
        return $p;
    }

}

?>
```

A continuación el programa **PDemo**, donde se ilustra el uso de la clase **CDemo**:

```
<?php

require_once 'CDemo.php';

$obj = new CDemo();
$a = 5;
$b = 3;

echo("$a + $b = " . $obj->suma($a, $b) . "<br>");
echo("$a * $b = " . $obj->producto($a, $b) . "<br>");
echo("$a ^ $b = " . $obj->potencia($a, $b) . "<br>");

?>
```

La ejecución del programa nos da el siguiente resultado:

$$\begin{array}{l} 5 + 3 = 8 \\ 5 * 3 = 15 \\ 5 ^ 3 = 125 \end{array}$$

2.3 Constantes e Interfaces

Se puede utilizar interfaces para declarar constantes y luego acceder a ellas desde cualquier parte del sistema. Aquí tenemos un ejemplo:

```
<?php
interface IParam {

    const COMPANY = "PeruDev";
    const SITE = "www.perudev.net";
    const MANAGER = "Gustavo Coronel";

}
?>
```

A continuación tenemos un ejemplo de cómo utilizar la interfaz:

```
<?php

header('Content-Type: text/html; charset=utf-8');

require_once 'IParam.php';

echo("Compañía: " . IParam::COMPANY . "<br>");
echo("Sitio Web: " . IParam::SITE . "<br>");
echo("Gerente: " . IParam::MANAGER . "<br>");

?>
```

El resultado es el siguiente:

```
Compañía: PeruDev
Sitio Web: www.perudev.net
Gerente: Gustavo Coronel
```

2.4 Las Interfaces se Pueden Extender

Una interfaz puede heredar otra mediante la palabra clave `extends`. La sintaxis es la misma que para la herencia de clases.

Cuando una clase implementa una interfaz que hereda otra interfaz, debe implementar todos los métodos dentro de la cadena de herencia de la interfaz.

Por ejemplo, tenemos la interfaz **IA**:

```
<?php
interface IA {

    function metodo1();
    function metodo2();

}
?>
```

Podemos tener una interfaz **IB** que extienda la interfaz **IA**:

```
<?php
require_once 'IA.php';

interface IB extends IA {

    function metodo3();

}
?>
```

La clase que implemente la interfaz **IB** debe implementar los tres métodos, tal como se ilustra a continuación:

```
<?php
require_once 'IA.php';

class CEstudio implements IA {

    public function metodo1(){
        echo("Implementación del método 1<br>");
    }

    public function metodo2(){
        echo("Implementación del método 2<br>");
    }

    public function metodo3(){
        echo( "Implementación del método 3<br>" );
    }

}
```

```
}  
  
}  
?>
```

Note usted que los métodos implementados en la clase **CEstudio** deben ser públicos.

El siguiente programa muestra el uso de la clase **CEstudio**:

```
<?php  
header('Content-Type: text/html; charset=utf-8');  
  
require_once 'CEstudio.php';  
$obj = new CEstudio();  
  
$obj->metodo1();  
$obj->metodo2();  
$obj->metodo3();  
  
?>
```

Su ejecución nos da el siguiente resultado:

```
Implementación del método 1  
Implementación del método 2  
Implementación del método 3
```

Si a modo de prueba eliminamos el método **metodo2()** de la clase **CEstudio** obtenemos el siguiente mensaje de error::

```
Fatal error: Class CEstudio contains 1 abstract method and must therefore  
be declared abstract or implement the remaining methods (IA::metodo2) in  
C:\PHP100\cap13\PjAprendizaje\CEstudio.php on line 19
```

Como conclusión, cualquier clase que implemente una interfaz, debe implementar todos los métodos que defina la interfaz, incluyendo todos los métodos heredados de otras interfaces.

3 Polimorfismo

3.1 Definición

Una misma operación puede implementarse de diferentes formas en clases distintas. Quiere decir que la semántica es común pero la implementación varía en cada clase.

3.2 Implementación Mediante Herencia

El polimorfismo se puede implementar mediante la herencia, como se muestra en la Figura 4, las clases **Circulo** y **Triangulo** implementan el método **calcularArea()** que se define en la clase abstracta **Figura**.

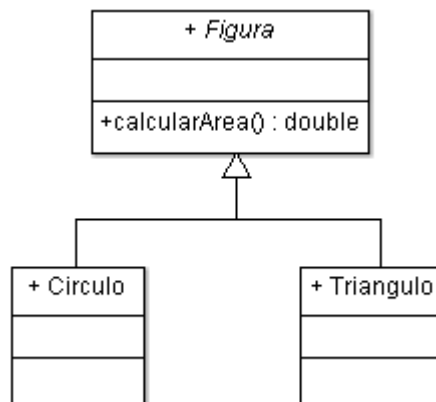


Figura 4 Polimorfismo mediante herencia.

3.3 Implementación Mediante Interfaces

También aplicamos polimorfismo cuando varias clases implementan la misma interfaz, como se muestra en la Figura 5, las clases **Empleado**, **Cliente** y **Producto** implementan la misma interfaz **Mantenimiento**.

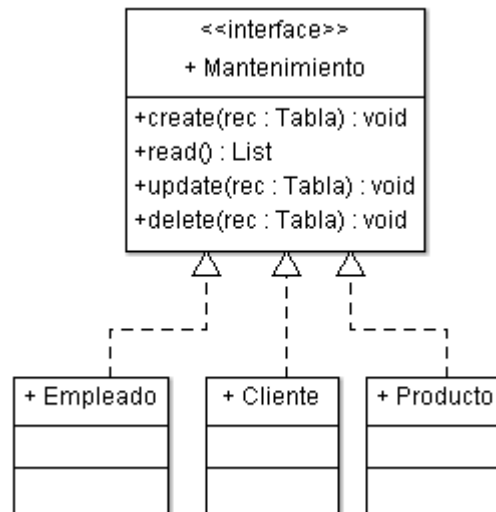


Figura 5 Polimorfismo mediante la implementación de una interfaz.

4 Proyecto: Perú Training

4.1 Requerimiento de Software

La institución educativa PerúTraining cuenta con dos tipos de trabajadores: Empleados y Docentes.

Los empleados cuentan con un sueldo fijo y depende del cargo que ocupa, según la siguiente tabla:

Cargo	Sueldo
Coordinador	3,500.00
Asistente	2,500.00
Secretaria	1,800.00

El sueldo del docente está en función a las horas que dicta, el pago por hora es de 50 Nuevos Soles.

El departamento de recursos humanos necesita una aplicación para calcular la bonificación que se debe pagar a cada trabajador según el siguiente cuadro:

Trabajador	Bonificación
Empleado	100% del Sueldo
Docente	60% del Sueldo

Y elaborar la planilla correspondiente.

4.2 Abstracción

El diseño está conformado por una clase abstracta de nombre **Trabajador** que representa cualquier tipo de trabajador, y de donde heredan las clases concretas **Empleado** y **Docente**, como se puede apreciar en la Figura 6.

Los métodos que tienen que ver con la planilla se definen en la interfaz **ITrabajador**, que es implementada por las clases **Empleado** y **Docente**, tal como se puede verificar en el diagrama de la Figura 6. En la misma interfaz **ITrabajador** se definen contantes que tienen que ver con la solución.

La clase **Planilla** representa una colección de todos los trabajadores de la empresa, incluso define un método para obtener el total de la planilla, una instancia de esta clase debemos mantener en sesión.

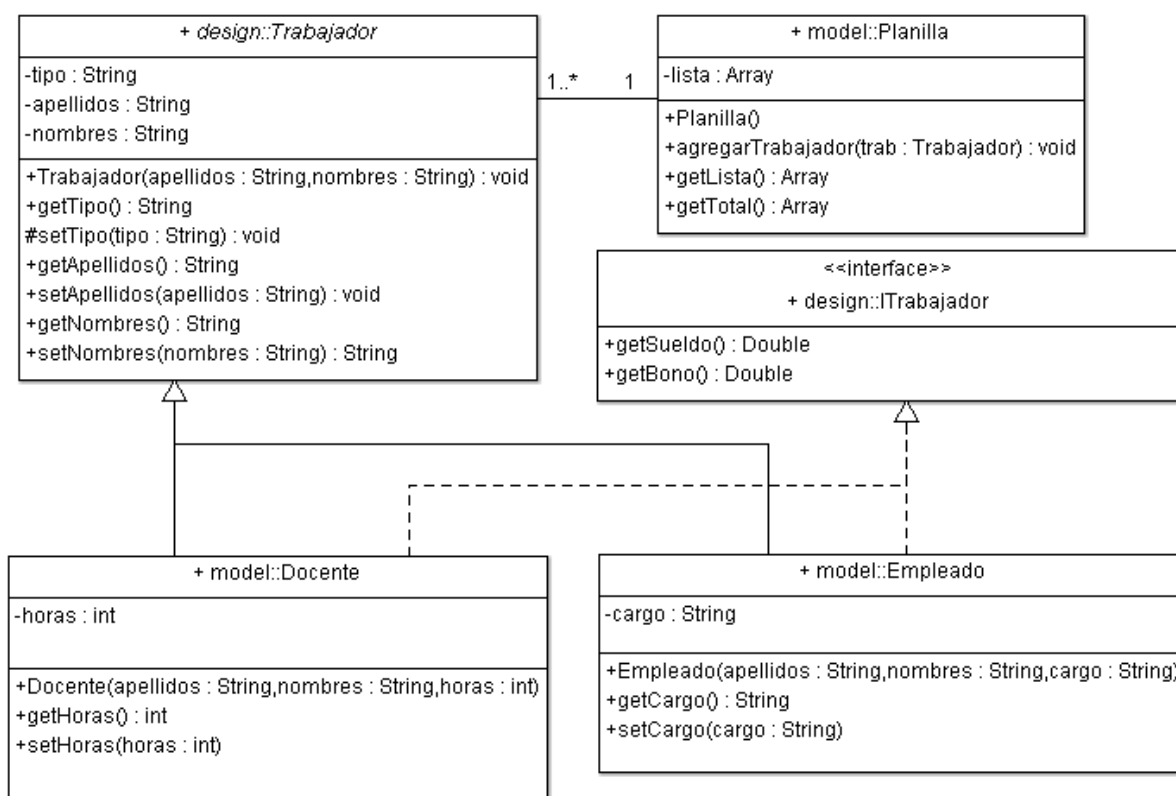


Figura 6 Diagrama de clases de la solución.

A continuación tenemos la interfaz de usuario (IU) para registrar un nuevo trabajador.



The screenshot shows the 'PerúTraining' web application interface. At the top, there is a navigation bar with three links: 'Registro', 'Planilla', and 'Salir'. Below this, the main heading is 'Registro de Trabajador'. The form contains the following fields: 'Tipo' (a dropdown menu with 'Docente' selected), 'Apellidos' (a text input field), 'Nombres' (a text input field), and 'Horas' (a text input field). At the bottom left of the form is a blue button labeled 'Grabar'.

Figura 7 IU para registrar un Docente.

El campo **Tipo** define el tipo de trabajador, si seleccionamos **Empleado** la IU se adecua al nuevo tipo de trabajador.



The screenshot shows the 'PerúTraining' web application interface. At the top, there is a navigation bar with three links: 'Registro', 'Planilla', and 'Salir'. Below this, the main heading is 'Registro de Trabajador'. The form contains the following fields: 'Tipo' (a dropdown menu with 'Empleado' selected), 'Apellidos' (a text input field), 'Nombres' (a text input field), and 'Cargo' (a dropdown menu with 'Coordinador' selected). At the bottom left of the form is a blue button labeled 'Grabar'.

Figura 8 IU para registrar un empleado.

La interfaz de la planilla es la siguiente:

PerúTraining

[Registro](#)
[Planilla](#)
[Salir](#)

Planilla General

Tipo	Apellidos	Nombres	Sueldo	Bono	Total
Empleado	Romero	Carlos	3500	3500	7000
Empleado	Ruiz	Mariela	2500	2500	5000
Empleado	Beltran	Claudia	1800	1800	3600
Docente	Coronel	Gustavo	5000	3000	8000
Docente	Valencia	Pedro	5000	3000	8000
Docente	Matsukawa	Sergio	5000	3000	8000
Docente	Marcelo	Ricardo	6000	3600	9600

Total: 49200

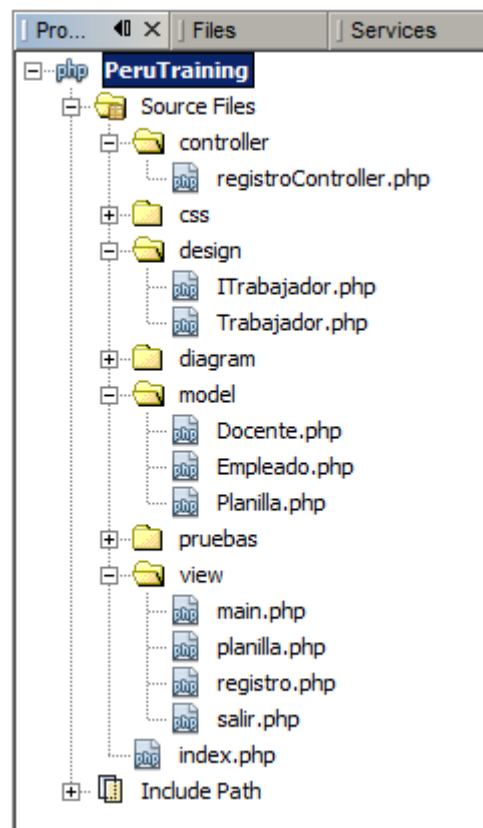
Figura 9 IU para la planilla.

4.3 Estructura del Proyecto en NetBeans

Como se puede apreciar en el gráfico de la derecha, tenemos cinco carpetas, en la carpeta **css** tenemos la hoja de estilos, en la carpeta **design** la interfaz **ITrabajador** y la clase abstracta **Trabajador**, en la carpeta **model** las clases **Docente**, **Empleado** y **Planilla**, en el carpeta **diagram** encontramos el diagrama en UML realizado con ArgoUML, en la carpeta **controller** encontramos el programa **registroController.php** que se encarga de llevar el control del registro de los trabajadores en una instancia de la clase **Planilla** que se guarda en sesión y en la carpeta **view** tenemos los programas que corresponden a las interfaces de usuario.

La carpeta **pruebas** no forma parte de la solución, aquí personalmente guardo programas de prueba, no debe ir a producción al igual que la carpeta **diagram**.

En la raíz del proyecto tenemos el programa **index.php**, éste es el que inicia la ejecución del proyecto, y en este caso es hacer un redireccionamiento al programa **main.php**.



4.4 Codificación

Archivo: PeruTraining\design\ITrabajador.php

```
<?php
interface ITrabajador {

    const TIPO_EMPLEADO = "Empleado";
    const TIPO_DOCENTE = "Docente";

    const EMPL_COORDINADOR = "Coordinador";
    const EMPL_ASISTENTE = "Asistente";
    const EMPL_SECRETARIA = "Secretaria";

    const SUELDO_COORDINADOR = 3500.0;
    const SUELDO_ASISTENTE = 2500.0;
    const SUELDO_SECRETARIA = 1800.0;
    const DOCENTE_PAGO_HORA = 50.0;

    const BONO_EMPLEADO = 1.00; // 100 %
    const BONO_DOCENTE = 0.60; // 60%

    function getSueldo();
    function getBono();

}
?>
```

Archivo: PeruTraining\design\Trabajador.php

```
<?php
abstract class Trabajador {

    private $tipo;
    private $apellidos;
    private $nombres;

    function __construct($apellidos = null, $nombres = null) {
        $this->setApellidos($apellidos);
        $this->setNombres($nombres);
    }

    public function getTipo() {
        return $this->tipo;
    }

    protected function setTipo($tipo) {
```

```
$this->tipo = $tipo;
}

public function getApellidos() {
    return $this->apellidos;
}

public function setApellidos($apellidos) {
    $this->apellidos = $apellidos;
}

public function getNombres() {
    return $this->nombres;
}

public function setNombres($nombres) {
    $this->nombres = $nombres;
}
}
?>
```

Archivo: PeruTraining\model\Docente.php

```
<?php

require_once '../design/ITrabajador.php';
require_once '../design/Trabajador.php';

class Docente extends Trabajador implements ITrabajador {

    private $horas;

    function __construct($apellidos = null, $nombres = null, $horas = 0) {
        parent::__construct($apellidos, $nombres);
        $this->setTipo(ITrabajador::TIPO_DOCENTE);
        $this->setHoras($horas);
    }

    public function getHoras() {
        return $this->horas;
    }

    public function setHoras($horas) {
        $this->horas = $horas;
    }

    public function getSueldo(){
        $sueldo = $this->getHoras() * ITrabajador::DOCENTE_PAGO_HORA;
```

```
        return $sueldo;
    }

    public function getBono(){
        $bono = $this->getSueldo() * ITrabajador::BONO_DOCENTE;
        return $bono;
    }
}
?>
```

Archivo: PeruTraining\model\Empleado.php

```
<?php

require_once '../design/ITrabajador.php';
require_once '../design/Trabajador.php';

class Empleado extends Trabajador implements ITrabajador {

    private $cargo;

    function __construct($apellidos = null, $nombres = null, $cargo = null) {
        parent::__construct($apellidos, $nombres);
        $this->setTipo(ITrabajador::TIPO_EMPLEADO);
        $this->setCargo($cargo);
    }

    public function getCargo() {
        return $this->cargo;
    }

    public function setCargo($cargo) {
        $this->cargo = $cargo;
    }

    public function getSueldo(){
        $sueldo = 0.0;
        $cargo = $this->getCargo();
        if( $cargo == ITrabajador::EMPL_COORDINADOR ){
            $sueldo = ITrabajador::SUELDO_COORDINADOR;
        } elseif( $cargo == ITrabajador::EMPL_ASISTENTE ){
            $sueldo = ITrabajador::SUELDO_ASISTENTE;
        } elseif( $cargo == ITrabajador::EMPL_SECRETARIA ){
            $sueldo = ITrabajador::SUELDO_SECRETARIA;
        }
        return $sueldo;
    }
}
```

```
public function getBono(){
    $bono = $this->getSueldo() * ITrabajador::BONO_EMPLEADO;
    return $bono;
}

}
?>
```

Archivo: PeruTraining\model\Planilla.php

```
<?php
class Planilla {

    private $lista;

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

    public function agregarTrabajador(Trabajador $trab) {
        $this->lista[] = $trab;
    }

    public function getList() {
        return $this->lista;
    }

    public function getTotal(){
        $total = 0.0;
        foreach ($this->lista as $trab) {
            $total += ($trab->getSueldo() + $trab->getBono());
        }
        return $total;
    }

}
?>
```

Archivo: PeruTraining\view\main.php

```
<?php
session_start();
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
```

```
<link rel="stylesheet" type="text/css" href="../css/estilos.css">
<title></title>
</head>
<body>
<h1>PerúTraining</h1>
<table border="1" cellspacing="0">
  <tr class="menu01">
    <td width="100"><a href="registro.php">Registro</a></td>
    <td width="100"><a href="planilla.php">Planilla</a></td>
    <td width="100"><a href="salir.php">Salir</a></td>
  </tr>
</table>
<p>&nbsp;</p>
<p>&nbsp;</p>
</body>
</html>
```

Archivo: PeruTraining\view\registro.php

```
<?php

session_start();
require_once '../design/ITrabajador.php';
if( isset($_REQUEST["msg"]) ) {
    $msg = $_REQUEST["msg"];
}

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <link rel="stylesheet" type="text/css" href="../css/estilos.css">
    <title></title>
    <script type="text/javascript">
      function funcCambioTipo(){
        var tipo = document.getElementById("tipo").value;
        var divCargo = document.getElementById("divCargo");
        var divHoras = document.getElementById("divHoras");
        if(tipo == "D"){
          divCargo.style.display = "none";
          divHoras.style.display = "";
        } else {
          divCargo.style.display = "";
          divHoras.style.display = "none";
        }
      }
    </script>
  </head>
```

```

<body>
  <h1>PerúTraining</h1>
  <table border="1" cellspacing="0">
    <tr class="menu01">
      <td width="100"><a href="registro.php">Registro</a></td>
      <td width="100"><a href="planilla.php">Planilla</a></td>
      <td width="100"><a href="salir.php">Salir</a></td>
    </tr>
  </table>
  <h2>Registro de Trabajador</h2>
  <form name="form1" method="post"
  action="../controller/registroController.php">
    <div class="divCampo">
      <label for="tipo" class="etiqueta">Tipo</label>
      <select name="tipo" class="campoEdicion" id="tipo"
      onChange="funcCambioTipo()">
        <option value="D" selected>Docente</option>
        <option value="E">Empleado</option>
      </select>
    </div>
    <div class="divCampo">
      <label for="apellidos" class="etiqueta">Apellidos</label>
      <input name="apellidos" type="text" class="campoEdicion"
      id="apellidos" size="30" maxlength="30">
    </div>
    <div class="divCampo">
      <label for="nombres" class="etiqueta">Nombres</label>
      <input name="nombres" type="text" class="campoEdicion"
      id="nombres" size="30" maxlength="30">
    </div>
    <div class="divCampo" id="divCargo" style="display:none;">
      <label for="cargo" class="etiqueta">Cargo</label>
      <select name="cargo" class="campoEdicion" id="cargo">
        <option value="<?php echo(ITrabajador::EMPL_COORDINADOR); ?>">
          <?php echo(ITrabajador::EMPL_COORDINADOR); ?>
        </option>
        <option value="<?php echo(ITrabajador::EMPL_ASISTENTE); ?>">
          <?php echo(ITrabajador::EMPL_ASISTENTE); ?>
        </option>
        <option value="<?php echo(ITrabajador::EMPL_SECRETARIA); ?>">
          <?php echo(ITrabajador::EMPL_SECRETARIA); ?>
        </option>
      </select>
    </div>
    <div class="divCampo" id="divHoras">
      <label for="horas" class="etiqueta">Horas</label>
      <input name="horas" type="text" class="campoEdicion"
      id="horas" size="10" maxlength="10">
    </div>
    <div class="divBotones">
      <input name="btnGrabar" type="submit" class="boton"

```



```
        id="btnGrabar" value="Grabar">
    </div>
</form>
<?php
if( isset( $msg ) ) {
    echo("<p>$msg</p>");
}
?>
</body>
</html>
```

Archivo: PeruTraining\view\planilla.php

```
<?php

session_start();
require_once '../model/Docente.php';
require_once '../model/Empleado.php';
require_once '../model/Planilla.php';

if( isset( $_SESSION["planilla"] ) ) {
    $planilla = unserialize($_SESSION["planilla"]);
}

?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <link rel="stylesheet" type="text/css" href="../css/estilos.css">
        <title></title>
    </head>
    <body>
        <h1>PerúTraining</h1>
        <table border="1" cellspacing="0">
            <tr class="menu01">
                <td width="100"><a href="registro.php">Registro</a></td>
                <td width="100"><a href="planilla.php">Planilla</a></td>
                <td width="100"><a href="salir.php">Salir</a></td>
            </tr>
        </table>
        <h2>Planilla General</h2>
        <?php if( isset($planilla) ) { ?>
        <table border="1" cellspacing="0">
            <tr class="tablaTitulo">
                <td width="80" height="19">Tipo</td>
                <td width="100">Apellidos</td>
                <td width="100">Nombres</td>
                <td width="70">Sueldo</td>
```

```
<td width="70">Bono</td>
<td width="70">Total</td>
</tr>
<?php foreach ($planilla->getLista() as $obj) { ?>
<tr class="TablaDato">
<td><?php echo( $obj->getTipo() ); ?></td>
<td><?php echo( $obj->getApellidos() ); ?></td>
<td><?php echo( $obj->getNombres() ); ?></td>
<td><?php echo( $obj->getSueldo() ); ?></td>
<td><?php echo( $obj->getBono() ); ?></td>
<td><?php echo( $obj->getSueldo() + $obj->getBono() ); ?></td>
</tr>
<?php } ?>
</table>
<p>Total: <?php echo( $planilla->getTotal() ); ?></p>
<?php } else { ?>
<p>No existen trabajadores en la planilla.</p>
<?php } ?>
</body>
</html>
```

Archivo: PeruTraining\view\sair.php

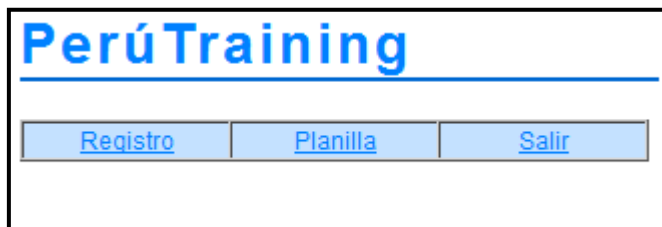
```
<?php
session_start();
session_unset();
session_destroy();
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<link rel="stylesheet" type="text/css" href="../css/estilos.css">
<title></title>
</head>
<body>
<h1>PerúTraining</h1>
<p>Sesión Finalizada.</p>
</body>
</html>
```

Archivo: PeruTraining\index.php

```
<?php
header("location: view/main.php");
?>
```

4.5 Ejecución del Proyecto

Cuando ejecutamos el proyecto inicialmente tenemos la siguiente interfaz:



The main menu of the PerúTraining system. It features the title 'PerúTraining' at the top. Below the title, there are three buttons: 'Registro', 'Planilla', and 'Salir'.

Para registrar un trabajador hacemos clic en la opción **Registro**, accedemos a la siguiente IU:



The 'Registro de Trabajador' form. It includes the title 'PerúTraining' and navigation buttons 'Registro', 'Planilla', and 'Salir'. The main heading is 'Registro de Trabajador'. Below this, there are input fields for 'Tipo' (a dropdown menu with 'Docente' selected), 'Apellidos', 'Nombres', and 'Horas'. A 'Grabar' button is at the bottom.

Para visualizar la planilla debemos ingresar a la opción **Planilla**, obtenemos la siguiente IU:



The 'Planilla General' table. It features the title 'PerúTraining' and navigation buttons 'Registro', 'Planilla', and 'Salir'. The main heading is 'Planilla General'. Below this is a table with 6 columns: Tipo, Apellidos, Nombres, Sueldo, Bono, and Total. The table contains 8 rows of employee data. At the bottom, there is a 'Total: 49200' label.

Tipo	Apellidos	Nombres	Sueldo	Bono	Total
Empleado	Romero	Carlos	3500	3500	7000
Empleado	Ruiz	Mariela	2500	2500	5000
Empleado	Beltran	Claudia	1800	1800	3600
Docente	Coronel	Gustavo	5000	3000	8000
Docente	Valencia	Pedro	5000	3000	8000
Docente	Matsukawa	Sergio	5000	3000	8000
Docente	Marcelo	Ricardo	6000	3600	9600

Total: 49200

Finalmente, para salir del sistema y borrar todos los datos de sesión ingresamos a la opción **Salir**.