INGENIERÍA DE APLICACIONES

Lenguajes Orientados a Objetos

Dra. María Luján Ganuza
mlg@cs.uns.edu.ar
DCIC - Depto. de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur, Bahía Blanca
2019

Temario

Lenguajes Orientados a Objetos

C++

C#

BV.NET

PHP

JavaScript

Lenguajes Orientados a Objetos

Los lenguajes orientados a objetos (LOO) pueden clasificarse en dos grandes grupos:

- Lenguajes **basados en clases** (*Class-Based Languages*)
- Lenguajes **basados en objetos** (*Object-Based Languages*)

Los **lenguajes basados en clases** forman el grupo mayoritario de los LOO, e incluye a los más populares.

Entre ellos, están los “históricos”

- *Simula*, considerado primer lenguaje orientado a objetos.

- *Smalltalk*, el primer lenguaje orientado a objetos con tipeo dinámico.

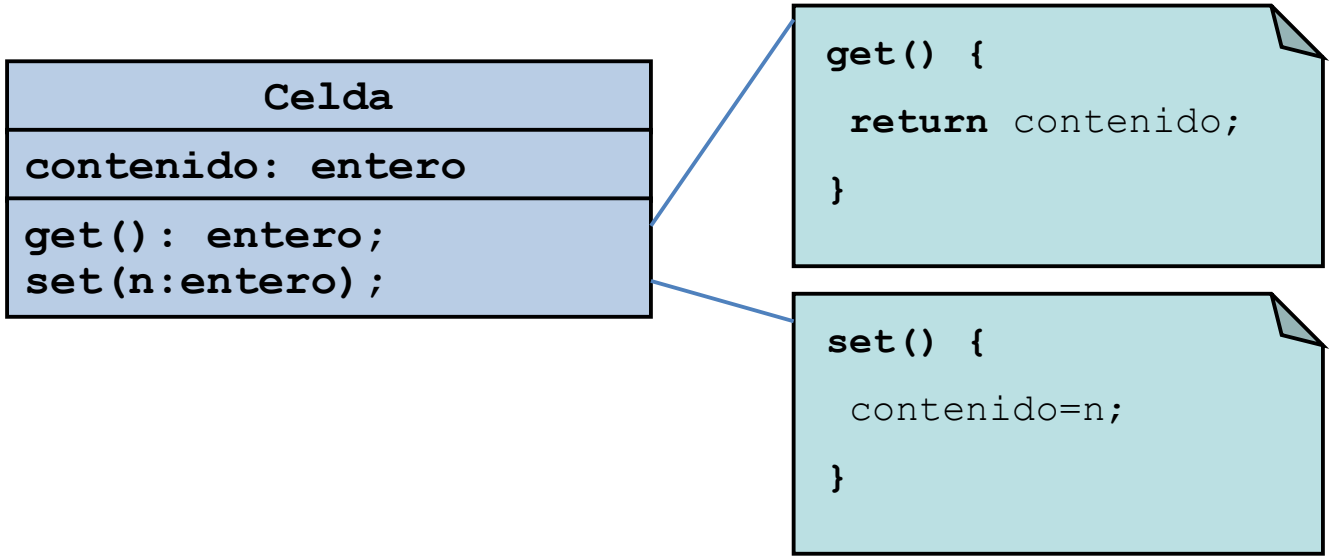
- *C++,* descendiente de *Simula*, y ampliamente utilizado en la industria.

- Java

Lenguajes Orientados a Objetos

Los lenguajes basados en clases (LBC) se centran en la noción de ***clase*** como descripción de los objetos.

Una ***clase*** describe la estructura de todos los objetos generados a partir de esa ***clase***.



Lenguajes Basados en clases

Los objetos son producidos a partir de la clase, la cual se interpreta como un *esquema estructural* de los objetos.

La mayoría de los lenguajes poseen creación explícita de objetos, por medio de operaciones especiales denominadas ***constructores***.

Los *constructores* inician implícitamente el proceso de creación física del objeto (memoria) y proveen algunas sentencias de inicialización.

```
celdita ← new Celda()
```

En algunos lenguajes los objetos pueden **destruirse** por medio de operaciones especiales denominadas ***destructores***, que inician el proceso de eliminación del objeto del sistema.

Lenguajes Basados en Clases

La destrucción de los objetos puede ser **explícita** (con operadores como *dispose*) o **implícita** (*garbage collector*). En ambos casos el destructor es invocado.

C++ *permite la definición de destructores y Java posee el servicio finalize() en la clase Object.*

Usualmente los objetos suelen manipularse por medio de **referencias** (*semántica por referencia*) aunque algunos lenguajes admiten objetos **expandidos** (*semántica por valor*).

- ***Java*** *utiliza únicamente referencias.*

- ***Eiffel*** *permite objetos por valor (subobjetos).*

Lenguajes Basados en Objetos

Los **lenguajes basados en objetos** pretenden ser más simples y más flexibles que los lenguajes basados en clases.

Aunque existen varios lenguajes, sólo uno es muy populares en la industria (JavaScript).

Proveen únicamente la noción de **objeto** y **despacho dinámico**.

No existe la noción de clase y por ende la de constructores como mecanismo de creación de objetos individuales

En los lenguajes basados en objetos, se define el tipo del objeto como ***interfaz***, y se ***implementan*** las operaciones en la declaración misma del objeto.

Lenguajes Basados en Objetos

Definición del tipo Cell

```
ObjectType Cell is
  var contents: Integer;
  method get(): Integer;
  method set(n:Integer);
end;
```

Declaración de un objeto *unaCelda* de tipo Cell

```
object unaCelda:Cell is
  var contents: Integer := 0;
  method get(): Integer is
    return self.contents
  end;
  method set(n: Integer) is
    self.contents := n
  end;
end;
```

Lenguajes Basados en Objetos

El rol del constructor puede cumplirlo una operación como la siguiente:

```
procedure newCell(m: Integer): Cell is
object cell: Cell is
    var contents: Integer := m;
    method get(): Integer is return self.contents end;
    method set(n: Integer) is self.contents := n end;
  end;
return cell;
end;
…
var cellInstance: Cell := newCell(0);
```

C++

C++

Es uno de los lenguajes más pupulares.

C++ *es el antecesor de* ***Java****.*

En sus comienzos, a modo de introducción se decía que ***Java = (C++) - -***

C++ *fue creado por Bjarne Stroustup en 1979.*

Agrega nociones orientadas a objetos al lenguaje ***C****, conservando los dos estilos de programación.*

Esto permitió la masificación de la OO en la industria, donde ***C*** *ya era un lenguaje popular.*

Clases en C++

Una clase en C++ tiene una ***declaración*** y una **definición** o ***implementación***.
La declaración tiene el siguiente formato:

```
class Nombre
{
// Cuerpo de la clase
}
```

```
class Nombre {
      private:
            // Miembros privados
      public:
            // Miembros públicos
      protected:
            // Miembros protegidos
}
```

El cuerpo contiene cero o más definiciones de ***miembros*** (*datos o funciones*).
Los miembros pueden ser ***privados***, ***públicos*** o ***protegidos***.

Clases en C++

En la ***declaración de la clase*** se incluyen los atributos y el signature o encabezado de las operaciones.

Puede agregarse implementación, pero sólo de operaciones simples.

En **C++** la *declaración de la clase* y la *definición* no necesariamente están en un mismo archivo...

`miClase.h`

Declaración de datos y métodos + posibles implementaciones <u>breves</u>

`miClase.cpp`

Implementación de métodos

Clases en C++

El formato de declaración de operaciones es:

```
tipo Nombre-clase :: Nombre_función (argumentos)
{
    // código
}
```

Las variables y objetos se declaran anteponiendo el tipo:

```
float cantidad;
boolean conexionActiva;
CuentaBancaria cbu123AB
Empleado lenny,karl;
```

La sintaxis de las sentencias de control es prácticamente la misma que en Java! :)

Clases en C++: declaración e implementación

```
class CuentaBanco {
   public:
      CuentaBanco();
      float saldo_cuenta();
      float retirar(float);
      void depositar(float);
   private:
      float saldo;
}
```

Archivo Cuenta.h

```
CuentaBanco::CuentaBanco(){
      saldo = 0.0;
}
float CuentaBanco::saldo_cuenta(){
      return saldo;
}
float CuentaBanco::retirar(float c){
      if (c<=saldo) {
            saldo = saldo - c;
            return c;
      } else {
            return 0;
}
void CuentaBanco::depositar(float c){
      saldo = saldo + c;
}
```

Clases en C++: creando objetos

C++ tiene dos operadores **new** y **delete** cuyo objetivo es la creación dinámica de objetos.

```
Empleado *homero;
homero = new Empleado;
```

Las llamadas calificadas (la invocación a métodos o atributos del objeto) se realizan utilizando el símbolo **->**

```
int edad = homero->edad();
```

Herencia en C++

En **C++** la **herencia** suele denominarse también ***derivación*** de clases.
Sustenta ***herencia simple*** *y* ***herencia múltiple***.

La sintaxis requerida es:

```
class Padre {
 //codigo clase padre
}
class Hija: [public|private|protected] Padre {
 //codigo clase hija
}
```

Los **modificadores de acceso** en la declaración de la herencia permiten la redefinición de visibilidades de la clase padre.

Herencia en C++

Derivación pública	
Acceso clase padre (base)	**Acceso clase hija (derivada)**
public	public
protected	protected
private	*no accesible*

Herencia en C++

Derivación protegida	
Acceso clase padre (base)	**Acceso clase hija (derivada)**
public	protected
protected	protected
private	*no accesible*

Herencia en C++

Derivación privada	
Acceso clase padre (base)	**Acceso clase hija (derivada)**
public	private
protected	private
private	*no accesible*

Ejemplo Herencia – clase Habitación

```
class Habitacion {
  public:
      Habitacion(const int);
      ~Habitacion();
      int size(void);
  private:
      int metros_cuadrados;
};
```

Habitacion.h

```
Habitacion::Habitacion(const int n)
{
      metros_cuadrados = n;
}

Habitacion::size()
{
      return metros_cuadrados;
}
...
```

Habitacion.cpp

Ejemplo Herencia – clase Oficina

```
class Oficina : public Habitacion{
  public:
      Oficina(const int, const int);
      ~Oficina();
      void listarCapacidad(void);
  private:
      int capacidad;
};
```

Oficina.h

```
Oficina::Oficina(const int size, const int capa)
      : Habitacion(size)
{
      capacidad = capa;
}
...
```

Oficina.cpp

C#

Redes de Computadoras

- En los últimos 20 años surge un *nuevo* escenario de ejecución: las **redes de computadoras**.
- Los sistemas ya no residen en una sola PC, sino “fraccionado” en varias computadoras que en conjunto conforman el sistema completo.
- Han surgido diversas propuestas para desarrollar software en este escenario.
  - **Java**
  - **.Net**

Redes de Computadoras

- **Java** es:
  Multiplataforma.
  Máquina virtual.
  Orientado a objetos.

- **.Net** ofrece:

  Máquina virtual, multiplataforma en potencia.
  Varios lenguajes, fáciles de integrar.
  Orientado a objetos en general.

  Uno de los lenguajes más populares de .NET es **C#**.

  Surge como una alternativa "*estilo Java*" para programar en el framework de **Microsoft**.

Definición de Clases

- **En Java**

```
class HelloWorld {
        public static void main(String[] args) {
                System.out.println("Hello, World!");
        }
}
```

- **En C#**

```
class HelloWorld {
        public static void Main(string[] args) {
                System.Console.WriteLine("Hello, World!");
        }
}
```

Definición de Clases

- **En Java**

```
class Foo extends Bar implements IFooBar {
      //cuerpo de la clase
}
```

- **En C#**

```
class Foo : Bar , IFooBar {
      //cuerpo de la clase
}
```

Referencia a la clase padre

desde la clase hija

- **En Java**

```
super.hashCode();
```

- **En C#**

```
base.hashCode():
```

Importación

- **En Java**

```
import java.util;
```

- **En C#**

```
using System.Net;
```

Clase Persona en C#

```
using System;
class Persona {
        private string miNombre ="N/A";
        private int miEdad = 0;
        // Declara una propiedad Nombre de tipo string:
        public string Name {
                get { return miNombre; }
                set { miNombre = value; }
        }
        // Declara una propiedad Edad de tipo int:
        public int Edad {
              get { return miEdad; }
              set { miEdad = value; }
        }

}
```

```
Persona p = new Persona();
p.Name = "Juan";
p.Edad = 34;
…
Console.Writeline(p.Edad);
```

Java y C#

C# conserva el estilo **Java** de definición de visibilidad de operadores
(*por entidades y no por bloques como en C++)*

La sintaxis de las sentencias es la misma.

Los dos proveen un operador que permite averiguar la clase base de un objeto
(Java: **`instanceof`**, C#: **`is`**).

Los dos poseen casting para polimorfismo.

Ninguno de los dos permite herencia múltiple.

Los dos usan el modificador *abstract* para las clases y operaciones abstractas.

VB.NET

VB.NET – Definición de clase

En **VB.NET** una clase es una forma de tipo de dato. Define un conjunto de miembros: campos, propiedades, métodos y eventos.

```
Public Class Employee
  Public EmployeeNumber As Integer
  Public FamilyName As String
  Public GivenName As String
  Public DateOfBirth As Date
  Public Salary As Decimal

  Public Function Format( ) As String
    Return GivenName & " " & FamilyName
  End Function

End Class
```

Acceso a datos y métodos

```
Dim emp As New Employee( )

emp.EmployeeNumber = 10
emp.FamilyName = "Grimes"
emp.GivenName = "Frank"
emp.DateOfBirth = #1/28/1965#
emp.Salary = 115000

Console.WriteLine("Employee Name: " & emp.Format( ))
Console.WriteLine("Employee Number: " & emp.EmployeeNumber)
Console.WriteLine("Birth: " & emp.DateOfBirth.ToString("D", Nothing))
Console.WriteLine("Salary: " & emp.Salary.ToString("C", Nothing))
```

```
Employee Name: Frank Grimes
Employee Number: 10
Date of Birth: Thursday, January 28, 1965
Salary: $115,000.00
```

Herencia en VBasic

```
Public MustInherit Class Shape
  Public Origin As Point
  Public Size As Extent
  Public MustOverride Sub Draw( )
  Public Sub Offset(ByVal Amount As Extent)
      Origin.X += Amount.XExtent
      Origin.Y += Amount.YExtent
  End Sub
…
End Class
```

Herencia en VBasic

```
Public Class Circle Inherits Shape
  Public Overrides Sub Draw( )
      'dibujar circulo ! !
  End Sub
End Class

Public Class Square Inherits Shape
  Public Overrides Sub Draw( )
     'dibujar cuadrado! ! !
  End Sub
End Class
```

PHP

PHP

El lenguaje **PHP** surge en 1995 bajo el nombre PHP/FI, creado por Rasmus Lerdorf.

Luego de liberar PHP, junto con Zeev Suraski y Andi Gutmans reecriben el intérprete y nace **PHP3** en 1997.

En 1999 reescriben el núcleo de **PHP** para obtener ***Zend Engine***.

En 2004 sale **PHP5**, con varias modificaciones, entre ellas un mejor modelo de objetos, deficiente en la versión anterior.

Hoy es usado por muchos sitios gracias al desarrollo de varios productos, la mayoría GNU: *phpBB*, *WordPress*, *MediaWiki*, *Joomla*, *Mambo*, etc.

PHP

La sintaxis y las características orientadas a objetos del lenguaje PHP(5) son fáciles de aprender, pues guardan similitud con lenguajes vistos anteriormente.

```
if (cond)
    { ... }
else
    { ...}
```

```
while (cond) {
...
}
```

```
do{
...
} while (cond)
```

```
for ($i=1; $i<=5; $i++)
{
  ...
}
```

PHP

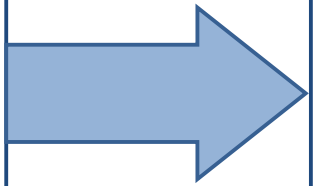
PHP es ***inscrustable*** en HTML, similar a **JavaScript.**
Todo código restante HTML se envía tal cual al cliente.

Tag inicial	Tag final
<?php	?>
<?	?>
<script language=”php”>	?>
<%	%>

PHP

```
<html>
<head>
  <title>Ejemplo</title>
</head>
<body>
  <?php
    echo "Hola Mundo!";
  ?>
</body>
</html>
```

hola.php



```
<html>
<head>
  <title>Ejemplo</title>
</head>
<body>
  Hola Mundo!
</body>
</html>
```

Resultado recibido por el cliente

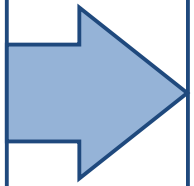
PHP

El script .php no necesariamente debe incluir código HTML...

```
<?php
 echo "<HTML>";
 echo "<TITLE>Ejemplo</TITLE>";
 echo "<BODY>";
 echo "Hola Mundo!";
 if date('w')=5
     { echo "Hoy es sabado"; }
 else
     { echo "Hoy no es sabado";}
 echo "</BODY></HTML>";

?>
```

todophp.php



```
<HTML>
<TITLE>Ejemplo</TITLE>
<BODY>
Hola Mundo! Hoy no es sabado
</BODY>
</HTML>
```

Resultado recibido por el cliente

PHP

```php
<?php
	function cuadrado($num) {
		return $num*$num;
	}
	$cartel1 = "El cuadrado de ";
	$cartel2 = " es ";
	$numero=4;

	echo $cartel1;
	echo $numero;
	echo $cartel2;
	echo cuadrado($numero);
	echo "<BR>";

	$numero=10;
	echo $cartel1.$numero.$cartel2.cuadrado($numero);

?>
```

Pueden implementarse funciones...

Clases en PHP

```php
<?php

class A
{
    public $var = 'un valor';

    function funcion1()  {
        //codigo
    }
    function funcion2()  {
        //codigo
    }

}
?>
```

Los objetos se crean con la palabra reservada **new** y el acceso a los miembros de los objetos es por medio del operador **->**

Para referenciar el objeto actual se utiliza la pseudo-variable $this.

Clases en PHP

```php
<?php
class SimpleClass
{
  // Declaración de atributo
  public $var = 'Hola Mundo!';

  // Declaración de método
  public function displayVar(){
        echo $this->var;
    }
}
?>
```

simpleclass.php

```php
<?php
include("simpleclass.php");

$a = new SimpleClass();
$a->displayVar();
$a->var = "Chau Mundo";
$a->displayVar();

?>
```

holachau.php

Herencia en PHP

- La herencia se denota con la palabra reservada ***extends***.
- PHP no admite herencia múltiple.
- Los métodos y miembros heredados pueden redefinirse, a menos que el ancestro incluya la palabra reservada ***final***.
- Una clase también puede ser ***final***.
- Es posible acceder a métodos de la clase padre por medio del prefijo ***parent::***

Herencia en PHP

```php
<?php
class SimpleClass
{
 // Declaración de atributo
 public $var = 'Hola Mundo!';

 // Declaración de método
 public function displayVar() {
        echo $this->var;
    }
}
?>
```

```php
<?php
class ExtendClass extends
SimpleClass
{
    function displayVar(){
      //redefinición
      echo "Heredando…\n";
      parent::displayVar();
    }
}

$extended = new ExtendClass();
$extended->displayVar();
?>
```

PHP también permite **clases abstractas** e **interfaces**.

Constructores en PHP

Los constructores deben tener el siguiente encabezado:

```
void __construct ( [mixed args [, ...]] )
```

```php
<?php
class BaseClass {
   function __construct() {
       print "In BaseClass constructor\n";
   }
}
class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();
       print "In SubClass constructor\n";
   }
}
$obj = new BaseClass();
$obj = new SubClass();
?>
```

Atributos en PHP

- Los **atributos** pueden ser **públicos**, **privados** o **protegidos**.
- *Los **atributos privados** no se heredan, por lo que un intento de acceso a ellos queda indefinido.*
- Los métodos pueden tener los mismos modificadores de visibilidad, con el mismo significado.

Atributos en PHP

```php
<?php
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';
    function printHello()    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}
$obj = new MyClass();
echo $obj->public;      // Funciona OK!
echo $obj->protected;   // Error fatal
echo $obj->private;     // Error fatal
$obj->printHello();     // Muestra todos
?>
```

Comparaciones en PHP

Existen dos operadores

==

Los objetos son iguales si poseen los mismos atributos y son instancias de la misma clase.

===

Las variables de objeto son idénticas si y sólo si refieren a la misma instancia de la misma clase.

JAVASCRIPT

JavaScript

El más popular de los lenguajes de script para la programación del lado cliente.

- Sintaxis simple, sin tipos de datos.
- Puede modificar dinámicamente el documento en el que se encuentra, por medio de una interfaz especial denominada DOM.
- Puede reaccionar ante eventos ocurriendo en la página en la que se encuentra (*onMouseOver,onClick,onLoad*,etc).
- Puede ser utilizado para la validación de datos.
- Puede detectar el browser del visitante y ayudar a optimizar la visualización
- Puede realizar pedidos adicionales a un servidor, en *background*, en forma asincrónica (AJAX).

JavaScript

```
<html>
 <head>
     <title>Pagina</title>
 </head>
 <body>
     <h2>Bienvenido</h2>
     <div> etc </div>

     <script>
          //sentencias
     </script>

 </body>
</html>
```

El código será **interpretado por el navegador**.
NO EL SERVIDOR

JavaScript

Podemos declarar variables con la palabra reservada **var**.

Estas variables no se declaran con tipos, lo cual es razonable al carecer de clases.

```
var robot = "C3P0";
var episodio=8;
var estreno = true;
```

Las sentencias son del estilo del lenguaje Java.

JavaScript

Las funciones se declaran de forma similar, pero sin mencionar los tipos, ausentes en el lenguaje

```
function nombre(param1, param2, p…, paramn) {
      // sentencias
}
```

Se invocan de la manera tradicional

```
nombre(p1, p2, ... , pn);
```

JavaScript - Ejemplo

```
<HTML>
	<HEAD>
		<TITLE>Ejemplo JS</TITLE>
	</HEAD>

	<BODY>

	<SCRIPT>
		var mostrar=true
	</SCRIPT>

	<p>Bienvenidos!</p>

	<SCRIPT>
		if(mostrar==true){ document.write("Hola! "); }
	</SCRIPT>

<p>Fin.</p>

</BODY>
</HTML>
```

JavaScript - Objetos

Es posible crear **objetos** en JavaScript.

Sin embargo, al ser un **lenguaje basado en objetos**, estos no se crean por medio de clases, sino por medio de operaciones que definen la estructura del objeto mismo.

```
function crearObj(param1, param2, p…, paramn) {
        this.atributo1 = param1;
        this.atributo2 = param2;
        …
        this.operacion1 = operacion1;
        this.operacion2 = operacion2;
}
```

Donde **operacion1**, **operacion2**, … son operaciones declaradas en el mismo ***script***.

La función trabaja como un constructor, que al mismo tiempo define los atributos y las operaciones del objeto.

JavaScript - Ejemplo

```
<HTML><HEAD><TITLE>Ejemplo JS</TITLE>
</HEAD>

<BODY>
Hola!<BR>

<SCRIPT >

  function imprimir(){
   document.write("<");
   document.write(this.coordx);
   document.write(",");
   document.write(this.coordy);
   document.write(">");
  }

  function crearPunto(x,y){
   this.coordx=x;
   this.coordy=y;
   this.imprimir=imprimir;
  }

</SCRIPT>
...
```

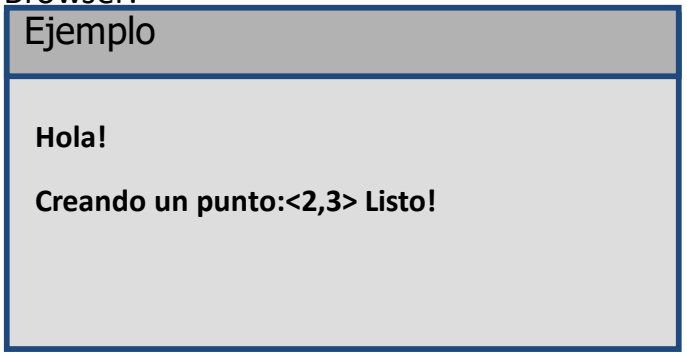
```
...
Creando un punto:

<SCRIPT >
  p1=new crearPunto(2,3);
  p1.imprimir();
</SCRIPT>

Listo!

</BODY>
</HTML>
```

Browser:



Material Bibliográfico

- Abadi, M., & Cardelli, L. A theory of objects. Springer Science & Business Media, 2012.

- Szyperski, C., Component Software: Beyond Object Oriented Programming. AddisonWesley, 2nd Ed., 2011

- Zamir, S., Ed., Handbook of Object Oriented Technology. CRC Press, 2000.

PRÓXIMA CLASE

Patrones de Diseño