Object-Oriented Programming

Rodrigo Augosto, Alejandro Gonzalez

November 16, 2010

This document has been prepared to guide learners of Object-Oriented Programming (**OOP**).

We used LaTeX to make this document and there are examples to each term involved using Python, Ruby, C#, Java and JavaScript

Contents

1 ProgrammingFundamentals **2**

2 Class **3**
- 2.1 Encapsulation 6
- 2.2 Inheritance 6
- 2.3 Polymorphism 6

3 Casting **6**

4 Class Abstracts **7**

5 Delegates **7**

6 Enumeration **8**

7 Generics **8**

8 Interface **8**

9 Lambda **8**

10 TDD **8**

1 ProgrammingFundamentals

- Existen dos tipos de Software: De sistema y de Aplicacin.
- Lenguajes de programacin pueden ser: De mquina, Bajo nivel y Alto Nivel.

- Tipo de Lenguajes:
  - Estructurados (C, Pascal, Basic)
  - Orientados a Objetos (C#, VB.NET, Smalltalk, Java)
  - Declarativos (Prolog)
  - Funcionales (CAML)
  - Orientados a Aspectos
  - Hbridos (Lisp, Visual Basic)
- Las Sentencias, describen acciones algortmicas que pueden ser ejecutadas, clasificadas en:
  - Ejecutables / No ejecutables
  - Simples / Estructuradas
- Una expresin es un conjunto de datos unidos por operadores que tiene un nico resultado (a = ((2+6) / 8) * 3).
- Las estructuras de control permiten alterar el orden del flujo de control
  - Estructuras de Control Selectivas; IF, CASE
  - Estructuras de Control Repetitivas: FOR, WHILE
- Existen diversos tipos de operadores:
  - Aritmticos: suma, resta, multiplicacin, etc.
  - De relacin: igual, mayor, menor, distinto, etc.
  - Lgicos: and, or, not, etc.
- Alcance o tipo de miebros se refiere a los campos, propiedades, mtodos, eventos, clases anidadas, etc
  - Public
  - Private
  - Internal
  - Protected
  - Protected Internal
- Las comillas dobles ("") delimitan strings y las comillas simples (') delimintan caracteres.

Bibliotecas Desarrollo -¿ Programa Fuente Compilacin -¿ Programa Objeto Link-Edicin -¿ Programa Ejecutable

??? Mtodos estticos: no requieren de una instancia para ser invocados. Se los llama mtodos de clase ???

??? Namespace de una clase ???

2 Class

Informalmente, un objeto representa una entidad (Fsica, Conceptual o Software) del mundo real, adems posee (segn Booch) Estado, Comportamiento e Identidad

La clase es el tipo del objeto, es decir, es una descripcin de un grupo de objetos con propiedades en comn (atributos), comportamiento similar (operaciones), misma forma de relacionarse con otros objetos (relaciones) y una semntica comn

* Atributos: representan los estados del objeto * Metodos: representan el comportamiento del objeto.

El ideal, purista, es que los metodos sean publicos y todos los atributos sean privados.

Todos los objetos necesitan de un constructor, que tiene el mismo nombre de la clase, el cual es un metodo que reserva memoria.

Usuarios coto = new User();

Puede existir un constructor que reciba parametros.

Usuario coto = new User("coto", "1234");

Java y c# inicializan los estados del objeto con valores nulos, vacios (0 en caso de numerico) o = en caso de numricos.

———————————o———————————

Sobrecarga de metodos "overloading" != sobreescritura de metodos "overwriting" (polimorfismo)

"overloading" no es propio de la POO, y consiste en una clase con varios mtodos con el mismo nombre pero diferente firma.

"overwriting" La clase base define mtodos, los cuales pueden ser reescritos por clases que heredan de ella.

User(); User(string); User(string, int, string);

public (modificador de acceso) -

Modificador de acceso: [public, private, protected] indica que un metodo o atributo puede ser accedido desde otra clase o solo internamente (private), es muy mala prctica declarar estados privados.

Atributos - Contructores - Metodos, es el orden adecuado dentro de una clase.

Metodos para acceder a los atributos de los objetos

Atributos:metodos:sobrecarga: constructos: porperties:modificadores de accesos (TODO)

Listing 1: Java - Class Example

```java
class ClassExample {
  private String name;
  public ClassExample(){

  }
  public static void main(String[] args){
    ClassExample ce = new ClassExample();
    ce.name = "John";

    ClassExampleInstance cei = new ClassExampleInstance();
```

```
    cei.name = "Peter";
  }
}

class ClassExampleInstance {
  private String name;

}
```

Listing 2: C# - Class Example

```
using System;
using System.IO;

class Program
{
    static void Main()
    {
        string p = @"C:\Users\Sam\Documents\Test.txt";

        string e = Path.GetExtension(p);
        if (e == ".txt")
        {
            Console.WriteLine(e);
        }
    }
}
```

Listing 3: Python - Class Example

```
class MyClass:
        """A simple example class"""
        i = 12345
        def f(self):
            return 'hello world'

x = MyClass()
print "Hola"
```

Listing 4: Ruby - Class Example

```
class Person
  attr_accessor :fname, :lname

  def initialize(fname, lname)
    @fname = fname
    @lname = lname
  end

  def to_s
    @lname + ", " + @fname
  end

```

```
  def self.find_by_fname(fname)
    found = nil
    ObjectSpace.each_object(Person) { |o|
      found = o if o.fname == fname
    }
    found
  end
end
```

Listing 5: JavaScript - Class Example

```
person = new Object()
person.name = "Tim Scarfe"
person.height = "6Ft"

person.run = function() {
  this.state = "running"
  this.speed = "4ms^-1"
}

//############################

timObject = {
  property1 : "Hello",
  property2 : "MmmMMm",
  property3 : ["mmm", 2, 3, 6, "kkk"],
  method1 : function(){alert("Method had been called" + this.
      property1)}
};

timObject.method1();
alert(timObject.property3[2])

for(x in timObject) alert( x + "-" + timObject[ x ] )

//############################

function cat(name) {
  this.name = name;
  this.talk = function() {
    alert( this.name + " say meeow!" );
  }
}

cat1 = new cat("felix");
cat1.talk() //alerts "felix says meeow!"

cat2 = new cat("ginger");
cat2.talk() //alerts "ginger says meeow!"
cat.prototype.changeName = function(name) {
  this.name = name;
}

firstCat = new cat("pursur");
firstCat.changeName("Bill");
firstCat.talk() //alerts "Bill says meeow!"
```

2.1 Encapsulation

Obj: No modificar estados internos de un objeto de forma directa, sino a travs de lods metodos expuestos.

el objeto tiene un estado interno que este representado por cada uno de sus atributos, el cual no puede ser cambiado forma directa, a mneos que se exponga un metodo de tipo publico que lo haga.

2.2 Inheritance

Obj: Este es un concepto fundamental para la POO y para el lenguaje java, ya que con este concepto significa que vamos a poder reutilizar codigo. Un ejemplo seria una clase Figura Geometrica, que tiene funciones como el calculo de su perimetro y de su area, y tiene como sub clase la clase Cuadrado, que era sus metodos de la clase Figura Geometrica que son en este caso perimetro y area, aca aprovechamos la reutilizacion de codigo. Pero en java no existe la herencia multiple como en otros programas como C/C++, aca es solo herencia simple, pero en java existe algo que simula esta herencia multiple que son las llamadas interfaces que posteriormente vamos a estudiar.

2.3 Polymorphism

Obj: Dado un mensaje despues de una instancia a una/varias subclases, todas debern responder de forma diferente.

Este concepto se base en que podemos utilizar varios metodos con el mismo nombre y con diferente funcionalidad. Por ejemplo de mi clase Vehiculos tenemos el metodo frenar, y tenemos sus sub clases Automovil y Motocibleta, ambos tienen el metodo frenar pero cada uno tiene una definicion diferente para cada clase. A esto se le denomina polimorfismo, mas adelante lo vamos a ver en las sobrecargas de funciones y redefiniciones de metodos en la herencia.

3 Casting

TODO

4 Class Abstracts

@@@ Pilares de la Orientacin a Objetos @@@

** Abstraccin **

La abstraccion es un metodo por el cual abstraemos, vale la redundancia, una determinada entidad de la realidad sus caracteristicas y funciones que desempean, estos son representados en clases por medio de atributos y metodos de dicha clase.

Ejemplo: Un ejemplo sencillo para comprender este concepto seria la abstraccion de un Automovil. Aca vamos a sacar de estas entidad sus caracteristicas por ejemplo: color, ao de fabricacion, modelo, etc. Y ahora sacamos sus metodos o funciones tipicas de esta entidad como por ejemplo: frenar, encender, etc.

A esto se le llama abstraccin.

** Encapsulamiento **

Atributos deben ser privados para otros objetos, exponiendolos solo a travs del comportamiento definido a travs de miembros pblicos.

Util para el control/validacin y respuesta ante cambios.

** Relaciones **

Los objetos contribuyen en el comportamiento de un sistema colaborando entre si a travs de sus relaciones.

Una Relacin de asociacin es una conexin entre dos clases que representa una comunicacin (e.g. Una Persona es Duea de un Vehculo)

Una Relacin de agregacin es una forma especial de asociacin donde un todo se relaciona con sus partes (e.g. Una Puerta es una parte de un Vehculo)

** Herencia **

Es un tipo de relacin entre clases en la cual una clase comparte la estructura y comportamiento definido en otra clase (Grady Booch)

@@@ Conceptos del Diseo Orientado a Objetos @@@

** Interfaces **

Recurso de diseo soportado por los lenguajes orientados a objetos que permite definir comportamiento.

La implementacin de una interfaz es un contrato que obliga a la clase a implementar todos los mtodos definidos en la interfaz.

** Polimorfismo **

Es la propiedad que tienen los objetos de permitir invocar genricamente un comportamiento (mtodo) cuya implementacin ser delegada al objeto correspondiente recin en tiempo de ejecucin.

5 Delegates

TODO

6 Enumeration

TODO

7 Generics

TODO

8 Interface

TODO

9 Lambda

TODO

Cuantas formas existen de expresar un puntero a una funcion

1. delegado on the fly
2. usando un metod ya existente
3. usando expresiones **lambda**

10 TDD

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle with these steps:

1. the developer writes a failing automated test case that defines a desired improvement or new function.
2. Run all tests and see if the new one fails.
3. Produces code to pass that test.
4. Run the automated tests and see them succeed.
5. Refactors the new code to acceptable standards.

Steps to create a unit test:

1. Establecer los datos de prueba y retorno
2. usar los datos de prueba para ejecutar el codigo que se esta probando
3. Generar las pruebas necesarias

Unit tests are so named because they each test one unit of code.

Listing 6: Example of Unit Test using C#

```
1 [Test()]
2 //Usada solo para chequear el tipo de la excepcion.
3 [ExpectedException(typeof(NullReferenceException))]
4 // IF "expression" returns null AND  "EmptyDataTemplate" is not null
      THEN
5 public void
    List_ExpressionReturnsNullAndEmptyDataTemplateisNotNull_ReturnsEmptyString
    () {
```

```
6     ValueToTest = .....
7   }
8
9
10  //Usada para chequear otras validaciones
11  public void List_Conditions_ReturnsValue() {
12    ValueToTest = .....
13    Assert(String.Empty, ValueToTest);
14  }
```