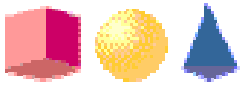
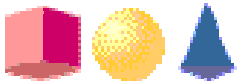
Programación Orientada a Objetos (POO)

Ing. Ramón Roque Hernández, M.C.
ramonroque@yahoo.com



1

Introducción



- Programación: Actividad compleja
- Sin comentarios…
- Crisis del Software
- Imponiendo orden al caos
- Programación Orientada a Objetos
- Definición formal de POO

2

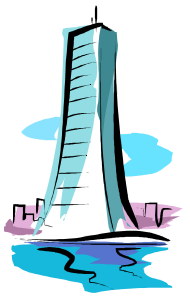
Programación: Actividad compleja

- Los problemas que se intentan resolver con el software implican elementos complejos propios del área a la que pertenecen.
- Además, es difícil gestionar el proceso de desarrollo de software.
- No hay estándares o límites para la flexibilidad en la construcción del software.

3

Sin comentarios…

- "Un constructor pensaría raramente en añadir un subsótano a un edificio ya construido de 100 plantas… Los usuarios de sistemas de software casi nunca lo piensan dos veces a la hora de solicitar cambios equivalentes…De todas formas (dicen ellos) es simplemente cosa de programar"



4

Crisis del software

- La incapacidad humana de dominar la complejidad del software conlleva a:
  - Proyectos retrasados
  - Proyectos que exceden el presupuesto
  - Proyectos deficientes que no cumplen los requerimientos

5

Imponiendo orden al caos

- "La técnica de dominar la complejidad se conoce desde tiempos remoros: ***divide et impera*** (divide y vencerás)".
  [Dijkstra]
- "Para entender un nivel dado de un sistema, basta con comprender unas pocas partes (no necesariamente todas) a la vez".
- Descomposición es la clave:
  - Descomposición algorítmica tradicional
  - Descomposición orientada a objetos

6

Programación Orientada a Objetos

- POO es un conjunto de técnicas que pueden utilizarse para desarrollar programas eficientemente.
- Los objetos son los elementos principales de construcción.

7

La POO es …

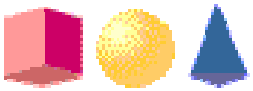
"Un método de implementación en el que los programas se organizan como colecciones **cooperativas** de **objetos**, cada uno de los cuales representan una **instancia** de alguna **clase** y cuyas clases son todas miembros de una jerarquía de clases unidas mediante relaciones de **herencia**"

Grady Booch

8

El modelo de Objetos

- Objetos en el mundo real
  - Propiedades
  - Métodos
- Abstracción
- Clases y Objetos
- Encapsulamiento
- Mensajes
- Constructores
- Destructores
- Herencia
  - Simple
  - Múltiple
- Clases Abstractas
- Sobreescritura
- Sobrecarga
- Polimorfismo



9

Objetos en el mundo real

Lavadora

Perro

Televisión

Persona

Factura

10

Podemos darnos cuenta que…

- Los objetos poseen características que los distinguen entre sí.
- Los objetos tienen acciones asociadas a ellos.

11

Ejemplo: PERRO

- Características:
  - Nombre: “FIDO”
  - Raza: “Chihuahua”
  - Color: “Café”
  - ….etc…
- Acciones:
  - Ladrar [“Guau Guau’]
  - Comer [“Chomp Chomp”]
  - Dormir [“Zzzzzzzz”]
  - …etc…

12

¿Cómo modelar un objeto real en un programa?

- Las "características" son PROPIEDADES o datos.
- Las "acciones" son METODOS u operaciones.

Objeto Perro "Real"

FIDO : Perro
Nombre: FIDO Raza: Chihuahua Color: Café
Ladrar() Comer() Dormir()

Abstracción de un objeto"Perro" en software

13

Abstracción

- Se refiere a "quitar" propiedades y métodos de un objeto y quedarse solo con aquellas que sean necesarias.

Objeto Perro "Real":
Propiedades:
(Nombre, Raza, Color, **etc.**)
Acciones o métodos:
(Ladrar, Comer, Dormir, **etc.**)

FIDO : Perro
Nombre: FIDO Raza: Chihuahua Color: Café
Ladrar() Comer() Dormir()

Abstracción de un "Perro"

Nótese que en la "Abstracción" del perro quitamos el "etc."

14

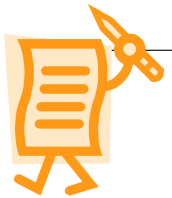
Clases y Objetos

- "FIDO" es UN "PERRO"
- "FIDO" es del TIPO "PERRO"
- "FIDO" es un OBJETO
- "PERRO" es la CLASE de "FIDO"

- "CHESTER" es OTRO "PERRO"
- "CHESTER" también es del TIPO "PERRO"
- "CHESTER" es otro OBJETO
- "PERRO" también es la clase de "CHESTER"

15

Clase



- Es una descripción de las características y acciones para un tipo de objetos.
- Una clase NO es un objeto. Es solo una plantilla o definición de objetos.

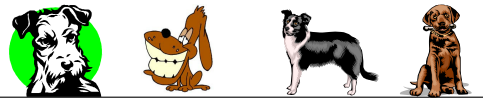
16

Clase

- Contiene todas las características comunes de ese conjunto de objetos
- Clase = Modelo = Plantilla = Esquema = Descripción de la anatomía de los objetos.
- A partir de una clase se pueden definir [crear] muchos objetos independientes con las mismas características.

17

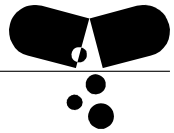
Objeto



- Unidad que combina datos y funciones.
  - Datos = Propiedades = Atributos = Características
  - Funciones = Métodos = Procedimientos =Acciones
- Los datos y funciones están **Encapsulados**.
- Posee un nombre único (identificador).
- Un objeto es del tipo de una clase
- "Un objeto es la instancia de una clase"

18

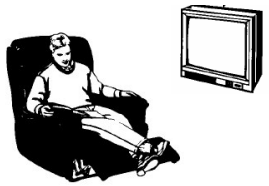
Encapsulamiento

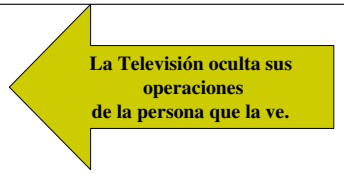


- Permite incluir en una sola entidad información y operaciones que operan sobre esa información.
- Permite:
  - Componentes públicos [Accesibles, Visibles].
  - Componentes privados [No accesibles, Ocultos].
  - Restricción de accesos indebidos.

19

Ejemplo: Encapsulamiento

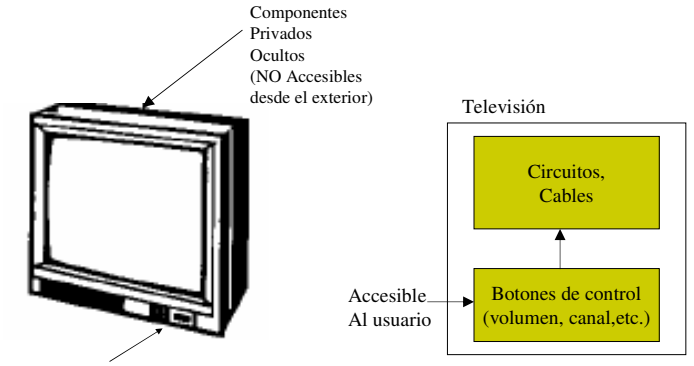




- Los objetos encapsulan lo que hacen. Ocultan la funcionalidad interna de sus operaciones, de otros objetos y del mundo exterior.

20

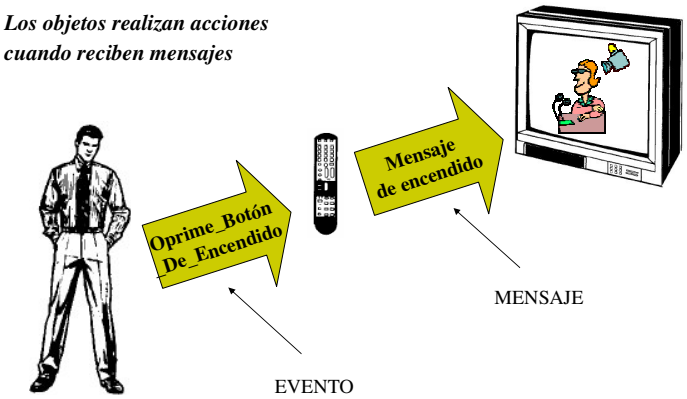
Ejemplo Encapsulamiento



21

Mensajes entre Objetos

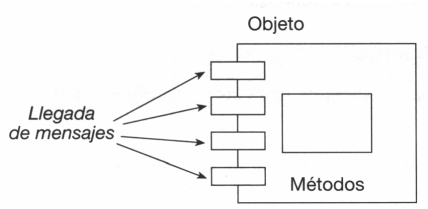
Los objetos realizan acciones cuando reciben mensajes



22

Mensajes: Comunicación entre objetos

- **Mensaje.-** Orden que se envía al objeto para indicarle realizar una acción.
- **Mensaje.-** Llamada a un método (o función) del objeto.



Al conjunto de mensajes a los cuales puede responder un objeto se llama **"Protocolo del Objeto"**

23

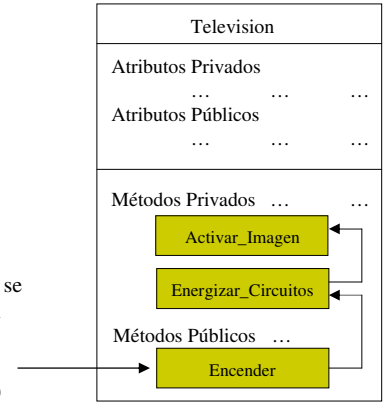
Anatomía de un mensaje

- Identidad del receptor
- Método que ha de ejecutar
- Información especial (argumentos o parámetros)

- Ejemplos:
  - miTelevision.Encender( )
  - miTelevision.Apagar( )
  - miTelevision.CambiarCanal(45)
  - miPerro.Comer("Croquetas")
  - miEmpleado.Contratar ("Juan", 3500)
  - miFactura.Imprimir( )

24

Ejemplo de envío de mensajes



1. El método **"Encender"** se invoca por un mensaje de otro objeto (una Persona)

3. **"Energizar_Circuitos"** Llama a **"Activar_Imagen"** enviándole un mensaje.

2. **"Encender"** llama a **"Energizar_Circuitos"** enviándole un mensaje.

25

Construyendo y destruyendo la imagen de la T.V.

Cada vez que se **enciende** la Television…
- Se deben energizar los circuitos
- Se debe activar el cinescopio

…Para posteriormente activar la imagen.

Cada vez que se **apaga** la Television…
- Se deben des-energizar los circuitos
- Se debe des-activar el cinescopio

…Para posteriormente apagar la imagen

26

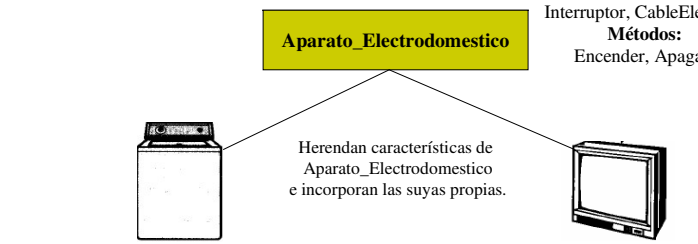
Constructores y Destructores

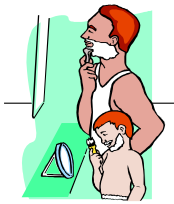
- Los objetos ocupan espacio en memoria; existen en el tiempo y deben crearse [instanciarse] y destruirse:
  - **Constructor.-** Operación que crea un objeto y/o inicializa su estado.

  - **Destructor.-** Operación que libera el estado de un objeto y/o destruye el propio objeto.

27

Herencia





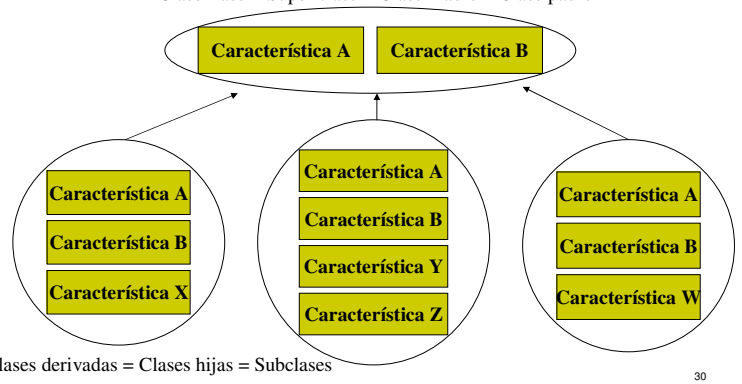
Herencia

- Capacidad para utilizar características previstas en antepasados o ascendientes.
- Permite construir nuevas clases a partir de otras ya existentes, permitiendo que éstas les "transmitan" sus propiedades.
- Objetivo: Reutilización de código.

29

Herencia - Jerarquía de clases

Clase Base = Super clase = Clase madre = Clase padre

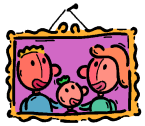


Clases derivadas = Clases hijas = Subclases

30

Tipos de Herencia

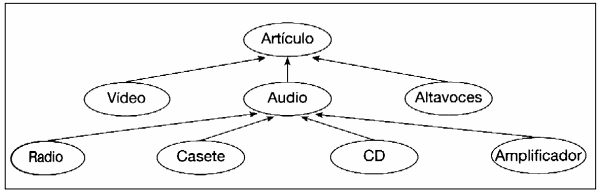
- **Herencia Simple.-** Una clase puede tener sólo un ascendiente. [Una subclase puede heredar de una única clase].



- **Herencia múltiple (en malla).-** Una clase puede tener más de un ascendiente inmediato. [Heredar de más de una clase].

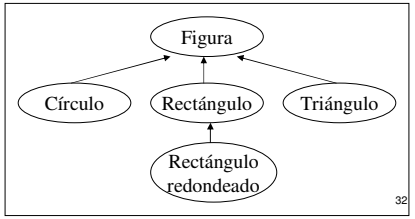
31

Herencia simple



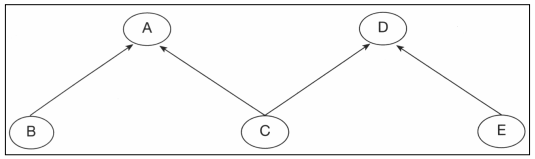
Ejemplo 1

Ejemplo 2

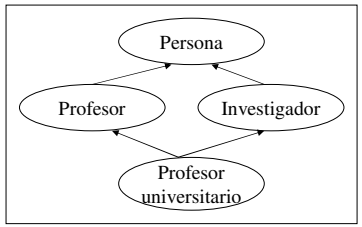


32

Herencia múltiple



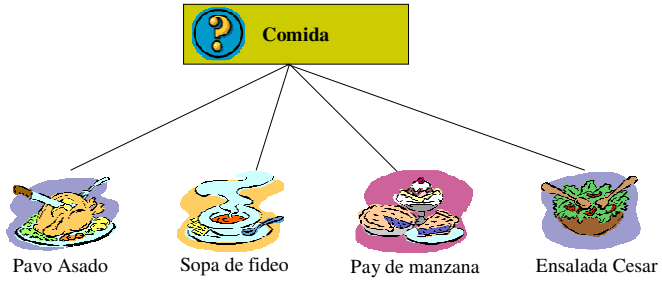
Ejemplo 1



Ejemplo 2

33

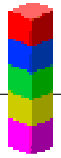
Clase abstracta



La "Comida" como tal, es solo un concepto abstracto que NO puede instanciarse.
Existen muchos alimentos que heredan sus características y
ellos SI pueden existir por si mismos.
"Comida" es una clase Abstracta.

34

Clase abstracta



- Es una clase que sirve como clase base común, pero NO puede tener instancias.
- Una clase abstracta solo puede servir como clase base (solo se puede heredar de ella).
- Sus clases "hijas" SI pueden tener instancias.

35

Anulación / Sustitución / Sobreescritura [Overriding]

Clase base

Articulos_Academicos
Propiedades: … … …
Metodos: **...** Abrir ( )

Herencia

miCarpeta y **miLibro** heredan el método **Abrir ( )** pero NO lo utilizan; sino que cada uno lo implementa nuevamente de manera distinta.

miLibro.Abrir ("Pagina", 15)

miCarpeta.Abrir ("Seccion", "Biologia", "Pagina", 3)

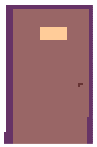
36

Anulación / Sustitución / Sobreescritura [Overriding]

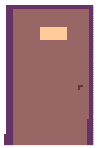
- Sucede cuando una clase “B” hereda características de una clase “A”, pero la clase “B” re-define las características heredadas de “A”.
- Propiedades y métodos pueden heredarse de una superclase. Si estas propiedades y métodos son re-definidos en la clase derivada, se dice que han sido “Sobreescritos”.

37

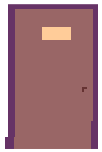
Sobrecarga [Overload]



miPuerta.Abrir (Adentro, Afuera)



miPuerta.Abrir (Afuera, Adentro)



miPuerta.Abrir ( )

38

Sobrecarga [Overload]

- Capacidad de utilizar el mismo nombre para un mismo método pero con diferentes firmas [número, orden y tipo de los parámetros].
- El código de programación asociado a cada sobrecarga puede variar.
- Ejemplo:
  - miEmpleado.Contratar(“Juan”, “Ventas”, 2500)
  - miEmpleado.Contratar(“Juan”)
  - miEmpleado.Contratar(“Juan”, 2500)

39

Polimorfismo

POLI = Múltiples MORFISMO = Formas

miCarpeta.Abrir(“Solo un poco”)

miCofre.Abrir( )

miSombrilla.Abrir( )

miCandado.Abrir(2, “Der”, 1, “Izq”)

miRefrigerador.Abrir(“Puerta de Abajo”)

miRegalo.Abrir( )

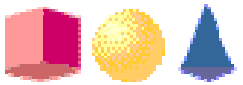
40

Polimorfismo

- Es el uso de un mismo nombre para representar o significar más de una acción.
- La sobrecarga es un tipo de Polimorfismo.
- Un mismo mensaje puede producir acciones totalmente diferentes cuando se recibe por objetos diferentes.
- Un usuario puede enviar un mensaje genérico y dejar los detalles de la implementación exacta para el objeto que recibe el mensaje.

41

Ejemplo práctico de modelado de objetos



42

Considerar el siguiente caso

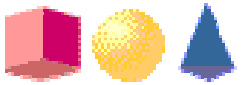
- Se desea realizar un sistema para capturar los datos de: Clientes, Empleados y Proveedores. Determinar las clases que utilizaría el sistema. Incluir Herencia.

Se considerarán los siguientes datos en la captura:

- CLIENTES:
  - Numero, Nombre, RFC, Direccion, Telefono, LimiteDeCredito
- EMPLEADOS:
  - Numero, Nombre, Direccion, Telefono, Salario, Antiguedad
- PROVEEDORES
  - Numero, Nombre, Direccion, Telefono, Email

43

Implementación del Modelo de Objetos en C#.NET



- Clases, Objetos, Propiedades, Métodos, Mensajes
- Constructores y Destructores
- Herencia
- Método Estático
- Sobrecarga

46

CLASES, OBJETOS, PROPIEDADES, METODOS, MENSAJES

```
class Programa
{
  public static void Main( )
  {
      Perro  unPerro = new Perro();
      unPerro.Nombre = "FIDO";
      unPerro.Raza = "Chihuahua";
      System.Console.WriteLine( unPerro.Nombre);
      System.Console.WriteLine( unPerro.Raza);
      unPerro.Comer ();
      unPerro.Ladrar ();
      unPerro.Dormir ();
  }
}
```

```
public class Perro
{           //VARIABLES INTERNAS
 private string nNombre;
 private string nRaza;

           //PROPIEDADES
 public string Nombre
 { get { return nNombre; }
   set { nNombre = value;}
 }
 public string Raza
 {  get { return nRaza;}
    set { nRaza = value;}
 }

           //METODOS
 public void Ladrar()
 {  System.Console.WriteLine ("...Guau Guau...");
 }
 public void Comer()
 { System.Console.WriteLine ("...Chomp Chomp...");
 }
 public void Dormir()
 { System.Console.WriteLine ("...Zzzzzzz....");
 }
}
```


Ejecución

```
FIDO
Chihuahua
...Chomp Chomp...
...Guau Guau...
...Zzzzzzz....
```


CONSTRUCTORES Y DESTRUCTORES

```
class Programa
{
  public static void Main( )
  {
      Perro  unPerro = new Perro();
      unPerro.Nombre = "FIDO";
      unPerro.Raza = "Chihuahua";
      System.Console.WriteLine( unPerro.Nombre);
      System.Console.WriteLine( unPerro.Raza);
      unPerro.Comer ();
      unPerro.Ladrar ();
      unPerro.Dormir ();
  }
}
```

```
public class Perro
{           //VARIABLES INTERNAS
 private string nNombre;
 private string nRaza;
           // CONSTRUCTOR
 public Perro()
 {         System.Console.WriteLine
                   ("....Se ha creado un perro...");
 }
           // DESTRUCTOR
 ~Perro ()
 {         System.Console.WriteLine
                   ("....Se ha destruido un perro...");

 }
           //PROPIEDADES
 public string Nombre
 { get { return nNombre; }
   set { nNombre = value;}
 }
 public string Raza
 {  get { return nRaza;}
    set { nRaza = value;}
 }
           //METODOS
 public void Ladrar()
 {  System.Console.WriteLine ("...Guau Guau...");
 }
 public void Comer()
 { System.Console.WriteLine ("...Chomp Chomp...");
 }
 public void Dormir()
 { System.Console.WriteLine ("...Zzzzzzz....");
 }
}
```


Ejecución

```
....Se ha creado un perro...
FIDO
Chihuahua
...Chomp Chomp...
...Guau Guau...
...Zzzzzzz....
....Se ha destruido un perro...
```

HERENCIA

```
using System;

class Programa
{
   // Programa Principal
   public static void Main()
   {
      Persona Persona1   = new Persona();
      Empleado Empleado1 = new Empleado();

      Persona1.Nombre = "Juan";
      Persona1.ImprimirNombre();

      Empleado1.Nombre = "Maria";
      Empleado1.Sueldo = 5000;
      Empleado1.ImprimirNombre();
      Empleado1.ImprimirSueldo();
   }
}
```

```
class Persona
{              // Variable local
   private string nombreLocal;

               // PROPIEDAD
   public string Nombre
   {  get { return nombreLocal;  }
      set { nombreLocal = value; }
   }

               //METODO
   public void ImprimirNombre()
   {    Console.WriteLine (" Me llamo " + nombreLocal);
   }
}

               //Clase Empleado hereda todo de la clase persona
class Empleado : Persona
{              //Variable Privada
   private double sueldoLocal ;

               //PROPIEDAD
   public double Sueldo
   {  get { return(sueldoLocal);  }
      set { sueldoLocal = value; }
   }

               // METODO
   public void ImprimirSueldo()
   {    Console.WriteLine (" Gano " + sueldoLocal);
   }
}
```

51

Ejecución

```
Me llamo Juan
Me llamo Maria
Gano 5000
```

52

Ejemplo de Método Estático

```
class Programa
{    public static void Main()
     {          ClaseEjemplo.MetodoEstatico();
     }
}

class ClaseEjemplo
{    public static void MetodoEstatico()
     {          System.Console.WriteLine (" ... Saludos....");
     }
}
```

```
EJECUCION:
... Saludos....
```

53

Ejemplo de Sobrecarga y Método Estático

```
class Programa
{     public static void Main( )
      {     Clase_Ejemplo.Saludar( );
            Clase_Ejemplo.Saludar ("Hola...con mi propio mensaje....");
      }
}

class Clase_Ejemplo
{     public static void Saludar( )
      { System.Console.WriteLine("...Hola!..Saludar( )...");
      }
      public static void Saludar(string Mensaje)
      { System.Console.WriteLine(Mensaje);
      }
}
```

```
EJECUCION
...Hola!..Saludar()...
Hola...con mi propio mensaje....
```

54

Bibliografía recomendada

- Fundamentos de programación: Algoritmos, Estructuras de datos y Objetos.
  Luis Joyanes Aguilar
  Mc Graw Hill
  Tercera Edición
  ISBN: 84-481-3664-0

- Superutilidades para C#
  Charles Wright, Kris Jamsa
  Mc Graw Hill
  ISBN: 84-481-3693-4

55