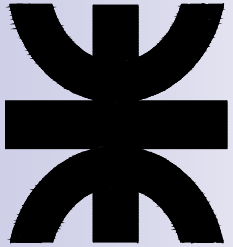


Universidad Tecnológica Nacional
Facultad Regional Resistencia

Paradigmas de Programación
Programación Orientada a Objetos

Modelo de Objetos. Contenidos.

- Fundamentos de la Programación Orientada a Objetos
- Objetos
- Programa
- Mensajes
- Abstracción y Encapsulamiento
- Clase
- Miembros de Clase y de Instancia
- Relaciones
  - Jerárquicas
  - Contractuales
- Redefinición
- Clases Abstractas
- Clases Concretas
- Sobrecarga

Modelo de Objetos. Grupos

- Grupo 1
  - Objetos y clases
- Grupo 2
  - Componentes de los objetos y Mensajes
  - Miembros de Clase y de Instancia
- Grupo 3
  - Relaciones de Herencia Simple y Múltiple. Conflicto de nombres.
- Grupo 4
  - Relaciones de Agregación o Ensamble.
- Grupo 5
  - Redefinición de métodos. Anulación o Sustitución
- Grupo 6
  - Sobrecarga
- Grupo 7
  - Clases Abstractas y clases concretas

Fundamentos de la Orientación a Objetos

- Causas que impulsan el desarrollo de la POO:
  - Los mecanismos de encapsulamiento de POO soportan un alto grado de reutilización de código, que se incrementa por sus mecanismos de herencia.
  - Aumento de **L**enguajes de **P**rogramación **O**rientados a **O**bjetos (LPOO).
- El paradigma OO se revela como el más adecuado para la elaboración del diseño y desarrollo de aplicaciones.
- Se caracteriza por la utilización del diseño modular OO y la reutilización del software.
- Especial énfasis en la abstracción (capacidad para encapsular y aislar la información del diseño y la ejecución).
- *Los objetos pasan a ser los elementos fundamentales en este nuevo marco.*

Orientación a Objetos
Definición

- *Conjunto de disciplinas que desarrollan y modelan software que facilita la construcción de sistemas complejos a partir de componentes mediante la modelización del mundo real.*

- *La Programación Orientada a Objetos es una metodología de diseño de software y un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (es decir, datos) y comportamiento (esto es, procedimientos o métodos).*

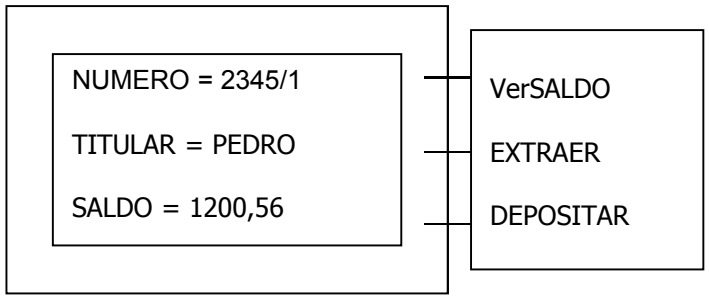
- *La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que se comunican entre ellos para realizar tareas.*

Propiedades más importantes

- **Abstracción**:
  - Una abstracción se centra en la vista externa de un objeto.
  - Separa el comportamiento esencial de un objeto de su implementación.
- **Modularidad**:
  - Propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas *módulos*), cada una las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.
  - Consiste en dividir un programa en módulos que se pueden compilar por separado, pero que tienen conexiones con otros módulos.
- **Encapsulamiento**:
  - Propiedad que permite asegurar que el contenido de la información de un objeto está oculta al mundo exterior: el objeto A no conoce lo que hace el objeto B y viceversa.
  - También se conoce como *ocultación de la información.*
- **Jerarquía**:
  - Propiedad que permite una ordenación de las abstracciones.
  - Las dos jerarquías más importantes de un sistema complejo son: *generalización/especialización* y *agregación*.
- **Polimorfismo** :
  - Propiedad que indica la posibilidad de que una entidad tome *muchas formas*.
  - Requiere *enlace dinámico*.

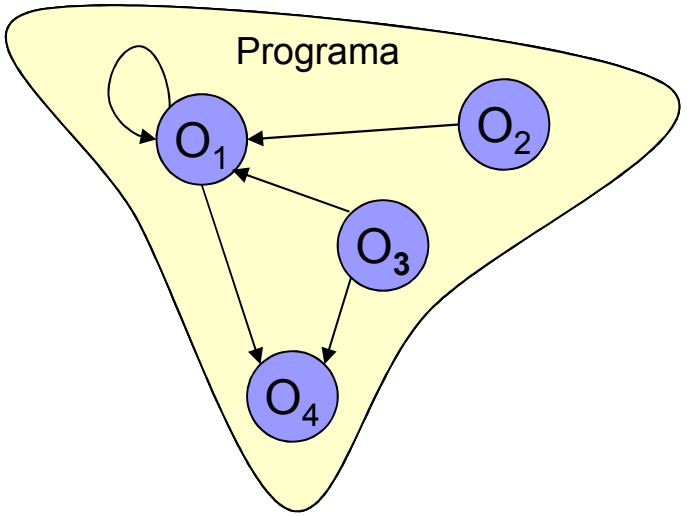
Objetos

- El elemento primordial del enfoque orientado a objetos es el objeto.
- Un objeto es una entidad que tiene un conjunto de responsabilidades y que encapsula un estado interno.
- Responsabilidades (o servicios) → se implementan mediante métodos
- Propiedades o atributos encapsulados → se implementan mediante valores.
- Un objeto tiene una IDENTIDAD, ESTADO y COMPORTAMIENTO



Programa Orientado a Objetos

- **Programa**
  - Un programa construido con el modelo de objetos es una colección de objetos cooperantes. La cooperación se lleva a cabo mediante mensajes.



- **Mensaje**
  - Un mensaje consiste en el nombre de una operación y de los argumentos por ella requeridos. Los mensajes constituyen el medio por el cual los objetos se comunican con el fin de lograr una acción cooperativa.

objeto.metodo (parámetros)

Abstracción y Encapsulamiento

- *Un objeto implementa:*
  - *"Una **abstracción:** denota las características esenciales de un objeto que lo distinguen de todas las otras clases de objetos y que por lo tanto proporcionan limites conceptuales bien definidos, con relación a la perspectiva del observador". [Booch]*
  - *"**Encapsulamiento:** es el proceso de esconder todos los detalles de un objeto que no contribuyen a sus características esenciales". [Booch]*

Clase

- Se dice que los objetos que comparten el mismo comportamiento pertenecen a la misma clase.
- Una clase es una especificación genérica para un número arbitrario de objetos similares.
- Se puede pensar a una clase como un molde para un tipo especifico de objetos.
- Las clases nos permiten describir en un lugar el comportamiento genérico de un conjunto de objetos, y entonces crear objetos que se comporten en esa forma cuando los necesitemos.

```
class Integer {
attributes:
  int i
methods:
  setValue (int n)
  Integer addValue (Integer j)
}
```

Objetos de una clase

- Los objetos que se comportan en la forma especificada por una clase se llaman instancias de esa clase.
- Todos los objetos son instancia de alguna clase.
- Una vez que se crea la instancia de la clase, se comporta como todas las otras instancias de su clase, capaz de realizar cualquier operación para la cual tenga métodos, inmediatamente después de haber recibido un mensaje.
- Puede también requerir a otras instancias, ya sea de la misma o de otras clases, que realicen otras operaciones en su nombre.
- Un programa puede tener tantas instancias de una clase en particular como sea necesario.
- Un objeto es una entidad concreta que existe en el tiempo y el espacio, una clase representa solo una abstracción, la "esencia" de un objeto.
- Las clases y los objetos son conceptos separados aunque están íntimamente relacionados.

Miembros de Clase y de Instancia

- Variables de instancia
  - Toda vez que se crea una instancia de una clase, el sistema crea una copia de cada variable de instancia de una clase.
  - Las mismas se acceden a través de cada objeto.
- Variables de clase
  - Generalmente se usa un modificador **static**.
  - El sistema almacena las variables de clase una sola vez. Una por clase sin tener en cuenta la cantidad de instancias que tenga la misma.
  - Todas las instancias comparten la misma copia de la variable de clase.
  - Se puede acceder a través de una instancia o de la clase misma.
  - Ej: Se pueden usar para guardar constantes porque como no pueden cambiar, solo se necesita una copia.

Miembros de Clase y de Instancia 2

- Métodos de instancia:
  - Estos métodos operan en las variables de la instancia del objeto actual, pero también tienen acceso a variables de clase.
  - Todos las instancias de una clase comparten la misma implementación de un método de instancia.
- Métodos de clase:
  - Algunos lenguajes usan un modificador **static**, otros tienen un bloque especial para su definición en las especificaciones de la clase.
  - No pueden acceder a las variables de instancia declaradas en la clase.
  - Pueden invocarse desde la clase.
  - No se necesita una instancia para llamar al método de la clase.
  - Ej: crear una clase con un método de clase que devuelva la fecha del sistema.

Relaciones

- **Jerárquicas**
  - Herencia: “es-un”
  - Ensamble: “todo-parte” ; “es-parte-de”

- **Contractuales, de colaboración o de asociación**: estas relaciones se dan cuando un objeto necesita para poder llevar a cabo en su comportamiento un servicio provisto por otro objeto.

Herencia

- Una clase comparte estructura interna y comportamiento definido en una o mas clases
- La herencia representa una jerarquía de abstracciones, en la cual una subclase hereda de una o más superclases.
- Es un mecanismo que acepta que la definición de una clase incluya el comportamiento y estructura interna definidos en otra clase más general.
- Hace posible la reutilización de código.
- Todos los servicios de una superclase están automáticamente disponibles para la subclase, sin necesidad de definirlos nuevamente.
- Se conoce también como generalización/especialización.

Herencia 2

- Una subclase hereda de su superclase:
  - Métodos
  - Variables de Instancia.
  - Acceso a variables de clase de la superclase.

- Una subclase contiene:
  - Variables y métodos heredados.
  - Nuevos métodos
  - Nuevas variables de instancia.
  - Métodos heredados cambiados.

Herencia. Tipos.

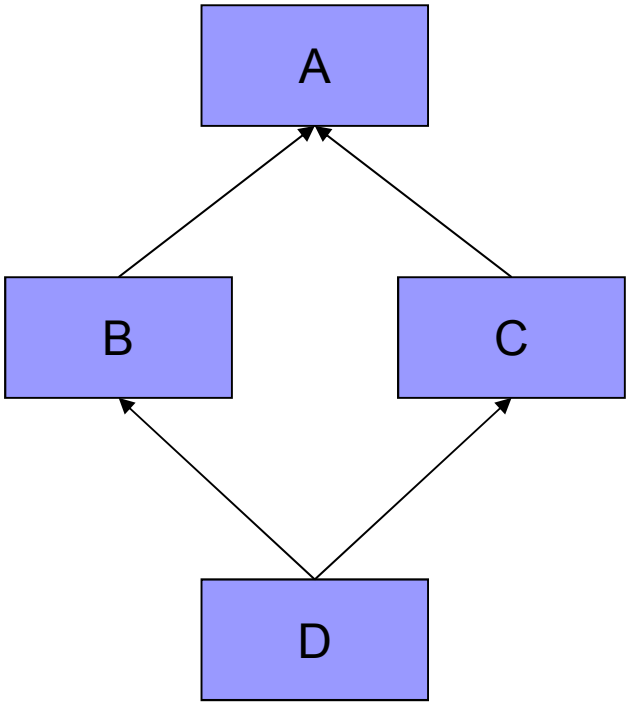
- Herencia **simple:** Es un mecanismo que permite que una clase A herede propiedades y métodos de una clase B. Diremos A hereda de B.
- Herencia **múltiple:** Una clase A hereda de más de una clase, diremos A hereda de B1, B2, …, Bn.
  - Ambos tipos deben agregar una pequeña porción de código por si sola para producir su propio y único tipo de objeto.

Herencia Múltiple. Conflicto de nombres.

- La herencia múltiple puede introducir lo que se denomina *conflicto de nombres en A*, si al menos dos de sus superclases (B1, B2, …, Bn), definen propiedades con el mismo nombre.

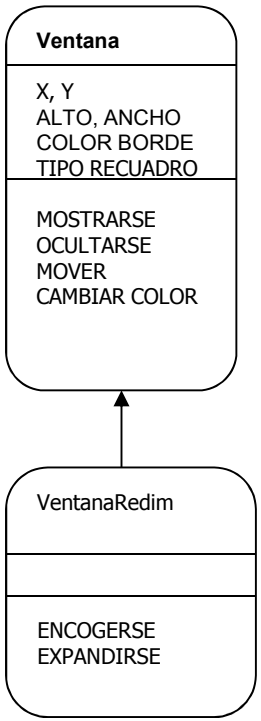
- Estos conflictos pueden resolverse de las siguientes formas:
  - El orden en que las superclases son enumeradas en la definición de la subclase, define la propiedad que será accesible. Las demás serán ocultadas.
  - Proveyendo una propiedad junto con la definición de los atributos que indica como se los usará desde sus superclases.

Herencia Múltiple. Conflicto de nombres.



- Este tipo de conflicto podría resolverse:
  - Heredando D con las propiedades de A
  - más las propiedades de B y C sin las propiedades que estas hereden de A.
  - Si B y C tienen propiedades con el mismo nombre, D vuelve a tener un conflicto de nombres debido a B y C.
- Heredar propiedades utilizando un camino (path) para indicar su origen. En este caso, D tendría dos copias de las propiedades de A; una heredada por A y otra por C.

Herencia. Ejemplo.



Todo objeto de la clase ventana tendrá dimensiones fijas especificadas en el momento de su creación y podrá MOSTRARSE, OCULTARSE, MOVERSE Y CAMBIAR DE COLOR.

Todo objeto de la clase VentanaRedim tendrá como estado interno heredado de ventana: X, Y, alto, ancho, color borde, tipo recuadro; y tendrá como comportamiento heredado: MOSTRARSE, OCULTARSE, MOVER, CAMBIAR COLOR. Agregará como comportamiento propio ENCOGERSE Y EXPANDIRSE. Es decir que, ventana redimensionable además de tener los mismos atributos y poder hacer las mismas cosas que ventana, puede también Encogerse y Expandirse.

Herencia. Respuesta a mensajes.

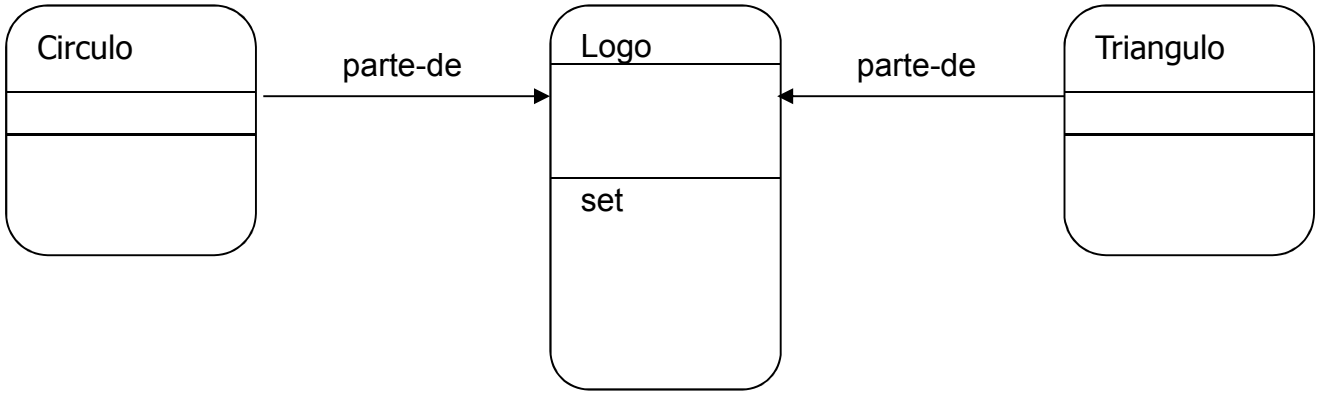
- El objeto recibe un mensaje.
- Se busca el mensaje en la clase del objeto receptor.
- Si el mensaje es encontrado, se ejecuta el método correspondiente.
- Si por el contrario, no se encuentra el mensaje en la clase del objeto, se lo busca en la superclase. Si no se lo encuentra en la superclase se lo busca en la superclase de la superclase, y así sucesivamente.
- Si finalmente el mensaje no es encontrado, se detiene la ejecución y se da un mensaje de error porque el objeto no entiende el mensaje.

Ensamble

- ***Es la relación "todo-parte" o "es parte de" en la cual los objetos que representan los componentes de algo son asociados a un objeto que representa el todo. [Rumbaugh]***

- Se dan entre objetos. Un objeto compuesto esta formado por objetos componentes que son sus partes.
- Cada objeto parte tiene su propio comportamiento y el objeto compuesto es tratado como una unidad cuya responsabilidad es realizar la coordinación de las partes.
- Las partes pueden o no existir fuera del objeto compuesto o pueden aparecer en varios objetos compuestos.

Ensamble. Ejemplo




```
class Logo {
attributes:
  Circulo circ
  Triangulo tri

methods:
  set (Punto p)
}
```

Relaciones Contractuales o de Colaboración

- Una empresa es un ensamble de divisiones, que a su vez están compuestas de departamentos.

- Una empresa no es un ensamble de empleados, puesto que empresa y personas son objetos independientes de la misma estatura.

Redefinición.

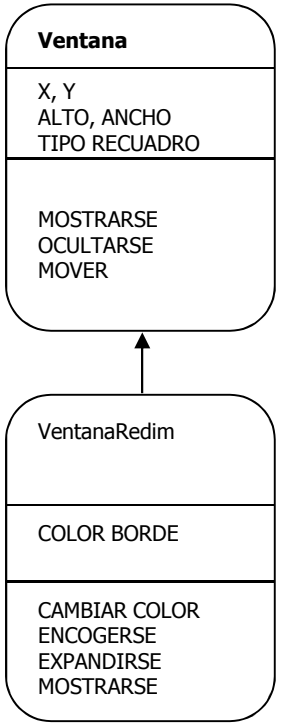
- Una subclase puede redefinir un método de la superclase simplemente definiendo un método con el mismo nombre.
- El método en la subclase redefine y reemplaza al método de la superclase.

- Para Qué?
  - Para especificar comportamiento que depende de la subclase
  - Para lograr mayor performance a través de un algoritmo más eficiente para la subclase o guardando un atributo de calculo dentro de la estructura interna del objeto, etc.

- ***Toda redefinición debe preservar la cantidad y tipo de parámetros utilizados en un mensaje y el tipo del valor devuelto.***

- ***Nunca debe redefinirse un método de manera que sea semánticamente inconsistente con el originalmente heredado.***

Redefinición. Ejemplo.



Todo objeto de la clase **Ventana** tendrá dimensiones fijas especificadas en el momento de su creación y podrá MOSTRARSE, OCULTARSE, MOVERSE. Además, el color del borde será el standard, y no podrá modificarse.

Todo objeto de la clase **VentanaRedim** agregará al estado interno heredado: color borde, y agregará al comportamiento heredado: CAMBIAR COLOR, ENCOGERSE y EXPANDIRSE. Y por ultimo, deberá redefinir el mensaje MOSTRARSE, debido a que el método heredado de su superclase no utiliza el atributo color borde para graficar la ventana.

Clases Abstractas

- Las clases que no han sido creadas para producir instancias de sí mismas se llaman clases abstractas.
- Existen para que el comportamiento y estructura interna comunes a una determinada variedad de clases pueda ser ubicado en un lugar común, donde pueda ser definido una vez (y más tarde, mejorado o arreglado de una sola vez, si es necesario).
- Las clases abstractas especifican en forma completa su comportamiento pero no necesitan estar completamente implementadas. Puede también especificar métodos que deban ser redefinidos por cualquier subclase.

Métodos Virtuales en clases Abstractas

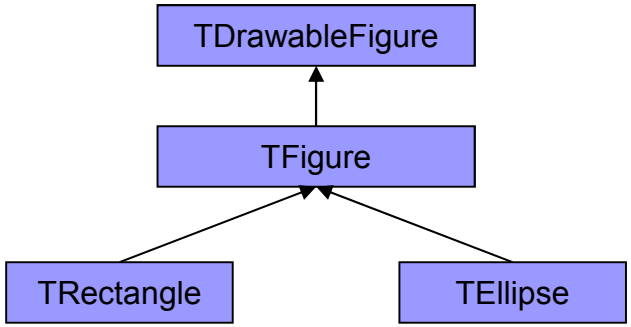
```
type

  TDrawableFigure = class
    procedure Draw; virtual;
  end;

  TFigure = class (TDrawableFigure)
    function Perimetro; virtual;
  end;

  TRectangle = class (TFigure)
    procedure Draw; override;
    function Perimetro; override;
  end;

  TEllipse = class (TFigure)
    procedure Draw; override;
    function Perimetro; override;
  end;
```




```
var
  Figure: TFigure;

begin
  Figure := TRectangle.Create;
  Figure.Draw;  // llama a TRectangle.Draw
  Figure.Destroy;
  Figure := TEllipse.Create;
  Figure.Draw;  // llama a TEllipse.Draw
  Figure.Destroy;
end;
```

Clases Concretas

- Las subclases concretas heredan el comportamiento y estructura interna de sus superclases abstractas, y agregan otras habilidades especificas para sus propósitos.
- Puede ser que necesiten redefinir la implementación por defecto de sus superclases abstractas.
- Estas clases completamente implementadas crean instancias de sí mismas.

Sobrecarga de métodos

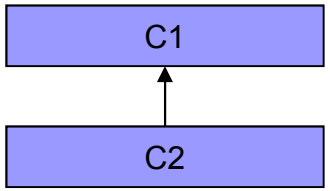
- Es la habilidad de dos o mas objetos de responder a un mismo mensaje, cada uno de su propia forma.
- La sobrecarga de métodos puede darse en una misma clase.
- Dos clases distintas pueden tener métodos con el mismo nombre.
- Ejemplo:
  - En un sistema bancario, donde tanto la clase CUENTA-CORRIENTE como la clase CAJA-DE-AHORRO responden a los mensajes EXTRAER(monto) o SALDO().

Sobrecarga de métodos. Ejemplo.

```
type

  C1 = class (TObject)
    procedure Test(I: Integer); overload; virtual;
  end;

  C2 = class (T1)
    procedure Test(S: string); reintroduce; overload;
  end;
```




```
SomeObject := C2.Create;
SomeObject.Test('Hola!');   // llama a C2.Test
SomeObject.Test(7);         // llama a C1.Test
```